Establishing a Relative Positioning System to Achieve Mobile Localization

By Brian W. Copeland

[Previous/Other Information: ie S.B., C.S. M.I.T., 2017]

Submitted to the
Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements of the Degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

Massachusetts Institute of Technology

June 2017

Author: _____
Department of Electrical Engineering and Computer Science
May 26, 2017

Certified by: _____
Chris Schmandt
Principal Research Scientist at MIT Media Lab
May 26, 2017

Accepted by: _____
Christopher J. Terman

Establishing a Relative Positioning System to Achieve Mobile Localization

By Brian Copeland

Submitted to the
Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements of the Degree of

Master of Engineering in Electrical Engineering and Computer Science

Abstract

There are an increasing number of situations in which the location of some digital device is known, but the methods to communicate with it are unknown. These situations can frustrate users and lead to the non-adoption to new, beneficial technologies. To combat this, I propose a relative positioning system between digital devices in order for each device to learn the relative locations of nearby devices.

I explore three separate methods of establishing this positioning system. The first uses ultrasonic range-finding to localize nearby devices. The second uses WiFi range finding between mobile devices and stationary WiFi access points. The final method uses machine vision to jointly localize two devices observing the same scene from different angle. Ultimately none of these methods were fully implemented, but an analysis is given for the advantages and disadvantages of using these methods.

Thesis Supervisor: Chris Schmandt
Title: Principal Research Scientist at MIT Media Lab

Acknowledgements

There are a number of people without whose help I would've been unable to finish this journey. First and foremost among these is Chris Schmandt. Not only did you agree to be my advisor in the first place, but when I lost faith in my ability to complete this you continued to encourage and believe in me.

I would also like to thank Chris Terman, the man who helped me through the first leg of this research. Without your guidance as I entered the world of academic research I surely would have faltered.

Great credit is also deserved by my close friends Berj, Guillermo, and Zara. I would not have even pursued this research had we not had countless discussions about it. Even more important though was the constancy and encouragement you offered whenever times were tough.

Lastly, none of this would have been possible without my family. Despite us being 3,000 miles apart for most of the last five years, your support has been unwavering and crucial to all my successes.

Table of Contents

1 Introduction **9**

1.1 Virtual Physical Correspondence Problem 10

1.2 Augmented Reality 12

1.3 Description of Project 13

1.3.1 Exploring GPS 14

1.3.2 Relative Positioning Systems 16

1.3.3 Acoustic RPS 18

1.3.4 Visual RPS 19

1.3.5 WiFi Ranging RPS 20

1.3.6 Modelling RPS Calculations 21

1.4 Potential Applications 21

1.4.1 Smartphone-Smartphone Interaction 22

1.4.2 Smartphone-IoT Interaction 23

1.4.3 IoT Smartphone Interaction 23

2 Related Work **25**

2.1 Ultrasonic Localization 25

2.2 Relative Positioning Systems 27

2.3 WiFi Localization 28
2.4 Vision-Based Localization 29
3 Implementation **31**
3.1 Solving VPCP Using Ultrasonic Ranging 32
3.1.1 Algorithm for Ultrasonic Ranging 32
3.1.2 Resolving Ultrasonic Pings 34
3.2 Solving VPCP Using WiFi 37
3.2.1 Algorithm for WiFi Ranging 37
3.2.2 Issues with WiFi Ranging 39
3.3 Solving VPCP Using Vision 41
3.4 Modelling Relative Positioning Systems 43
3.4.1 Modelling Ultrasonic RPS 45
3.4.2 Modelling WiFi RPS 47
4 Analysis **50**
4.1 Analysis of Ultrasonic Ranging 50
4.1.1 Analysis of Ultrasonic Resolution Issue 50
4.1.2 Feasibility of Ultrasonic Ranging in Establishing an RPS 53
4.2 Analysis of WiFi Ranging 54
4.2.1 Notable Characteristics of WiFi Networks in Experiments 54
4.2.2 Effectiveness of Algorithms 56
4.3 Analysis of Machine-Vision for Solving VPCP 59
5 Future Work **61**

5.1 Expansions to Previously Discussed Methods 61

5.2 New Methods 62

6 Conclusion **65**

References **67**

List of Figures

1-1 Mock-up of VPCP in Augmented Reality 13

3-1 Comparison of High and Low Frequencies 36

3-2 Screenshot of WiFi Application's UI 39

3-3 Two Images of a Scene with SIFT Features Overlaid 43

3-4 UI for Idealized Model of RPS 44

3-5 Examples of Success and Failure in Idealized Model 46

3-6 Idealized Model with Particle Filter 49

4-1 Recorded and High-Pass Filtered Sound Recording 51

4-2 Particle Filter Model at Startup and After Localization 57

4-3 Output of Scene Comparison App 59

Chapter 1

Introduction

The computer industry, since the widespread adoption of the internet, has split into two distinct spheres. The first sphere is that of the virtual world. We exchange messages and tweets, look things up, and consume media all with no physical interaction, save that between our fingers and our keyboards. All the interactions we have with our devices which have no effect on the physical world around us I will define as being in a virtual world.

The second sphere which is a focus of the computer industry is the physical world. By this I do not mean the construction or appearance of our computers and cellphones, but rather the means in which we can change the physical world by interacting with our electronic devices. You can order a product from a retail store and have it appear in front of you a few days (or even hours) afterwards. You can order a car to drive you from place to place, and have it pick you up mere minutes later. These types of interactions where you cause a physical change in your environment by performing an action on an electronic device I will define as existing in the physical sphere of computer interaction.

These two spheres are for the most part distinct. Let us say that, after talking to someone, you decided to connect with them on social media. You would have to look

them up on a social media site the same way you would any person that you hadn't met, or that wasn't right in front of you.

As another example, suppose you were tasked with remotely controlling a television with a computer or a phone. This technology has been well developed via technologies like bluetooth; however, difficulties may arise if there are multiple televisions in proximity to you and you need to decide which one is the right one. Let us suppose, to make your task easier, that each television even has a distinct name when you are trying to connect to one. You would still be put in the position in which you know the physical location of the television you desire, but you don't know its virtual name.

1.1 Virtual Physical Correspondence Problem

The above two example exemplify what I define as the Virtual Physical Correspondence Problem (VPCP). To put it more explicitly, I define Virtual Physical Correspondence Problem as any situation where a user wants to interact with an object whose location is known, but whose *name* is unknown. I further go on to define *name* in this situation not as what we call an object in casual conversation, but rather what we call an object when we wish to interact with it in the virtual world.

To tie this back to the television example, in casual conversation I would call the television I wish to connect to simply a television. However, its name will generally be

something more identifying, such as *Samsung Electronics UN55MU6300* or *Brian's Living Room TV*.

As another example of how I use the term *name*, let us suppose you are in the social media scenario from the previous section in which you want to connect to someone you just met via social media. In this case you, or more accurately your mobile device, would have to communicate with the other person's mobile device to complete the connection. The physical location then is just the location of the other person's mobile device, and the *name* is the device ID of the other person's mobile device that your phone would use to communicate with it.

This problem may seem relatively isolated and unusual at the moment, but I believe there are two factors which will make the VPCP more prevalent. The first factor is the burgeoning influence of the Internet of Things (IoT). The Internet of Things will cause more and more objects in our world have electronic components which connect to the internet, and consequently more objects with known locations, but unknown *names*.

The second factor which will increase the prevalence of situations where the VPCP is present is the continued refinement of augmented reality technologies. Augmented Reality (AR) provides a natural way to interact with an object's physical location and virtual properties concurrently. The full potential of AR in relation to the VPCP will be discussed in the following section.

1.2 Augmented Reality

Augmented and virtual reality are often used interchangeably, so I will attempt to define AR in the context it will be used for the rest of this work here. The Merriam Webster Dictionary defines Augmented Reality as “an enhanced version of reality created by the use of technology to overlay digital information on an image of something being viewed through a device (such as a smartphone camera).” Of particular note here is that an image of the real world must be presented to the user. This is in contrast to virtual reality, where the user by design has no concept of what is happening in the real world.

AR can be extremely effective in solving the physical virtual correspondence problem. On the simplest level consider the case where a user wants to know the *name* of an object they are looking at in their field of view. This is generally infeasible, but if the user is using AR the *name* of the object can be projected onto their AR interface. An example of this phenomena is shown in Figure 1-1.

This is a simple, yet rather clunky means of solving the VPCP. In this method a user would have to exit the current AR experience just to interact virtually with a real world object, such as the phone in Figure 1-1. In a sophisticated AR experience, interaction with real-world objects could be done by touching a screen location corresponding to the real-world object, or even just looking at the object.

Figure 1-1
A mock-up example of the VPCP being solved in Augmented Reality. In this image a mobile phone (BLU Advance 5.0) sits on a table. A virtual label sticking out from the phone is superimposed on the image, much in the way it would be with an AR interface.

One last thing to note about AR is that the only hardware components strictly necessary to create an augmented reality experience are a camera and a screen. Thus, AR can be created by hardware as varied as Google Glass, Microsoft's HoloLens, and regular smartphones. This fact influenced a design imperative for my project, namely that whatever I create should be able to run on most, if not all, hardware capable of supporting AR.

1.3 Description of Project

The ultimate goal of my project was to create an AR experience similar to that shown in Figure 1-1. Effectively I wanted to design and create a system where two smartphones could interact and find both the virtual *name* of their counterpart, and the physical

location. This is equivalent to solving the VPCP in the simple situation with only two devices.

1.3.1 Exploring GPS

One way this could be solved is through GPS. Using this method, both phones would start off by determining their GPS coordinates, and communicating these to the other phone. Then, each phone would determine the relative direction and distance to the other phone. Next, each phone would figure out which way it was facing using its internal compass and/or gyroscope. Finally, each phone would figure out if the other phone was in its field of view, and if so project it to the corresponding location on its screen.

This method, though relatively simple to implement, has a number of systemic issues. Most of these issues relate to the fact that GPS is used as the primary means of determining location. Some of these issues include:

- **Poor Accuracy:** Accuracy of the system is limited by the precision of GPS data, and specifically the accuracy of GPS in phones. As shown in Chapter 2 this can be as low as 10 meters on smartphones.
- **Inability to Function Indoors:** GPS has many additional error sources when working indoors, making it far too inaccurate for my uses.

The environment in which solving the VPCP would be most useful has two notable characteristics. The first is that there should be multiple devices nearby which have virtual or online means of interaction. If there were just one other device in the vicinity that had these means, it would be simple to connect to it because it would be the only one. Currently there is a much higher prevalence of IoT and mobile devices indoors than outdoors.

The second characteristic of an environment which solving the VPCP would be useful is that a person interacting with the system should be able to see the object they wish to interact with. This means the objects which the VPCP must be solve on are often within a few meters of each other.

These two characteristics combined mean that GPS in its current state on smartphones is a poor method to use in order to solve VPCP. To make matters worse, many IoT devices do not have any GPS hardware on them. This would mean that in order for an mobile device to interact with these devices in an AR setting, each IoT device would need to know its precise location. Calibrating the global position of each IoT device would be infeasible as the number of these devices grows.

1.3.2 Relative Positioning Systems

If you recall from from the beginning of Section 1.3 my ultimate goal was to create a system where two smartphones could determine both the virtual *name* and physical location of the other smartphone. The disadvantages of using GPS detailed in Section 1.3.1 forced me to find an alternative to GPS as a method for one mobile device to find the location to of another mobile device. However, one key insight I obtained by considering GPS was that it was not necessary to know the *global* position of the two smartphones, but only their position *relative* to each other.

Specifically, in the GPS scenario after I used GPS to determine the coordinates of each smartphone, I would only use those coordinates to determine the distance and direction between the two phones. Each smartphone needed to have knowledge only of the distance and direction to the other smartphone. This led me to the conclusion that a Relative Positioning System (RPS) would be sufficient.

In a relative positioning system, each node (in this case a smartphone) in the system records its position only in reference to the other nodes in the system. This has a couple potential advantages over a global position system, where each each node has a well defined and distinct position.

The first advantage is that a relative positioning system is not reliant on outside sources to determine the position. The nodes within the system are both capable and required to determine their relative position by themselves. Not only does this give a relative positioning system flexibility to work anywhere, but it also means that it will persevere even if outside infrastructure (such as GPS satellites) fail.

Another potential advantage of relative positioning systems is that someone outside and not connected to the system does not necessarily know the global location of the nodes in the system. This is particularly good for maintaining privacy of location, an increasing concern in the modern era. An observer outside of the network may be able to determine your proximity to other people, but have no clue about your absolute location. This anonymity of location breaks down however if someone in the network discloses both their global and relative position.

The majority of my time on this project was spent exploring and evaluating methods and algorithms to establish a RPS for the two smartphone scenario I detailed at the beginning of Section 1.3. In total I explored three different methods of establishing a RPS: acoustic, visual, and WiFi triangulation.

1.3.3 Acoustic RPS

In the acoustic method of establishing an RPS, I had two cell phones in a relatively enclosed environment communicate via two separate media. The first medium was sound: One phone would send out a short ultrasonic ping. Simultaneously that phone would send out a message over Bluetooth. The other phone would record the time at which it received both signals, and determine the distance apart using the delay.

If the distance between the two phones was successfully determined, the next step would have been determining how far and in which direction each phone moved. This would be done via dead reckoning, a process which combines the data from a number of sensors on the phone to make an approximation for how far and in which direction movement occurred.

Finally, the distance data and the dead reckoning data for each phone at a number of time steps was to be combined which would allow triangulation to be used to determine the direction between the two phones. Unfortunately I was never able to implement this fully. I had an inextricable issue in resolving the ultrasonic ping, which I describe more fully in Chapter 3.1 and 4.1.

1.3.4 Visual RPS

The visual method to establish a relative positioning system is based loosely on how you might describe the location of an object to a friend. For example, say I wanted to tell my friend where my television was. I might say “It is above the TV stand, across from the couch.” This method establishes a position by finding salient points in the room which both you and your friend can easily recognize.

This is approximately what SIFT, the primary algorithm I used to visually establish a RPS, attempts to do. Given an image it finds points which can consistently be identified in an image, even if the image is taken from a different depth or angle. Then if two different phones are concurrently taking a picture of the same area in the room, they will be able to communicate to each other this fact. Furthermore, they will be able to use the differences in the locations of the salient points identified to find the relative angle and distance between the two phones.

This method can in theory give all the information needed to establish a RPS between the two phones. However, in practice I found the SIFT algorithm to be too computationally expensive to be feasibly implemented in a real-time setting on the phone I was using.

1.3.5 WiFi Ranging RPS

The final method I explored in order to establish a relative positioning system between two phones was WiFi triangulation. This method is similar to the acoustic method in 1.3.3 in that a key step is determining the distance between two devices. However, in this case I calculated the distance between a mobile device and a WiFi access point instead of between the two mobile devices themselves. This distance can be approximated using the signal strength from a WiFi router measured on the phone.

After calculating this distance, my plan was to use the triangulation method used in the acoustic RPS method to determine the distance and direction between each phone and each WiFi router. Then, knowing the distance and direction between each phone and a WiFi router, I would combine these measurements to calculate the distance between the two phones themselves. My hope was to average this calculation over a large number of routers to get a more accurate distance calculation.

However, I did not implement the steps detailed in the paragraph above on a smartphone, or any other mobile device. In order to create and refine an algorithm to get the distance and direction to a WiFi router, I created a model representing the interactions between different devices in the real world. This model is described in the next section.

1.3.6 Modelling RPS Calculations

After a great deal of trial and error, I realized that implementing and testing triangulation and localization algorithms on the mobile devices was far too complicated until I understood them fully. In order to both increase my comprehension of these algorithms and to refine them in a quick and easy way I created a simple python of two smartphones interacting with the world and each other to solve the VPCP.

This model had a number of advantages over implementing and testing these algorithms in the real world. One such advantage was that I could change the amount of noise in my measurements and see how the accuracy of my algorithms changed. Another advantage of the python model was that it was much quicker to create new and test new algorithms. Overall, this model was integral both for determining the accuracy constraints of my system, and for helping me to develop more efficient algorithms.

1.4 Potential Applications

The main goal of my research was to solve the virtual physical correspondence problem; that is, to allow a user to determine the virtual identity or name of nearby device with only knowledge of its location. There are many potential applications of solving this VPCP, but in the next sections I will characterize a few of these applications by which devices are solving the VPCP.

1.4.1 Smartphone-Smartphone Interaction

The first category of applications of solving VPCP is where a smartphone successfully determines the location and *name* of a nearby smartphone. Furthermore, in this situation I will consider that both smartphones are being held and used by distinct users. Solving the VPCP in this situation allows either user to establish a virtual connection to the other user.

An example of this virtual connection is the social media application discussed in 1.1. In this example, a user would be able to connect to another nearby user on some form of social media just by knowing their location.

Another application here requires both users to be using their smartphones in an AR mode: ie in a situation where they are looking at the world through the screen on their phone. Say in this situation one user augments the image on his phone by overlaying an image of a poster or presentation onto the screen. Since they are in an AR mode, the position of this poster or presentation on the user's screen corresponds to a real location. Since this user’s phone is knows the location of the other smartphone in this scenario, his phone can communicate to the other phone exactly where the poster or presentation is in physical space. In such a way, both users in this scenario could see

the same poster or presentation in the same spot, despite the fact it only exists on their smartphones.

1.4.2 Smartphone-IoT Interaction

Another class of potential applications of creating a system that can solve VPCP involves situations where a user with a smartphone is trying to interact with an IoT device such as a smart TV or smart thermostat. The user will generally know the location of the IoT device, but it may not be easy or intuitive to interact with this device through conventional means.

However, if the user's smartphone has a correspondence between the location and name of the nearby IoT device, then the user can interact with this device easily with just knowledge of the location. As the scope of the IoT grows in the coming years, this method of interaction could become much more useful.

1.4.3 IoT-Smartphone Interaction

A final class of applications of my research that I describe here involves interactions between an IoT device such as a sensor and a user holding a smartphone. However, as opposed to the situation in 1.4.2, in this example the sensor is attempting to find the physical location and identity of the user with a smartphone.

An example of where this would be useful would be in a retail setting. If a user was interested in hearing information about sales a store was offering and was physically in the store, the store could give targeted advertising based on the location of the user in the store. This could benefit the user in getting them cheaper products, and the store in increasing customer loyalty and improving customer experience.

Chapter 2

Related Work

One of the inspirations for this work was a project created by the media lab at MIT called “Reality Editor” [1]. This project aimed to ease the interaction between humans and smart devices. They did this by solving what I call the virtual physical correspondence problem, but in a different way than I attempted. The researchers working on this project put unique visual identifiers on different smart devices that a computer could recognize. This allowed them to then observe these devices through a smartphone camera and determine what device they were looking at. The researchers could then easily manipulate certain aspect of the smart device (such as increase the volume of a speaker, or turn lights off).

2.1 Ultrasonic Localization

There are many existing systems that succeed in localizing objects using high-frequency sound waves [2] [3] [4] [6] [7] [9]. The majority of these systems require ultrasonic beacons to be placed in key locations in a room. These beacons are then used to triangulate the position of a mobile device moving in the room or area.

One of the variations in these systems' architecture is precisely how the distance between mobile device and beacon is measured. One common way this is done is by measuring the difference in time of flight (ToF) between the ultrasonic signals from different beacons [2] [3]. Another common method is to record the angle of arrival (AoA) of the incoming signal from the beacons [2]. A final method is to concurrently send out a radio wave concurrently with the ultrasonic sound blip and measure the time difference of arrival [6] [7].

Another difference between different implementations of ultrasonic localization systems is which actor in the system is emitting the noise. Generally the stationary agents will emit ultrasonic sound, and the mobile agent will simply record it and calculate its position. However, in some systems [2] the mobile agent acts as the ultrasonic beacon and the stationary agents record the sound.

Overall ultrasonic localization systems with stationary beacon can achieve high degrees of accuracy. In ideal environments these systems can achieve errors as small as 1-2 cm [3] [4], but 10 cm is more accepted in non-ideal real-world circumstances with multipath reflections.

One disadvantage of these systems is that they require significant time and resource overhead to set up. Not only do ultrasonic emitters/receivers need to be purchased and

set up, but the system often has to be calibrated for the environment it is meant to operate in [4] [6].

2.2 Relative Positioning Systems

Obtaining the relative location of different entities without nodes of known location has been accomplished in a number of systems [9] [10] [11] [12]. The methods used to establish this RPS vary wildly from one implementation to another. Some use GPS [10], others use ultrasonic ranging [9] [11] [12], others use radio signals like WiFi and bluetooth [9] [13], and many of these systems use a multitude of these methods.

Another common approach to establishing a RPS between different devices rely on dead reckoning, or calculating a devices new position by integrating data from an internal acceleration sensor, an internal compass, and its previous position [8] [9]. Although this is probably the oldest method for establishing, it is not alone sufficient to track a smartphone as it is carried by a person. This is because of a phenomena called drift, in which small inaccuracies in accelerometer readings lead to large inaccuracies in the estimated location over time.

Many of the acoustic systems to establish an RPS have relatively high accuracy [9] [11] which is on the order of centimeters. However, these systems primarily operate at close

ranges, generally < 3 meters. Although this is an ideal distance for file sharing [11], it is too close a range for many of the applications I describe in Chapter 1.

2.3 WiFi Localization

One common way to establish the location of one or multiple mobile devices indoors is through WiFi Localization [14] [15] [16] [17] [24]. WiFi networks and access points are prevalent in most indoor environments these days. Additionally, WiFi access points have the dual advantage of being stationary and sending out signals which can be used to estimate how far away they are.

The most widespread method for WiFi localization is called WiFi fingerprinting [16] [17]. The basic idea behind this method is that each location will have a certain set of WiFi access points nearby, along with thresholds for the signal strength (RSSI) for these access points. The advantages of this method are that it is relatively simple and very accurate. However, the main disadvantage is that it is time-consuming to get the fingerprint for a lot of locations. To get this data, someone needs to walk to every location in the database. This is increasingly being accomplished using various forms of crowdsourcing [16] [17].

A less common method for WiFi localization is proposed by [14]. They propose a method to colocate a user and WiFi access points in an environment using RSSI values combined with the dead-reckoning measurements made by a user's smartphone. Although the accuracy here is considerably worse than WiFi fingerprinting (~4m as opposed to ~1m), it offers the advantage that no start up work is needed to create the system.

2.4 Vision Based Localization

Another possible way for mobile devices such as smartphones to establish their location when GPS is unavailable or inaccurate is using machine vision [18] [20] [21]. The most common way to do this is to have someone walk around an area taking pictures of their surroundings. These pictures are then stored in a database along with the precise location they were taken at. When a user want to figure out where they are in the building, they simply take a picture of their surroundings and query the database for images similar to the picture they took.

This method is actually fairly similar to the WiFi fingerprint localization method discussed in Chapter 2.3. In both methods, the system is set up by walking around a space and recording some characteristic of the environment. In the WiFi fingerprinting method this characteristic is the name and strength of nearby WiFi routers, while in the machine vision method the characteristic being recorded is the visible appearance.

This means that the machine vision method will also suffer from some of the same disadvantages of the WiFi fingerprinting method. These disadvantages include the fact that it is time-consuming to initialize the system, and that the precision is limited by the size of the database being used.

Unfortunately for the machine-vision method, finding similar images to a given image can not be done with a simple image correlation function. This is because the images in the database may have been taken at a different zoom or a different angle. Furthermore, doing a simple correlation function to tell if two images are similar would grow too time-consuming as the database of images grew in size. This is often overcome by using SIFT or SURF [18] [20].

SIFT (Scale Invariant Feature Transform) [19] and SURF (Speeded Up Robust Features) are two similar algorithms for finding and characterizing key points in an image. One of the key benefits of using these algorithms instead of a simple image correlation is that the same keypoints will often be found on images taken at different levels of zoom, and even at different angles. Another benefit is that instead of storing a whole image or set of images for each location in a building, you only need to store the key points found by SIFT or SURF.

Another way to avoid naive image correlation between an image taken by a mobile device and an image in the database is to compare the color histograms [21]. This

method is even cheaper and faster than the SIFT/SURF method, however it suffers from the disadvantage that the precision is generally worse. On top of that, it is more prone to errors in area with uniform color appearance.

Chapter 3

Implementation

If you recall from chapter 1.3, the main goal of my research was to create a system where two smartphones could determine each other's relative position, effectively solving the virtual physical correspondence problem in a simple case. In 1.3 I briefly described the methods I experimented with to accomplish this goal. In this Chapter, I will elaborate on these methods.

While exploring methods to solve the VPCP, I primarily experimented with a HTC Desire 626s and a BLU Advance 5.0. These were both smartphones on the lower end of the quality spectrum. I believe this fact may have impacted the results of my experiments. As an example, I think my inability to resolve the ultrasonic signals (as described in 3.1) was a result of the poor audio detection on the BLU Advance. Despite this, I considered it important that I used lower quality hardware. One of my design goals was to enable a process to solve VPCP on any device; so working with devices on the lower end of the

spectrum means the methods I used would have a higher chance of working on the rest of devices I might try.

3.1 Solving VPCP Using Ultrasonic Ranging

The first method I tried to solve the VPCP was to have two cell phones determine their relative signals using a combination of ultrasonic pings and a bluetooth connection. To do this, I created an android application that both smartphones would simultaneously run.

3.1.1 Algorithm for Ultrasonic Ranging

The basic outline of the algorithm for my android application is described below:

1. The two smartphones begin the process by pairing over bluetooth. Bluetooth connections using the standard android API generally have a ‘master’ and a ‘slave’ device. The master device can be connected to a couple different slave devices, however each slave device can only be connected to one master device. During my research, I arbitrarily set one device to be the master and the other to be the slave.

2. Next, the master smartphone emits an ultrasonic ping. This ping is a tone which lasts for 100 ms. and is replayed every second. The frequency of this tone is 20,000 Hz. I choose this frequency because it was just above the human hearing threshold.
3. Simultaneously the master smartphone sends out a message via bluetooth to the slave smartphone. The purpose of this message is to inform the slave smartphone when the ultrasonic signal was sent.
4. After the slave receives both the bluetooth and the ultrasonic signal from the master, it takes the difference in time of arrivals (ToA) to compute the distance between the two devices. It uses the following formula :

$$distance = \frac{\Delta ToA * s * c}{c - s}$$

Where ΔToA is the time difference between the ultrasonic and bluetooth signal, s is the speed of sound, and c is the speed of light.

5. The slave then responds to the master with the value it calculated for the distance between the two devices.
6. The slave and the master both use their internal accelerometers, gyroscopes, and compass's to determine their approximate change in position over the next second. When they next send information via bluetooth (steps 3 and 5) they include this change in position.
7. Finally, each device combines a few specific pieces of data to find the location of its partner. These pieces of datum are:
    - The current distance between the two smartphones

- The distance between the two smartphones at some previous time step
- The net direction and distance each smartphone has moved since that previous time step

Unfortunately, I was unable to get step 4 of this process working correctly. Specifically, I was unable to resolve ultrasonic pings on my slave device. There are a number of possible reasons that this could have happened, and they will be discussed further in Chapter 4.1. However, for now it should be noted that I only implemented steps 1-5 on this particular android application. Step 6 was never fully implemented, and step 7 was only implemented in a simulated environment on my computer. More about this simulation will be described in Chapter 3.4

3.1.2 Resolving Ultrasonic Pings

After I was initially unable to have one phone pick up the ultrasonic pings sent out by the other phone, there were a number of methods I tried to correct the issue. These methods are all different ways to answer the question of whether or not the phone is recording a high frequency sound. These methods are the Fast Fourier Transform, zero crossings, and a high pass filter.

The first method I implemented to determine if an ultrasonic ping was being sent was the Fast Fourier Transform (FFT). The purpose of this algorithm is to describe a signal

in terms of the frequencies that make up the signal. In my application, I was checking the audio signal recorded by the phone to check if the 20,000 Hz frequency was a main component in the audio signal. If it was, then my application would conclude that the other phone had just sent an ultrasonic ping.

Unfortunately, I was never reliably able determine if the 20 KHz frequency was present in the audio signal. Depending on the thresholds I set, the application would either determine that ultrasonic pings were being sent at random times, or that no ultrasonic pings were being sent at all.

Another potential disadvantage of using the FFT to determine when ultrasonic pings were sent is that the FFT requires a certain minimum length of the audio segment in order to determine if certain frequencies are present. This is an issue in my application because my application relied on knowing precisely when an 20KHz frequency started. The FFT could only give a rough approximation for this time.

These factors caused me to pivot to my next method of resolving ultrasonic pings: zero-crossings. The intuition behind zero crossings is that when a high frequency signal is being recorded the air pressure and thus the voltage recorded by the audio receiver will switch very rapidly between positive and negative values (where 0 is the base).

To implement zero-crossings, I had my application look at a small window of the audio data and keep track of how many times the signal went from either a positive to a negative value, or from a negative to a positive value. If this was above a certain threshold I would consider that an ultrasonic signal was being sent. I started with the threshold at 20 crossings per 100 data samples, with data being sampled at 44100 Hz.

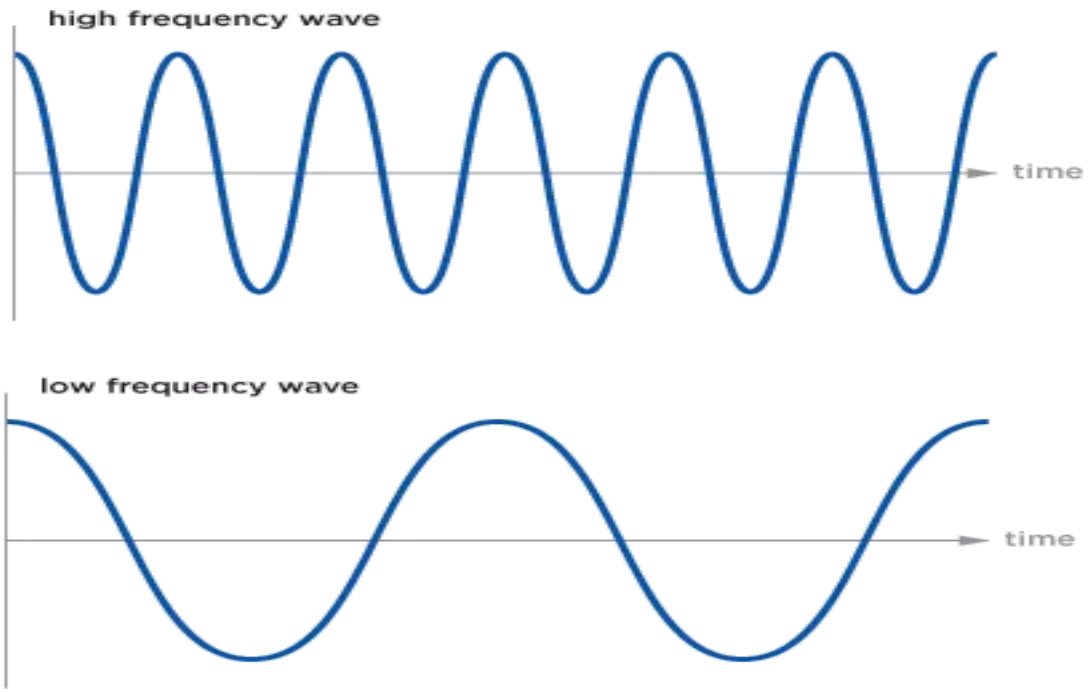

Figure 3-1
A comparison of high frequency (top) and low frequency (bottom) waves. During the same time period, the high frequency wave switches sign many more times than the low frequency wave.

However, similar to the FFT method neither this nor any other threshold proved sufficiently accurate to resolve the ultrasonic ping.

The final method I tried was to use a high-pass filter to resolve the ultrasonic pings. The high pass filter was implemented in software on the android application and was designed to filter signals whose frequencies were under 18 KHz. This method gave even worse results than the previous methods, as the signal to noise ratio was virtually non-existent. After discovering this, I decided to abandon the ultrasonic method of finding the distance between the two smartphones and move onto other methods.

3.2 Solving VPCP Using WiFi

Another method I implemented to enable two smartphones to find the relative position of each other was WiFi ranging. In this method, each smartphone would first try to determine the location of nearby WiFi access points. After this was done, the phones would notify each other about the distance and direction to nearby WiFi access points. After comparing results, both phones would be able to find an approximate location for the other phone.

3.2.1 Algorithm for WiFi Ranging

Similarly to the algorithm I describe in 3.1.1, I implemented this as a standalone android application, which both phones were to run. Additionally, as happened with the ultrasonic ranging application, I never completed this application fully. The algorithm I give below is what I would have implemented had I not run into difficulties:

1. Determine the distance to nearby WiFi access points using the Received Signal Strength Indicator (RSSI). It was necessary to use only nearby WiFi access points because various form of error, namely multi-path propagation errors, greatly reduce the accuracy of distance calculations over long distances. In my experiments I limited the RSSI signal to 80, which in ideal circumstances is close to 25 m. An example of this interface is shown in Figure 3-2.
2. Each smartphone then waits for one second, during which it uses dead reckoning to calculate how far it has moved. Note that the 1 second here is an arbitrary amount of time chosen by me for convenience purposes. It has not been optimized or changed at all.
3. After 1 second, each smartphone repeats step 1 to get a new distance estimate between it and the nearby WiFi access points.
4. Each smartphone then uses these three pieces of information--its current distance to a beacon, the amount it travelled in the past second, and the distance to the beacon a second ago-- to calculate the direction to that beacon.
5. Each smartphone sends the list of nearby access points, along with its position relative to them, to its partner smartphone.
6. Upon receiving its partners message, each smartphone find all WiFi access points that both smartphones are close to. For each of these access points, the smartphone combines the displacement between it and the access point to the displacement between the other phone and the access point using vector addition. For each access point, this quantity is an estimate of the relative

location of the other smartphone. These quantities are then averaged together to get a refined estimate of the relative location.

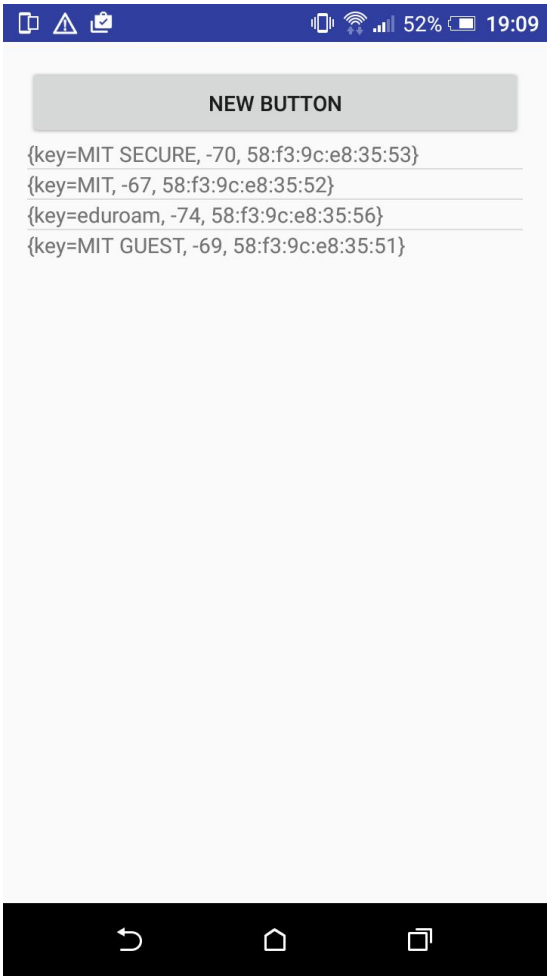

Figure 3-2
A screenshot of the user interface for the application that I coded to determine the range between a WiFi access point and the device running the application. At the top of the application the "New Button" button initiates the process of recalculating the distance to nearby access points. In the gridview below that are the names, RSSI strengths, and id's of these access points.

3.2.2 Issues with WiFi Ranging

After implementing step 1 of the algorithm described in 3.2.1, I decided to stop and test how accurate the calculated distance between WiFi access point and smartphone was. I started with a qualitative sanity test. Although a quantitative would have been a more rigorous test of my system, at the time I was only interested in whether my system was working or not.

I started this qualitative test by measuring the consistency of the results. When I tested in a space with many different access points in close proximity (MIT's campus) I observed that the RSSI and consequently the calculated distance varied greatly from one measurement

to the next. Specifically, I noticed a few jumps which the RSSI changed by over 30. At the distances I was measuring, this would have been an equivalent distance jump of about 15 meters.

Upon further investigation I realized that one cause of this issue was that the device device id for the network sometimes changed. For example, the id of the access point for the MIT network changed from '58:f3:9c:e8:35:53' to '58:f3:9c:e8:35:56'. Note here that the two id's are identical except for the last number. From this, I concluded that the access point I was receiving a signal from changed from the first time I accessed the system to the second time. Normally this isn't an issue when you are connected to a WiFi network as the network will do its best to make sure you are connected with the same access point until the signal degrades too much. This issue only came up in my application because my application was exploring the connection data from different networks instead of just connecting to one.

This led me to the conclusion that I would have to rethink how I got the RSSI from nearby access points. In the meantime, I then worked on creating an efficient and accurate algorithm to complete steps 4-6 of the algorithm described in Chapter 3.2.1. These steps roughly correspond to how I would integrate the data from the WiFi ranging of the two phones with the dead-reckoning information of the two phones. This ended up being more complicated than I at first anticipated, and I never returned to resolve the issue with the changing access points.

3.3 Solving VPCP Using Vision

The final method I attempted to use to solve VPCP was machine vision. Specifically, my goal was to enable two different smartphones to look at the same scene from different angles and infer their relative positions based on the position of objects in the scene. Although there have been a few localization schemes based upon machine vision, I believe my design was markedly different from most of these.

Most of the current vision localization techniques (described in Chapter 2.4) rely on creating a database of images over a large area. A user can then compare an image they take to this database and figure out where they are. However, for my uses only the relative position two users is important. This led me to design an application where no central database is necessary and users can communicate directly from phone to phone.

I implemented this algorithm again as an android application. This time however I relied heavily on the opencv library [22] for android as it had a number of convenient functions. Despite the fact that it was difficult to set up my android application with opencv, I found that opencv was critical in order to get my system to work.

I started off by creating a system that would evaluate whether two images belonged to the same scene, that is whether the same objects were present in both images. To do this I used the SIFT algorithm, which is described in Chapter 2.4. SIFT identified salient points in the images, that is points that could consistently be recognized from different angles and levels of zoom.

After identifying the salient points in both images, the next step was to find a correspondence between the two sets. There are two common algorithms to find this correspondence: RANSAC (random sample consensus) and FLANN (fast library for approximate nearest neighbors). I decided to use FLANN for this task because it was generally considered to be faster and because it was already implemented in opencv.

After successfully creating this application I decided to test its accuracy on both pictures of the same room, and pictures of different rooms. An example of two pictures of the same room from different angles is given in Figure 3-3. Chapter 4.3 features these results and the analysis stemming from them, but here I encountered a different problem. The process to run both SIFT and FLANN took about 5 seconds per iteration. Not only would this eat up a large amount of computational power for any application based off of this, but it would mean the output would be at least 5 seconds off. I found this to be an unacceptable amount of lag, and so decided to abandon this method of solving VPCP.

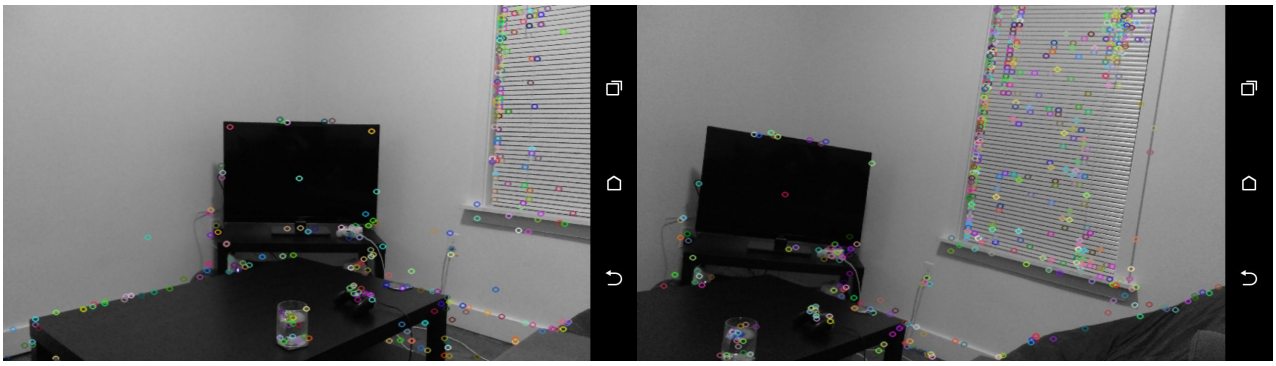

Figure 3-3

A screenshot of the user interface for the application I created using SIFT to determine if two images were of the same scene. Figure 4a (left) shows the scene from one angle, and Figure 4b (right) shows the scene from a slightly different one. The colored dots overlaid on the scene are all the salient points discovered by SIFT for both images.

3.4 Modelling Relative Positioning Systems

In Chapters 3.1 and 3.2 I discussed my efforts to create a relative positioning system between two mobile devices using some range-finding metric. In 3.1 I attempted to find the distance between the two smartphones themselves, while in 3.2 I attempted to find the distance between each smartphone and nearby WiFi access points. These methods were similar in that even had I succeeded in the distance finding metrics of these two methods to work, I would still have to create an algorithm or formula to find the actual displacement between the two smartphones.

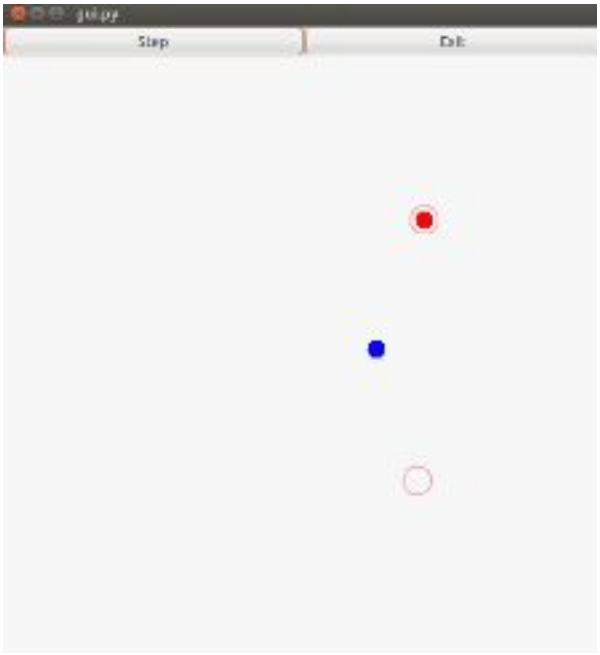

Figure 3-4
This is an example output of the model I created to represent an idealized world where my range-finding metrics were successful. Pressing the 'step' button simulates moving one step forward in time. The filled blue and red circles are two different smartphones, and the pink circles are guesses for the location of the red smartphone.

In both cases I decided that I would try to model an idealized version of the world where the range-finding methods in all the implementations were working correctly. The inherent similarity between the implementations described in 3.1 and 3.2 allowed me to use basically the same model for both scenarios. A sample picture from the model is shown in Figure 3-4.

This model was created in python because I have found that python is the easiest language to rapidly prototype in and it is the one I have the most familiarity with. I additionally used the 'pygtk' [23] library to help build the simple UI shown in Figure 3-4. Overall the ability to rapidly prototype helped me not only to build the model quickly, but also to change it and to add new algorithms quickly.

3.4.1 Modelling Ultrasonic RPS

In the first situation in which my model is representing the algorithm from 3.1, my model starts out by creating two phones (modelled as filled circles) in two different places on the screen. After this, my model waits for the user to press the 'Step' button to advance to the next time step. The step button effectively moves the simulation 1 second further in time by moving each device, telling each device how far it moved in the past second, and telling each device the distance to its partner device. After this, both devices attempt to calculate the relative position of the other smartphone.

Eventually I was able to do this very consistently in my idealized model. However, I ran into two issues on the way. The first issue was figuring out how to do step 7 described in 3.1.1-- that is how to combine all the given data and find out the location of the other device. To solve this I had to refer back to high school geometry, specifically the law of cosines:

$$c^2 = a^2 + b^2 - 2 * a * b * cos(\angle ab)$$

where c, a, and b are all sides of triangle and $\angle ab$ is the angle between the side a and b.

The second issue I ran into was that this method gave two possible results for the location of the other smartphone in the simulation. This is evident in Figure 3-3 where

there are two pink circles, only one of which is actually on top of the other smartphone.

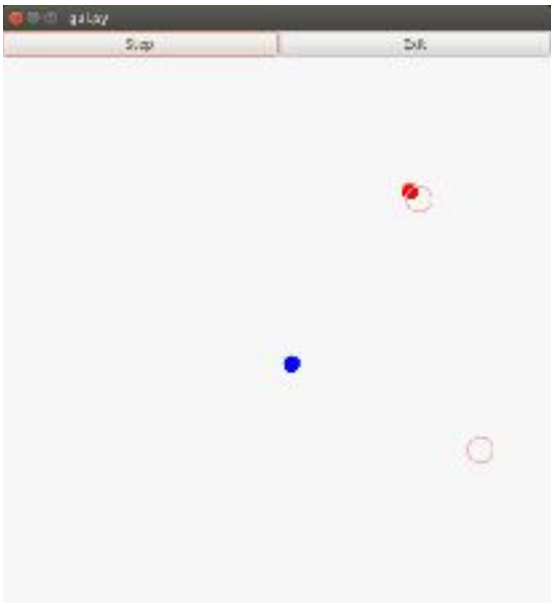

Figure 3-5(a)
In this situation the blue device attempts to predict the location of the red device. This is different from Figure 5 in that a small amount of noise is added to the distance calculation between the two devices. However, in this case noise does not change the accuracy of the prediction by too much.

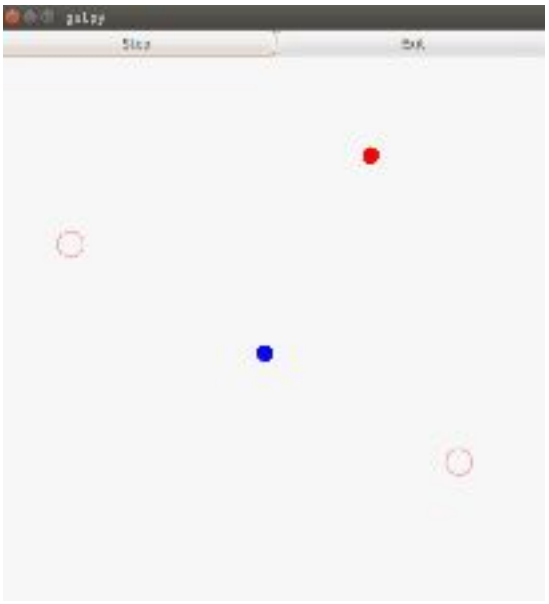

Figure 3-5(b)
As opposed to Figure 6a, about ¼ of the time when a small amount of noise is added, my algorithm's prediction is very far from reality. This most commonly occurs when the devices are moving directly toward or away from each other.

To solve this problem, I was forced to keep track of the previous two time steps (instead of just one) and use this extra information to 'break the symmetry' of the system, making sure it only returned the one correct result.

After solving both of these issues my model worked great--as long as there was no noise added. I decided next to see how my model would react when noise was added to the system. To do this, I changed the distance being fed into both smartphones in my model by a small random amount. Normally this lead to the phones in my model

mis-guessing the location of the other phone by a small amount, as I predicted it would. However, sometimes my model would then be off by a very large amount as pictured in Figure 3-5(b). I decided that I needed to use a different algorithm, and decided to explore possibilities as I modified my simulation to support the WiFi Ranging RPS.

3.4.2 Modelling WiFi RPS

Modelling the WiFi relative positioning system described in 3.2 was much the same as 3.1, with the added twist that separate stationary WiFi access points had to be modelled. By itself this didn't change the code too much, but it would end up leaving a large amount of bloat on the GUI as I tried to render each phone's guess for each new WiFi access point. Eventually I decided to limit the number of WiFi access points to just a few in simulations because of this fact.

One thing that changed a good deal between modelling the ultrasonic RPS and the WiFi RPS was how each phone in my model calculated where the other phone was. Recall that in the WiFi RPS case the first step was for each phone to determine the location of the nearby WiFi access points. I had earlier discovered that the algorithm that I had planned to use here, namely using the law of cosines to integrate the data from two time steps, had an unacceptable high error rate when noise was introduced. Instead, I decided to make the noise a part of my algorithm and use a technique called particle filters.

The basic idea behind this technique was that instead of trying to figure out exactly where a WiFi access point was, each smartphone would keep track of a probability distribution of where it might be. Each smartphone kept track of a large number of points, which were initially randomly distributed around the smartphone. After each time step, each smartphone would update its set of points. If a point was too close or too far to the smartphone after the new time step, ie if the distance between it and the smartphone was different than the measured distance between the smartphone and the access point, then it was removed. If the point was good, ie if it was about as far away from the phone as the access point was, then it was kept and there was a chance of adding a new point close to that point.

Although this algorithm takes a bit longer than the algorithm developed in 3.4.1, it could be much more accurate over the long run. This algorithm suffers from its own suite of difficulties though. One of the issues was that it takes a good deal longer than the algorithm implemented in 3.4.1. This is a result of the program having to do calculations on thousands, and often tens of thousands of particles. The calculation time can be reduced by reducing the number of particles, but only at the price of accuracy. As the total computation time is still less than one second per step, I decided to leave the default number of particles as 1,000 per WiFi access point in my model.

The other issue that this algorithm suffers from is setting parameters. There were a couple of key points in this algorithms which parameters were needed for, such as:

Figure 3-6
Another example output of the model I created to represent an idealized world where my range-finding metrics were successful. The blue (bottom-middle) and red (top-middle) circles are the location of the mobile phones, and the green (right) is the location of a WiFi access point. The black dots are the locations of the particles representing the blue phones guess of where the WiFi access point is.

- How close to the measured distance a particle had to be to be considered an accurate particle
- How much error to add to a 'good' particle in order to make new particles which would also be good

These parameters were problematic to set because there was a constant tradeoff between the ability for the particles to quickly 'migrate' to the right answer, and the ability to stay at the right answer when arbitrary errors were introduced.

Chapter 4

Analysis

In the previous chapter I described the applications and models I created over the course of my research. In this chapter I will go on to provide analysis of how well or poorly these applications worked, and to discuss whether the given performance is good enough for my needs. I will do this by examining individually each of the components described in Chapter 3.

4.1 Analysis of Ultrasonic Ranging

In Chapter 3.1 I described both the algorithm that I planned to implement for ultrasonic ranging, and the difficulties I had while implementing this algorithm. Recall that one aspect I had particular difficulty with eas resolving when one phone was transmitting a high-pitch sound. In 4.1.1 I will described my analysis of this problem, while in 4.1.2 I will discuss the feasibility in general of using ultrasonic methods to establish an RPS.

4.1.1 Analysis of Ultrasonic Resolution Issue

As described in Chapter 3.1.2, I implemented three separate methods on my android application to resolve when an ultrasonic signal was sent. These methods were a FFT, zero-crossings, and a high-pass filter. After none of these methods were able to effectively recognize ultrasonic pings from another device, I decided to do a closer analysis of what was going on. Specifically, I was worried about whether my code for these algorithms had been implemented correctly, so I decided to send my data to my computer where I would use Matlab to analyze it.

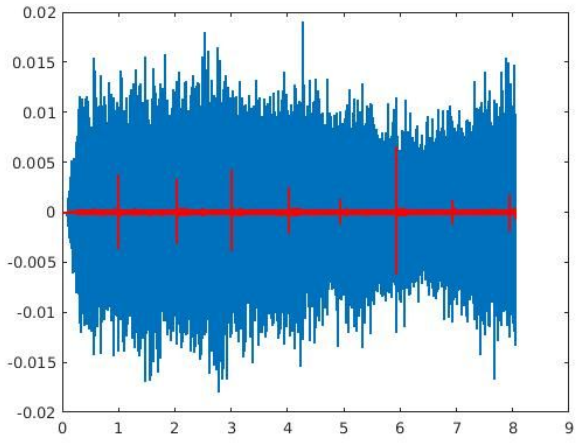

Figure 4-1
This graph shows two separate phenomena. The first is the raw input which was recorded by my computer's microphone (blue). The second is this input after being passed through a high-pass filter (red). There are clear peaks in the red signal, happening at one second intervals. These peaks have varied magnitude, but all are less than that of the blue waveform.

Unfortunately recording the sound data on the android application was a non-trivial task, so I decided to have my computer record the ultrasonic signal instead. After recording, I imported the sound file into Matlab and ran a high pass filter on it. The result of this is shown in Figure 4-1.

From Figure 4-1 we can clearly see that my computer's microphone was able to resolve when an ultrasonic signal was being sent. These correspond to whenever a red bar appears in Figure 4-1, and happens at the same frequency (once per second) as these

ultrasonic signals were being sent. From this, I concluded that at the very least the high-pass filter implementation was feasible on some devices.

One thing that should be noted is that the strength of the filtered signal varied greatly from one ultrasonic burst to the next. As an example, the the magnitude of the filtered signal at t=6 seconds is about three times the magnitude at t=5 seconds. I had a number of hypotheses as to what the issue could be here, detailed below:

- The smartphone emitting the ultrasonic pings may have a faulty speaker, or a speaker that was not designed to work at those frequencies.
- There was a great deal more noise in the room than I originally thought, compromising the ultrasonic signal.
- The geometry of the room of the room led to multi-path propagation errors which compromised the signal strength.
- My computer itself had a microphone not designed to consistently work at these high frequencies.

Given the fact that both my phone and my android application had issues resolving these ultrasonic pings, albeit the android device to a greater extent, I concluded that the issue was probably a combination of poor ultrasonic transmitting and receiving hardware on the android phones in my experiment. This makes a lot of sense when the lower quality of these phones is taken into account.

4.1.2 Feasibility of Ultrasonic Ranging in Establishing an RPS

Ultrasonic localization has been successfully used in a number of different systems to localize a mobile device or agent. However, in just one situation that I was able to find [7] was it used in a relative positioning system as opposed to an absolute positioning system with stationary ultrasonic beacons of known location. There are a couple reasons which I believe ultrasonic ranging is not ideal for a relative positioning system, and especially for the applications I was hoping to create. These reasons are detailed below:

1. **Noise:** A relative positioning system will have a higher utility the more agents, or devices which are being localized, are in it. As the number of devices in this ultrasonic system grows, there will be more and more noise in the high-frequency spectrum. This noise is unavoidable as all devices need to establish their location by transmitting ultrasonic signals.
2. **Hardware Standardization:** One of the main issues I had in implementing this application was the hardware on the smartphones I was using was inconsistent. Most applications don't rely as heavily on some of the nitty gritty specifications of smartphone hardware. This would add a whole new plane of compatibility complexity to mobile application design.

3. **Audibility Issues:** Although I designed my application to use a frequency above the normal threshold for humans, it is still possible there exist some individuals who could hear these noises. More relevantly, many pets can hear noises at this frequency. The noises being constantly sent by this kind of system would be sure to annoy anyone or anything that could hear it.

Although ultrasonic positioning systems can be powerful, my research into their applications in relative positioning systems has led me to the conclusion that these two technologies should not be used in conjunction.

4.2 Analysis of WiFi Ranging

In this section I will provide a more in-depth analysis of my efforts to create a relative positioning system using WiFi signal strength as recorded on smartphones in the same area. In 4.2.1 I will focus on some of the interesting characteristics of the WiFi networks I had access to in my experiments, while in 4.2.2 I will discuss the effectiveness of some of the algorithms I implemented in the model of this system explained in Chapter 3.4.

4.2.1 Notable Characteristics of WiFi Networks in Experiments

There were two interesting characteristics of the WiFi networks which were used for localization by my application. The first of these has to do with how my application

selected which WiFi points to access. Recall from Chapter 3.2 that the first task of the app I created was to find the distance towards nearby WiFi access points. As it was doing this, I noticed an interesting behaviour in which the distance to a given WiFi network would abruptly change.

My hypothesis for why this is the case is that when my application remeasured the distance between a nearby access point and itself, it chose a different access point. This was most likely the case because my application determined classified access points based on their network name. As an example, there were many access points nearby which were broadcasting the “MIT” WiFi network, but my application was treating them as the same. In order to overcome this problem, the networks and access points need to be accessed in a different way.

Another interesting characteristic of the WiFi access points being used in my experiments was that one access point was often broadcasting multiple networks. This was first observed when a sparsity of nearby access points led to multiple different networks having about the same RSSI, or measured distance. Upon closer inspection, the id of the access points for these networks was similar in all but the last digit, or 4 bits. From this, I concluded that the same physical access point was creating all of the virtual access points to the networks that the application observed.

This fact can be leveraged to great use by applications using WiFi ranging. The distances that such a WiFi ranging application calculates are inherently noisy to begin with. However, this noise can be reduced by repeating the experiment or taking multiple measurements from the same access point. When one physical access point transmits multiple WiFi networks, nearby WiFi ranging applications can combine the measurements from these different networks and get a more accurate assessment of the distance the access point is away.

I believe this insight has application beyond the WiFi ranging application I proposed in Chapter 3.2. An example of where this could be used effectively is in WiFi fingerprinting localization systems. If these systems were able to effectively discern when a group of virtual access points were all coming from the same physical access point, these systems could eliminate some of the noise in their database by averaging the RSSI of all of access points which were in the same physical location.

4.2.2 Effectiveness of Algorithms

As discussed in Chapter 3.4, a python model representing how different devices such as smartphones and WiFi access points would interact with each to establish an RPS was created. The most effective algorithm for determining the location of stationary WiFi access points by the mobile phones in this model was particle filters. However, one of the key issues this algorithm suffered from was finding a tradeoff between

localizing the access point quickly and continuing to keep track of it even when there is noise.

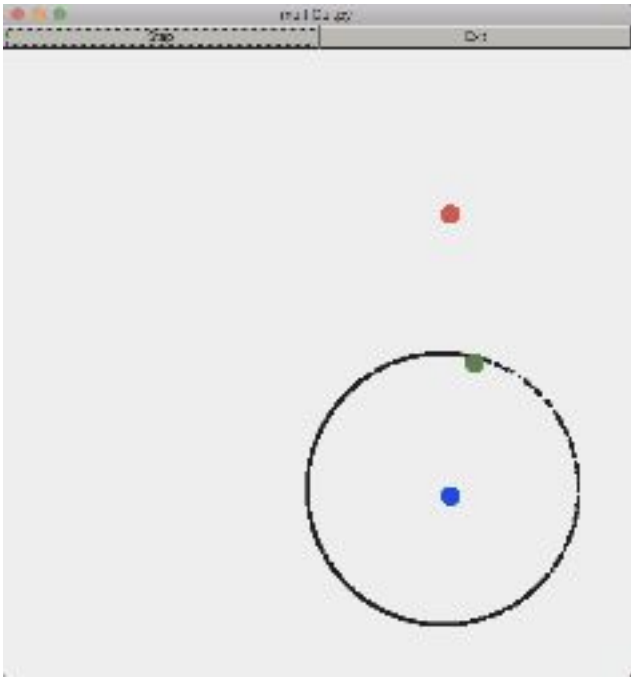

Figure 4-2(a)
A screenshot of the the GUI of the model described in 3.4. This is the situation right after startup where the blue phone (bottom dot) has no clue where the green (middle dot) router is. This is evident by the fact the black dots, ie the particles, are uniformly distributed around the blue phone.

Figure 4-2(b)
A screenshot of the the GUI of the model described in 3.4. This screenshot was taken after the algorithm had run for a couple of iterations. In this case, the blue (bottom dot) phone has determined the location of the green (middle dot) access point, evident by the particles all being very close to the green access point.

The system starts off in an unlocalized state, that is where each phone does not know where any of the access points are. An example of what this looks like is shown in Figure 4-2a. Each phone in the model must then quickly determine where the access points are, allowing them to determine the other phone's location. Let us call this the

localization phase. An example of successfully doing this is shown in 9b. Once the phone has located where an access point is, the particles representing the position of the access point should move a lot less. If they continued to move at the same rate, then noise could cause each phone to readjust its assessment of the location of the access point.

One way to solve this issue is to change the behaviour of the particles based on how far into the simulation you are. During the first few iterations they would be updated in a way so that more focus was put on aligning with the accuracy of the measured distance between smartphone and access point. After localization had been achieved, these particles would then pay more attention to where they were previously, or effectively making smaller jumps if their distance from the smartphone did not align with the measured distance to the access point.

There is, however, a tradeoff to using this method. This tradeoff is that after localizing the smartphones interpretation of where the access point is wrong, then it will continue to be wrong. This problem is not dissimilar from the problem in information theory of choosing a threshold to minimize false negatives and false positives. However, it is made more complex by the fact there are many ways that our system could be wrong while thinking it is right, ie a disproportionate amount and variety of false positives.

Overall, I believe that a particle filter is a good fit for this application. Although it has a few disadvantages, namely that it requires a good deal of computation and the right parameters to update the particles, it was the best algorithm for dealing with noise. In general I think finding an algorithm that accomplishes the task of mobile localization via WiFi ranging is tough, as the information that is trying to be inferred is more complicated than the information being given to the algorithm.

Figure 4-3(a)
The results of my scene comparison app when two pictures of the same scene are compared. Not only are there more matches between the SIFT features, but the ratio of accurate matches to total matches is higher compared to 10b.

Figure 4-3(b)
The results of my scene comparison application when the scenes are different. There are fewer matches and a lower ratio of good matches as compared to 10a.

4.3 Analysis of Machine-Vision for Solving VPCP

My analysis of the machine-vision based application I created was focused on determining if my algorithm could distinguish two different rooms, or scenes. I took a picture of one room, then another of either the same room from a different angle, or a

different place altogether. In the best case scenario, I would get results similar to those in Figure 4-3, where my algorithm reported a marked increase in the similarity for the pictures which were of the same room.

However, this turned out to be the exception rather than the rule. In most cases the metric I used to determine if two images were of the same scene--the 'Ratio' parameter in the image-- was close to 0.4 for both images. Given how inaccurate just the scene recognition was, I decided not to implement a more complicated method of matching up SIFT features between two images.

I believe the main reason that accuracy was consistently low was that the SIFT features were not as salient as I believed them to be. I originally thought that SIFT would pick out the same locations as keypoints in a scene, even if seen from two different angles. However, upon looking more closely at the actual keypoints found by SIFT I found that these locations were often displaced by a small amount between the two images. This displacement could then cause SIFT's characterization of the keypoints to be markedly different.

Chapter 5

Future Work

There are many promising avenues of research in relation to solving the virtual physical correspondence problem. Some of these avenues were explored in the previous two chapters, but there are many which I did not have the time not resources to explore. In the following chapter I will discuss some of these methods. Chapter 5.1 will discuss methods which build off of or are related to the methods I implemented in Chapter 3, while 5.2 will explore different technologies.

5.1 Expansions to Previously Discussed Methods

One avenue for future research is a comparative study of different smartphones in order to evaluate the differences between their speakers and microphones. Specifically, this study would be most relevant if it studied the ultrasonic capabilities of these devices. I hypothesized in Chapter 4.1 that faulty or ineffectual hardware on the smartphones I

was using. A more thorough study on this matter could prove or disprove that hypothesis.

Another promising direction for future work is to refine the algorithm and parameters I use for WiFi ranging in 4.2. The current method uses particle filtering with a set of parameters optimized for the simulated model I created. The effectiveness of particle filters in a real test environment for this problem has yet to be explored.

There are also some potential improvements to the particle filter algorithm in the WiFi ranging model I created. This algorithm in its current form can be thought of as being two distinct steps. The first step is for each phone to determine the relative displacement between itself and the nearby WiFi access points. The second step is to combine this information and determine the location of the other smartphone.

However, once the location of the other smartphone is known, each smartphone can potentially get a better guess on where the nearby access points are. For example, say that phone A was able to accurately localize a WiFi access point, but the phone B incorrectly determined the location of that access point. After discovering the location of phone A and phone A's location relative to the access point in question, phone B could update its model of the location of this access point.

5.2 New Methods

The essence of solving the virtual physical correspondence problem is enabling two mobile devices to localize each other. Although I focused on doing this with relative positioning systems as opposed to absolute or global positioning systems, the latter two can still be used to solve this problem. For example, one potential way to solve this problem is to have the two devices use gps to determine their position, and then communicate it to each other. Although some of the disadvantages of using GPS were put forth in Chapter 1, there are other absolute positioning systems that are more suited to this task.

In particular, WiFi fingerprinting can yield accurate positions for users inside buildings--or other places GPS is less effective. Over the past few years it has been used to augment GPS, but it has often not had the same accuracy. However, there are now multiple research projects and some commercial ventures underway whose aim is to improve the accuracy of this technology through crowdsourcing.

One potential disadvantage of WiFi fingerprinting is that the location is measured in discrete units defined by the database of WiFi fingerprints. That is, if I walked across a room, a WiFi fingerprinting localization method would put me in a number of distinct locations, one after another. Since my phone would get details about its location through this service, from its point of view its movement would appear as a sequence of jumps, as it jumps from one location in the fingerprint database to the next. More

importantly, any phone communicating with my phone would see my movement in the same way--as a sequence of jumps.

One way this issue could be solved is by using the inertial measurement units like accelerometer and compass embedded in the smartphone. If the smartphone could convey its approximate location obtained through WiFi fingerprinting along with its movement, an outside observer could get a more accurate fix on its location. More importantly, it wouldn't appear to jump from one place to another, but rather to move smoothly. Kalman filters, similar to the particle filters I used in my implementation, would be an ideal candidate to combine the fingerprint data with the inertial data to create a smooth and accurate interpretation of the situation.

Overall I think this idea, combining WiFi fingerprint localization with inertial information from a smartphone, is the best candidate for solving the VPCP. Not only does it require little if any additional infrastructure, but it also has the potential to be very accurate if precise WiFi fingerprint databases are kept.

Chapter 6

Conclusion

In this thesis I described my research into relative positioning systems with focus on their applications to mobile localization. I primarily focused on three unique ways of establishing relative positioning systems. The first method used ultrasonic time of flight to determine the distance between two devices, then dead reckoning to establish their relative position. In the second method, mobile devices used WiFi RSSI from nearby routers to establish their relative position. The final method featured machine vision, and relied on two mobile devices observing the same location, and then inferring their relative position from differences in their observation.

None of these methods were able to achieve the results I desired. However, my exploration of these methods provides future researchers with more insight into what methods might or might not work to solve mobile localization problems. My insights into this field also helped provide guidance into what the best next steps are to solve these problems.

The overall goal of this research was to solve what I termed the virtual physical correspondence problem. This problem boils down to a mobile device learning about the location of some other mobile device that it interacts with. If this can be done by both mobile devices, then these devices can provide their users with a more informative and more interactive view of the electronic world around them.

References

[1] Valentin Heun, James Hobin, Pattie Maes. "Reality Editor: Programming Smarter Objects" UBICOMP 2013.

[2] S. Flores, J. Geiß, M. Vossiek, "An ultrasonic sensor network for high-quality range-bearing-based indoor positioning", *2016 IEEE/ION Position Location and Navigation Symposium (PLANS)*, pp. 572-576, 2016.

[3] Sewan Kirn; Younggie Kim; “Robot Localization Using Ultrasonic Sensors”, Proceedings of IEEEIRSl, September, 2004

[4] Do-Eun Kim; Kyung-Hun Hwang; Dong-Hun Lee; Tae-Young Kuc, “A Simple Ultrasonic GPS System for Indoor Mobile Robot System using Kalman Filtering“, SICE-ICASE 2006 International Joint Conference, October, 2006

[5] Jim Pugh; Alcherio Martinoli, “Relative Localization and Communication Module for Small-Scale Multi-Robot Systems”, IEEE International Conference on Robotics and Automation, May 2006

[6] L. Marton, C. Nagy, Z. Biro-Ambrus, and K. Gyorgy, “Calibration and measurement processing for ultrasonic indoor mobile robot localization systems,” in Proc. of IEEE International Conference on Industrial Technology, 2015, pp. 131–136.

[7] S. Pang, and R. Trujillo, “Indoor localization using ultrasonic time difference of arrival[C],” Proc. IEEE Southeastcon. pp. 1-6, 2013.

[8] Bong-Su Cho; Woo-sung Moon; Woo-Jin Seo; Kwang-Ryul Baek, “A Dead Reckoning Localization System for Using Inertial Sensors and Wheel Revolution Encoding”, Journal of Mechanical Science and Technology, November, 2011

[9] Haojian Jin; Christian Holtz; Kasper Hornbaek, “Tracko: Ad-hoc Mobile 3D Tracking Using Bluetooth Low Energy and Inaudible Signals for Cross-Device Interaction”, Proceedings of the 28th annual ACM symposium on User interface software and technology, November, 2015

[10] Lim, Hun-Jung; Chung, Tai-Myoung. “Performace evaluation of relative

positioning based on low-cost GPS and VANET”, Proc. ICITST, December 2011.
[11] J.-W. Qiu, C. C. Lo, C.-K. Lin, and Y.-C. Tseng, “A d2d relative positioning system on smart devices,” in IEEE Wireless Communications and Networking Conf.(WCNC), 2014.
[12] J. O. Oh, M. S. Lee and S. K. Lee, “An Acoustic-Based Relative Positioning System for Multiple Mobile Devices,” Proc. of the 4th International Conference on Computer Sciences and Convergence Information Technology, 2009, pp. 1565-1570.
[13] Rafael G. Aranha and Rui M. Rocha, “Real-Time Relative Positioning with WSN”, SENSORCOMM08, pp.276-281, Sep 2008.
[14] Shafiee, Mahsa; Klukas, Richard, “Joint Access Point and User Localization Using Unlabeled WiFi RSS Data”,Position, Location and Navigation Symposium (PLANS), 2016 IEEE/ION, April 2016.
[15] Hernandez, N., Ocana, M., Alonso, J., and Kim, E., "WiFi-based Indoor Localization and Tracking of a Moving Device", Ubiquitous Positioning Indoor Navigation and Localization Based Service (UPINLBS), pp. 281-289, 2014.
[16] J. Niu, B. Wang, L. Cheng, and J. J. Rodrigues, “Wicloc: an indoor localization system based on wifi fingerprints and crowdsourcing,” in 2015 IEEE International Conference on Communications (ICC). IEEE, 2015, pp. 3008–3013.
[17] V. Radu and M. K. Marina, “Himloc: Indoor smartphone localization via activity aware pedestrian dead reckoning with selective crowdsourced wifi fingerprinting,” in 2013 International Conference on Indoor Positioning and Indoor Navigation (IPIN),. 2013
[18] Guan, Kai; Ma, Lin; Tan, Xuezhi et. al., “Vision-Based Indoor Localization Approach Based on SURF and Landmark”,Wireless Communications and Mobile Computing Conference (IWCMC), 2016 International.
[19] Lowe, David G., “Distinctive Image Features from Scale-Invariant Keypoints”, International Journal of Computer Vision, 2004.
[20] J. Z. Liang, N. Corso, E. Turner, and A. Zakhor, “Image based localization in indoor environments,” in Computing for Geospatial Research and Application (COM. Geo), 2013 Fourth International Conference on. IEEE, 2013, pp. 70–75.
[21] Nishkam Ravi, Pravin Shankar, Andrew Frankel, Ahmed Elgammal, and Liviu Iftode, “Indoor Localization using Camera Phones,” in Mobile Computing Systems and Applications, 2006.
[22] Bradski, G. “opencv_library”, Dr. Dobb's Journal of Software Tools, 2000.
[23] Henstridge, James; Dahlin, Johan, “Pygtk”, ‘https://github.com/GNOME/pygtk’.
[24] P. Bahl, V. N. Padmanabhan, “RADAR: An In-Building RF-based User Location

and Tracking System” 19th Joint Conference of the IEEE Computer and Communications Societies, volume 2, pages 775-784, 2000