Mobile Localization for Augmented Reality

Brian Copeland
Advisor: Prof. Chris Terman

1. Introduction

We live in a world of increasingly ubiquitous computing. All around us there are devices we can interact with digitally, ranging from mobile phones to wearables to everyday devices such as key chains. However, interacting with these devices often requires that we first identify their digital location: some non-physical way to interact with them.

There are many instances where a device's physical location is known, while its digital location is not. For example, if I wanted to send a message to a phone next to me, I would first have to learn the phone number of that phone. As another example, if I wanted to connect my phone to a smart appliance (such as a TV) I would first have to find that device from a list of bluetooth enabled devices that are near me.

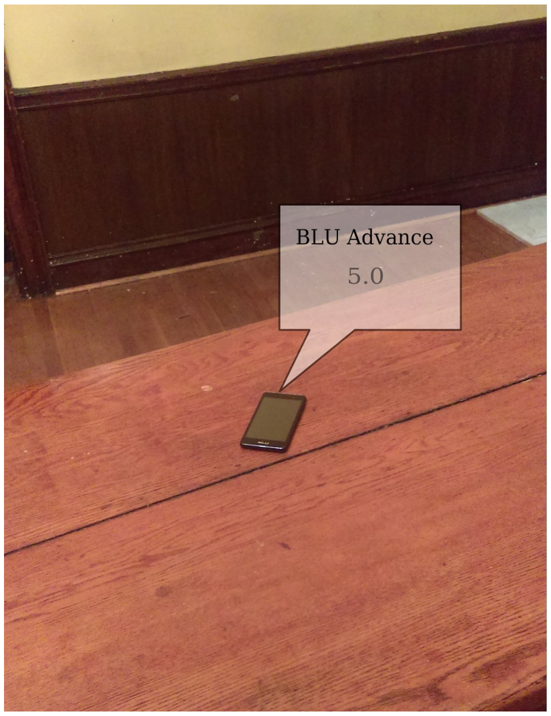
1.1 Goals

The goal of my research was to enable communication between two devices in a way such that a user only needs to know the physical location of the device they want to interact with. More specifically, a user would be exposed to an interface showing the world behind the device, along with tags of devices in view. An example of this interface is shown in Figure 1.

I see a number of possible applications of this technology. The first class of these are related to social media. Let us say two people meet and want to connect on a networking site like LinkedIn or Facebook. Normally they would have to search for the other person, even though they are in the same room.

If instead each person's mobile device knew where the other’s mobile device was located, the two people could use the augmented reality interface shown in Figure 1 to interact with the other person’s digital identity. Connecting would be as simple as tapping their screen.

Another class of possible applications stemming from this research is the class of problems where someone wishes to interact with another compute-enabled devices, such as a smart TV or other smart appliances. Some research has been in this area by



the Reality Editor project in the MIT Media Lab [1]. This project tags smart devices with a code that a phone's camera can read, allowing it to then know the digital address of the smart device. My idea goes a step further in that you no longer need to physically tag the smart device in order to connect to it.

1.2 Original Approach

My original approach to this problem was to create the entire system as an Android application on mobile phones. I had two phones that would use ultrasonic pings to determine their distance apart. They would also use accelerometers and magnetometers to estimate the distance and direction they had moved. These metrics can be combined to determine the relative location of the other phone.

I broke this process down into the following steps:

1. Create and send a short ultrasonic (20 kHz) ping every second.
2. Send a electromagnetic BLE signal at the same time as the ultrasonic signal so difference in time of arrival between the sound signal and electromagnetic signal can be measured.
3. Receive the ultrasonic and electromagnetic and interpret the distance between the two devices.
4. Use each phone's sensors to determine the dead reckoning location traveled in the past second.
5. Combine the dead-reckoning information with the distance information to determine each device's relative location.

After completing steps 1 and 2, I encountered technical difficulties in receiving ultrasonic pings in step 3. I was unable to reliably filter out the ultrasonic signal from the rest of the noises picked up by each phone's microphone. I discuss the details of this process and the methods I tried in Section 3.1. However, I continued to work on this project through a slightly alternative approach.

1.3 Secondary Approach

After encountering difficulties filtering the auditory data received by each phone, I turned my attention to modelling step 5 of my original approach. That is, I created a virtual environment on my computer in which I programmatically added two virtual devices and had them interact. Each devices uses relative distance data and movement data from the virtual environment to determine the location of the other device. I was primarily interested in the accuracy of these estimations in the presence of environmental noise, and the results are shown in Section 3.2.

2. Previous Work

There has been a great deal of research already into the field of localization of mobile agents. Most of this research has focused around localizing autonomous robots. I describe some of this research in section 2.1, while focussing on localization using ultrasonic signals. In section 2.2 I move on to discuss ad-hoc localization where only relative position is important. It is this latter category of study that my research has been in.

2.1 Localization Using Ultrasonic Signals

Many ultrasonic sensor systems such as [2] and [3], can accurately localize a mobile entity with the aid of stationary nodes in know location. When using ultrasonic signals to provide the distance metrics, there seem to be two primary approaches. The first is to synchronize the communicating devices by sending an RF (usually bluetooth) signal along with the ultrasonic signal. The distance between the two devices is then calculated using the speed of sound and the time difference in arrival between the two signals. This is the approach used in [2] and the approach that I use.

The second approach uses round trip travel times (RTT). When one phone receives an ultrasonic signal from the other phone, it immediately responds with an ultrasonic signal. Once the first phone receives the signal, it can calculate the RTT as the time difference between the time it sent and received the signal, and then use that to estimate the distance between the two phones.

2.2 Ad-Hoc Localization

Obtaining the relative location of different entities without nodes of known location has considerably less research, but has been accomplished with varying accuracy [4]. The approach used in [4] relies very heavily on dead reckoning, or calculating a devices new position by integrating data from an acceleration sensor [5]. This is unfortunately not accurate enough for most commercial sensor hardware.

There has also been research aimed at combining different types of location data to get the relative position to another device [5]. Research conducted at Yahoo Labs went further to combine dead reckoning, ultrasonic range-finding, and RSSI of Bluetooth signals. This study was able to obtain localization accuracy within a few percentile [5], albeit at a relatively close range (< 2m). I use very similar techniques, but expand the range to a couple of meters.

3. Design and Results

As I mentioned in the Introduction, my original approach was to make the full localization/augmented reality system as a full-fledged android app. In section 3.1 I explain in more detail how the app was to be developed and implemented. In 3.2 I discuss which features of the app I was able to implement, and in section 3.3 I discuss which features I was unable to implement. In section 3.4 I attempt to explain why I had difficulty implementing some of the features.

In sections 3.5 I explain my setup to modelling my ad-hoc localization scheme in software as opposed to on physical hardware. Finally, in section 3.6 I give the results of my modelling efforts.

3.1 Original Approach

My app is designed around updating its model of the world every second. It is required to have discreet updates because the ultrasonic signal can not be sent out continuously and still be used to measure distance. My goal was for the app to do the following operations every second:

1. Send out an ultrasonic ping lasting 0.1 seconds at 20 kHz.
2. Send out a BLE packet concurrently with the ultrasonic ping. This packet would also contain information such how much the device moved.
3. Receive the ultrasonic ping and BLE packet and determine the distance to the other device by looking at the time of arrival of each signal.
4. Integrate the last second of accelerometer and magnetoscope information to determine by how much and in which direction the phone moved.
5. Combine this dead-reckoning information with the distance to the other device with that recorded at other time steps and use triangulation to determine where the other device is.

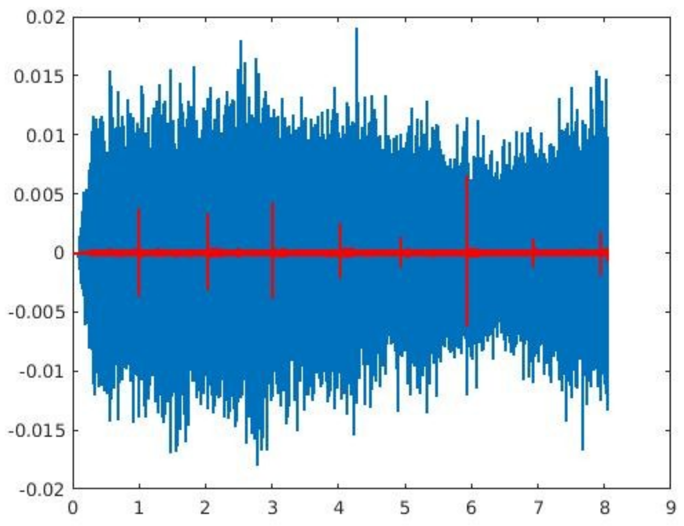
There were other possible choices for the update time other than one second, however one was not only conceptually simple, but also fast enough to see live updates of the system. If this update time is too fast then there are complexity issues and problems separating pings from distinct time steps. If the update time is too slow the system is very sluggish.

3.2 The Good

In this section I discuss the features of my app that I was able to successfully implement. The first of these was the ability to send out an ultrasonic signal every second. To do this I first created the type of signal that I wanted, which was a simple 20 kHz sine wave signal of duration 0.1 seconds. I created this resource using the help of this [6] website.

I was then successfully able to play this sound in my application using pre-existing android tools. Since the sound I was playing was above the threshold of human hearing I decided to check to make sure the sound was actually playing. To do this, I recorded one devices pre-existing recording software to record the other device. I then transferred the raw input to my computer, where I analyzed it using MATLAB. Figure 2 shows the resulting analysis

The blue line is the raw input and the red line is the filtered input. Clearly the signal is being sent and the background noise is low enough to detect the ultrasonic signal, provided some filtering is applied.



After that I started working on the bluetooth connection between the two devices. I used BLE protocols to pair the two devices, and then later send signals to the other device. Like the ultrasonic playback, I used pre-existing android API's for this process. The difficult part was actually choosing which of the many android bluetooth api's was appropriate to use.

3.3 The Bad

In this section I describe what I wanted to get, but was unable to do in the scope of this project. The major roadblock for me was receiving the ultrasonic ping.

The first step in every method I tried was to get the raw audio data. This is collected by an android phones microphone as an array of positive and negative samples of the air pressure: basically a .wav file. If theses samples change very fast then there is a high-frequency wave being recorded; whereas if they change very slowly there is a low frequency wave. I was only interested in looking at very high frequency waves, especially those around 20 kHz.

The first method I tried was called zero-crossings. This method entailed counting the total number of times the samples switched from a positive to negative number, or vice versa. If a strong, high-frequency sound is being recorded than there should be a lot of these zero-crossings. This method worked well when the two phones were closer than one meter apart, but I could not generalize it to work for larger distances.

The second method I tried was to get the Fourier transform of the sound signal around the frequency I was looking at (20 kHz). I would then be able to measure the value of that fourier transform and determine exactly when the ultrasonic signal was received. I tested this method out at a much lower frequency (1 kHz) with great success, however when I tried it at my target frequency I was unable to resolve the signal.

The final method I tried to determine when the phone received an ultrasonic signal was using a high-pass filter. This was the same method that I used to filter the sonic frequencies out when doing my MATLAB analysis of figure 2. Having already made the filter in MATLAB, I was fairly confident that I could adopt the same code to make the same type of filter on my android device. However, I was stymied here too and was unable to get the resolution I got from my MATLAB analysis.

3.4 Why Ultrasonic Resolution Was Unsuccessful

I will now pause for a moment and discuss some of my theories for why I was never able to successfully identify ultrasonic signals on my android app. This was surprising for me, especially in light of the fact that I was successfully able to identify these signals on my computer in MATLAB, as shown in Figure 2.

The most likely explanation for disparity between the analysis on my phone and computer was that the sound was recorded differently. When I recorded the sound data that I analyzed on my computer, I used a built-in application on my phone. When I was analyzing the sound data inside my android application, I used a lower-level android api to do the recording. It is possible these two software systems differed in some non-trivial way, complicating the analysis.

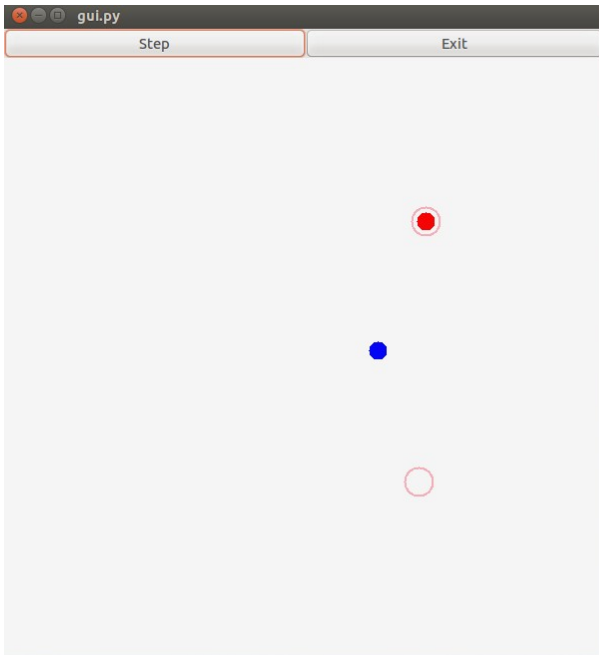
Another possibility is that the ultrasonic signals being sent are themselves much less reliable than I expected. As you can see in Figure 2, not all the peaks in the high frequency data are the same size. In fact, one of them is low enough to be easily confused with the background noise. It is possible that with a noisier background and devices which are farther away, the strength of the ultrasonic ping is low enough to consistently be confused by the background noise.

3.5 Ad-Hoc Mobile Localization Model

After finding myself stymied by my original approach of achieving ad-hoc localization using real devices, I switched approaches to modelling the localization process that two real devices would need to undergo to determine each other's position. I created a virtual world in which two virtual devices can interact with each other in a similar way that they would be able to in the real world. Specifically, each virtual device knows only the distance to the other device, the distance it moved in the last time step (dead reckoning), and the distance the other device moved in the last time step.

My goal was to create an algorithm that can be used by one device to find the other devices relative location based only on these three pieces of information. This is basically a triangulation problem, but instead of using three or four different beacons of known location, I use the previous locations and distances between the two devices. Similar to triangulation, trigonometric properties of the current and previous locations and change of location are used in my algorithm to calculate the location of the corresponding device.

In my model, I update the location of each device by randomly moving it a small distance at each time step. The amount moved by each device is on average about 1/15th of the distance between the two devices. This random movement is necessary to get data about previous locations and distances between the two devices. In the real world too some kind of movement between the two devices is necessary for each device to determine the others position.



Shown in Figure 3 is the visualization of the model I created. The two solid circles are the locations of the two virtual devices. The two slightly larger outlined circles are the best guesses for the location of the other virtual device.

3.6 Results of Modelling

As can be seen in Figure 3, the virtual devices participating in the ad-hoc mobile localization have two guesses for where the other device can be. This is because my original algorithm uses data from only one time step. When only one time step is used, there is a symmetry between the two guesses that can only be broken by obtaining more data.

However, Figure 3 does clearly show that my algorithm accurately calculates the location of the other virtual device, even if it can only narrow it down to two locations. My algorithm is able to achieve this accurate prediction on every time step, assuming there is no noise.

I also applied my model in the face of a noisy environment. Specifically, I added a slight amount of noise to the distance calculated between the two devices. Most of the time my algorithm was still fairly accurate and placed one of its guesses in approximately the right place (Figure 4), but about a quarter of the time both guesses were a considerable distance away from the actual location (Figure 5).

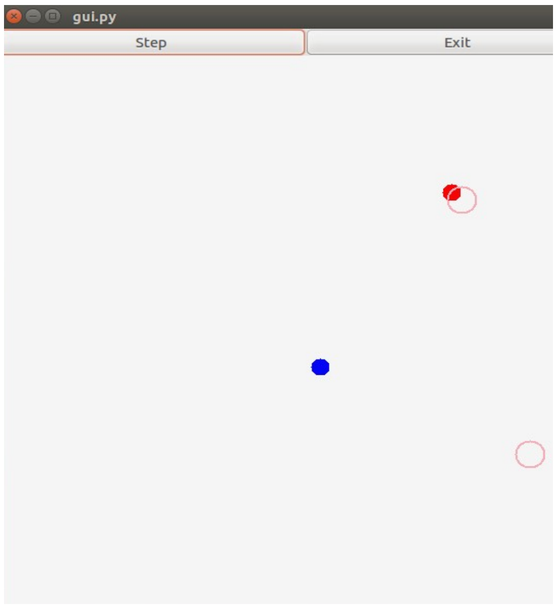


Figure 4
In this situation the bottom-left virtual device again attempts to predict the location of the other device. This is different from Figure 3 in that a small amount of noise is added to the distance calculation between the two devices. However, this does not change the accuracy of the prediction by too much.

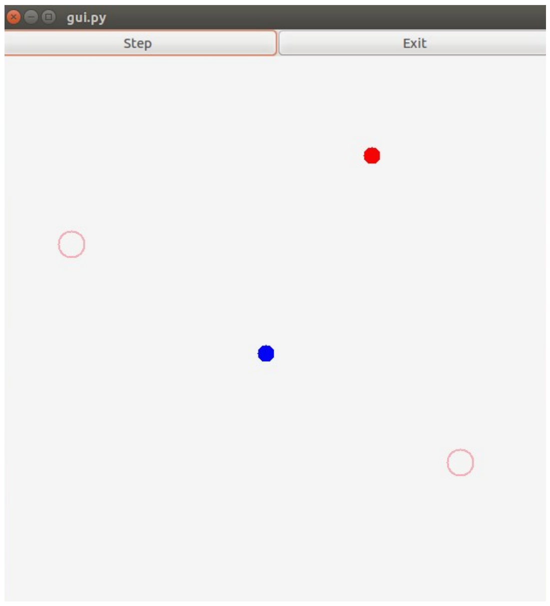


Figure 5
As opposed to in Figure 4, about ¼ of the time when a small amount of noise is added, my algorithm's prediction is very far from reality. This most commonly occurs when the devices are moving toward or away from each other.

4. Future Work

This project is not yet complete, so the simplest future steps are to implement the unfinished components and remove some of the bugs. In section 4.1 I discuss the known bugs and unimplemented features in of my original application. In Section 4.2 I discuss bugs with my algorithm for modelling ad-hoc localization.

4.1 Unimplemented Features of Android Application

The biggest problem with my system as is is ultrasonic detection--that is detecting when an ultrasonic ping was received. This problem and my attempts at solutions are explained in depth in section 3.3 and 3.4. Unfortunately the architecture of my app means that it will not be successful without this functionality. As such, there are two additional features that were not implemented for lack of ultrasonic detection. The first of these is the ability to estimate distance travelled from the phone's accelerometer and magnetoscope. The second of these is the integration of the distance to the other device from the ultrasonic detection with the dead reckoning calculated from the phone's sensors.

The algorithm necessary for this integration is outlined in section 3.5. With all of these features fully implemented, you would have a system in which each phone knows the relative position of the other phone. For example, one phone might say that the other phone is 2.3 meters away at an angle of 67 degrees from North. This data can be used for a number of real-world applications, discussed in 4.2.

4.2 Bugs with Ad-Hoc Localization Algorithm

If you recall from Section 3.6, there were two issues with my Ad-hoc localization algorithm. The first of these was that it was unable to determine which of two potential locations the other device was located. The second was that it sometimes gave very inaccurate answers in a noisy environment.

I anticipate that both of these issues will be solved with the incorporation of multiple different time steps into my algorithm. My algorithm currently only uses data from the current time step, leading to the symmetry issues with the two different locations. This can be easily solved by using just one more time step’s worth of data.

The accuracy issue can also be solved if multiple time steps’ of data are used. If a mostly accurate position from one time step is known, the next time steps prediction differs by a very large distance, this newer prediction is probably incorrect. As a way to determine the actual location, methods such as Kallman filtering can be used. Applying these methods to the ad-hoc localization system I use here would be a great future step to improve accuracy.

5. Contributions

1. Created app that can send ultrasonic pings that can be used for range-finding.
2. Developed an algorithm that will be able to perform ad-hoc localization on mobile devices.
3. Created a virtual testing environment to test ad-hoc localization between two devices [7].

References

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

[2] 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

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

[4] 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

[5] 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

[6] AudioCheck.com, “Sine Tone Generator”, http://www.audiocheck.net/audiofrequencysignalgenerator_sinetone.php.

[7] Github.com, “bcope/UAP_Model”, https://github.mit.edu/bcope/UAP_Model.