Enhancing the Sensing Capabilities of Mobile and Embedded Systems

Daniel George Graham

Williamsburg, Virginia

Bachelor of Science, University of Virginia, 2010
Masters of Engineering, University of Virginia, 2011

A Dissertation presented to the Graduate Faculty
of the College of William and Mary in Candidacy for the Degree of
Doctor of Philosophy

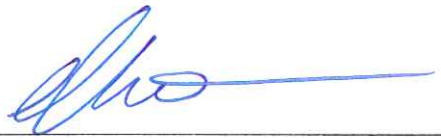
Department of Computer Science

The College of William and Mary
May 2016

APPROVAL PAGE

This Dissertation is submitted in partial fulfillment of
the requirements for the degree of

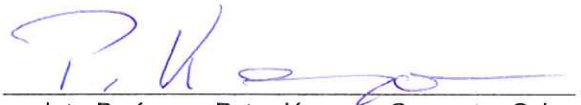
Doctor of Philosophy



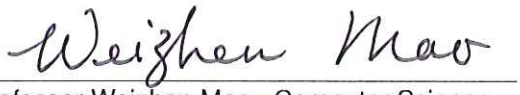
Daniel George Graham

Approved by the Committee, December 2015

Committee Chair
Associate Professor Gang Zhou, Computer Science
The College of William and Mary

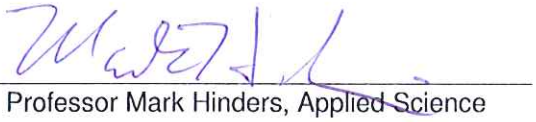


Associate Professor Peter Kemper, Computer Science
The College of William and Mary



Professor Weizhen Mao , Computer Science
The College of William and Mary

Associate Professor Haining Wang, Computer Science
The College of William and Mary



Professor Mark Hinders, Applied Science
The College of William and Mary

ABSTRACT

In this work, we aim to develop new sensors and sensing platforms that facilitate the development of new mobile and embedded devices. Mobile and embedded devices have become an integral part of our everyday lives and the sensing capabilities of these devices have improved throughout the years. Developing new and innovative sensors and sensing platforms will provide the building blocks for developing new sensing systems. In an effort to facilitate these innovations we have developed two new in-air sonar sensors and a new reconfigurable sensing platform. The first in-air sonar sensor is designed for ranging applications and uses the phone's microphone and rear speaker to generate a wide beam of sound. The second in-air sonar sensor is an external module which uses a narrow beam of sound for high resolution ranging. This ranging information is then combined with orientation data from the phone's gyroscope,magnetometer and accelerometer to generate a two dimensional map of a space. While researching ways of enhancing the sensing capabilities of mobile and embedded devices, we found that the process often requires developing new hardware prototypes. However, developing hardware prototypes is time-consuming. In an effort to lower the barrier to entry for small teams and software researchers, we have developed a new reconfigurable sensing platform that uses a code first approach to embedded design. Instead of designing software to run within the limited constraints of the hardware, our proposed code-first approach allows software researchers to synthesize the hardware configuration that is required to run their software.

TABLE OF CONTENTS

Acknowledgments **iv**

Dedication **v**

List of Tables **vi**

List of Figures **vii**

1 Introduction **1**

2 A Software Sonar Ranging Sensor for Smartphones **3**

2.1 Introduction 3

2.2 Background 5

2.3 Related Work 6

2.4 Design 10

2.4.1 Generating the Signal 10

2.4.1.1 Pulse Compression 11

2.4.1.2 Windowing the Pulse 14

2.4.2 Capturing the Signal 14

2.4.2.1 Sampling Constraints and Hardware Requirements . . 15

2.4.2.2 The Concurrency Problem 15

2.4.3 Processing the Signal 16

2.4.4 Frequency Domain Optimization 16

2.4.5 Peak Detection and Reverberation 19

2.4.6 Increasing the minimum detection range . . . 20
2.4.7 Temperature's Impact on the Speed of Sound . . . 21
2.4.8 smartphone Application . . . 23
2.5 Performance Evaluation . . . 24
2.5.1 Evaluating the Impact of Temperature and Reverberation . . . 24
2.5.2 Evaluating the Detection of multiple Objects . . . 28
2.5.3 Evaluating Real-time Performance and System Usage . . . 29
2.6 Conclusion . . . 31

3 A SONAR Ranging Attachment for 2D Mapping 32
3.1 Introduction . . . 32
3.2 Related Work . . . 35
3.3 Design Section . . . 36
3.3.1 External Attachment . . . 37
3.3.1.1 Modulator . . . 38
3.3.2 Demodulation . . . 39
3.3.2.1 Demodulation Algorithm . . . 41
3.3.3 2D Map Generation . . . 43
3.3.4 Possible Applications . . . 43
3.4 Evaluation . . . 44
3.4.1 Evaluating Accuracy . . . 44
3.4.2 Evaluating Resolution . . . 46
3.4.3 Sources of Error . . . 46
3.4.4 Energy Consumption . . . 47
3.4.5 Testing other devices . . . 47
3.4.6 Testing in an irregular environment . . . 49
3.5 Conclusion . . . 49

4 A code first approach to designing embedded systems 50

4.1 Introduction 50

4.2 Related Work 53

4.3 Software Design 55

4.3.1 A Code-First Approach to Embedded System Design 56

4.3.2 Modular Middleware Architecture for Embedded Systems . . . 57

4.4 Configuration Generation Process 59

4.4.1 Introducing Mathematical Abstraction and Notation 60

4.4.2 Abstracting Libraries 61

4.4.3 Specifying the Constraints 63

4.4.4 Automating Microcontroller Selection 67

4.4.5 Generating the Hardware Configuration 67

4.4.6 Automating the Process 69

4.4.6.1 Parsing Step 70

4.4.6.2 Constraint Checking Step 70

4.4.6.3 Configuration Generation Step 71

4.4.6.4 Schematic Generation Step 71

4.5 Evaluation 72

4.5.1 Step Counter 72

4.5.2 Environment Monitoring Smart watch 73

4.5.3 Infrared Indoor Localization Device 74

4.5.4 System's Limitations 75

4.5.5 Considerations for a Commercial Solution 76

4.6 Conclusions 77

5 Conclusion and Future Work 78

ACKNOWLEDGMENTS

I would like to thank all the people who supported me throughout this process. I would especially like to thank my Dad (Errol Graham) for taking the time to proof read my writing, my Mom (Angelique Graham) for her kind counsel and my sister (Dominique Graham) for her loving support. I would also like to thank, Dr. Denys Poshyvanyk, for his support during my first two years of research and Dr. Jason McDevitt, for helping to fund this research. I would also like to thank Dr. Michael Tierney and Jason Pully for employing me while I wrote my dissertation. I would also like to thank my co-authors: Jeffery Buffkin, David Nguyen, Ed Novak and George Simmons. I would also like to say thank you to: Shadrack Antwi, Jacqulyn Johnson, Jesse Laeuchli and the Tans, for their friendship and my girlfriend, Raquel Plummer, for her encouragement and support. Finally, I would like to thank my adviser, Dr. Gang Zhou, for his guidance and patience during the years. I have learned a great deal from him and this body of work would not have been possible without him.

To my parents: Angelique and Errol Graham, and to my sister Dominique Graham

LIST OF TABLES

2.1 Chirp properties table 12

3.1 Compatible Devices. 48

4.1 An example of the superset P' 62
4.2 An example of the superset S' 62
4.3 An example of library mappings 68
4.4 An example of the port mappings 68
4.5 The systems and associated Libraries 75
4.6 The systems and associated libraries 75

LIST OF FIGURES

2.1 This figure shows an overview of the process that the sonar system uses to calculate the distance from the system to an object. 6

2.2 Figure (a) shows the original pulse that was transmitted. Figure (b) shows the signal that is received by the microphone, and Figure (c) shows the result of the filtering process. These figures are illustrative and have been simulated using a sample rate of 44.1kHz 7

2.3 The figure shows an overview of the sonar system's architecture. . . . 11

2.4 These figures show the time and frequency domain representation for the windowed and unwindowed linear chirp 13

2.5 Figure (a) shows the noisy signal that was captured at approximately 2.5 meters from the wall. Figure (b) shows the filtered signal. Figure (c) shows the result of applying the envelope detection algorithm. Figure (d) shows the detection threshold values applied to another sample taken at 1.5 meters. A Nexus 4 smartphone was used to collect these readings by measuring the distance to an outdoor wall on a college campus. 18

2.6 The figure shows the error in meters vs. the threshold value. These readings were collected using a Nexus 4 smartphone. 20

2.7 The figures show an example of how pulse compression helps reduce the minimum distance at which objects can be detected. Figure (a) shows the combination of the two pulses that are received by the microphone. The first pulse is a linear chirp 4kHz to 8kHz with a sweep time of 10ms while the second pulse has the same parameters but is attenuated by a factor of 0.4. Figure (b) shows the result of the filtering process. All signals shown in this figure were sampled at a rate of 44.1kHz. 22
2.8 Figure (a) shows a screen shot of the sonar sensor application running on the Nexus 4. Figure (b) shows the experimental setup that was used in the evaluation 23
2.9 These figures show pictures of all three environments. Figure (a) shows the outdoor environment. Figure (b) shows the large carpeted indoor classroom and figure (c) shows the small indoor room 3.2m by 3.1m. 26
2.10 These figures show the measurements that were taken in all three environments. Figure (a) shows the results for the outdoor environment. Figure (b) shows the results for the large carpeted indoor classroom and figure (c) shows the results for the small indoor room 3.2m by 3.1m. 27
2.11 This figure shows a picture of three the wall that form the sides of the wheel chair ramp 29

2.12 Figure (a) shows the output of the filtering process when the readings were taken directly facing the wall. Figure (b) shows the output of the filtering process when the readings were taken at an oblique angle to the wall. The horizontal line represents a detection threshold of $2 * 10^{10}$. These readings underestimate the target since they are not temperature adjusted, the speed of sound is assumed to be 330 m/s 29

2.13 The figure shows the amount of time that it takes to obtain a single sonar reading. No Opt represents not optimizations. Opt 1 represents the Frequency domain optimization, Opt 2 represents caching the FFT of the pulse and Opt 3 represents limiting the number of samples 30

3.1 Figure (a) is a picture of the $3.3V$ battery, the modulator circuit, and the male $3.5mm$ male headphone jack adapter. Figure (b) is a picture of the transducer module. 33

3.2 Shows a system diagram of the proposed SONAR system, which is comprised of: 1) an external SONAR module; 2) a commodity smartphone; and 3) the software application running on the phone. . . 37

3.3 Shows the signal that is reconstructed when the pulse width modulated signal is fed directly to the phone. 38

3.4 Shows a circuit schematic of the modulation circuit. RC is the label for the received carrier. PW is the label for the pulse-width modulated signal. MIC is the label for the output of the modulator that is connected to the microphone input on the phone. 39

3.5 Figure (a) shows the carrier frequency that is generated on the phone, (b) shows the pulse that is generated by the transducer module, and (c) shows the modulated signal which encodes the distance information. These measurements were taken using an object that was 43 cm away from the sensor. 40
3.6 Shows the 8000 samples obtained from a single reading of the buffer. . 41
3.7 Shows a single reading from the buffer, i.e. x_1 from figure 3.6. 41
3.8 A screen shot of the mapping application showing the information on distance (in inches), direction (in degrees, with 0° at North), pitch (in degrees, with 0° at horizontal, i.e. to the ground), and the resulting polar plot. 43
3.9 Figures (a), (b), and (c) show the results for a small room, large room, and outside. Ten measurements were taken at each distance; the error bars represent the standard deviation of these measurements. A_s represents the average error for software SONAR module and A_h the average error for the software module 44
3.10 Figures (a), (b) and (c) show the polar values obtained by the external SONAR module in a small room, a large room and outdoor environment, respectively. The application divides the compass into eighteen 20 degrees sections. A measurement taken in a given section will associate that value with the center of that section. All measurements shows are in meters and all angles are in degrees 45
3.11 Shows a picture of the stairwell in which the measurements where taken 48
3.12 This figure shows the map that was reconstructed from the measurements taken in the stairwell. All measurements are in meters and directions are in degrees. 48

4.1 Shows the layout of the mainboard (E-unit) 51
4.2 An overview of the middleware architecture 57
4.3 An example program written using the modular middleware that displays the temperature on the LCD. 60
4.4 Shows an example of a port conflict on the UART port of the microcontroller reference design. BLE represents a bluetooth module. 61
4.5 Modification of indoor temperature sensor to include GPS and Bluetooth. For this example we assume the implementation of GPS module and it corresponding library. 66
4.6 Figure (a) shows a temperature sensor with LCD display. Figure (b) shows an overview of the steps in automating the synthesis process. . 69
4.7 Figure (a) (b) (c) show the different possible configurations of the platform. Figure (d) shows a screenshot of the accompanying Android application 72
4.8 Figure (a) shows the results of the case study, in which two participants (Person1, Person2) wore both the prototype and a Fitbit for two days. Figure (b) shows the results of the localization experiment. When the signal is high it indicates that the prototype received the signal and when the signal is low it indicates that the prototype has not received the signal. Figures (c) and (d) show graphs of the light and temperature readings collected over a 1 hour period at 10 minute intervals. During the first 50 minutes the device was placed indoors and during the final 45 minutes the device was placed outside. 73

Enhancing the Sensing Capabilities of Mobile and Embedded Systems

Chapter 1

Introduction

Mobile and embedded devices have become an integral part of our everyday lives and the sensing capabilities of these devices have improved throughout the years. Developing new and innovative sensors and sensing platforms will provide the building blocks for developing new sensing systems. In this work, we explore ways of enhancing the sensing capabilities of mobile and embedded devices, by focusing on designing new sensors and sensing platforms. In particular we developed two in-air sonar sensors and a reconfigurable sensing platform.

The first in-air sonar sensor is a software based sonar ranging sensor that uses the phone's microphone and rear speaker. The system can be decomposed into three steps: a signal generation step, a signal capture step and a signal processing step. During the signal generation step the system creates an encoded pulse known as a linear chirp. This chirp is then passed to the phone's rear speaker which oscillates and creates a pressure wave that travels through the air until it encounters an object. Once the wave encounters an object it is reflected. During the signal capture step the reflected wave is recorded and converted to a vector. Once the signal has been captured it is processed in the final step and the distance to the object is calculated by measuring the elapsed time between the initial pulse and its reflection. Designing a software based sonar sensor for smartphones presents a collection of unique challenges, including: concurrent management of hardware

buffers, real-time processing and hardware sampling limitations. These challenges and their solutions are discussed in chapter 2.

Though sonar ranging using smartphones is interesting, it would be more interesting to be able to generate a two dimensional map of an environment using smartphone based in-air sonar. However, the sound beam produced by the software sonar ranging sensor is too wide for mapping applications. To mitigate this, we developed an external sonar module that plugs into the headphone jack of a smartphone. The readings from the module are combined with information from the phone's compass and gyroscope to construct a two dimensional map of a space.

In addition to exploring ways of enchaining the sensing capabilities of mobile devices, we also explored ways of enchaining the sensing capabilities of embedded devices. During our exploration we found that research in embedded sensing is limited by the constraints of the available hardware platforms, since software developers are only able to develop software that runs within the limited constraints of the hardware. Furthermore, developing new hardware platforms for specific sensing applications is time consuming. In this work we attempt to address the problem by presenting a code-first approach to the design of embedded systems. A code-first approach allows a software developer to design an embedded system by first developing the software that the system is required to run. Once the software has been developed an interpreter is used to generate the hardware configuration that is required to run the application. This lowers the barrier to entry for software developers by abstracting the underlying hardware.

The rest of this dissertation is structured as follows. Chapter 2 discusses the software based sonar sensor. Chapter 3 discusses the design of the external sonar module. Chapter 4 discusses our proposed code-first approach to embedded design, before concluding the work in Chapter 5.

Chapter 2

A Software Sonar Ranging Sensor for Smartphones

2.1 Introduction

Sensors on mobile devices have allowed developers to create innovative mobile applications. For example, the use of GPS localization allows developers to create applications that tailor their content based on the user's location [60]. Other sensors such as the proximity sensor help to improve the user's experience by disabling the touch screen when it detects that the user has placed the phone next to his or her ear. This prevents buttons from accidentally being pressed during a phone call [49]. Since the release of Android 1.5, Google has added application program interface (API) support for eight new sensors [13]. These sensors include: ambient temperature sensors, ambient pressure sensors, humidity sensors, gravity sensors, linear acceleration sensors and gyroscopic sensors.

Developing new and innovative sensors for smartphones will help open the field to new possibilities and fuel innovation. In particular, developing sensors that allow smartphones to perceive depth is key. Google's Advanced Technology and Projects Team share this vision. They are currently working on a smartphone that uses custom hardware to perceive depth. The project is nicknamed: "Project Tango" [3]. Engineers at NASA have also

partnered with the Google team to attach these phones to robots that will be sent to the international space station [5]. However, Google's structured light sensor does not perform well in outdoor environments, since the light from the sun interferes with the sensor. In this chapter we explore the possibility of using sonar to provide depth sensing capabilities in both indoor and outdoor environments and address two unique research questions: *1) How do we design a sonar sensor for smartphones using only the phone's existing hardware? 2) How do environmental factors such as noise, reverberation and temperature affect the sensor's accuracy?.*

The proposed sonar sensor uses the smartphone's rear speaker and microphone, and implements the sonar capabilities on a software platform. The software process is comprised of three major steps: 1) a signal generation step, 2) a signal capture step, and 3) a signal processing step. During the signal generation step, the phone's rear speaker emits a pulse. The pulse forms a pressure wave which travels through the air until it encounters an object, which then reflects the pulse and scatters it in multiple directions. During the signal capture step, the microphone captures the reflected pulse, and the distance to the object is determined by calculating the time between the pulse and its reflection.

However, factors such as noise and multipath propagation negatively affect the system's ability to accurately identify the reflected pulse. To address this we use a technique called pulse compression. Pulse compression is the process of encoding the pulse with a unique signature. This unique signature makes it easier to distinguish the pulse from external noise [45]. The pulse is recovered by calculating the cross correlation between the noisy signal and the pulse.

In addition to being corrupted by noise, a pulse may sometimes overlap with another pulse. This occurs when objects close to the system begin to reflect parts of the wave while it is still being transmitted. This limits the minimum distance at which an object can be detected. Encoding the pulse helps to reduce this distance by allowing the filtering process to distinguish between overlapping pulses.

The sonar sensor was evaluated using three metrics: accuracy, robustness, and real-

time performance. The accuracy of the sonar sensor was evaluated by comparing the distances reported by our sensor with known distances. The sensor accurately measured distances within 12 centimeters. The robustness of the sensor was evaluated by comparing the sensor's accuracy under different noise and reverberation conditions in different environments. Finally, the sensor's real-time performance was evaluated by measuring the time that it takes to process a signal and return a measurement when different optimizations are applied. By using a collection of optimizations we were able to reduce the processing time from 27 seconds to under two seconds.

In-air sonar has been extensively studied in the literature and supports a vast array of sensing capabilities beyond simply ranging. State of the art systems can determine the 3D positions of objects [15] and can even ascertain properties of these objects [34]. However, these techniques cannot simply be ported to smartphones. Implementing these techniques on smartphones presents a collection of unique challenges and therefore requires a measured and systematic approach. In this chapter we begin by exploring ranging applications

2.2 Background

A sonar system can be decomposed into three steps. Figure 2.1 shows a simulated example of these steps. During the first step, the system generates a pulse. This pulse travels through the air until it encounters an object. Once the pulse encounters an object, it is reflected by the object. These reflected waves then travel back to the system which records the reflected pulse. The time difference between the initial pulse and the reflected pulse is used to calculate the distance to the object. Since the speed of sound in air is known, the distance to an object can be calculated by multiplying the time difference between the initial pulse and the reflected pulse by the speed of sound, and dividing the result by two. We need to divide by two because the time difference between the reflected pulse and the initial pulse accounts for the time that it takes the wave to travel from the phone to the

object and back.

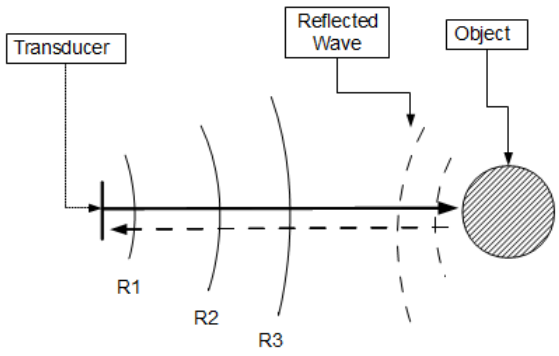


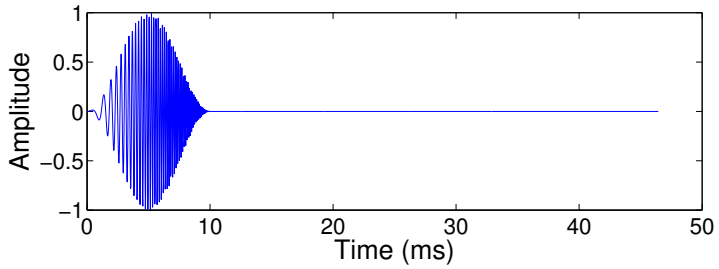
Figure 2.1: This figure shows an overview of the process that the sonar system uses to calculate the distance from the system to an object.

The reflected pulse will contain noise from the environment. This noise is reduced by filtering the signal. Figure 2.2 shows the signals that are generated or recorded at each step.

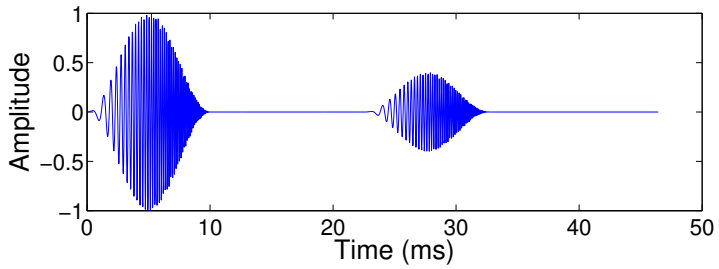
2.3 Related Work

Figure 2.2(a) shows the signal that is transmitted from the phone's rear speaker, while figure 2.2(b) shows the signal that is received by the phone's microphone. In figure 2.2(b) the received signal contains both the initial pulse and the reflected pulse, the phone's microphone will pick up both the transmitted signal and its reflection. This is common in sonar systems where both the transmitter and receiver are located close to each other. Such sonar systems are called monostatic sonar systems. In figure 2.2(b) the second pulse represents the reflected pulse. Figure 2.2(c) shows the output of the filtering process. The peaks in the resulting filtered signal correspond to the location of the pulse in the original signal. The filtering process will be discussed in detail in section 2.4.4.

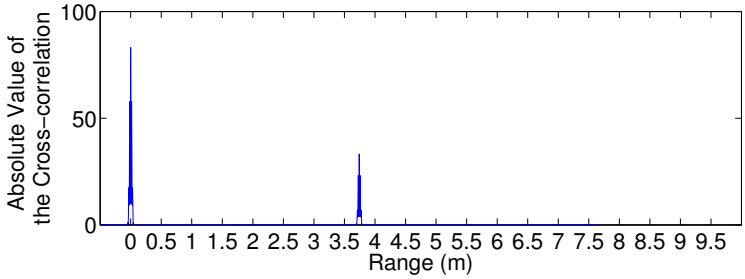
In figure 2.2(b) the reflected signal has a smaller amplitude than the initial pulse because some of the energy has been lost. This is because as sound travels through free space its energy per square meter dissipates as a fixed amount of energy gets spread over



(a) Transmitted pulse



(b) Sound recorded



(c) Result of filtering process

Figure 2.2: Figure (a) shows the original pulse that was transmitted. Figure (b) shows the signal that is received by the microphone, and Figure (c) shows the result of the filtering process. These figures are illustrative and have been simulated using a sample rate of 44.1kHz

a larger surface area. This dissipation is governed by the inverse wave square law [38]. Figure 2.1 provides a visual explanation of the inverse wave square law. As the wave moves from location $R1$ to $R3$, its energy density decreases since the same amount of energy is spread over a larger surface area.

As the wave travels further from the transmitter, its power density decreases. If an object is too far away, the energy density of the wave that encounters the object may not

be enough to generate a reflected wave that can be picked up at the receiver. Distance is not the only factor in determining the amount of energy that is reflected. The amount of energy that is reflected is also determined by the composition and cross section of the object. Larger objects have larger cross sections and therefore reflect more energy, while smaller objects have smaller cross sections and therefore reflect less energy. Because objects with larger cross sections reflect more energy, they can be detected at larger distances. However, objects with smaller cross sections can only be detected at smaller distances because they reflect less energy. Another key insight for sonar systems is that large flat surfaces act like mirrors and mostly reflect sound waves in the direction of their surface normal. This property is known as the mirror effect.

Sonar systems attempt to accommodate for these limitations by designing special speakers and microphones. To improve the range of sonar systems, speakers are designed so that they focus the speaker's output. The concept of focusing the speaker's output is known as the speaker's gain. Focusing the speaker's output allows sound waves to travel further, in a specified direction. Sonar systems also attempt to improve their range by being able to pick up weaker signals. Just as objects with large surface areas are able to reflect more energy, microphones with large surface areas are able to receive more energy. The concept of a microphone's surface area is known as the microphone's aperture. Once the wave reaches the microphone, it is only able to pick up a subsection of the waves energy. Sonar systems use an array of microphones to increase the receiving surface area thus increasing the microphone's aperture. Now that we have developed an intuition for sonar systems, we will compare the state of the art in-air sonar systems with our proposed system, highlighting the key differences and technological challenges that arise when implementing a sonar system on a smartphone.

In 1968 D. Dean wrote a paper entitled "Towards an air Sonar" in which he outlined some of the fundamental challenges of designing in-air sonar [27]. These challenges included acoustic mismatch and wind effects. Since Dean's paper several in-air sonar systems, have been develop for a variety of applications. These systems include: ultrasonic

imaging [39], ultrasonic ranging for robots [20] and SODAR (SOnic Detection And Ranging) systems that measure atmospheric conditions [18]. However, all of these systems have been implemented using custom hardware. By using custom hardware these systems are able to address many of the challenges associated with in-air sonar systems. This is where our system is different. The sonar sensor that we proposed does not use any custom hardware and must compensate for the limitations of the commodity hardware in everyday smartphones.

The earliest occurrence of a smartphone based ranging sensor in the literature occurred in 2007 when Peng et al. proposed an acoustic ranging system for smartphones [63]. This ranging sensor allowed two smartphones to determine the distance between them by sending a collection of beeps. The sensor was accurate to within centimeters. The sensor is a software sensor and only uses the front speaker and microphone on the phone. Our sensor is different from the sensor in [63] because it allows smartphones to determine the distance from the phone to any arbitrary object in the environment.

In 2012, researchers at Carnegie Mellon University proposed a location sensor that allowed users to identify their specific location within a building [50]. The system proposed by the researchers used a collection of ultrasonic chirps that were emitted from a collection of speakers in a room. A smartphone would then listen for these chirps and use this information to locate a person in a room. The phone was able to do this by using the chirps from the speakers to triangulate itself. For example, if the smartphone is closer to one speaker than another it will receive that speaker's chirp before it receives the chirp from another speaker. Since the locations of the speakers are known and the interval of the chirps are also known, the phone is able to use the chirps to triangulate its location. Our system is different from this one, since it attempts to determine the location of the smartphone relative to another object.

Other researchers have also implemented in-air sonar systems on other unconventional systems. For example, researchers at Northwestern University have implemented a sonar system on a laptop [72]. Other researchers have also uploaded code to Matlab central that

implements a sonar system on a laptop by using Matlab's data acquisition framework [17]. The closest sensor to the proposed sensor is an iphone application called sonar ruler [24]. The application measures distances using a series of clicks. The application does not filter the signal and requires the user to visually distinguish the pulse from the noise. Our sensor is different from the sonar ruler application because our sensor filters the signal and does not require the user to manually inspect the raw signal. Removing the need for user input allows the proposed sensor to be abstracted using an API. Being able to abstract the sensor using an API is important because it allows the sensor to be easily used by other applications.

2.4 Design

The system is comprised of three major components: 1) a signal generation component, 2) a signal capture component and 3) a signal processing component. Figure 2.3 shows an overview of these components. The signal generation component is responsible for generating the encoded pulse. This component is comprised of two sub-components: a pulse/chirp generation component and a windowing component. The second component is the signal capture component. The signal capture component records the signal that is reflected from the object. The third component is the signal processing component. The signal processing component filters the signal and calculates the time between the initial pulse and its reflection. This component is comprised of two sub-components. The first component is the filtering component and the second sub-component is the peak detection component. We discuss each component in detail in the following sections.

2.4.1 Generating the Signal

The signal generation process is comprised of two subprocesses. The first subprocess generates an encoded pulse, while the second subprocess shapes the encoded pulse. We

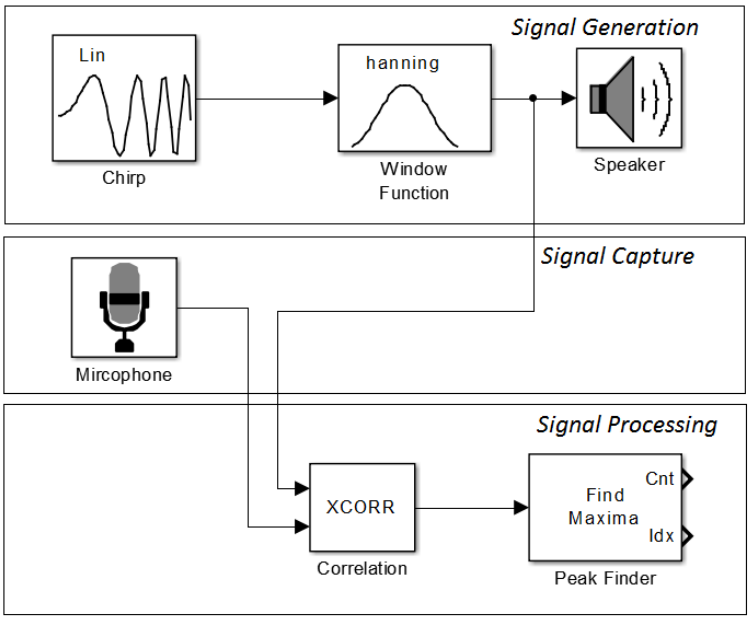


Figure 2.3: The figure shows an overview of the sonar system's architecture.

discuss each part of the process in a separate subsection. We begin by discussing the pulse encoding process which is also called pulse compression.

2.4.1.1 Pulse Compression

Pulse compression makes it easier to recover a pulse by encoding the pulse with a unique signature. The pulse can be encoded using amplitude modulation or frequency modulation. Amplitude modulation is the process of encoding a wave by increasing or decreasing the amplitude of sections of the wave, while frequency modulation is the process of varying the frequency of different sections of the wave. The state of the art pulse compression approach uses frequency modulation to create an encoded pulse, since frequency modulation is less susceptible to noise [25]. The encoded pulse is known as a linear chirp. A linear chirp is a signal whose frequency increases linearly from a starting frequency to an ending frequency.

Figure 2.4(a) shows an image of the linear chirp in the time and frequency domain. The signal was sampled at 44.1kHz, so each sample index represents $0.227\mu s$. The signal

starts at a low frequency and progresses to a higher frequency. Now that we have discussed the intuition behind a linear chirp, we will look at how the signal is generated. A linear chirp can be expressed as a sinusoidal function. Equation 2.1 describes how the amplitude of a linear chirp signal varies with time. The value f_0 represents the initial frequency while the value k represents chirp rate (how quickly the frequency increases) and ϕ_o represents the phase of the pulse. The chirp rate can be determined using equation 2.2.

$$x(t) = \sin\left[\phi_0 + 2\pi\left(f_0 * t + \frac{k}{2} * t^2\right)\right] \tag{2.1}$$

In equation 2.2 the values f_0 and f_1 represent the starting and ending frequencies, respectively. The value t_1 represents the duration of the pulse.

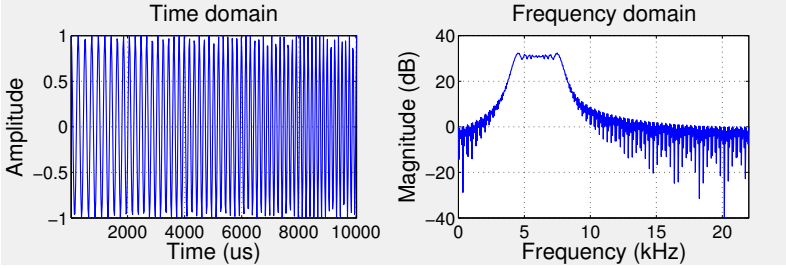
$$k = \frac{f_1 - f_0}{t_1} \tag{2.2}$$

f_0	4kHz
f_1	8kHz
t_1	0.01s
ϕ_0	0
Sample Rate	44.1kHz

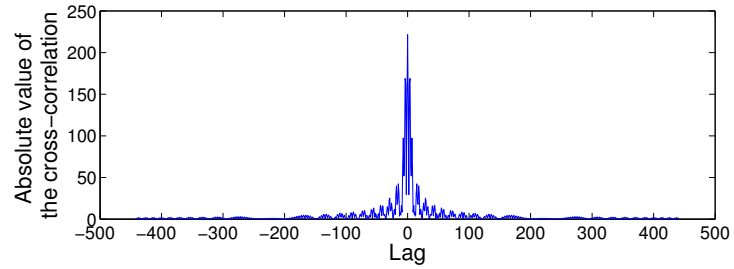
Table 2.1: Chirp properties table

Equation 2.1 represents the continuous time representation of a linear chirp. However, the signal must be discretized before it can be played by the phone's speaker. The signal is discretized by sampling equation 2.1 at specific time intervals. The time intervals are determined based on the sample rate and the pulse duration. In our implementation, a linear chirp was generated using the parameters shown in table 2.1. Though we have selected a linear chirp with these properties it is important to note that other linear chirps can be used with different frequencies and sweep times.

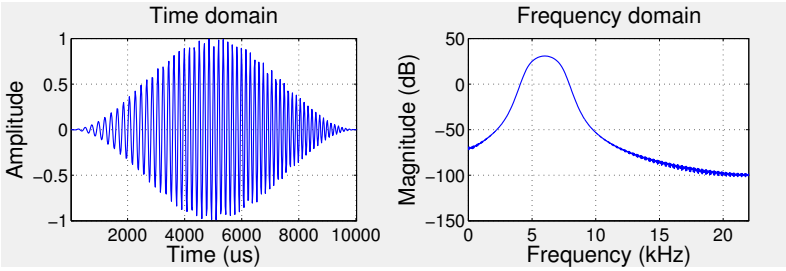
Now that we have discussed how the encoded pulse is generated, we will discuss how the pulse is shaped in the next section. This process of shaping the pulse is known as windowing.



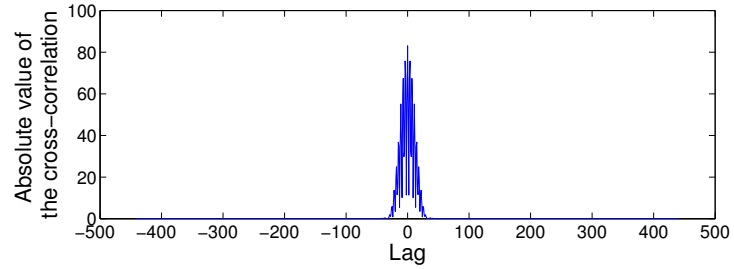
(a) Unwindowed linear chirp time and frequency domain sampled at 44.1kHz



(b) Unwindowed linear chirp autocorrelation showing sidelobes



(c) Windowed linear chirp time and frequency domain sampled at 44.1kHz



(d) Windowed linear chirp autocorrelation showing no sidelobes

Figure 2.4: These figures show the time and frequency domain representation for the windowed and unwindowed linear chirp

2.4.1.2 Windowing the Pulse

The windowing process is the second subprocess of the signal generating process. The windowing subprocess shapes the pulse by passing it through a window. Shaping the pulse improves the pulse's signal to noise ratio by improving the peak to side lobe ratio. This becomes clearer when we compare the autocorrelated representation of the windowed signal in figure 2.4(d) with the unwindowed signal in figure 2.4(b). Notice that the autocorrelated representation of the windowed signal does not contain the additional peaks/sidelobes that are in the autocorrelated representation of the unwindowed signal.

The signal in figure 2.4(c) was windowed using a hanning window [55]. The pulse is shaped by multiplying it by a hanning window of the same length. The hanning window is described by equation 2.3. In equation 2.3, N represents the number of samples in the window and n represents the index of a sample. Since the pulse is $0.01s$ and the sample rate is 44.1kHz, the window has a length of 441 samples. The discretized pulse is shaped by performing an element-wise multiplication between the discretized pulse vector and the discretized hanning window. Finally, the discretized shaped pulse is sent to the speaker's buffer to be transmitted.

$$H[n] = 0.5 * (1 - cos(\frac{2 * \pi * n}{N - 1})) \tag{2.3}$$

2.4.2 Capturing the Signal

Once the system has transmitted the pulse, the next step is capturing the pulse's reflection. The signal capture component is responsible for capturing the signal's reflection. However, accurately capturing the signal possess two unique challenges. The first challenge is working with the constraints of the phone's sampling rate and the second challenge is concurrently managing the hardware's microphone and speaker buffers.

2.4.2.1 Sampling Constraints and Hardware Requirements

The range of frequencies that can be recovered by the phone is limited by the maximum sampling rate and frequency response of the hardware. This is because in order to recover a wave we must sample at more than twice the wave's frequency. This means that the frequencies that can be contained in the linear chirp are limited by the sampling rate of the microphone and speaker. The microphone and speaker on the nexus 4 has a maximum sampling rate of 44.1kHz. This means that without the use of compressive sensing techniques it is only possible to generate and record a maximum frequency of $22,050Hz$, since Nyquist sampling theorem states that we must sample at twice the frequency of signal that we are attempting to recover. To ensure that we remain within the sample range of most phones, we use a linear chirp that ranges from 4kHz to 8kHz. Limiting the frequency range of the linear chirp allows us to address the sampling constraints of the hardware. In addition to the sampling constraints of the hardware, the phone's speaker and microphone have frequency response constraints. This means that they are only able to generate and receive a limited range of frequencies. This frequency response depends heavily on the make and model of the microphone and speaker, which can vary drastically among devices. To mitigate this we select a frequency range for the chirp that is slightly above the range of the human voice. So most smartphone microphones and speakers should be able to transmit and receive the pulse.

2.4.2.2 The Concurrency Problem

State of the art sonar systems have the ability to concurrently manage the microphone and speaker buffers. Synchronizing the buffers is important for sonar systems because ideally the system would start recording immediately after it has finished sending the pulse. Starting the recording immediately after the pulse is transmitted provides a baseline for calculating the time difference between when the initial pulse was sent and when the reflected pulse was received. If the buffers are not well managed, the recording may contain

the initial pulse, and the time index of the reflected pulse will not accurately represent the time between the initial pulse and the reflected pulse. The android operating system does not allow for real-time concurrent management of the microphone and speaker buffers so synchronizing them is challenging. This means that we must find a way to accurately calculate the time between the initial pulse and reflected pulse without managing the buffers in real-time.

We solve the concurrency problem by starting the recording before the pulse is transmitted. Starting the recording before transmitting the pulse allows us to record the initial pulse as well as the reflected pulse. We can then calculate the distance by calculating the time between the first pulse and the second pulse, since we have recorded both. This solution works because the proposed sonar system is monostatic which means that both the microphone and the speaker are located on the same device.

2.4.3 Processing the Signal

Now that we have explained how the sonar system generates a pulse and captures its reflection, we can discuss how the captured signal is processed. The process of analyzing the signal is comprised of two subprocesses. The first process is the filtering process. The signal is filtered by calculating the cross correlation between the known signal and the noisy signal. The result of the filtering process is passed to the peak detection process, which detects the peaks in the output and calculates the distance between each peak. The distance between peaks is used to calculate the distance between an object and the sonar system.

2.4.4 Frequency Domain Optimization

The cross-correlation process works by checking each section of the noisy signal for the presence of the known signal. Each section of the noisy signal is checked by calculating the sum of the element-wise multiplication between the known signature and the signal. Equa-

tion 2.4 describes this process. The $\star$ notation represents the cross-correlation operation and, the value $f^{'}$ represents the complex conjugate of f.

$$(f \star g)[n] = \sum_{m=-\infty}^{\infty} f^{'}[m]g[n+m] \tag{2.4}$$

Calculating the cross-correlation of the signal can be computationally expensive. This is because, the algorithm that is normally used to calculate the cross-correlation of two signals has an algorithmic complexity of $O(n^2)$ (Assuming that both signals (f and g) have the same length). However, it is possible to optimize the algorithm by performing the computation in the frequency domain.

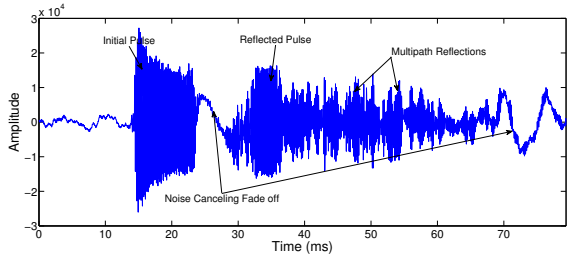
Equation 2.5 shows how to compute the cross-correlation of two signals in the frequency domain. The $\mathscr{F}$ is the mathematical notation for the Fourier transform, and $\mathscr{F}^{-1}$ represents the inverse Fourier transform.

$$(f \star g) = \mathscr{F}^{-1}[\mathscr{F}[f]^{'} \cdot \mathscr{F}[g]] \tag{2.5}$$

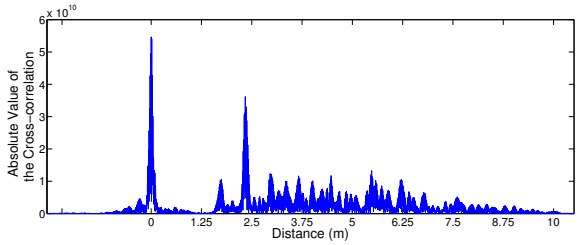
Matches are represented by large peaks in the output of the filtering process. Higher peaks represent better matches. Figure 2.5(a) shows a noisy signal which contains the initial pulse and its reflection while figure 2.5(b) shows the output of the filtering process. Notice that the output of the filtering process contains two prominent peaks. These peaks correspond to both the initial pulse and its reflection.

Computing the correlation in the frequency domain allows for faster computation since the algorithm has a lower algorithmic complexity $O(nlog(n))$. The process can be further optimized since the Fourier transform of the known signal has to be computed only once. Reducing the number of samples will also result in faster computation times.

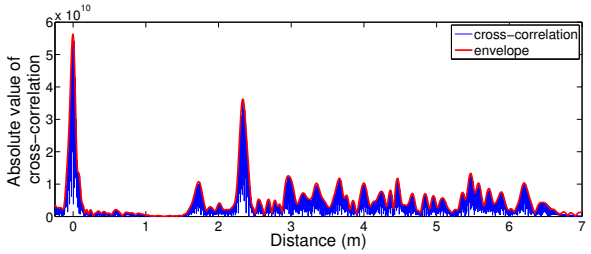
Now that we have discussed the filtering process we can discuss the process that is used to detect the peaks in the filter's output.



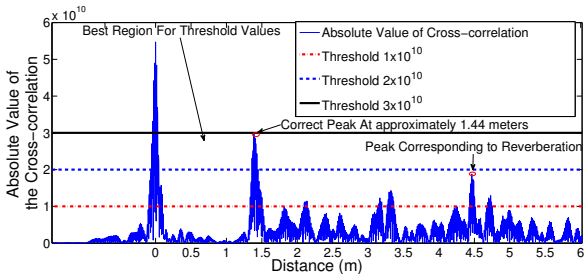
(a) Noisy signal captured using the Nexus 4



(b) Filtered signal



(c) Envelope detection



(d) Threshold selection

Figure 2.5: Figure (a) shows the noisy signal that was captured at approximately 2.5 meters from the wall. Figure (b) shows the filtered signal. Figure (c) shows the result of applying the envelope detection algorithm. Figure (d) shows the detection threshold values applied to another sample taken at 1.5 meters. A Nexus 4 smartphone was used to collect these readings by measuring the distance to an outdoor wall on a college campus.

2.4.5 Peak Detection and Reverberation

Performing peak detection on the cross-correlation function is difficult, since the resulting function is jagged and contains several small peaks. To mitigate this we calculate the envelope of the cross correlation by calculating the analytic signal of the frequency domain multiplication of the pulse and the received signal. The analytic signal is calculated by setting the negative frequencies to zero and doubling the positive frequencies. Once the analytic signal has been calculated the absolute value of the inverse Fourier transform is calculated to determine the final envelope. Figure 2.5(c) shows an example of the envelope.

The output of the filtering process is a collection of peaks. The sonar system needs to automatically detect these peaks and calculate the distance between them. In an ideal case the signal would only contain as many peaks as there are objects in the wave's path and a simple peak detection algorithm could be applied. However, factors such as noise cause the filtered signal to contain other peaks.

To account for the noise in the filtered signal, we propose a new peak detection algorithm that selects the most prominent peaks. By only considering peaks above a fixed threshold it is possible to remove the number of peaks that correspond to noise. This threshold can be determined empirically. We define a peak as a point which is higher than its adjacent points.

Algorithm 2.1: Peak Detection Algorithm

```
Input: array, threshold
Output: PeakArray
for i=0; i<array.length; i++ do
    if array[i] > array[i-1] and array[i] > array[i+1] then
        if array[i] ≥ threshold then
            PeakArray.add(i);
        end
    end
end
```

We propose a peak detection algorithm for detecting the peaks in the cross-correlation result. The threshold that we chose for the Nexus 4 was $22 * 10^9$. This threshold provides

the best trade-off between accuracy and range. We selected this threshold empirically by taking 10 samples in a room at different distances. We then selected the detection threshold that resulted in the least number of peaks above the line. The height of a peak is a reflection of the quality of the received signal. The last peak corresponds to the object that is the furthest from the smartphone, while the first peak represents the object that is closest to the phone. Figure 2.5(d) shows the threshold superimposed on the filtered signal. Furthermore, figure 2.6 shows the error in meters verse the threshold that was selected.

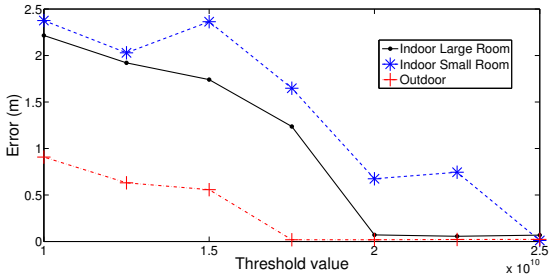


Figure 2.6: The figure shows the error in meters vs. the threshold value. These readings were collected using a Nexus 4 smartphone.

We have considered a collection of approaches for determining the threshold including other functions that are more complex than the proposed linear one. However, since the height of the correlation peak also depends on the sensitivity of the microphone the user needs to be able to calibrate the device by adjusting the threshold. A simple linear function makes this calibration process easier. Figure 2.8(a) shows the user interface slider that is used to adjust this threshold.

2.4.6 Increasing the minimum detection range

The minimum distance at which a sonar system can identify an object is limited. If objects are too close to the system they will reflect parts of the linear chirp while other parts are still being transmitted. This is because the linear chirp is not transmitted instantly but rather over a period of $10ms$.

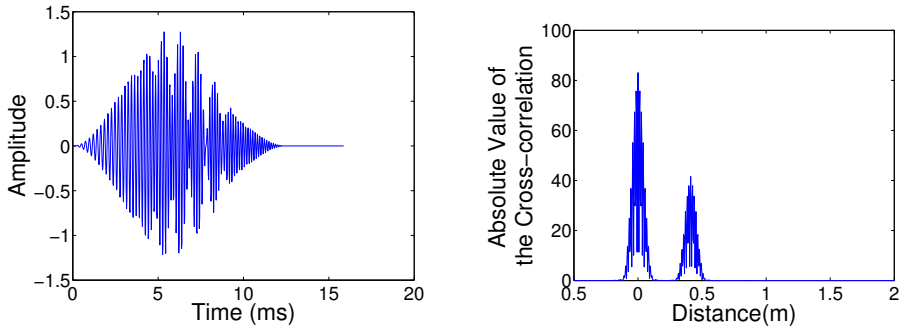
Consider the example shown in figure 2.7(a). This figure shows an illustration of the signal that is received when a pulse is reflected from an object at $0.41m$, overlaps with the initial pulse. Notice that it is not possible to visually distinguish the initial pulse from its reflection. This is because as the first linear chirp is generated it begins to travel through the air until it encounters an object at 0.41 meters, $2.5ms$ later. The object will then reflect the waves and the reflected waves will arrive at the system $5ms$ later. This is problematic since the system is recording both the pulse and the reflection. The reflected signals will interfere with the signal that is being generated since the generated signal has a duration of $10ms$. This means that the sonar system cannot detect objects within a 1.65 meter radius.

There are two ways to decrease the minimum distance at which an object can be detected. The first method is to reduce the duration of the pulse. Reducing the duration of the pulse reduces the amount of time that subsequent reflections have to overlap with the pulse. However, reducing the duration of the pulse increases the signal to noise ratio. The second method is pulse compression.

Pulse compression allows the cross-correlation process to identify pulses even when they overlap without increasing the signal to noise ratio. Figure 2.7(b) shows the signal that results from two overlapping pulses and the output of the filtering process. The two peaks correspond to the correct location of the pulses. Notice that even-though the pulses overlapped the filtering process was able to recover two distinct peaks, because they were encoded. The only thing that limits the minimum distance now is the width of the autocorrelation peak.

2.4.7 Temperature's Impact on the Speed of Sound

Environmental factors such as temperature, pressure and humidity affect the speed of sound in air. Because these factors affect the speed of sound in air, they also affect the accuracy of a sonar system. The factor that has the most significant impact is temperature [78] [54]. In this section, we show how the ambient temperature can significantly affect



(a) Time domain (b) Cross-correlation result

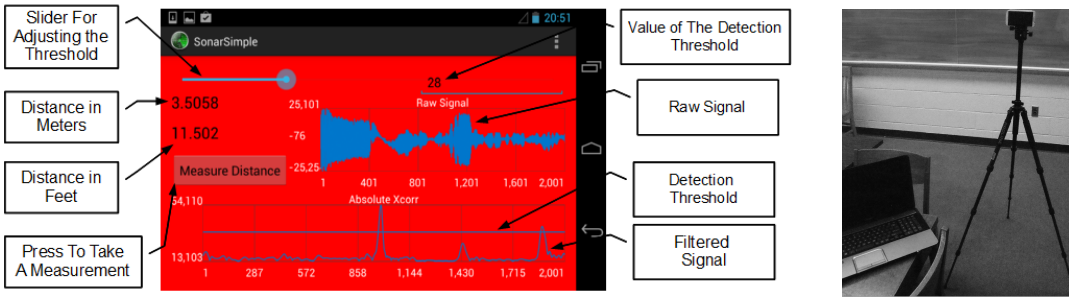
Figure 2.7: The figures show an example of how pulse compression helps reduce the minimum distance at which objects can be detected. Figure (a) shows the combination of the two pulses that are received by the microphone. The first pulse is a linear chirp 4kHz to 8kHz with a sweep time of 10ms while the second pulse has the same parameters but is attenuated by a factor of 0.4. Figure (b) shows the result of the filtering process. All signals shown in this figure were sampled at a rate of 44.1kHz.

the accuracy of a sonar system. We also propose a method for increasing the system's accuracy by using the phone's temperature sensor to estimate the air's temperature.

Equation 2.6 from [9] describes the relationship between the speed of sound and the air's temperature. Where T_c represents the air temperature and $v(T_c)$ represents the speed of sound as a function of air temperature.

$$v(T_c) \approx 331.4 + 0.6 * T_c \tag{2.6}$$

From equation 2.6 we can see that underestimating the temperature will result in a speed of sound that is slower than its actual speed. Underestimating the speed of sound will cause objects to appear further away than they actually are, while overestimating the temperature will overestimate the speed of sound thus causing objects to appear closer than they actually are.



(a) Application screenshot

(b) The experimental setup

Figure 2.8: Figure (a) shows a screen shot of the sonar sensor application running on the Nexus 4. Figure (b) shows the experimental setup that was used in the evaluation

2.4.8 smartphone Application

Figure 2.8(a) shows a screen shot of the sonar application. The top graph shows the raw signal that was captured by the microphone. The number in the top left shows the distance in meters. The number below it shows the distance in feet. The graph at the bottom shows the absolute value of the filtered signal.The highest peak represents the initial pulse while the second highest peak represents the reflected pulse. The x-axis in both the top and bottom graphs represent the sample index. The graphs have been designed so that the user is able to zoom in the graphs by pinching the display. Subsequent readings are overlaid on the original graph. This allows the user to easily validate the peaks. The horizontal line in the bottom graph represents the detection threshold. Only peaks above this threshold are considered by the peak detection algorithm. The slide bar on the top left is used to adjust the threshold and the value in the box next to it displays the value of the detection threshold. New measurements are taken by pressing the "Measure Distance" button. If the output of the filtering process does not contain peaks above the threshold, the system will report a measurement of zero. The application will also display a message to the screen that asks the user to take another measurement. Since graphs for each measurement are overlaid on each other, the user may reset the display by simply rotating the phone.

To obtain a good reading the user must not cover the phone's microphone or rear

speaker. If the user covers the phone's rear speaker, the sound from the rear speaker will be muffled and the chirp signal will be suppressed. If the user covers the phone's microphone, the reflected pulse will not be received. It is also important to calibrate sensor by adjusting the threshold as previously described. The source code and application package(APK) file for this application is available on github [1]

2.5 Performance Evaluation

We evaluate the sonar system using three metrics: accuracy, robustness and real-time performance. We evaluate the accuracy of the system by measuring known distances under different temperature and reverberation conditions. The accuracy of the system is then determined by comparing the known values to the measured values. All measurements that are reported in this section were collected using a Nexus 4 smartphone

2.5.1 Evaluating the Impact of Temperature and Reverberation

In this subsection we explain the process that we used to concurrently measure the sonar system's accuracy and robustness. We measure the sonar sensor's accuracy by using it to measure known distances. Ten measurements were taken at distances between 1 and 4 meters. This range was divided into 0.5 meter increments: 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4. To ensure that the measurements were taken in the same position every time, the phone was placed on a tripod. The tripod was placed at a height of 1.53 meters. It is important to note that stability is not a requirement. The phone does not need to be held perfectly still to obtain a reading. However, since we are evaluating the accuracy of the sonar system, we wanted to ensure that the readings were taken at the same distance every time. Figure 2.8(b) shows a picture of the phone on the tripod. We also evaluated the sensor's accuracy in different environments. In particular, we examined the sensor's performance in environments that were expected to have different levels of

[1] `http://researcher111.github.io/SonarSimple`

reverberation and ambient noise. It is well known that reverberation affects the accuracy of sonar systems [14]. Reverberation occurs when a reflected signal is scattered in multiple directions and reflects off other objects in the environment. Some signals may arrive at the receiver later since they have taken a longer path. The signal processing component at the receiver must decide among these multiple reflections. We evaluate the sensor's robustness by repeating the measurement process above for environments that are expected to have different levels of reverberation and ambient noise. Figure 2.9 shows a picture of each of these environments.

Figure 3.9(c) shows the results of the outdoor measurements. The y-axis represents the measured distance, while the x-axis represents the known distance. The dotted line represents a perfect match between measured distances and the known distances. Ideally the sensor's readings should perfectly match this line. The bars represent the standard deviation of the measurements. Each point on the graph represents the average of the 10 readings. The solid black lines represent the readings that were taken using $330m/s$ as the speed of sound. The green lines represent the readings that were taken using the temperature adjusted speed of sound. Notice that most of the non-temperature adjusted readings are slightly below the ideal line. This is because the speed of sound is temperature dependent. Sound travels at $330m/s$ at -1^oC, however it travels faster at higher temperatures for example $343.6m/s$ at 21^oC. Since the system is assuming that sound waves are traveling slower than they actually are, the system under estimates the distance. Some smartphones have temperature sensors. These sensors can be used to improve the accuracy of the system.

The outdoor environment is expected to have the least reverberation, since it is an open environment in which other reflecting surfaces are far away. Ten measurements were taken at fixed distances between 1 and 4 meters. Once the ten measurements have been taken, the tripod is moved to the next 0.5 meter increment. The process is repeated until measurements have been taken at all seven distances, for a total of 70 readings. When the measurements were taken, there was low ambient noise, light foot traffic and conversation.

(a) Outdoor

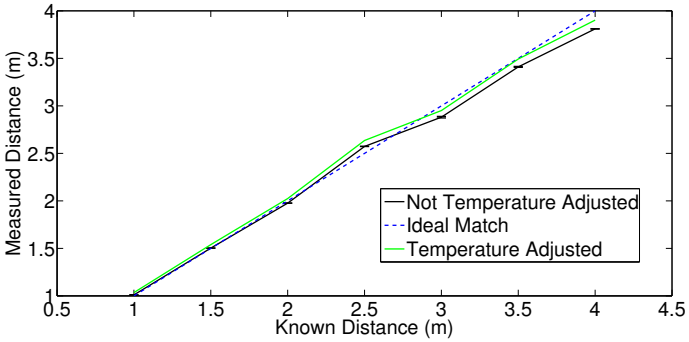
(b) Large room

(c) Small room

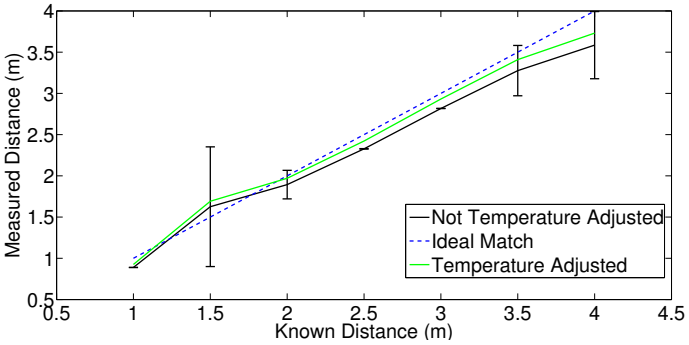
Figure 2.9: These figures show pictures of all three environments. Figure (a) shows the outdoor environment. Figure (b) shows the large carpeted indoor classroom and figure (c) shows the small indoor room 3.2m by 3.1m.

The second environment is a large indoor carpeted classroom. This environment is expected to have more reverberation than the outdoor environment, since it contains several chairs and tables. For this experiment the tripod is set up facing a wall in the classroom. The measurement process is repeated until all 10 measurements were obtained for all seven distances. The measurements were plotted in a similar fashion to the outdoor results. Figure 2.10(b) shows a plot of these results. Notice the indoor measurements underestimate the ideal line even more than the outdoor environment. This is because the classroom is warmer than the outdoor environment and therefore the sound waves are traveling faster. Notice that the measurement at 1.5 meters has a high standard deviation. This is due to the effects of reverberation. The indoor classroom also contained ambient noise such as the rumbling of the air conditioning unit.

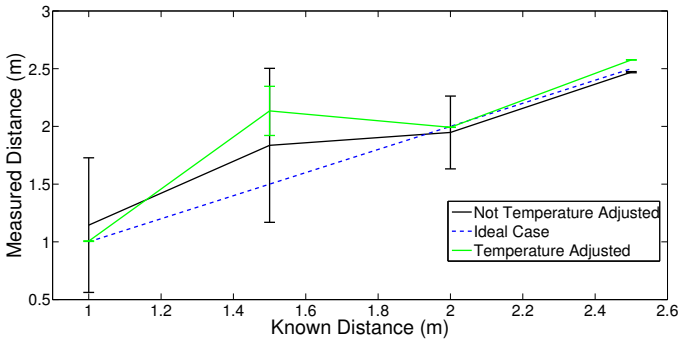
The final environment is a small indoor room that is not carpeted. The room is 3.2 meters by 3.1 meters and has a ceiling that is 2.9 meters high. All the walls in the



(a) Outdoor



(b) Large room



(c) Small room

Figure 2.10: These figures show the measurements that were taken in all three environments. Figure (a) shows the results for the outdoor environment. Figure (b) shows the results for the large carpeted indoor classroom and figure (c) shows the results for the small indoor room 3.2m by 3.1m.

room are solid brick. This room is expected to have the highest level of reverberation. Since the room is small and contains tables and chairs, the sound waves are expected to

bounce off multiple objects thus resulting in a high level of reverberation and interference. Figure 2.10(c) shows the results for the experiments performed in the room. Notice the high standard deviation. This is expected since the room is expected to have a high degree of reverberation. The standard deviation is larger at smaller distances in the small room because as the phone is placed closer to the wall the reverberation effects are greater, due to the proximity of the neighboring walls. The room also contains ambient noise from the air conditioning unit.

The results in Figure 2.10 lead us to conclude that this system works well in environments that have low reverberation such as outdoor environments and large rooms but does not work well in areas that have high reverberation such as small rooms.

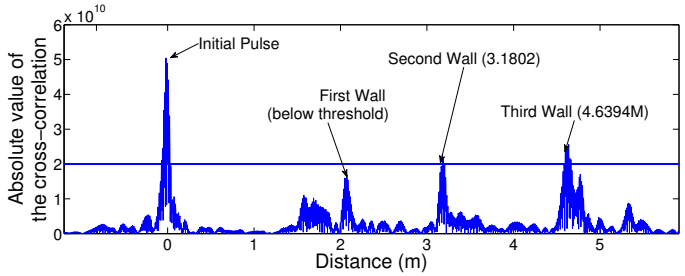
2.5.2 Evaluating the Detection of multiple Objects

In this experiment, we use the sonar sensor to detect the distance from the phone to three walls that form the sides of a wheel chair ramp. Figure 2.11 shows a picture of the walls. The output of the filtering process is shown in figure 2.12(a). Notice that, there are four main peaks. The first peak corresponds to the initial pulse while the second, third and fourth peaks correspond to the first, second and third walls, respectively. Each Wall is approximately 1.7 meters apart. The phone was placed 1.7 meters in front of the first wall so that the distances from the phone to each wall would be 1.7, 3.4 and 5.1 meters respectively. In figure 2.12(a) each peak is labeled with the distance it represents. It is important to note that this is a single sample and averaging the additional samples will increase accuracy as previously shown.

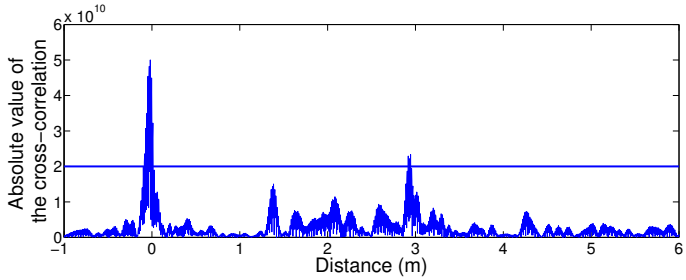
The experiment was repeated but this time the phone was placed at an oblique angle to the wall, figure 2.12(b) shows the result of the experiment. Notice that the system does not detect the walls when the readings are taken at an oblique angle (approximately 140^o) to the wall. This is because of the mirror effect. Since large planar surfaces reflect sound in the direction of their surface normal, the reflected sound does not get reflected back to the phone, hereby preventing the walls from being detected.

Figure 2.11: This figure shows a picture of three the wall that form the sides of the wheel chair ramp

.



(a) Directly facing the wall



(b) Oblique angle to the wall

Figure 2.12: Figure (a) shows the output of the filtering process when the readings were taken directly facing the wall. Figure (b) shows the output of the filtering process when the readings were taken at an oblique angle to the wall. The horizontal line represents a detection threshold of $2 * 10^{10}$. These readings underestimate the target since they are not temperature adjusted, the speed of sound is assumed to be 330 m/s

2.5.3 Evaluating Real-time Performance and System Usage

In this subsection we evaluate the real-time performance of the sonar sensor. In particular we focus on the time that it takes to obtain a reading. The most computationally expensive part of processing a signal is calculating the cross-correlation between the captured signal

and the known pulse. In this section we discuss three optimizations and evaluate how each optimization affects the system's real-time performance.

We focus on three optimization strategies: Opt 1) Performing the cross-correlation calculation in the frequency domain. Opt 2) Caching the frequency domain representation of the pulse. Opt 3) Only processing a subsection of the signal. Figure 2.13 summarizes the result of our experiments.

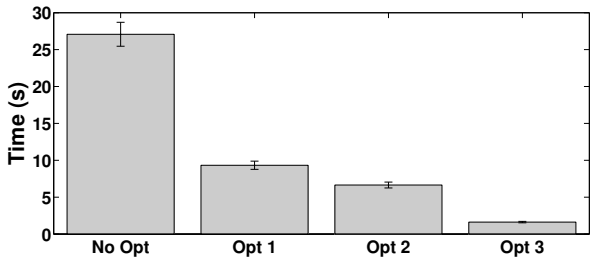


Figure 2.13: The figure shows the amount of time that it takes to obtain a single sonar reading. No Opt represents not optimizations. Opt 1 represents the Frequency domain optimization, Opt 2 represents caching the FFT of the pulse and Opt 3 represents limiting the number of samples

Calculating the cross-correlation in the time domain is computationally expensive and has an algorithmic complexity of $O(n^2)$. It takes an average of 27 seconds to return a result. However, calculating the cross-correlation in the frequency domain has an algorithmic complexity of $O(nlog(n))$. Performing the calculation in the frequency domain reduces that average time from 27.01 seconds to 9.33 seconds. This is equivalent to a 290% reduction in the amount of time that it takes to return a reading. However, it still takes over 9.33 seconds to return a result. Ideally we would like to get the response time to under 2 seconds. In an effort to reduce the time we introduced the second optimization Opt 2. This optimization reduces the processing time by caching the Fast Fourier transform of the pulse. Since the transform of the known pulse does not change, we only have to calculate its transform once. This reduces the average processing time by 2.68 seconds resulting in an average processing time to 6.65 seconds. However, this is still above the ideal value of 2 seconds. We further optimize the process by limiting the number of samples that we

consider to 2048. This does not affect the accuracy of the system but it does limit the system's range to approximately 4 meters. Limiting the number of samples reduces the processing time to 1.62 seconds. This is 0.38 seconds below the ideal processing time of 2 seconds.

2.6 Conclusion

The proposed sonar sensor is comprised of three components: a signal generation component, a signal capture component and a signal processing component. Designing a sonar system for smartphones presented two unique challenges: 1) concurrently managing the buffers and 2) achieving real-time performance. We addressed the concurrency problem by starting the recording before transmitting the pulse. This allowed us to capture the pulse along with its reflected pulses. Doing this allowed us to determine the index of the pulse and reflections by filtering the signal. We addressed the real-time performance problem by reducing the algorithmic complexity of the filtering process from $O(n^2)$ to a $O(nlog(n))$ by performing the cross-correlation calculation in the frequency domain.

Finally, we evaluated our sonar sensor using three metrics: accuracy, robustness, and efficiency. We found that the system was able to accurately measure distances within 12 centimeters. We evaluated the robustness of the sensor by using it to measure distances in environments with different levels of reverberation. We concluded that the system works well in environments that have low reverberation such as outdoor environments and large rooms but does not work well in areas that have high reverberation such as small rooms. In the future, we plan to investigate strategies for improving the sonar sensor's accuracy in environments with high reverberation. We also evaluated the system's real-time performance. We found that by performing three optimizations we were able to reduce the computation from 27 seconds to under 2 seconds. In the future we will be releasing the sonar application on the android market. This will allow us to test the sensor's performance on different hardware platforms

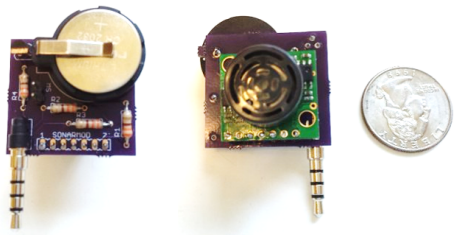
Chapter 3

A SONAR Ranging Attachment for 2D Mapping

3.1 Introduction

Smartphones have allowed companies to replace custom embedded solutions with more cost effective smartphone attachments. For example, Honeywell's Point-of-Sale (POS) attachment has allowed vendors to replace expensive, custom handhelds with standard smartphones and relatively cheap hardware attachments [1]. Recently, there has been emerging interest in developing attachments that provide depth sensing capabilities. Allowing smartphones to sense depth, opens mobile computing research to a variety of interesting possibilities.

In this paper, we attempt to solve the problem of providing depth sensing capabilities to smartphones by designing an external SONAR attachment. Ideally, we would like to leverage existing research by using in-air SONAR modules from the area of robotics. However, there are currently no hardware or software interfaces that allow us to connect existing sensors to a smartphone. Thus, in an effort to leverage existing sonar modules, we have designed a hardware and software interface that allows us to connect an existing SONAR sensor (like the LV-MaxSonar-EZ 1), to a smartphone.



(a) Rear view (b) Front view

Figure 3.1: Figure (a) is a picture of the $3.3V$ battery, the modulator circuit, and the male $3.5mm$ male headphone jack adapter. Figure (b) is a picture of the transducer module.

Currently, there are three possible approaches for connecting an external attachment to a smartphone: bluetooth, the micro USB port, and the headphone jack. Connecting the attachment using a bluetooth connection allows the attachment to remain physically untethered from the smartphone but increases the energy consumption and cost of the attachment. Connecting the attachment using a USB connection allows the external attachment to be powered by the smartphone. However, since not all modern phones directly support the USB standard (e.g. the Apple iPhone), these USB attachments will only be compatible with a subset of attachments.

Current systems which utilize the headphone jack have been designed to interface with external modules by simply reading analog values from the sensor. However, if the sensor produces a logic level signal, the analog to digital converter (ADC) on a typical smartphone is unsuited from reading the signal. This presents a challenge, since the SONAR sensor we use is a digital sensor, and therefore encodes its information using logic level signals [4]. We address the smartphone's limitations by proposing a hybrid hardware/software modulator that converts these logic level signals into modulated continuous wave (MCW) signals, which are similar to the modulation strategy that was used by Samuel Morse when he developed Morse code telegraphy [66]. These signals can be read by the smartphone. When thinking about how MCW modulation works in the proposed context, it is important to consider the symbol space. Morse code has a symbol space of size two: a long beep and a

short beep. Intuitively, it is possible to think of the proposed scheme as having a symbol space of n, where n represents the resolution with which the pulse is measured. Our proposed modulator is a key component in the design of our external attachment, since it is what makes communication between the attachment and the phone possible.

Our external SONAR attachment is comprised of an ultrasonic transducer, a microcontroller, a modulator, and a standard "coin cell" battery. The ultrasonic transducer is responsible for generating the ultrasonic signal, which travels through the air until it is reflected by a nearby surface. The reflected wave is then captured by the transducer. Once the signal has been captured, the microcontroller calculates the distance to the object by examining the time delay between the initial signal and its reflection. Once the distance has been determined, the microcontroller generates a pulse-width modulated signal whose length is directly related to the measured distance. This signal is then fed to the modulator, which converts the pulse-width modulated signal into a continuous wave modulated (MCW) signal, which is sent to the smart device through the headphone jack. The device can read these signals and decodes them, in software, to receive the distance information. Once the distance is decoded, the next step is to determine the direction of the received wave. This is done using the phone's gyroscope, accelerometer, and magnetometer to determine the phone's current orientation. By taking a collection of repeated measurements it is possible to construct a simple two dimensional map of the space.

In this paper, we make the following contributions:

- We design and build an in-air SONAR module that can be paired with commodity smartphones;

- We propose the use of a hybrid hardware/software modulator for communicating between the external module and the smartphone, via a standard 3.5mm headphone jack;

- We propose, implement, and evaluate a linear time algorithm for demodulating the

MCW signal.

- We build a prototype of our system and evaluate its performance, showing precision on the order of inches;

- We propose and implement an approach for combining the range and directional information to generate a two dimensional map of a space.

3.2 Related Work

In 2010, researchers at the University of Michigan explored a way of connecting external devices to smartphones via the headphone jack, called hijack [46]. A unique aspect of their design was that they harvested energy from the headphone jack to power the microcontroller which was responsible for processing sensor data and performing modulation tasks. They also proposed using a frequency shift key encoding scheme to communicate with the phone over the headphone jack, where it was demodulated using software installed on the phone. Several external mobile attachments has been developed using the hijack platform [59] [74] [41]. However, it is also possible to design a modulator that does not require the use of a processor. Such a modulator, would provide the software application on the phone with control of carrier frequency and with it, the baud rate. The modulator in this paper uses a hybrid hardware/software approach to modulate the signal and does not require the a microcontroller. Since the modulator design does not include a microcontroller there is also no need to implement any firmware. The proposed hybrid hardware/software modulator is comprised of a collection of simple analog components which are used to modulate a carrier supplied by the smartphone.

Project tango, by Google's advance technology and applications group, integrates a 3D depth sensing camera into a smartphone. Hereby providing the smartphone with the ability to generate three dimensional maps of space [7]. The inclusion of this sensor in smartphones has opened a collection of new accessibility applications, including indoor

navigation and obstacle detection. Developing depth sensing sensors for smartphones will help advance the field of assistive technologies. Researchers have also explored ways of providing depth sensing capabilities without the use of specialized hardware, instead choosing to implement software based SONAR ranging sensor solutions [32]. These sensors use a collection of chirps to determine the distance from the sensor to an object in space. However, these software based sensors are limited by the hardware constraints of the phone, since microphones and speakers are located at different locations on different phones. Creating external sensors will help mitigate this constraint and facilitate the development of assistive technologies such as SONAR canes. Though SONAR based navigation devices such as SONAR canes [31] already exist, providing attachments for devices that are already carried by the visually impaired will replace custom embedded solutions.

3.3 Design Section

The system is comprised of three major components: 1) an external SONAR module; 2) a commodity smartphone; and 3) the software application running on the phone. Figure 3.2 shows a diagram of the system's architecture. The external SONAR module is designed to use a collection of ultrasonic chirps to measure the distance to an object. Once the distance has been measured, the external module generates a modulated analog signal with the encoded information, and transmits it to the phone's hardware, through the headphone jack. The smartphone receives the signal and converts it to a digital vector using its own, on-board analog to digital converter (ADC). In addition to sampling and converting the signal, the smartphone is also responsible for tracking the phone's orientation and, by extension, the attachment's orientation, using its own internal gyroscope, accelerometer and magnetometer. The range and direction information is then combined by a software application to generate a direction vector. This vector represents the distance and direction of the item or surface, which reflected the original ultrasonic signal. These distance vectors can then be combined to generate a two dimensional map of a space.

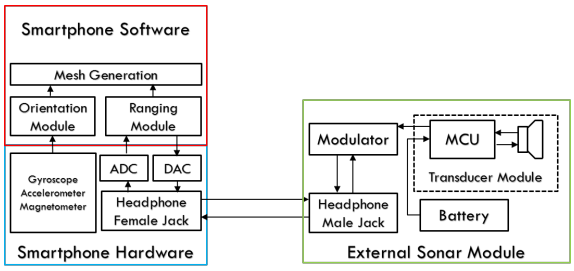


Figure 3.2: Shows a system diagram of the proposed SONAR system, which is comprised of: 1) an external SONAR module; 2) a commodity smartphone; and 3) the software application running on the phone.

3.3.1 External Attachment

There are currently no commercial, off the shelf (COTS) interfaces that allow us to connect a SONAR sensor to a smartphone. Since, there are no existing interfaces or drivers, we have designed and built a headphone jack interface and designed a MCW modulation scheme, that allows us to connect an existing SONAR module to a smartphone. In this section we describe the hardware that comprises the headphone jack interface, and the SONAR transducer module.

The external module hardware, (shown in Fig. 3.1) is comprised of three major components: a transducer module (the LV-MaxSonar-Ez 1), a modulator circuit, and a 3.5mm male headphone jack adapter [4]. The transducer module contains a microcontroller and an ultrasonic transducer. The transducer module is responsible for generating the ultrasonic chirps, and calculating the distance of the object by measuring the elapsed time between the chirps. This distance is encoded as a pulse-width modulated signal where each $147\mu s$ represents one inch. This signal is a logic level signal, and it cannot be decoded by the phone if it is directly fed to the phone's headphone jack. Figure 3.3 shows the signal that is decoded by the phone's ADC when the pulse width modulated signal is fed directly to the phone. Notice that it is difficult to determine the width of the pulse. We suspect that this poor reconstruction may be due to the fact that the phone's ADC is proceeded by AC coupling. However, we cannot confirm this because the phone's

hardware is proprietary.

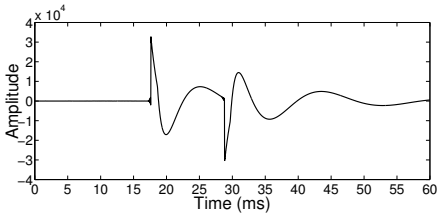


Figure 3.3: Shows the signal that is reconstructed when the pulse width modulated signal is fed directly to the phone.

The modulator, is responsible for transforming the pulse-width modulated signals from the transducer module into an alternating current (AC) signal that the phone can decode. This is done by designing a circuit that generates a modulated continuous wave from the pulse-width modulated signal. The modulated continuous wave is then fed to the phone via its headphone jack.

3.3.1.1 Modulator

The modulator takes two inputs, a carrier and the pulse-width modulated signal, and produces a modulated continuous wave (MCW). It is possible to generate the carrier using a hardware oscillator, such as a crystal oscillator, which uses the mechanical vibrations of a piezoelectric crystal to generate a periodic signal. Crystal oscillators are commonly used in radios to generate the carrier frequency [56]. However, including this hardware increases the size and cost of the platform. Instead, we implement the oscillator in software by leveraging the functionality of the phone's digital to analog converter (DAC) to generate the carrier. Once the carrier has been generated, it is fed to the modulator through the headphone jack. Figure 3.5 shows the signals that are involved in the modulation process.

It is important to select the correct frequency for the carrier. Selecting a carrier frequency that is too high may prevent the ADC from converting the signal, since the ADC may not support the sampling rate that is required. Selecting a carrier frequency that is too low may not provide enough resolution to accurately decode the length of the

signal. The datasheet of the transducer module indicates that the pulse width modulated signal has a resolution of $147\mu s$, where each $147\mu s$ corresponds to one inch [4]. Ideally, we would like to design the carrier so that each peak in the carrier corresponds to a single inch as well. This means that we would need to create a 6.8kHz carrier. However, the frequency of this carrier is too high for the analog to digital converter (ADC) on the phone to accurately recover the peaks in the signal. To mitigate this, we selected a frequency of 3.4 kHz. This is half the original, which means each peak corresponds to two inches. However it is possible to get a resolution of an inch at this frequency by also counting troughs, since each trough represents half the distance between two peaks.

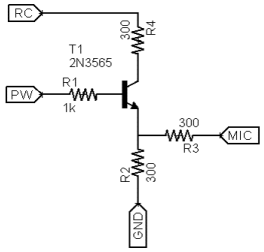
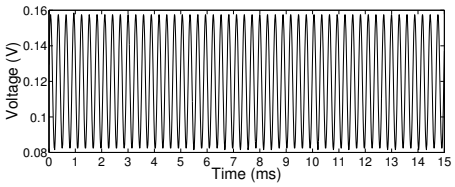


Figure 3.4: Shows a circuit schematic of the modulation circuit. RC is the label for the received carrier. PW is the label for the pulse-width modulated signal. MIC is the label for the output of the modulator that is connected to the microphone input on the phone.

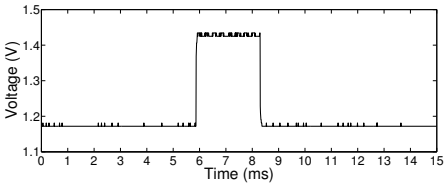
Figure 3.4 shows a circuit schematic of the modulator. The modulator is designed to turn the carrier frequency on and off. Once the pulse-width modulated signal goes high, the transistor saturates, and the carrier is transferred to the phone. Once the pulse width signal goes low, the transistor enters cut-off mode, and the carrier is truncated. By switching the carrier frequency on and off as the pulse transitions from low-to-high and high-to-low, it is possible to generate an AC signal that has approximately the same length as the pulse width modulated signal.

3.3.2 Demodulation

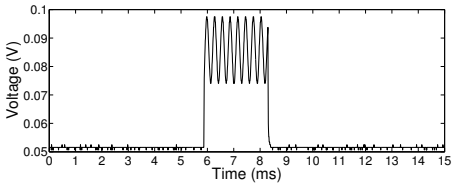
The external module continuously and asynchronously transmits distance readings to the phone, where they get stored in the phone's audio buffer. An application running on the



(a) 3804 kHz carrier frequency



(b) Pulse-Width Modulated Signal



(c) Modulated Continuous Wave Signal

Figure 3.5: Figure (a) shows the carrier frequency that is generated on the phone, (b) shows the pulse that is generated by the transducer module, and (c) shows the modulated signal which encodes the distance information. These measurements were taken using an object that was 43 cm away from the sensor.

phone reads from this buffer in chunks of 8000 samples (equivalent to about $181ms$ at a $44.1kHz$ sample rate).

Since the external module is generating readings every $50ms$, it is possible to have at most approximately three signals in each chunk, that have not been truncated. Figure 3.6 shows a plot of the values from the signal buffer. The signals x_1, x_2, x_3, and x_4 represent four different readings. Notice that signal x_4 is truncated because of the buffer size, and therefore its length does not reflect the correct measurement, so it must be discarded. For a more complicated design, a producer consumer design pattern can be used to create a continuous buffer. In the ideal case, the remaining signals, x_1, x_2, and x_3 would be the same length. However, we see a stabilization error in the digital signal when we try to decode it. Since the smartphone's hardware is proprietary, we cannot easily tell exactly what is causing the transient response, but we think it may be due to AC coupling, since the direct current (DC) bias is removed in the received signal. Figure 3.7 shows a zoomed in section of one of the signals that has been reconstructed on the phone. Notice that it takes approximately $4ms$ for the signal to stabilize, from the 31ms mark to 35ms. This may be due to the transient response of the pulse-width component that is still present

in the signal. However, it is possible to mitigate this during the demodulation process. Since, the first and last peaks are always reconstructed correctly, and the frequency of the carrier is known, the number of peaks in the signal can be calculated using the, known, time between the first peak and the last peak. This approach removes any errors that are introduced by peaks that are not correctly reconstructed in the signal.

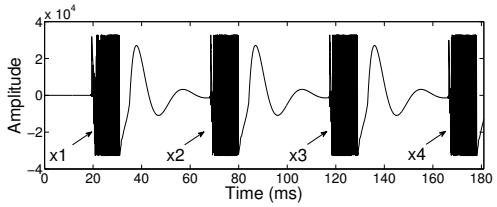


Figure 3.6: Shows the 8000 samples obtained from a single reading of the buffer.

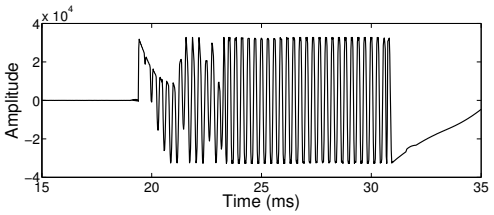


Figure 3.7: Shows a single reading from the buffer, i.e. x_1 from figure 3.6.

3.3.2.1 Demodulation Algorithm

Once the signal has been modulated by the external attachment it needs to be demodulated by the software on the smartphone, so that the signal can be converted back to a distance reading. To handle this decoding process, we use a linear time demodulation algorithm. The algorithm works by counting the number of peaks that are above a particular threshold. Recall that the frequency of the carrier was selected carefully, so that each peak corresponds to two inches (0.051 meters). The threshold is automatically selected by the software to be 50% of the maximum value in the buffer. Having the software automatically select the threshold will help mitigate the differences in the sound interfaces on different devices.

Algorithm 3.1: Calculate signal width

```
Input: array, threshold
Output: PeakArray
first = true; firstIndex = 0; lastIndex = 0 ; for i=0; i<array.length; i++ do
    if  array[i] > array[i-1] and array[i] > array[i+1]
    || array[i] < array[i-1] and array[i] < array[i+1] then
        if array[i] ≥ threshold || array[i] ≤ -threshold then
            peakSeen =true;
            count++;
            if first then
                firstIndex = i;
                first = false;
            end
            else
                lastIndex = i;
            end
        end
        if peakSeen then
            hops++;
        end
        if hopExpire < hops then
            if count > 2 then
                distances.add(lastIndex - firstIndex);
            end
            hops = 0;
            count = 0;
            peakSeen = false;
            first = true; firstIndex = 0; lastIndex = 0 ;
        end
    end
end
```

3.3.3 2D Map Generation

Since we collect information on both the distance and direction of an object from the phone, we can generate a two dimensional map of a space by taking multiple measurements in different directions. Recall that the phone's orientation can be determined by using its internal magnetometer, gyroscope, and accelerometer. We map a room by rotating the phone through a single 360° rotation. As a convention we define 0° to be North and values increase in a clockwise direction (e.g. East is 90°). Since the magnetometer is noisy, the direction values are discretized into 20° increments [52, 69]. The application divides the compass into eighteen sections of 20°. A measurement taken in a given section will associate that value with the center of that section. We can then generate a relatively stable measurement by using the values of the phone's pitch to help compensate for the phone orientation.

We implemented the design on a Nexus 4 smartphone. Figure 3.8 shows a screen shot of the application. The application shows the distance, pitch and direction values, as well as a polar plot generated from a collection of measurements.

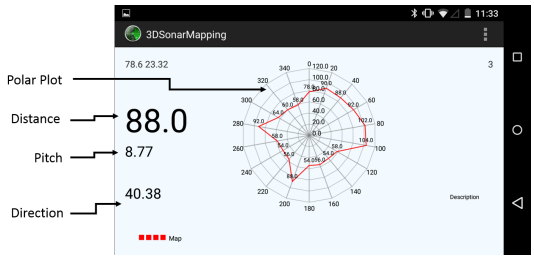
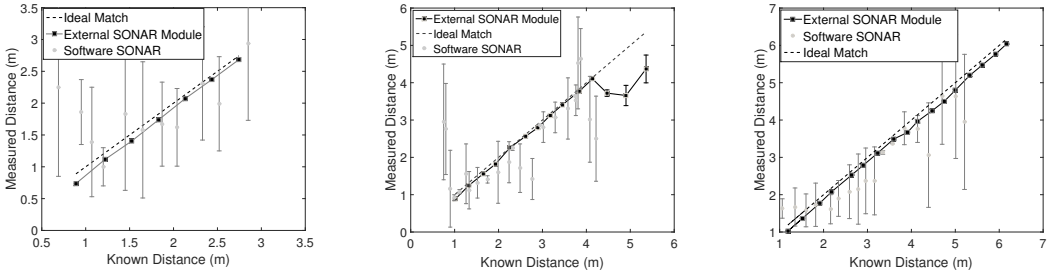


Figure 3.8: A screen shot of the mapping application showing the information on distance (in inches), direction (in degrees, with 0° at North), pitch (in degrees, with 0° at horizontal, i.e. to the ground), and the resulting polar plot.

3.3.4 Possible Applications

Now that we have shown that it is possible to do depth sensing using SONAR on smartphones, the question naturally arises: "What can this be used for?". We have several ideas for possible applications of depth sensing using SONAR. The system may be used to



(a) Small Room $A_s = 0.42$, $A_h = 0.09$

(b) Large Room $A_s = 0.22$, $A_h = 0.07$

(c) Noisy Outside Area $A_s = 0.38$, $A_h = 0.16$

Figure 3.9: Figures (a), (b), and (c) show the results for a small room, large room, and outside. Ten measurements were taken at each distance; the error bars represent the standard deviation of these measurements. A_s represents the average error for software SONAR module and A_h the average error for the software module .

create a SONAR cane that allows people with visual impairments to navigate a space. The system can also be used as a tool for realtors, interior designers, or engineers to quickly get an estimate of the dimensions of the room.

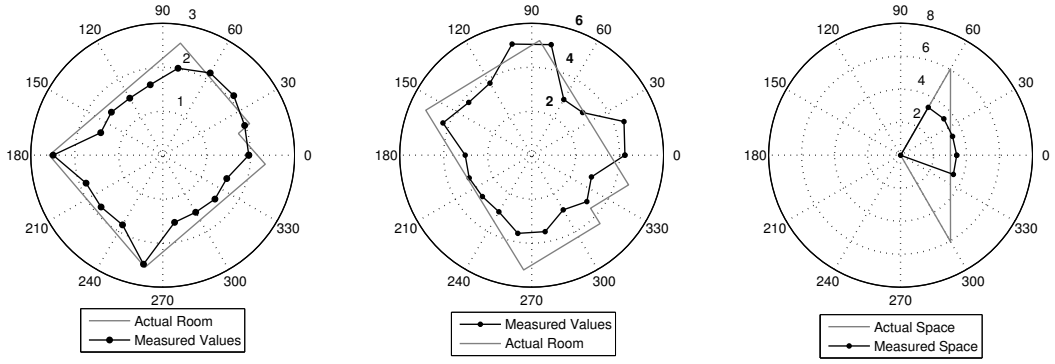
3.4 Evaluation

We evaluated our system using two metrics, accuracy and spatial resolution, in a small room, a large room, and an outdoor environment. In measuring the external SONAR attachment's accuracy, we compared it to our previous software based SONAR sensor [32] and a commercial sonar measuring device called the Strait-Line Laser Tape [10]. The external attachment achieved much lower standard deviations at each measured distance and smaller error when compared to the software based SONAR from our previous work.

In evaluating the spatial resolution, we took a collection of measurements in different directions to generate a polar plot of the area around the phone (as described in section 3.3.3), and compared the plot to the actual floorplan of the room.

3.4.1 Evaluating Accuracy

We evaluated the accuracy of our external SONAR by placing it on a tripod and taking measurements at different distances in three environments (For all tests we used the Nexus 4). At each distance, ten measurements were taken using both the external SONAR



(a) Polar Plot, Small Room (b) Polar Plot, Large Room (c) Polar Plot, Outside

Figure 3.10: Figures (a), (b) and (c) show the polar values obtained by the external SONAR module in a small room, a large room and outdoor environment, respectively. The application divides the compass into eighteen 20 degrees sections. A measurement taken in a given section will associate that value with the center of that section. All measurements shows are in meters and all angles are in degrees

module and the software based SONAR module. The average value and standard deviation of these readings were plotted against the actual distance for all three environments. The average error for both the external module A_h and the software based SONAR application A_s were calculated, excluding the break down regions, (i.e. less than one meter and more than four meters for the software SONAR, and less than $\frac{1}{2}$ meters for the external sonar module.

The first environment was a small room (approximately 3.36 by 3.92 meters). Figure 3.9(a) shows the results of this experiment. For the software SONAR sensor, we see decreased performance at values lower than one meter. When we compare the result of the values obtained using the external SONAR attachment, we noticed that the values have a much smaller standard deviation and are more accurate. The smaller beam of sound, that is produced from the external sensor, helps mitigate the reverberation effects.

The second environment was a large carpeted indoor room (approximately 8.61 by 6.10 meters). Figure 3.9(b) shows the results of this experiment. The software based SONAR module performs much better in a large indoor room, but begins to break down after 3.5

meters. As for the attachment, as the distance increases, so does the beam width. At 5.5 meters the beam is wide enough to pick up other objects in the space, like desks and chairs. This causes the accuracy of the sonar sensor to decrease as distance from the wall increases.

The final environment was an outdoor area with moderate building construction noise in the background. Figure 3.9(c) shows the results of this experiment. From these results we can see that the external SONAR module outperforms the software based sensor. Even though external module does sometimes underestimate the distance, it continued to perform up to its design limit of approximately six meters. This design limit is determined by the maximum distance of the transducer module which, from the datasheet, we know is approximately 20 feet, or 6.1 meters. Though it is possible increase the system's range by purchasing a new transducer module; such modules are normally much larger and have a large cone that surrounds the buzzer. In an effort to maintain a small, convenient form factor, we have opted to not use a smaller transducer module.

3.4.2 Evaluating Resolution

In this subsection, we evaluate the spatial resolution of the system by generating a two dimensional map of a space and displaying it on a polar plot. Figure 3.10(a) shows a polar plot that was obtained by placing the smartphone on a tripod in the middle of the small room and rotating it 360^o. We then measured the room, using a compass to determine the room's orientation, and calculated its polar coordinates. These values were then overlaid on the measured values in figure 3.10(a). We repeated this process for the other two environments and plotted in the results in figures 3.10(b) and 3.10(c).

3.4.3 Sources of Error

There are two main sources of noise when generating polar plots. The first source is noise from the gyroscope and magnetometer. The magnetometer determines the smartphone's direction with respect to magnetic north by sensing the earth's magnetic field. However,

the magnetometer is noisy and sometimes the direction is not determined correctly. We believe that this is the reason for the distorted corners in the large room measurement. The other source of error may be due to the fact that surfaces reflect sound at an angle to the surface normal. This means that the system will be fairly accurate when taking measurements of an object that is directly in-front, but as the angle from the receiver to the object increases, the accuracy will decrease, since less energy is being reflected in the direction of the receiver.

3.4.4 Energy Consumption

Another unique aspect of our design is that it is internally powered and does not require power from the phone. The only energy that is consumed by the phone is the energy that is associated with processing the information that is generated by the sensor. Since there are several other processes on the phone, it is difficult to isolate the energy that is consumed by only the demodulation process. The notion of harvesting energy from the phone to power the external sensor is interesting, though we do not do it for this sensor, other researchers and engineers have adopted this approach. For example, the highjack prototype developed by researchers at the University of Michigan uses energy harvesting [46]. However, we did not opt for this approach due to the additional complexity associated with implementing it. Implementing this energy harvesting approach would include the addition of a bridge rectifier to convert the AC signal to DC signal, and a voltage multiplier to step up the signal to the required voltage.

3.4.5 Testing other devices

So far throughout our evaluation we have only used the Nexus 4. But the question remains, *Will this solution generalize to other devices?* In an effort to answer this question, we evaluated our interface design on six different devices. The results of these tests are summarized in Table 3.1. We found that the interface mismatch issue existed on four of the six devices that we tested. We also tested the mapping application on these devices,

Device	OS	Mismatch	App Worked	App Failed
LG Nexus 4	5.1.1	✓	✓	
Samsung 5	4.4.2			✓
Samsung 4	4.4.4			✓
Nexus 5	5.1.1 & 4.4.3	✓		✓
LG L90	4.4.4	✓		✓
LG MyTouch QC 800	4.0.1	✓	✓	

Table 3.1: Compatible Devices.

and found that the application worked on two of the six devices that we tested. We are currently searching for the hardware or software issues that may have caused the failures in these other devices. One possible source of failure may be due to the operating system automatically lowering the volume when the headphone jack is plugged in. This makes it difficult to get a strong enough carrier for the modulator. Future versions of the hardware may need to include an amplifier and buffer on the interface board.

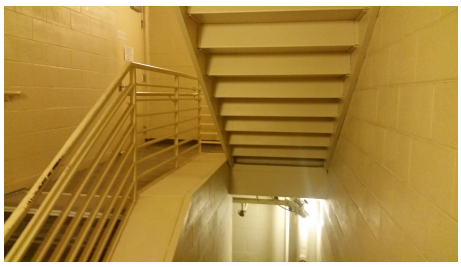


Figure 3.11: Shows a picture of the stairwell in which the measurements where taken

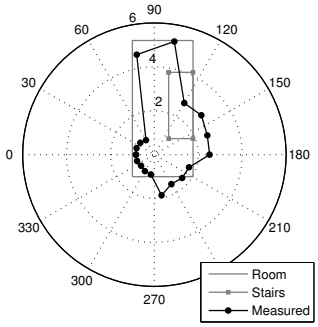


Figure 3.12: This figure shows the map that was reconstructed from the measurements taken in the stairwell. All measurements are in meters and directions are in degrees.

3.4.6 Testing in an irregular environment

So far we have tested the external attachment in simple environments. In this section we evaluate the performance of the system in a stairwell. Figure 3.11 shows a picture of the stairwell and Figure 3.12 shows the results of the map that was generated. From the figure we notice that errors from the magnetometer cause the map to be slightly skewed. For example, in Fig. 3.12, the points at 100° and 260°. We also notice that sensor picks up the middle of the staircase that is blocking the rear wall. We can also see that sensor does not pick up the lower rear wall due to the narrow beam width. All of these factors along with the irregular nature of the stairwell result in a moderate reconstruction of the space.

3.5 Conclusion

In this paper, we proposed a design for a smartphone compatible external SONAR module as an extension on our previous research into software based SONAR sensors for smartphones [32].

We designed a hybrid hardware/software modulator that allows the external attachment to communicate with the phone via the headphone jack. The modulator design is unique because it does not use a hardware oscillator to generate the carrier frequency but instead uses the software on the phone and the phone's DAC. The modulator encodes the range information from the external module using a modulated continuous carrier. This information is then decoded and combined with information from the phone's gyroscope, accelerometer, and magnetometer to generate a two dimensional map of the space.

A possible avenue for future research is exploring ways of improving the hybrid modulator, by using a photo relay as a switch instead of transistor. This would mitigate the stabilization time in the buffer signal by potentially eliminating the DC bias from the transient response of the pulse-width component.

Chapter 4

A code first approach to designing embedded systems

4.1 Introduction

In 1988, the chief scientist at Xerox PARC, Mark D. Weiser, coined the term ubiquitous computing. He envisioned a future of ubiquitous computing that he called:"Calm Computing". He believed that computers should create calm by being quiet and invisible servants that help us to be more efficient in a way that feels intuitive [76]. His vision has inspired several new computing devices, including wearable devices.

These wearable devices have changed the way we perceive personal computing devices. Devices such as the Galaxy Gear [8], the Pebble [6] and the Fitbit [2] have created new ways for us to track our health and check our mail. However, as researchers and engineers begin to explore the potential of wearable devices, they are faced with the challenge of developing and testing custom hardware prototypes.

As we begin to consider the question of prototyping devices by generating hardware from code, there are two fundamental contexts in which the question should be considered. The first context is automatically generating hardware prototypes by analyzing the code they are required to run, thereby allowing software developers with limited hardware

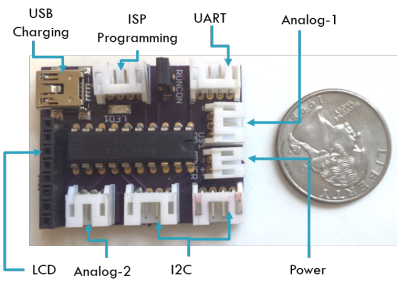


Figure 4.1: Shows the layout of the mainboard (E-unit)

experience to develop their own prototypes. The second context, is that of hardware/-software co-design, where the experienced hardware engineer is interested in optimizing a hardware design by utilizing information from the code that the platform is required to run. Thanks to research done by several researchers, we know a great deal about problems related to hardware/software co-design [73] [35] [30] [26] [28] [71].

In this paper, we consider the first context and attempt to reduce the time that it takes to develop a hardware prototype by proposing a code-first approach to the design of embedded systems. A code-first approach allows a software developer to begin developing a hardware prototype by first writing the code that it will run. After the code has been written, it can be analyzed to determine the hardware configuration that is required. Once the configuration has been determined, a list of modules and their appropriate ports are displayed and the system can be configured by plugging in the appropriate modules. After the software developer has tested the code on the configured hardware prototype, he or she can then automatically generate the design files that are required to fabricate a custom board. This is possible because the platform is comprised of a collection of modular software and hardware components. The hardware components consist of a main board with several hardware modules, while the software component is a modular middleware which consists of stateless libraries, which abstract each hardware component. This modular abstraction creates a direct relationship between the software module and the hardware module, thereby allowing us to synthesize a hardware configuration from a software definition. A code-first approach to designing embedded systems can be achieved by creating

a direct mapping between a stateless modular middleware and modular hardware.

The intuition behind the board's design is that there are a collection of interfaces (i.e. SPI, I^2C, and UART interfaces) that components normally use to interface with microprocessors. By abstracting these interface communication protocols and directly exposing the associated pins, it is possible to design a board that allows software developers to automatically generate the hardware configurations that are required to run their software.

Figure 4.1 shows a picture of the main board, which we call the E-unit. The E-unit is comprised of four expansion ports. Each port is designed to be compatible with a specific interface and therefore accommodates a particular type of module. The first port is designed to be compatible with I^2C interfaces and accommodates modules (such as an accelerometer and gyroscope). The second port is designed to interface with UART-based modules (such as a Bluetooth module). The third port is designed to interface with analog sensing modules (such as a humidity sensor or finger pulse sensor). The fourth and final port is an SPI port that is designed to be compatible with a display module.

In the past, hardware platforms have helped to catalyze innovation [40] [65]. In 2005, researchers at the University of California Berkeley released the Telos mote along with the TinyOS operating system. This platform provided computer scientists with the tools to design and evaluate new protocols for wireless sensor networks and devices. Unlike the Telos platform, our proposed platform is reconfigurable. This means that components can be added or removed from the platform. Reconfigurable platforms for prototyping large-scale devices are becoming increasingly popular.

This paper makes four contributions:

- it presents a platform for prototyping wearable computing devices;
- it proposes a code-first approach to the design of embedded systems that allow programmers to synthesize hardware configurations from software definitions;
- it presents a method for automatically generating custom schematics from a hardware configuration; and

- it presents an approach for automatically identifying port conflicts in hardware/-software co-designs.

4.2 Related Work

Research into rapid prototyping strategies can be divided into two major categories: fixed platform approaches and hardware/software co-design approaches. Fixed platforms use predefined designs, while hardware/software code design approaches use co-synthesis strategies to generate efficient hardware and software partitions.

The hardware/software co-design approaches that are the most similar to our design can be grouped into two subcategories: 1) Interface-Based Designs and 2) Platform-Based Designs. One of the earliest papers on Interface-Based Design by J. Rowson et al. proposed a methodology for separating a component's behavior from how it communicates with other components in the system. Separating components in this way makes it easier to formally verify the component's behavior [67]. Since then, several researchers have explored this interface based design paradigm [62] [61] [23]. Though our platform uses an interface based approach that is similar to previously proposed approaches, our approach extends the interface based paradigm beyond the verification of a single component to the synthesis of an entire system.

The second hardware/software co-design approach is a Platform-Based Design approach. A Platform-Based approach encourages the reuse of pre-designed components through the use of automatic mapping tools [68] [44]. These automatic mapping tools use a layered approach to isolate and map an abstracted top layer description to a more detailed lower layer implementation. Consider the example of a field programmable gate array which uses a compiler to provide isolation and automatic mapping from the top layer VHDL abstraction to the lower layer implementation of logic blocks. In our approach, the libraries provide the abstraction for hardware/software partitions and the automatic mapping to lower level hardware implementation is done by the automatic configuration

generation process.

In addition to many hardware/software co-design approaches, several fixed platform based approaches have also been proposed. Several companies and researchers have developed platforms that allow researchers and engineers to quickly prototype and test their new ideas. One of the earliest prototyping platforms was the phidgets platform, which was developed in 2001 [33]. Afterwards, in 2003, Plessl et al. advocated for the inclusion of field programmable gate arrays (FPGAs) in sensor nodes [64]. They presented a sensor hardware architecture which coupled an fpga with a CPU. By including the fpga and allowing the CPU to configure it, they were able to dynamically configure the chips. Our approach does not use a reconfigurable chip. Instead, we focus on extending the capabilities of the platform by adding and removing external modules.

The earliest occurrence of an extensible platform that we found in the literature was the MetaCricket. [53]. The MetaCricket consisted of a main board which connected to other devices and sensors. The board consisted of a main master controller and a supporting slave controller. The external sensors and expansion boards connected to the master controller while the slave controller controlled the board's internal components. Following the release of MetaCricket, researchers at Stanford University introduced the GoGoBoard in 2004 [70]. The GoGoboard was a low cost programmable control and sensing board. The GoGoBoard was designed to be used as a learning resource in developing countries. With this goal in mind, the researchers focused on ensuring that the GoGoBoard could be assembled in developing countries. This decision influenced the board's design and the components that were selected. Unlike the GoGoBoard and the MetaCricket, our platform consists of both hardware and software components which make the process of prototyping easier.

In 2005, researchers at UC Berkeley proposed the Telos platform along with the TinyOS operating system [51]. The release of the reference platform and operating system sparked innovation in sensor network research. Following the release of the Telos platform, researchers at the UC Berkeley released an updated platform called the TelosB. And in

2009, researchers at UC Berkeley [47] extended the capacity of the TelosB motes by creating two extension boards. The first extension board comprised of a triaxial accelerometer and a biaxial gyroscope. The second extension board provided electrocardiogram (ECG), and electrical impedance pneumography (EIP) functionality to the TelosB platform. These expansion boards demonstrated the flexibility of the TelosB motes. However, as new computing form factors emerge, such as body networks and wearable devices, researchers will need a platform that allows them to prototype devices for these new applications. Unlike the Telos and TelosB motes, our proposed platform can be extended using off-the-self components instead of custom extension boards. More recently, in 2014, researchers at the University of Florida have developed a reconfigurable RFID sensing tag [57]. The platform provides three pins that can be used to connect sensors to the platform. The platform also has an RFID antenna and Cortex M3 microcontroller.

Large companies have also attempted to develop reconfigurable platforms. For example, in 2008 Shimmer began developing their wearable computing development kit [21]. The Shimmer kit was a flexible health sensing kit which consisted of a collection of prebuilt expansion modules. Following the release of the Shimmer development kit, researchers at Microsoft proposed the Gadgeteer platform in 2011. The Gadgeteer platform is an extensible platform that allows researchers and industry professionals to quickly prototype hardware devices [75]. The Gadgeteer's main board is designed to use a collection of prebuilt modules which can be plugged into the main board. Unlike the Gadgeteer and Shimmer platforms, our proposed platform does not require custom modules. It works with off-the-shelf devices.

4.3 Software Design

This section is divided into two subsections. In the first subsection, we present our proposal for a code-first approach to developing embedded systems. In the second subsection, we discuss the design of the modular middleware architecture that is used in our reconfigurable

platform.

4.3.1 A Code-First Approach to Embedded System Design

A code-first approach allows software developers to write an application without worrying about configuring its resource dependencies. For example, Microsoft's entity framework allows software developers to abstract database models as classes [22]. Once the user compiles the program, the framework will automatically generate the database's structure from the class definitions. This increases the programmer's productivity since she does not need to manage the database by writing SQL queries to create, update and delete tables. Instead, the entity framework ensures that the database is compatible with the programmer's implementation.

A code-first approach to embedded system design allows the framework to configure the hardware platform by analyzing the code for hardware dependencies. These dependencies are then used to generate the hardware configuration that is needed to run the application. This is possible because of the modular architecture that maps one software module directly to one hardware component. This one-to-one mapping allows hardware dependencies to be determined by analyzing the software dependencies. However, it is not sufficient to only analyze software dependencies, the microcontroller may have physical constraints that may prevent a valid software model from being executed. For example, the software developer may include two UART libraries in their program. Though this is a valid software model some microcontrollers such as the MPG430G2553 can only accommodate a single UART module without the use of resource sharing hardware. To ensure that our approach only produces valid hardware configurations we need to verify that the software models are compatible with our hardware platform and microcontroller. In the following sections, we discuss the design of the modular architecture and demonstrate how it can be used to construct the software models.

4.3.2 Modular Middleware Architecture for Embedded Systems

Developing embedded software is tedious. The software developer needs to know what control registers to set and what data registers to read. This means that the developer needs to have intimate knowledge of the chip's architecture and the board's layout, and therefore must spend hours reading datasheets. In an effort to reduce the burden on the software developer, several researchers have proposed a collection of hardware abstractions to help address this problem [36] [37]. We take a similar approach by proposing a modular middleware architecture that allows the software developer to quickly build software for our platform. Figure 4.2 shows an overview of the modular middleware architecture. The middleware is comprised of a collection of libraries/software modules. Unlike previous systems that associate modules with generic functionality [29], our middleware architecture associates software modules directly with hardware modules. This allows the middleware to be tailored directly to the hardware platform's current configuration, resulting in a smaller code size.

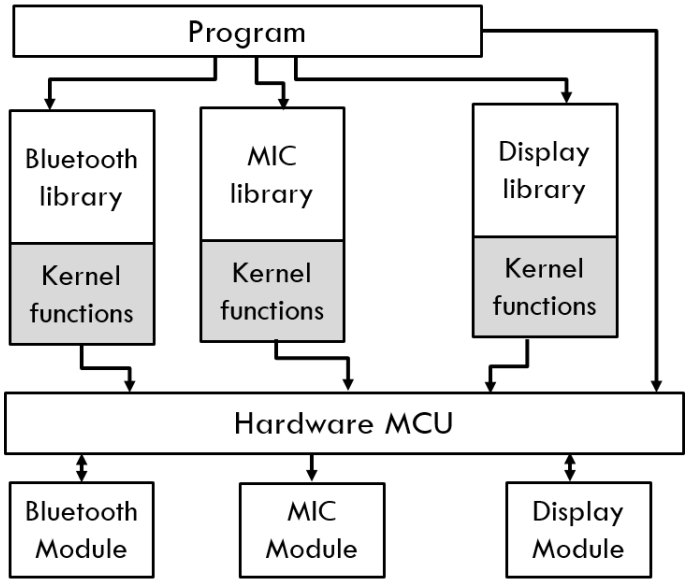


Figure 4.2: An overview of the middleware architecture

Each software module is responsible for abstracting the hardware configurations of

its corresponding hardware module. In particular, each software module is responsible for three tasks: 1) managing the appropriate hardware control and data registers, 2) performing the calculations associated with the module, and 3) managing the system's power. For example, the light-sensing software module is responsible for setting the control registers associated with controlling the MCU's analog to digital converter (ADC) and CPU. Since the sensing module is allowed to set the CPU's control registers, it can disable the CPU to save power while it is waiting for the ADC to settle on a value. Once the ADC has settled on a value, the sensing module can wake up the CPU and perform the necessary calculations.

All software modules are required to be stateless. This means that the libraries do not assume that the MCU's control or data registers are in a particular state. This is an important requirement since memory limitations prevent us from using an operating system to provide resource management and protection. The absence of an operating system would be concerning if the platform was required to run multiple programs simultaneously. However, our platform is designed for specialized embedded applications where resources such as memory are too limited to support an operating system.

Extending the middleware is relatively easy, since it is a collection of decoupled stateless modules. The stateless nature of these modules allows a developer to extend the platform without impacting the other libraries. If the software developer chooses to use these libraries in addition to controlling the registers, he or she is able to do so without impacting the middleware since each module is stateless. Regrading the design of the libraries, there are two seminal questions which we believe need to be addressed. The first question is how do we partition functionality between software and hardware and how does this affect the performance of the design? This problem of dividing the systems functionality between hardware and software is known as the "partitioning problem". Several researchers have studied this problem and have proposed ways to achieve the fastest or lowest cost solution [43] [77] [58] [30].

This implies that there are several partition options depending on the constraints

of our design. So this leads us naturally to the second question. How will a software developer with limited hardware experience choose and configure these different hardware implementations? This is where the library abstraction proves to be extremely useful. One advantage of creating a direct mapping between the hardware modules and the software libraries is that new hardware/software partitions can be selected by including different libraries. This means that a software developer can select the appropriate partition by simply including the library that meets their performance and cost constraints, without having any knowledge of the underlying hardware.

4.4 Configuration Generation Process

In this section, we present an example that shows how a hardware configuration can be generated from a software definition. In particular, we go through a detailed example of how to implement an indoor temperature sensor with an LCD using the proposed code-first approach. The example code in figure 4.3 shows a program that was written in C using our modular middleware architecture. The program begins by including three software modules/libraries. The first is an LCD module which abstracts the display component. The second module is the temperature sensing module. And the third module is the helper module which includes helper functions. These helper functions include the $board\_init()$ function that abstracts the setup of the Watchdog Timer and the $convertADC()$ function that converts longs to formatted strings.

Lines 10-19 represent the program's running loop. In the loop the program instructs the LCD screen to go to position (0,2) and write the string "AT". Once this step has completed, the program calls the tempSense module which sets the appropriate control registers, reads the appropriate data registers and returns the result, which is then converted to a string and displayed. The LCD software module supports unique characters with special codes, for example $0x7f$ represents the degree ^{o} character.

```
#include ``LCD.h''
#include ``tempSense.h''
#include ``helpers.h''

void main(void) {
        unsigned long degrees;
        char* reading;
        board_init();
        LCD_init();
        while(1) {
                LCD_gotoXY(0,2);
                LCD_writeString(``AT:'');
                degrees = tempSense();
                LCD_writeChar(`(');
                reading = convertADC(degrees,1);
                LCD_writeString(reading);
                LCD_writeChar(0x7f);
                LCD_writeString(``C'');
    }
}
```

Figure 4.3: An example program written using the modular middleware that displays the temperature on the LCD.

4.4.1 Introducing Mathematical Abstraction and Notation

There are currently solutions for partitioning software and hardware to create a performance optimized design using microcontrollers [30]. However, we were unable to find any research that addresses the problem of port conflicts that occur when implementing a collection of partitioned designs. A port conflict occurs when two partitions are included in the same design and require the same port. Figure 4.4 shows an intuitive example of a port conflict between two hardware partitions. If a port conflict occurs, it is not possible to implement the design given current hardware/software partitions. Engineers at Texas instruments have developed a tool called pinMux that helps resolve conflicts in complex designs [12]. However, the port requirements must be specified before the software is written. There are also cases in which the pinMux tool is unable to find a solution and therefore port conflicts must be manually resolved by modifying the design to more

efficiently use the ports. In this section, we present an approach for solving this port conflict problem by abstracting the problem as a constraint solving problem. The process is comprised of three steps:

1. abstracting the collection of hardware/software co-design libraries as a set, which represents the system's configured state;
2. representing the microcontroller port limitation as a collection of constraining set functions; and
3. applying the constraint set functions to the system set to automatically identify the software/hardware partition libraries that create port conflicts in the design.

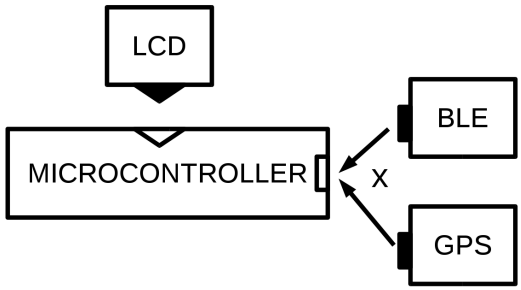


Figure 4.4: Shows an example of a port conflict on the UART port of the microcontroller reference design. BLE represents a bluetooth module.

4.4.2 Abstracting Libraries

The first step in the process of identifying port conflicts is abstracting the hardware/software co-design libraries as a set representing the system's state. These state abstractions are best explained by an example. Throughout our explanation, we will use the code in figure 4.3 to show how each abstraction applies to the specific example.

We can abstract each library as a tuple comprised of a library identifier λ and a set of properties P where P is a subset of the defined superset P'.

$$t = (\lambda, P = \{p_1, p_2..p_n\}) \tag{4.1}$$

We define t_λ as representing the library identifier of the tuple and t_p as representing the property set in the tuple. Each value p_i describes a particular property of the library. These properties are the requirements that the associated hardware partition needs to interface with the software library. Table 4.1 shows a collection of example properties. For example, p_1 may mean that the library is a UART based library, while property p_2 may mean that the component has an operating voltage of $3.3V$. Consider the "LCD.h" library from figure 4.3, it can be expressed in tuple form as: $(\lambda_3, \{p_3, p_4\})$. Table 4.2 shows a collection of example library identifiers and their associated properties.

property	description
p_1	UART module
p_2	Sensing ADC Module
p_3	3.3V power requirement
p_4	display module
p_5	on-board module
p_6	5v power requirement

Table 4.1: An example of the superset P'

library identifier	description	properties	include statement
λ_1	Bluetooth UART Library	$\{p_1, p_3\}$	"Bluetooth.h"
λ_2	IR Sensing	$\{p_2, p_3\}$	"IRSense.h"
λ_3	LCD display module	$\{p_3, p_4\}$	"LCD.h"
λ_4	Temperature Sensing module	$\{p_3, p_5\}$	"tempSense.h"
λ_5	GPS UART Library	$\{p_1, p_3\}$	"Bluetooth.h"

Table 4.2: An example of the superset S'

Now that we have presented an abstraction for the software libraries and their associated properties, we can use this approach to model the software system S as a collection of tuples that are a subset of the defined superset of tuples S'.

$$S = \{(\lambda_1, P_1), (\lambda_2, P_2)...(\lambda_n, P_n)\} \tag{4.2}$$

Consider the following example code shown in figure 4.3. Now that we have defined the supersets P' and S', we can determine the system's tuple representation by examining the library dependencies of the program. The program may contain other libraries that are not associated with hardware modules. By checking the libraries in the program against a known set of libraries it is possible to extract the relevant libraries from the program. Consider the program in figure 4.3 it has three library dependencies: "LCD.h", "tempSense.h" and "helpers.h". We construct the tuple set by checking for the library in the tuple superset S'. If the library is found we add it to the set S. For example, the "tempSense.h" and "LCD.h" libraries are both in the superset S' so we add them to the subset S. However, the "helper.c" library is not in the superset S' so we do not add it to the subset S. This results in the following set:

$$S = \{(\lambda_3, \{p_3, p_4\}), (\lambda_4, \{p_3, p_5\})\} \tag{4.3}$$

4.4.3 Specifying the Constraints

Now that we have abstracted the hardware/software co-design libraries as a set representing the system's state, we need to represent the microcontroller's port constraints, so that they can be checked against the system's state. Recall that a port conflict occurs when there is a discrepancy between the system's state and the microcontroller's port constraints. For example, a program may include two libraries that have the UART property. This means that both libraries require the use of USCI module on the microcontroller. However, the msp430g2553 microcontroller does not have the required pins to support two UART devices, consequently even though the software definition of the program is correct, the hardware is unable to run it because the microcontroller does not have the required number of ports to support the design. In order to verify that a hardware configuration can be generated from the software model, we must show that the software model does not violate any of the hardware constraints. Formally, we can think of these constraints

as a set of set functions

$$C = \{c_1(S), c_2(S), ..c_n(S)\} \tag{4.4}$$

A constraint $c_i(S)$ is considered to be satisfied if it returns an empty set $c_i(S) = \{\}$. If a constraint is not satisfied, we say that it is violated. For a software model to be considered valid it must satisfy all of the constraints in the set C. Each hardware platform will have its own collection of constraints. For example, the proposed platform has a port constraint for UART modules p_1. The platform can only accommodate one UART module. We call this type of constraint a uniqueness constraint.

$$c_1(S) = \{t | t \in S \wedge \exists t^{'} \in S : p_1 \in t_p \wedge p_1 \in t_p^{'} \wedge t \neq t^{'}\} \tag{4.5}$$

Recall that a constraint is considered to be satisfied if it returns an empty set. This empty set check, could be associated with the constraint function by having the function simply return a Boolean. For example, equation 4.5 could have been written more simply as shown in equation 4.6.

$$c_1(S) = \exists! t \in S : p_1 \in t_p \tag{4.6}$$

However, we want the constraint function to identify the violating tuples, so we express it as shown in equation 4.5.

Another type of constraint is a universal property constraint. An example of a universal property constraint is a power constraint, which requires all modules to meet the 3.3V power requirement. Equation 4.7 shows an example of a universal property constraint for the property p_3.

$$c_2(S) = \{t | t \in S \wedge p_3 \notin t_p\} \tag{4.7}$$

We can define the violation set V as the set of the constraints that a system S violates.

$$V(S, C) = \{c | ((c \in C) \wedge c(S) \neq \{\})\} \tag{4.8}$$

We say a system S is consistent with a constraint set C if the violation set is empty. An empty violation set means that the system set does not violate any of the hardware constraints. This formalization is important because as the complexity of the system grows we expect that these abstractions will form the basis for explicitly specifying constraints, though we do not expect to supplant formal systems such as TLA [48]

Now that we have defined the concept of a constraint, let us consider it within the context of the hardware platform. Because of the microcontroller's architecture and the design of the reference platform, the platform has three uniqueness constraints for properties p_1, p_2, p_4 and one universal property constraint for property p_3. Now let us consider how these constraints can be used to check the system set from our running example, shown in equation 4.3

To ensure that the set represents a valid software model, we must ensure that all constraints are satisfied. First, let us consider the uniqueness constraint shown in equation 4.5. The uniqueness constraint requires that a property p_i is only found in a single tuple in the set S. In the set S, the property p_4 occurs only once so the constraint is satisfied. Since properties p_1 and p_2 are not in the set, their uniqueness constraints are also satisfied. Now that we have shown that all the uniqueness constraints are satisfied, we need to show that the universal property constraint in equation 4.7 is satisfied. The universal property constraint requires that all tuples in S have the property p_i. This constraint is satisfied for property p_3 since all the tuples in S have the property p_3. Since all the constraints in C are satisfied, the software model is considered to be valid.

It may be difficult to grasp the purpose of the constraints in a scenario where they are not violated. So let us consider a system where the constraints are violated. To see how the constraints could be violated, let us extend the temperature display example to include a GPS module and a bluetooth module. Figure 4.5 shows the code for the new design. By following the procedure outlined, we can define a new system set S_2 which presents this new GPS and bluetooth capable system. Equation 4.9 shows definition of the new set.

```
#include ``LCD.h''
#include ``tempSense.h''
#include ``Bluetooth.h''
#include ``GPS.h''
#include ``helpers.h''

void main(void) {
        unsigned long degrees;
        char* reading;
        char* gpsReading;
        board_init();
        LCD_init();
        while(1) {
                LCD_gotoXY(0,2);
                LCD_writeString(``AT:'');
                degrees = tempSense();
                LCD_writeChar(`(');
                reading = convertADC(degrees,1);
                LCD_writeString(reading);
                LCD_writeChar(0x7f);
                LCD_writeString(``C'');
                gpsReading = GPS_getValueString();
                bluetooth.send(reading);
                bluetooth.sending(gpsreading);

    }
}
```

Figure 4.5: Modification of indoor temperature sensor to include GPS and Bluetooth. For this example we assume the implementation of GPS module and it corresponding library.

$$S_2 = \{(\lambda_3, \{p_3, p_4\}), (\lambda_4, \{p_3, p_5\}), (\lambda_1, \{p_1, p_3\}), (\lambda_5, \{p_1, p_3\})\} \tag{4.9}$$

Now we have a definition for the system set, we can once again apply the constraints. Notice that this time the uniqueness constraint is violated by both the GPS module and the bluetooth module, since the GPS module and bluetooth module both require the use of the UART ports and there are not enough pins on the microcontroller to accommodate. Notice also that the constraint does not simply return true or false but instead returns the

violating tuples: $\{(\lambda_1, \{p_1, p_3\}), (\lambda_5, \{p_1, p_3\})\}$. These are then returned to the programmers as an error, notifying them of the libraries that have violated the limitation of the reference platform. Once these violation have been identified, the programmer may select new libraries that implement similar functionality but do not violate the port constraint of the microcontroller. For example, the programmer may choose a GPS implementation whose hardware/software partition uses an I^2C interface instead.

4.4.4 Automating Microcontroller Selection

Currently our system is designed to be a proof of concept, but commercial alternatives would allow engineers to select from a wide variety of microcontrollers. This means that these constraints could be used to inform processor selection. If a particular microcontroller does not meet the constraints of the application, these constraints could be used to search for a microcontroller that might be capable of running the application. In this case, the problem becomes an optimization problem where the system is attempting to find the best possible microcontroller whose constraints meet the requirements of a given application. Since companies like Texas instruments currently offer over 40 different microcontrollers [11], approaches that provide automatic microcontroller selection would be extremely useful.

4.4.5 Generating the Hardware Configuration

Now that we have validated the software model, we can begin synthesizing the hardware configuration. The synthesis process is comprised of two steps. The first is to determine the list of hardware modules that are needed. The second step is to determine which ports each module plugs into. Our modular middleware architecture simplifies the first step by providing a mapping from the software libraries to the hardware modules. Table 4.3 shows the mapping of the libraries to the hardware modules. The design of the main board simplifies the second step by providing four unique expansion ports. Once we determine what hardware modules are required, we can look up the associated port in table 4.4.

library identifier	hardware id	mapped module
λ_1	α_1	Bluetooth UART module
λ_2	α_2	IR Sensing Module
λ_3	α_3	LCD Display module
λ_4	α_4	on-board temperature module
λ_5	α_5	Accelerometer Gyroscope module

Table 4.3: An example of library mappings

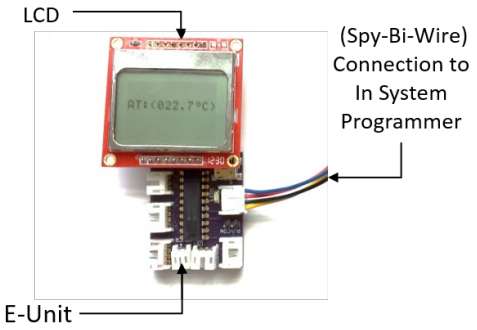
hardware id	port id	port mapping
α_1	Υ_1	UART Port
α_2	Υ_2	Sensing ADC Port
α_3	Υ_3	Display Port
α_4	Υ_4	on-board no port
α_5	Υ_3	Sensing ADC Port

Table 4.4: An example of the port mappings

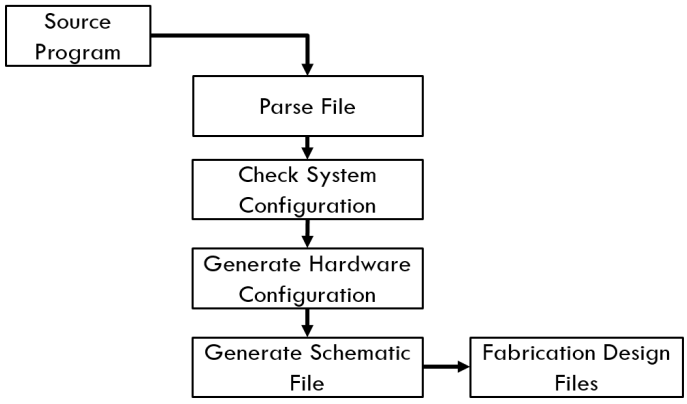
Now let us consider the example in figure 4.3, we determined that the software model for this code was $S = \{(\lambda_3, \{p_3, p_4\}), (\lambda_4, \{p_3, p_5\})\}$. Formally, we can think of the synthesis process as a set function that converts the software model set S to a hardware model set H. We define a hardware model set H as a collection of tuples,

$$H = (\alpha_1, \Upsilon_1), (\alpha_2, \Upsilon_4)...(\alpha_n, \Upsilon_m) \tag{4.10}$$

where α_i represents the hardware module id and Υ_i is the port identifier. We perform the first step of the synthesis process by identifying the corresponding hardware modules for libraries λ_3, λ_4 which are α_5, α_4 respectively. Now that we know what hardware modules we need, we can look up the corresponding ports. This results in the tuple set $H = P\{(\alpha_3, \Upsilon_3), (\alpha_4, \Upsilon_4)\}$. From this configuration, the software developer knows to plug the LCD module into the display port and that the temperature sensor is already a part of the main board. The process of generating the hardware configuration from the software model can be easily automated since it is only a collection of look-ups. Figure 4.6(a) shows a picture of the resulting hardware configuration.



(a) Temperature Sensor



(b) An overview of the process

Figure 4.6: Figure (a) shows a temperature sensor with LCD display. Figure (b) shows an overview of the steps in automating the synthesis process.

4.4.6 Automating the Process

In this subsection, we present the system architecture that implements the proposed formalization. Figure 4.6(b) shows an overview of the system's architecture. We implement an alpha version of the system by developing an eclipse plug-in that is compatible with Code Composure Studio [42].

The system is comprised of four stages: a parsing stage, a constraint checking stage, a configuration generation stage, and a schematic generation stage. A key insight for implementing the system is realizing that the super sets S, H and P can be represented as relational tables. By representing the sets as relational tables, it is possible to express and

validate the system constraints as queries against these relations. This SQL-based architecture also allows the system to be updated as new libraries and constraints are added, since the database can be stored in a central location and queried remotely. However, before constraints can be validated using queries, the system set must be constructed. The system set is constructed by parsing all the files in the project directory and extracting the relevant include statements.

4.4.6.1 Parsing Step

During the parsing step the system reads all the files in the project folder and extracts all of the libraries that are included in the system. This is done by examining the include statements in the project files. Once all the libraries have been extracted, the libraries that are not associated with the middleware must be removed before the system definition can be generated. This list is filtered by querying the tables associated with the S relation. The libraries that exist within the P relation are kept and those that do not, are discarded. The remaining libraries represent the system set S. This system set is then represented as an arraylist of library ids, and is encapsulated as a member of a system node class.

4.4.6.2 Constraint Checking Step

Now that the system set has been determined, the next step is validating this system set. A valid system set is one that does not violate any of the constraints associated with the system. Since each constraint can be expressed as a single SQL statement, it can be further abstracted as a visitor class which operates on the system node. When a visitor operates on the system node it returns a violation set containing all of the libraries that violate the constraint that is associated with the visitor. This violation set is then added to a global violation set that is associated with the system node class. Once all the visitors have visited the system node, and if the global violation set is empty, the system set is considered valid.

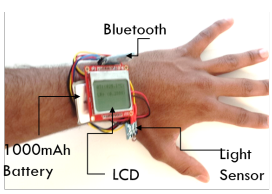
4.4.6.3 Configuration Generation Step

Now that the system set has been validated, the next step is generating the hardware configuration. The hardware configuration specifies which hardware modules connect to a particular port. Recall that there is a one-to-one relationship between hardware modules and software libraries. This direct relationship allows us to query the S and H relations to determine the appropriate port and hardware module.

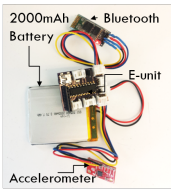
4.4.6.4 Schematic Generation Step

Once the programmer has finished testing the hardware prototype by configuring the platform, she may want to fabricate a custom hardware prototype. However, in order to do this she will need a custom schematic that represents her prototype. In this subsection we explain that it is possible to automatically generate a custom schematic using the proposed code-first approach. Schematic generation is possible because each hardware module maps directly to a schematic block and since each hardware module directly maps to a library, each library also maps to a schematic block.

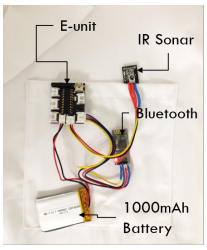
Intuitively, this can be thought of as generating a schematic of the current hardware configuration while removing the unnecessary sections. A hardware schematic is normally represented using a .sch file. Fortunately, the .sch file is a xml based file. Since the schematic for each hardware module is known, each module can be represented as a xml block. Because of this, each block can be added as a child in the .sch xml files. Once the schematic block is added to the schematic, it needs to be appropriately wired. Given that the tree based hardware configuration already presents the appropriate wiring, the wiring for a particular schematic block can be determined by examining the port associated with the appropriate parent node. Since the connections from the ports to the modules are fixed, the appropriate wiring can simply be added to the schematic by looking at the wiring for the associated port.



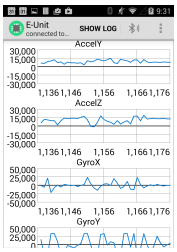
(a) Environmental monitoring watch



(b) Step counter



(c) Infrared system



(d) Android application

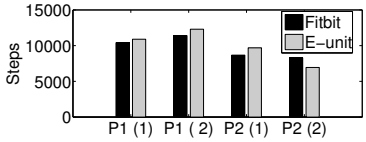
Figure 4.7: Figure (a) (b) (c) show the different possible configurations of the platform. Figure (d) shows a screenshot of the accompanying Android application

4.5 Evaluation

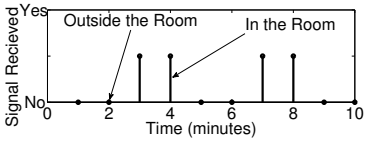
This section is divided into three subsections. In the first subsection, we evaluate the design by building a step counter that communicates with an Android application. In the second subsection, we evaluate the flexibility of the design by (1) prototyping a smartwatch that was designed to monitor the user's environmental exposure to temperature and light; and (2) prototyping an indoor tracking module which tracks the user's location using a collection of landmark infrared transmitters.

4.5.1 Step Counter

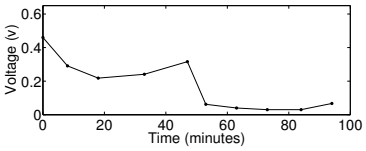
To test the system on a nontrivial example, we developed a prototype step counter and compared the results to the Fitbit. Figure 4.7(b) shows a picture of the prototype step detector. The prototype was programmed to use a windowed peak detection algorithm [19]. We evaluated the prototype by having two participants wear the step counter for two days between the hours of 9am to 9pm. The results of each participant are shown in



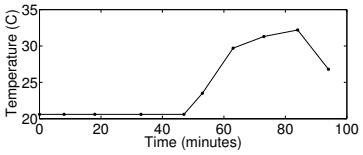
(a) Results of the step detection experiment



(b) Readings from the IR sensor



(c) Readings from the light sensor



(d) Readings from the temperature sensor

Figure 4.8: Figure (a) shows the results of the case study, in which two participants (Person1, Person2) wore both the prototype and a Fitbit for two days. Figure (b) shows the results of the localization experiment. When the signal is high it indicates that the prototype received the signal and when the signal is low it indicates that the prototype has not received the signal. Figures (c) and (d) show graphs of the light and temperature readings collected over a 1 hour period at 10 minute intervals. During the first 50 minutes the device was placed indoors and during the final 45 minutes the device was placed outside.

figure 4.8(a).

To accompany the wearable platform we build an Android application that is compatible with the platform. Figure 4.7(d) shows a screenshot of the application. The application displays a real-time update of the readings that it receives from the platform. The readings are sent to the application from the module via Bluetooth. Once the Android application receives the values, it updates the appropriate section of the interface.

4.5.2 Environment Monitoring Smart watch

We prototyped a smartwatch designed to monitor the user's ambient temperature and light exposure. The watch was constructed using four components: the LCD display module, 3.7 V battery pack, the Bluetooth module and the mainboard. Once the components were placed on the mainboard, we developed software that captured the light and temperature readings and displayed them on the LCD screen. Figure 4.7(a) shows a prototype of the

watch. Figures 4.8(c) and 4.8(d) shows the results of the tests.

4.5.3 Infrared Indoor Localization Device

The third device that we prototyped was an indoor infrared localization device. This wearable device localizes an individual by using unique infrared signatures from landmark infrared devices. Each landmark device is placed in a separate room and produces a unique infrared signal. As the user moves from room to room an infrared sensor located on the wearable device picks up the unique infrared signature of the landmark device. Since this infrared signature is unique to each room, this information can be used to localize the user.

We used our platform to quickly prototype this device using the mainboard and three modules: the infrared sensor, the Bluetooth module and the 3.7V battery. Figure 4.7(c) shows a picture of the final prototype. This prototype is designed to fit in the user's shirt pocket, with the infrared sensor sitting slightly above the rim of the pocket so that it can pick up the infrared signatures from the landmark devices. The wearable device communicated with a smartphone using the Bluetooth module. However, if users would like to explore a lower power option they may elect to remove the Bluetooth module and replace the large 3.7V battery with the smaller 3.3V coin battery. Though the Bluetooth module is compatible with the 3.3V voltage battery, the smaller 3.3V coin does not have the capacity to sustain the Bluetooth module for long periods. So instead of using the Bluetooth module to transmit readings to a smartphone, the software developer can record the signature by writing the signature values to the chip's internal flash memory. These values can be retrieved later by connecting the platform to a PC and reading the chip's internal flash memory. In our evaluation, we used an infrared LED to present the landmark devices. We tested the device by walking into the room with the landmark device for two minutes and then walking out and waiting two minutes. The device was polled at one minute intervals. Figure 4.8(b) shows the results of the experiment.

System	Library ID
Watch	“Bluetooth.h” , “LCD.h”, “tempSense.h”, “lightSense.h”
IR system	“Bluetooth.h”, “irSense.h”
Step Counter	“AccelerometerRead.h” ,“Bluetooth.h”

Table 4.5: The systems and associated Libraries

Recall that constraints are fixed and are associated with the processor and do not change based on the application.

Constraint Type	Properties
Universal Property Constraint	p_1, p_2, p_4
Uniqueness Constraint	p_3

Table 4.6: The systems and associated libraries

4.5.4 System's Limitations

One of the requirements of this approach is that the hardware/software partitions are sufficiently isolated. If the components are not sufficiently isolated they can affect the performance of the microcontroller and the other components. We tested this isolation requirement by connecting a motor controller circuit that was not sufficiently isolated to the E-unit. We also connected a bluetooth module that was sufficiently isolated. We noticed failures in the bluetooth module when both components where operated at the same time.

Another limitation of the system is the fixed power requirement. Since the ports on the reference design only provide 3.3V, it is not possible to connect a module that has a 5V requirement. Other researchers have proposed a custom port design that has two power supplies: a 3.3V supply and 5V [75]. Components that connect to a reference design simply select the appropriate power pin. Adopting this alternate design may help address these issues, but may also increase the complexity of the schematic generation process.

4.5.5 Considerations for a Commercial Solution

Interrupt vectors may prove problematic since the libraries are stateless and therefore cannot rely on the state of the microcontroller's interrupt vectors. This can be resolved by using wait loops (spin locks) in place of interrupts. These spin locks limit the performance of the system where real-time performance is required. However, this can be mitigated by using a real-time operating system with appropriately partitioned hardware/software libraries [16].

In an ideal case we would like to support a variety of operating systems and languages. Supporting languages like python and java would make the approach more accessible to a wider variety of developers. However, the memory limitations of the microcontroller used in our prototype make it difficult to run an operating system and java virtual machine, without which, it is impossible to run java byte code. Nonetheless, it is possible to use third party C libraries with our custom middleware, as long as the libraries do not violate the memory constraints of the chip on the prototype and operate within the constraints of the 16 bit MCU. If the proposed approach is adopted it can be improved, and factors such as the target operating systems and languages may be used to inform the selection of microcontrollers.

Facilitating a commercial implementation of the schematic generation process would require a repository of partitioned hardware/software co-design libraries that adhere to a specific format.

The software developer may select from two options when programming the platform. The software developer can choose to compile, load and debug the code directly from the command line using the GCC toolchain for the MSP430 chips, or use an Integrated Development Environment (IDE). Our preferred IDE is Code Composure Studio (CCS). We have designed the platform so that the MCU can be programmed while the chip is still on the board. This means that the software developer does not have to remove the chip to program it. Instead the developer can connect the board to the in-system-programmer

using the 2 JTAG pins in the programming port.

4.6 Conclusions

In this paper, we presented a code-first approach to designing embedded systems. The proposed approach allows software developers to create hardware prototypes by first writing the code that the prototype is required to run. Once the code has been written, it can be analyzed and an appropriate hardware configuration can be generated. By using library abstractions to create a one-to-one mapping between software and hardware modules, it is possible to determine the hardware dependencies by examining the libraries included in the program. However, there are several challenges associated with this approach. For example, limitations in the microcontroller's architecture could make a given hardware configuration infeasible, even though it is running a valid program. For example, a port conflict may occur when two libraries, whose associated hardware modules use the same port, are both included in a program. In this case it is not possible to generate a viable hardware configuration because there are not enough ports to accommodate all of the modules. To address this issue, we presented an approach for automatically identifying port conflicts. We evaluated our design by building three prototypes: 1) a step counter, 2) an environmental monitoring smartwatch, and 3) an infrared indoor location system.

Chapter 5

Conclusion and Future Work

In this work we proposed two new sensors and a new sensing platform for mobile and embedded devices. The first sensor that we proposed uses the phone's rear speaker and microphone to implement a software based sonar sensor. We found that our implementation was accurate to within 12 centimeters and was suitable for ranging applications up to 4 meters. We found that the sensor performed well in outdoor environments but did not perform well in small indoor environments, due to the effects of reverberation. We also found that though a wide beam was acceptable for ranging applications, it was not suitable for more precise mapping applications. In an effort to create a narrow beam, we designed an external sonar module that plugs into the headphone jack of a smartphone. Readings from this module were then combined with readings from the phone's gyroscope, magnetometer and accelerometer, and used to generate a two dimensional map of a space. The key contribution of this work was the design of a new hybrid hardware/-software modulator that allowed a digital sonar sensor to communicate with a smartphone via the headphone jack. We believe that our hybrid hardware/software modulator can be improved by replacing the transistor with a photorelay, hereby providing prefect isolation and fast switching.

While developing these new sensors, we noticed that the ability to develop both new hardware and software is key to extending the sensing capabilities of mobile and embedded

devices. However, it is difficult for software developers to develop new sensing hardware. To mitigate this, we proposed a code-first approach to the design of embedded systems. Instead of designing software to run within the limited constraints of the hardware, our proposed approach allows software developers to synthesize the hardware configuration that is required to run their software. In the future, as software synthesis methods improve, researchers will be able to specify large scale systems and develop dynamic virtual machines and emulators, which consist of both virtualized sensors and processors. These virtual machines will allow researchers and engineers to evaluate and test their hardware systems entirely in software. Dynamic virtual machines will also provide a common interface for all embedded programs regardless of the intricacies of the hardware or its configuration. These enhancements in the synthesis and specification of embedded devices will result in tighter development cycles and better products. We hope that as developers use our device and sensors to prototype new ones this notion of code-first development will become a standard for specifying systems. However, as a preliminary step, we believe that extending the modular architecture to a real-time operating system, like FreeRTOS [16], will provide the necessary hardware/software abstractions to allow the system to run multithreaded programs.

Bibliography

[1] Captuvo sl42 enterprise sled for apple iphone 5th generation. https://www.honeywellaidc.com/en-US/Pages/Product.aspx?category=enterprise-sleds-for-apple-devices&cat=HSM&pid=captuvosl42v5. Accessed: 2015-06-14.

[2] fitbit flex product page. https://www.fitbit.com/flex. Accessed: 2014-07-8.

[3] Introducing project tango. https://www.google.com/atap/projecttango/#project. Accessed: 2014-08-11.

[4] Lv-maxsonar-ez series. http://maxbotix.com/documents/LV-MaxSonar-EZ_Datasheet.pdf. Accessed: 2015-06-23.

[5] Nasa sending google's project tango smartphone to space to improve flying robots. http://www.theverge.com/2014/4/19/5629254/nasa-google-partnership-puts-tango-smartphone-on-spheres-robots. Accessed: 2014-08-11.

[6] Pebble Smartwatch pebble steal. https://getpebble.com/steel. Accessed: 2014-07-8.

[7] Project tango. https://www.google.com/atap/project-tango/about-project-tango/. Accessed: 2015-06-15.

[8] Samsung Galaxy gear. http://www.samsung.com/us/mobile/wearable-tech/SM-V7000ZKAXAR. Accessed: 2014-07-8.

[9] Speed of sound in air. http://hyperphysics.phy-astr.gsu.edu/hbase/sound/souspe.html, note = Accessed: 2014-12-05.

[10] Strait-line. http://www.irwin.com/uploads/documents/36_laser_tape_25.pdf. Accessed: 2015-06-23.

[11] Texas Instruments development kits and software for low-power mcus. http://www.ti.com/lsds/ti/microcontrollers_16-bit_32-bit/msp/tools_software.page. Accessed: 2015-10-14.

[12] Texas Instruments pinmuxtool. http://www.ti.com/tool/PINMUXTOOL. Accessed: 2015-10-14.

[13] Sensors Overview. http://developer.android.com/guide/topics/sensors/sensors_overview.html, 2013. [Online; accessed 22-November-2013].

[14] Douglas A Abraham and Anthony P Lyons. Simulation of non-rayleigh reverberation and clutter. *Oceanic Engineering, IEEE Journal of*, 29(2):347–362, 2004.

[15] Huzefa Akbarally and Lindsay Kleeman. A sonar sensor for accurate 3d target localisation and classification. In *ICRA*, pages 3003–3008, 1995.

[16] Richard Barry. *Using the FreeRTOS real time kernel: a practical guide*. Real Time Engineers, 2010.

[17] Robert Bemis. Sonar demo. http://www.mathworks.com/matlabcentral/fileexchange/1731-sonar-demo, 2014.

[18] SG Bradley. Use of coded waveforms for sodar systems. *Meteorology and Atmospheric Physics*, 71(1-2):15–23, 1999.

[19] Agata Brajdic and Robert Harle. Walk detection and step counting on unconstrained smartphones. In *Proceedings of the 2013 ACM International Joint Conference on Pervasive and ubiquitous computing*, pages 225–234. ACM, 2013.

[20] Wolfram Burgard, Dieter Fox, Hauke Jans, Christian Matenar, and Sebastian Thrun. Sonar-based mapping with mobile robots using em. In *MACHINE LEARNING-INTERNATIONAL WORKSHOP THEN CONFERENCE-*, pages 67–76. MORGAN KAUFMANN PUBLISHERS, INC., 1999.

[21] Adrian Burns, Barry R Greene, Michael J McGrath, Terrance J O'Shea, Benjamin Kuris, Steven M Ayer, Florin Stroiescu, and Victor Cionca. Shimmer–a wireless sensor platform for noninvasive biomedical research. *Sensors Journal, IEEE*, 10(9):1527–1534, 2010.

[22] Pablo Castro, Sergey Melnik, and Atul Adya. Ado. net entity framework: raising the level of abstraction in data programming. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 1070–1072. ACM, 2007.

[23] Pai Chou, Ross Ortega, Ken Hines, Kurt Patridge, and Gaetano Borriello. ipchinook: An integrated ip-based design framework for distributed embedded systems. In *Proceedings of the 36th annual ACM/IEEE Design Automation Conference*, pages 44–49. ACM, 1999.

[24] Laan Consulting Corp. Sonar ruler. `https://itunes.apple.com/us/app/sonar-ruler/id324621243?mt=8`, 2014.

[25] Murray G Crosby. Frequency modulation noise characteristics. *Radio Engineers, Proceedings of the Institute of*, 25(4):472–514, 1937.

[26] G De Michell and Rajesh K Gupta. Hardware/software co-design. *Proceedings of the IEEE*, 85(3):349–365, 1997.

[27] DS Dean. Towards an air sonar. *Ultrasonics*, 6(1):29–32, 1968.

[28] Stephen Edwards, Luciano Lavagno, Edward A Lee, and Alberto Sangiovanni-Vincentelli. Design of embedded systems: Formal models, validation, and synthesis. *Readings in hardware/software co-design*, 86, 2001.

[29] Dawson R Engler, M Frans Kaashoek, et al. *Exokernel: An operating system architecture for application-level resource management*, volume 29. ACM, 1995.

[30] Rolf Ernst, Jörg Henkel, and Thomas Benner. Hardware-software cosynthesis for microcontrollers. *Readings in hardware/software co-design*, pages 18–29, 2002.

[31] René Farcy, Roger Leroux, Alain Jucha, Roland Damaschini, Colette Grégoire, and Aziz Zogaghi. Electronic travel aids and electronic orientation aids for blind people: technical, rehabilitation and everyday life points of view. In *Conference & Workshop on Assistive Technologies for People with Vision & Hearing Impairments Technology for Inclusion*, page 12, 2006.

[32] Daniel Graham, George Simmons, David Nguyen, and Gang Zhou. A software based sonar ranging sensor for smart phones. In *Internet of Things Journal*, page 1. IEEE, 2015.

[33] Saul Greenberg and Chester Fitchett. Phidgets: easy development of physical interfaces through physical widgets. In *Proceedings of the 14th annual ACM symposium on User interface software and technology*, pages 209–218. ACM, 2001.

[34] Jan-Eric Grunwald, Sven Schörnich, and Lutz Wiegrebe. Classification of natural textures in echolocation. *Proceedings of the National Academy of Sciences of the United States of America*, 101(15):5670–5674, 2004.

[35] Rajesh K Gupta and Glovanni De Micheli. Hardware-software cosynthesis for digital systems. *Design & Test of Computers, IEEE*, 10(3):29–41, 1993.

[36] Vlado Handziski, Joseph Polastre, Jan-Hinrich Hauer, and Cory Sharp. Flexible hardware abstraction of the ti msp430 microcontroller in tinyos. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 277–278. ACM, 2004.

[37] Vlado Handziski, Joseph Polastre, Jan-Hinrich Hauer, Cory Sharp, Adam Wolisz, and David Culler. Flexible hardware abstraction for wireless sensor networks. In *Wireless Sensor Networks, 2005. Proceeedings of the Second European Workshop on*, pages 145–157. IEEE, 2005.

[38] William M. Hartmann. *Signals, sound, and sensation*. Springer, 1997.

[39] Michael P Hayes. Ultrasonic imaging in air with a broadband inverse synthetic aperture sonar. 1997.

[40] Jason L Hill and David E Culler. Mica: A wireless platform for deeply embedded networks. *Micro, IEEE*, 22(6):12–24, 2002.

[41] Dezhi Hong, Ben Zhang, Qiang Li, Shahriar Nirjon, Robert Dickerson, Guobin Shen, Xiaofan Jiang, John Stankovic, et al. Demo abstract: Septimucontinuous in-situ human wellness monitoring and feedback using sensors embedded in earphones. In *Information Processing in Sensor Networks (IPSN), 2012 ACM/IEEE 11th International Conference on*, pages 159–160. IEEE, 2012.

[42] Texas Instruments. Code composer studio users guide. *Texas Instruments Literature Number SPRU328B*, 2000.

[43] Asawaree Kalavade and Edward A Lee. A global criticality/local phase driven algorithm for the constrained hardware/software partitioning problem. In *Proceedings of the 3rd international workshop on Hardware/software co-design*, pages 42–48. IEEE Computer Society Press, 1994.

[44] Kurt Keutzer, Jan M Rabaey, A Sangiovanni-Vincentelli, et al. System-level design: orthogonalization of concerns and platform-based design. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 19(12):1523–1543, 2000.

[45] JR Klauder, AC Price, S Darlington, and WJ Albersheim. July 1960." the theory and design of chirp radars,". *The Bell System Technical Journal*, 39(4).

[46] Ye-Sheng Kuo, Sonal Verma, Thomas Schmid, and Prabal Dutta. Hijacking power and bandwidth from the mobile phone's audio interface. In *Proceedings of the First ACM Symposium on Computing for Development*, page 24. ACM, 2010.

[47] Philip Kuryloski, Annarita Giani, Roberta Giannantonio, Katherine Gilani, Raffaele Gravina, V-P Seppa, Edmund Seto, Victor Shia, Curtis Wang, Posu Yan, et al. Dexternet: An open platform for heterogeneous body sensor networks and its applications. In *Wearable and Implantable Body Sensor Networks, 2009. BSN 2009. Sixth International Workshop on*, pages 92–97. IEEE, 2009.

[48] Leslie Lamport. *Specifying systems: the TLA+ language and tools for hardware and software engineers*. Addison-Wesley Longman Publishing Co., Inc., 2002.

[49] Nicholas D Lane, Emiliano Miluzzo, Hong Lu, Daniel Peebles, Tanzeem Choudhury, and Andrew T Campbell. A survey of mobile phone sensing. *Communications Magazine, IEEE*, 48(9):140–150, 2010.

[50] Patrick Lazik and Anthony Rowe. Indoor pseudo-ranging of mobile devices using ultrasonic chirps. In *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems*, pages 99–112. ACM, 2012.

[51] Philip Levis, Sam Madden, Joseph Polastre, Robert Szewczyk, Kamin Whitehouse, Alec Woo, David Gay, Jason Hill, Matt Welsh, Eric

Brewer, et al. Tinyos: An operating system for sensor networks. In *Ambient intelligence*, pages 115–148. Springer, 2005.

[52] Jó Agila Bitsch Link, Paul Smith, Nicolai Viol, and Klaus Wehrle. Footpath: Accurate map-based indoor navigation using smartphones. In *Indoor Positioning and Indoor Navigation (IPIN), 2011 International Conference on*, pages 1–8. IEEE, 2011.

[53] Fred Martin, Bakhtiar Mikhak, and Brian Silverman. Metacricket: A designer's kit for making computational devices. *IBM Systems Journal*, 39(3.4):795–815, 2000.

[54] John Minkoff. Signals, noise, and active sensors-radar, sonar, laser radar. *NASA STI/Recon Technical Report A*, 93:17900, 1992.

[55] Sanjit KK Mitra. *Digital signal processing: a computer-based approach*. McGraw-Hill Higher Education, 2000.

[56] Alyosha Molnar, Benson Lu, Steven Lanzisera, Ben W Cook, and Kristofer SJ Pister. An ultra-low power 900 mhz rf transceiver for wireless sensor networks. In *Custom Integrated Circuits Conference, 2004. Proceedings of the IEEE 2004*, pages 401–404. IEEE, 2004.

[57] Mohammad S. Islam Muhammad S. Khan and Hai Deng. Design of a reconfigurable rfid sensing tag as a generic sensing platform toward the future internet of things. *IEEE Internet of Things Journal*, 1(4), 2014.

[58] Ralf Niemann and Peter Marwedel. An algorithm for hardware/software partitioning using mixed integer linear programming. *Design Automation for Embedded Systems*, 2(2):165–193, 1997.

[59] Shahriar Nirjon, Robert F Dickerson, Qiang Li, Philip Asare, John A Stankovic, Dezhi Hong, Ben Zhang, Xiaofan Jiang, Guobin Shen, and

Feng Zhao. Musicalheart: A hearty way of listening to music. In *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems*, pages 43–56. ACM, 2012.

[60] James Nord, Kåre Synnes, and Peter Parnes. An architecture for location aware applications. In *System Sciences, 2002. HICSS. Proceedings of the 35th Annual Hawaii International Conference on*, pages 3805–3810. IEEE, 2002.

[61] Roberto Passerone, Luca De Alfaro, Thomas A Henzinger, and Alberto L Sangiovanni-Vincentelli. Convertibility verification and converter synthesis: Two faces of the same coin. In *Proceedings of the 2002 IEEE/ACM international conference on Computer-aided design*, pages 132–139. ACM, 2002.

[62] Roberto Passerone, James A Rowson, and Alberto Sangiovanni-Vincentelli. Automatic synthesis of interfaces between incompatible protocols. In *Proceedings of the 35th annual Design Automation Conference*, pages 8–13. ACM, 1998.

[63] Chunyi Peng, Guobin Shen, Yongguang Zhang, Yanlin Li, and Kun Tan. Beepbeep: a high accuracy acoustic ranging system using cots mobile devices. In *Proceedings of the 5th international conference on Embedded networked sensor systems*, SenSys '07, pages 1–14, New York, NY, USA, 2007. ACM.

[64] Christian Plessl, Rolf Enzler, Herbert Walder, Jan Beutel, Marco Platzner, Lothar Thiele, and Gerhard Tröster. The case for reconfigurable hardware in wearable computing. *Personal and Ubiquitous Computing*, 7(5):299–308, 2003.

[65] Joseph Polastre, Robert Szewczyk, and David Culler. Telos: enabling ultra-low power wireless research. In *Information Processing in Sensor Networks, 2005. IPSN 2005. Fourth International Symposium on*, pages 364–369. IEEE, 2005.

[66] John G Proakis, Masoud Salehi, Ning Zhou, and Xiaofeng Li. *Communication systems engineering*, volume 1. Prentice-hall Englewood Cliffs, 1994.

[67] James A Rowson and Alberto Sangiovanni-Vincentelli. Interface-based design. In *Proceedings of the 34th annual Design Automation Conference*, pages 178–183. ACM, 1997.

[68] Alberto Sangiovanni-Vincentelli. Defining platform-based design. *EEDesign of EETimes*, 2002.

[69] Hyojeong Shin, Yohan Chon, and Hojung Cha. Unsupervised construction of an indoor floor plan using a smartphone. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 42(6):889–898, 2012.

[70] Arnan Sipitakiat, Paulo Blikstein, and David P Cavallo. Gogo board: augmenting programmable bricks for economically challenged audiences. In *Proceedings of the 6th international conference on Learning sciences*, pages 481–488. International Society of the Learning Sciences, 2004.

[71] Mani Bhushan Srivastava and Robert W Brodersen. Rapid-prototyping of hardware and software in a unified framework. In *Computer-Aided Design, 1991. ICCAD-91. Digest of Technical Papers., 1991 IEEE International Conference on*, pages 152–155. IEEE, 1991.

[72] Stephen P Tarzia, Robert P Dick, Peter A Dinda, and Gokhan Memik. Sonar-based measurement of user presence and attention. In *Proceedings of the 11th international conference on Ubiquitous computing*, pages 89–92. ACM, 2009.

[73] Karl Van Rompaey, I Bolsens, Hugo De Man, and D Verkest. Cowarea design environment for heterogenous hardware/software systems. In *Proceedings of the conference on European design automation*, pages 252–257. IEEE Computer Society Press, 1996.

[74] Sonal Verma, Andrew Robinson, and Prabal Dutta. Audiodaq: turning the mobile phone's ubiquitous headset port into a universal data acquisition interface. In *Proceedings of the 10th ACM Conference on Embedded Network Sensor Systems*, pages 197–210. ACM, 2012.

[75] Nicolas Villar, James Scott, Steve Hodges, Kerry Hammil, and Colin Miller. . net gadgeteer: a platform for custom devices. In *Pervasive Computing*, pages 216–233. Springer, 2012.

[76] Mark Weiser and John Seely Brown. The coming age of calm technology. In *Beyond calculation*, pages 75–85. Springer, 1997.

[77] Wayne H Wolf. Hardware-software co-design of embedded systems [and prolog]. *Proceedings of the IEEE*, 82(7):967–989, 1994.

[78] George SK Wong and Tony FW Embleton. Variation of the speed of sound in air with humidity and temperature. *The Journal of the Acoustical Society of America*, 77:1710, 1985.