MarcoPolo: Determining Relative Location Using Accelerometer and GPS data in a Smart-Phone based Wireless Sensor Network

Abhishek Sethi

CMSC684

Contents

Introduction

Curiosity is a part of human nature. As humans, we strive to develop new techniques to study the unfamiliar world around us, and attempt to visualize it. We understand the world based on our 5 senses, but visual images and quantifications help humans understand the world best, hence a picture speaks a thousand words. In order to visualize the hidden phenomena surrounding us, we develop various sensors, such as light sensors, temperature sensors, barometers, etc. These sensors provide a number and visual meaning to the otherwise hidden events occurring on a daily basis. Using these sensors in our homes is relatively convenient, but not interesting. In order to understand the hidden parts of the world, such as the deep sea, heat of a volcano, etc, we must venture there. Humans cannot survive in such an environment, but we can still deploy these sensors as a network, and have them communicate with each other to help humans quantify the world around us. This is where wireless sensor networks (WSNs) come in.

A wireless sensor networks are defined as a collection of sensors (also referred to as nodes) connected and dynamically configured in ad hoc topologies to provide support for detection and monitoring applications [7]. These nodes can have any number of sensors, only limited by the imagination, and amount of energy that is provided for the node. Multiple sensors on a single node is feasible, but not always efficient. For example, 5 sensors on a node may provide more information, but at the cost of draining the already miniscule battery 5 times faster. If battery were not an issue, then any amount of sensors can be placed on the node. However, in almost everyone's pocket sits a powerful sensor node that efficiently uses multiple sensors, and is energy efficient because humans constantly keep the node charged. This node is the everyday smartphone.

We take smartphones for granted, using them for texting, calling, application use, email, etc. Smartphones have hundreds of features, and the hardware itself is advancing faster than expected. For example, the newest Samsung Galaxy S6 contains 2 CPU's, and each one is a quad core running at 1.5 GHz and 2.1 GHz, respectively [6]. This hardware is powerful in comparison to quite a few laptops, and this device is less than ¼ the size of a normal laptop. Along with the powerful CPU come sensors in each phone. The sensors on a phone vary between manufacturers, but the most common are accelerometer, gyroscope, proximity sensor, and an ambient light sensor. Quite a few of these sensors are new, but the accelerometer was employed well before any of the other sensors were implemented in modern smartphones.

An accelerometer is a small sensor found in almost all smartphones today. This device provides information on static acceleration of the sensor due to gravity [1]. The signals come based on gravity affects from the x, y, and z axes. There are many applications for this sensor, such as detecting what whether your phone is landscape or portrait mode, how fast are objects moving so airbags are deployed at the appropriate time, etc. The one issue with working with an

accelerometer is that the signal is noisy, and the device itself is sensitive to movement. To overcome this, various transforms of the data can be used. In this case, the Discrete Fourier Transform (DFT) was used, which is discussed later. Once this data is processed, it is easy to use and understand. Knowing this information, the project presented uses accelerometer signals to determine 3D movement of the phone (the user of the phone), and then creates a WSN using cellphones as the sensor nodes to communicate with other sensor nodes in order to determine relative locations. This is simply a proof of concept, and can be further elaborated and studied.

Application

Introduction

The application is an Android based phone application. Due to the ease of working with Android and Java, this was the platform of choice. Also, the availability of cellphones impacted the choice of platform. The application is simple. It utilizes Java classes, as well as various XML files. These files are provided in the additional material along with the report. The nodes communicate using WiFi Direct, and one node serves as the base station for the remaining nodes, collecting information and using that to determine relative locations. Simple computation is performed at each node to process the accelerometer data. Full implementation notes, and source code are provided along with this report.

Communication

In order to network the nodes together, Wifi direct was used as the means for connecting the nodes. Originally, the idea was to use Bluetooth or NFC communication, but both options had their disadvantages over Wifi direct. Using Bluetooth, this would have limited my transfer speed as Bluetooth is slower than Wifi. Bluetooth also requires passcode entry, which would have taken time and phone energy to enter, and finally connect with another device. The energy costs, and time to do the connection made Bluetooth inferior to Wifi direct. The limitation for NFC was the extra technology that was necessary to communicate nodes, and also the learning curve associated with using NFC to have nodes communicate with each other. There was a learning curve for Wifi direct, but Wifi direct did not require me to buy additional hardware for communication.

The overall structure for communication is simple. The application itself implements a one-way communication mechanism. It is a basic server-client setup. One node will establish itself as the server, the other will establish itself as the client. This may seem to lack luster, but the beauty is while using Wifi direct. Wifi direct allows for “groups” to form, and peers to be found.

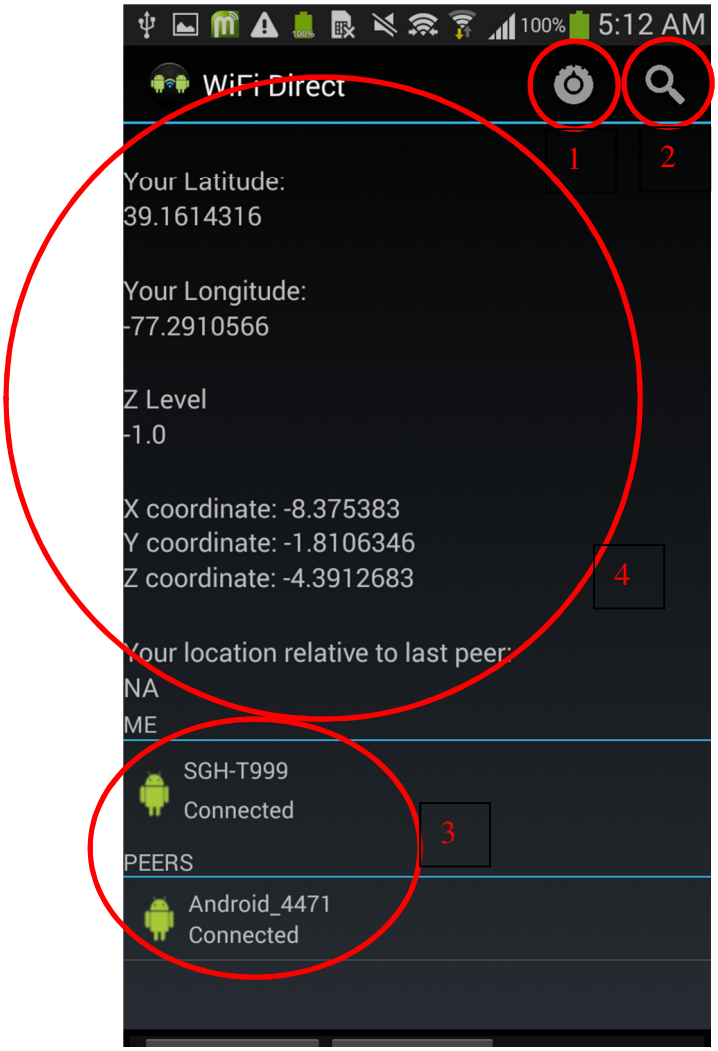

Figure 1: Screenshot of application when booted up

Figure 1 shows the simple layout of the application when started up. The red circles are important features of the application. The circle labeled 1 simply takes you to the Wifi settings so that the Wifi can be turned on or off as need be. Circle 2 is peer discovery. This button allows a user to find any other Wifi direct devices connected on the same wireless network. This allows the user to develop groups with the other peers, and establish connections as need be. Along with discovering peers, this button also updates the user's location to the latest location update. It does not have the ability to turn on the location; it simply uses the latest location update to set the latitude and longitude of the phone. This refers back to what was the latest location update that the user received. Circle 3 shows the peer discovery. Initially, there will be no peers available, but as users use the peer discovery button (circle 2), their device names will show up, and whether or not they are connected to this current device.

One concept worth noting is that the user picks the peers for the application. This also allows the user to select devices that he/she trust, rather than possible malicious devices that the phone

automatically selects. This process can be automated, but for the purposes of this project, it is not necessary. Finally, the data and information is shown in circle 4. This area shows your current location in longitude and latitude, as well as another location termed the "z level". The "z level" is discussed further in the computation section. The application also displays real time accelerometer data that has been corrected for the acceleration of gravity ($g = 9.81\ m/s^2$). Finally, this area also provides information on the location relative to the last peer that was selected. This is the purpose of the application. Through networking, it is able to exchange data, and obtain relative information of where it is to a previous peer. If that peer is still live, the user can select to receive data once again, but this is again dependent on if the peer is live. The purpose of this information is elaborated in the computation section. Overall, once a peer is selected, the client side can send data over to the server side, and the server will receive information updates and recompute values to determine the relative position of the client to the server.

The data that should be passed from the client to server is a text file found in the external storage of the phone. The path is the **(external storage)/com.example.android.wifidirect/data.txt**. This text file contains information on location updates and accelerometer updates post processing. A sample data file is provided along with the code. This helps the server to determine what the relative location is, and can validate the calculations done by the client on whether or not it moved appropriately. This is great for 1 sided communication, but in order to obtain 2 sided communication, the intent of a node to become the group owner is required. This means that once the intent of one node is higher than the other, it will receive the updates and be able to update its own information instead of just providing information. Overall, wifi direct seems to be a good choice for a solution that this project requires, rather than Bluetooth or NFC technology.

Computation

2 sensors are used in this project. The first is the GPS location, which is set to use the most accurate location possible, so GPS is require. The second is of course the accelerometer, used to measure travel distance, and do many more tasks. The GPS data was used in its raw form. The latitude and longitude were provided by the Google Play Services API [4]. These values were used to calculate relative distances by a simple subtraction. For nodes 1 and 2, the relative location would simply be ($latitude_2 - latitude_1$) and the same for longitude. If the number is positive, then that implies we are more towards the left of the map (the negative x/y axes, respectively) since our location is less than the peer's location. The opposite is true if the number turns out negative. The interesting number in figure 1 is the z location. This location is calculated using the accelerometer data in order to differentiate whether the phone/user is moving up/down a staircase. Moving up a staircase will increase the z location, whereas moving down decreases it. Unfortunately, this will not provide an exact position as to where the user is in a building, but it is a relative position to where the user started when the application as opened. The main use of this data could be to determine how many stair cases the user crossed in a day, if a sum instead of negative/positive is used. Also, this information can be used to construct a 3D map as opposed

to Google map's 2D map only. In the relative location portion, these elements are also shown to provide relative information to the last peer that was contacted.

Along with the location information, real time data from the accelerometer is provided for the user to see how the accelerometer works with the application.

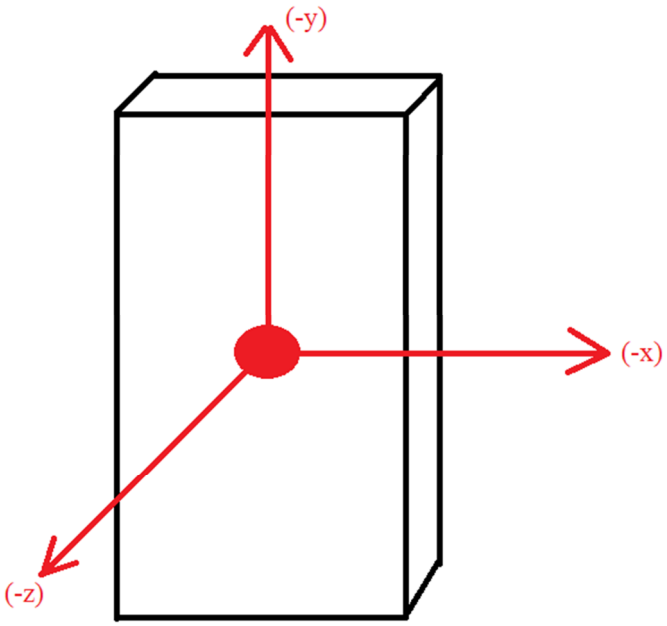

Figure 2: How the accelerometer responds to movement of the cellphone. The direction of the axes are all labeled as negative

Figure 2 shows how the cellphone position and movement affects accelerometer data. Moving in either of the arrows direction will cause the accelerometer to produce a signal in high magnitude relative to gravity, but the value will be negative instead of positive. This did not cause much issue, since a difference was used to classify movement as opposed to using the actual phone orientation. The accelerometer was easy to understand, but when it came to using the data, post data collection processing was required.

As stated before, the signal produced by the accelerometer is noisy. There is a low signal to noise ratio when dealing with accelerometer data, so most of the data seems to just be waves, from which no distinct pattern can be discerned. Figure 3 shows a perfect example of this.

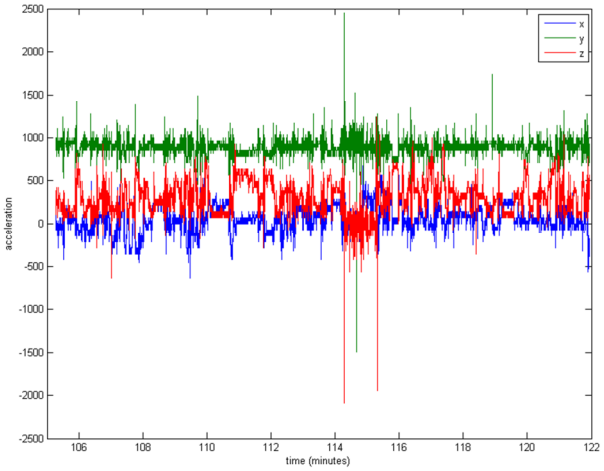

Figure 3: Sample accelerometer data taken from a different source [5]

In order to remedy this situation, some post processing is required. There are many signal processing techniques that are out of the scope of a computer science student, but one that stood out was the Fourier transform. In particular, the use of the Discrete Fourier Transform (DFT), was implemented in this project. The DFT was chosen because it processes the data in such a way that it allows the user to view periodicities in the data, so that most of the noise is removed, while trends and discrete values can be seen.

$$f_k = \frac{1}{N}\sum_{n=0}^{N-1} F_n \, e^{2\pi i k n/N}.$$

Figure 4: The formula for the DFT

This was the formula implemented in Java for the DFT. The implementation is provided in the code along with this report. A sample result of the DFT applied to our accelerometer data is shown in figure 5.

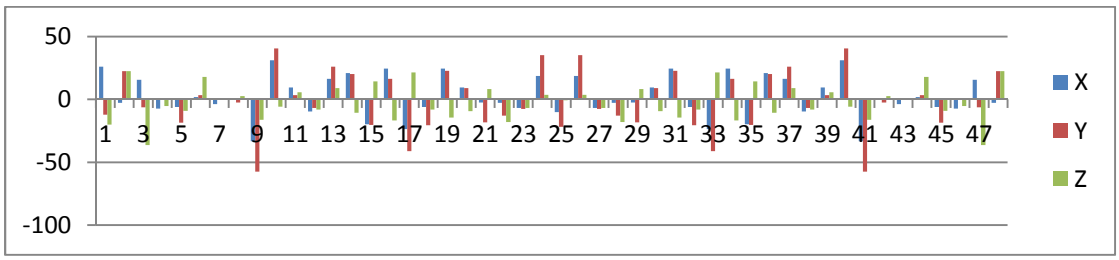

Figure 5: Sample DFT applied to some data collected from the accelerometer

Clear trends can be seen in figure 5, especially when compared to other values, as shown in the results section. Once the DFT was performed, the summation of the x, y, and z values were done.

Initially, my intention was to do an average, but the problem with the average was that the small values in the middle of the signal would pull the overall average down. This caused issues when attempting to determine whether a phone was moving between levels or not. Instead, a simple summation of the signal across each axes individually was performed, and each axes was individually analyzed. This provided clear distinction between walking and moving up a staircase. The main difference is if a phone was in a pocket, then the leg will begin to move at a different angle and speed to propel the person up the stair case, thus activating different axes on the accelerometer compared to walking. When analyzing the data, there was a noticeable activation in the z-axis and x-axis portions of the accelerometer based on the results of the DFT. This allowed for a distinction between walking vs going up the stairs. Unfortunately, the application does not distinguish going up the stairs from running. This is further discussed in the results section, but it is an important distinction that allowed for the user to increment/decrement the z level appropriately. Further refinement was required to distinguish between going up vs down the stairs. This was done by noticing that the magnitude of the summation of intensities post-DFT was almost the same, but going down the stair case showed a slightly more intense signal, meaning this was the small distinction that was desired. This is further shown in the results section. The full code shows the calculations performed. Overall, the combination of location services, accelerometer data, and Wifi direct has made this project successful as a proof of concept.

Materials & Results

Materials

In order to test the application, android devices were required. Unfortunately, due to budget restrictions, only 2 devices were used to prove the usage of this application. The first is an HTC One M7, AT&T version rooted and changed over to the T-Mobile version. The specifications can be found on the GSMArena website [3]. This phone is running the latest Android update, Lollipop 5.0.2. The other phone is a T-Mobile Samsung Galaxy S3, running an older version of Android (version 4.3). The interesting thing about this phone is that Wifi Direct is already built into it, but for the purposes of this project, it was not utilized. The code for the application was developed on Google's Android Studio IDE. The phone that was in use was placed in my pocket so that the long side lay on the bottom of my pocket, as I either walked around, or walked up the stairs. Based on timing with a stopwatch on traveling up and down the staircase, it took about 9.5 seconds for me, so data from the accelerometer was collected for about 12 seconds before processed and studied.

Accelerometer

As stated before, the accelerometer data had to be processed via the DFT before any calculations were done. Once that step was completed, calculations could then be done. Data from each experiment using the DFT and accelerometer is shown below.

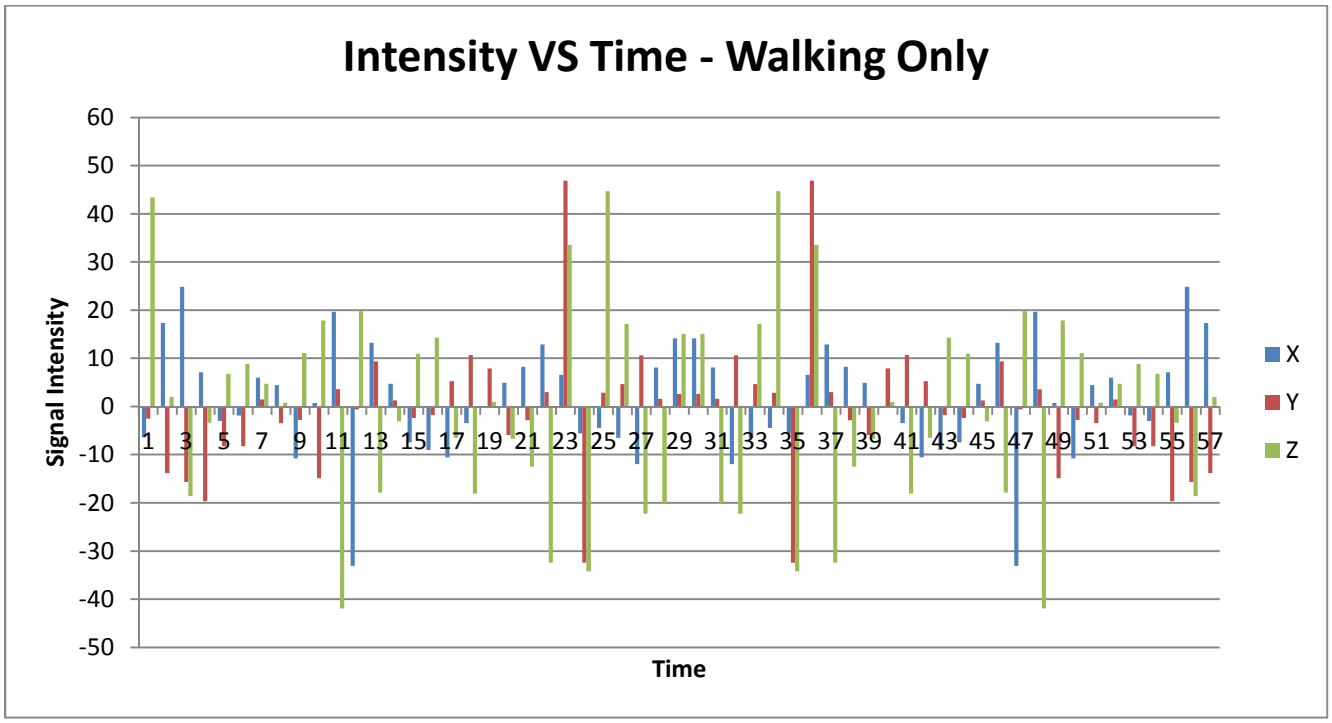

Figure 6: A graph showing the change in signal intensity during walking.

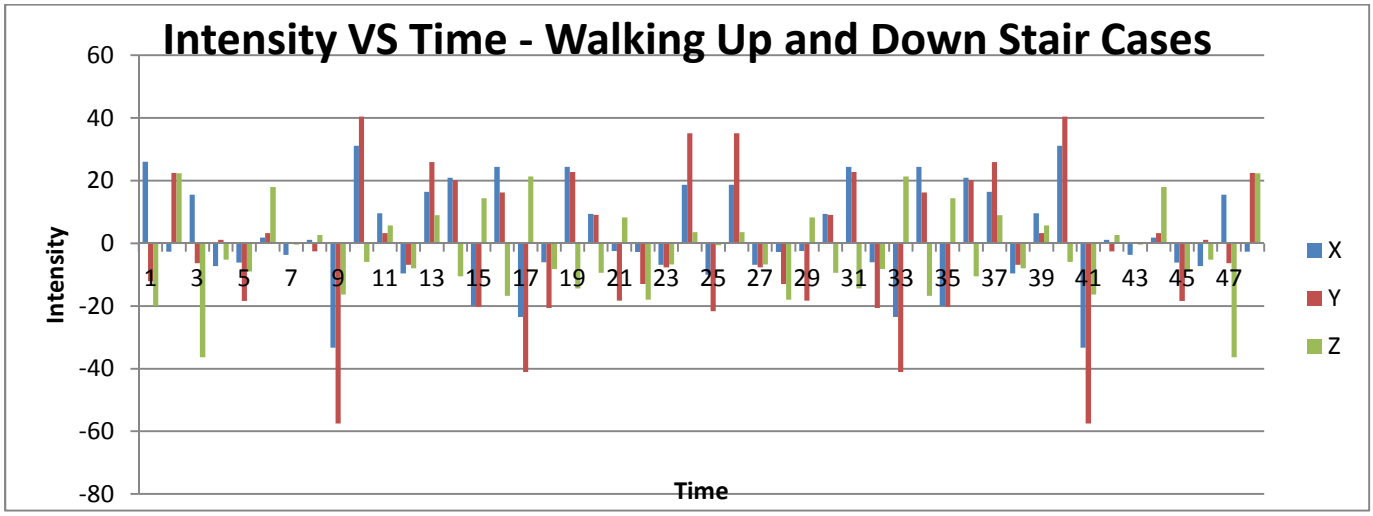

Figure 7: Graph showing the change in signal intensity over time while walking up and down the stairs at a moderate pace.

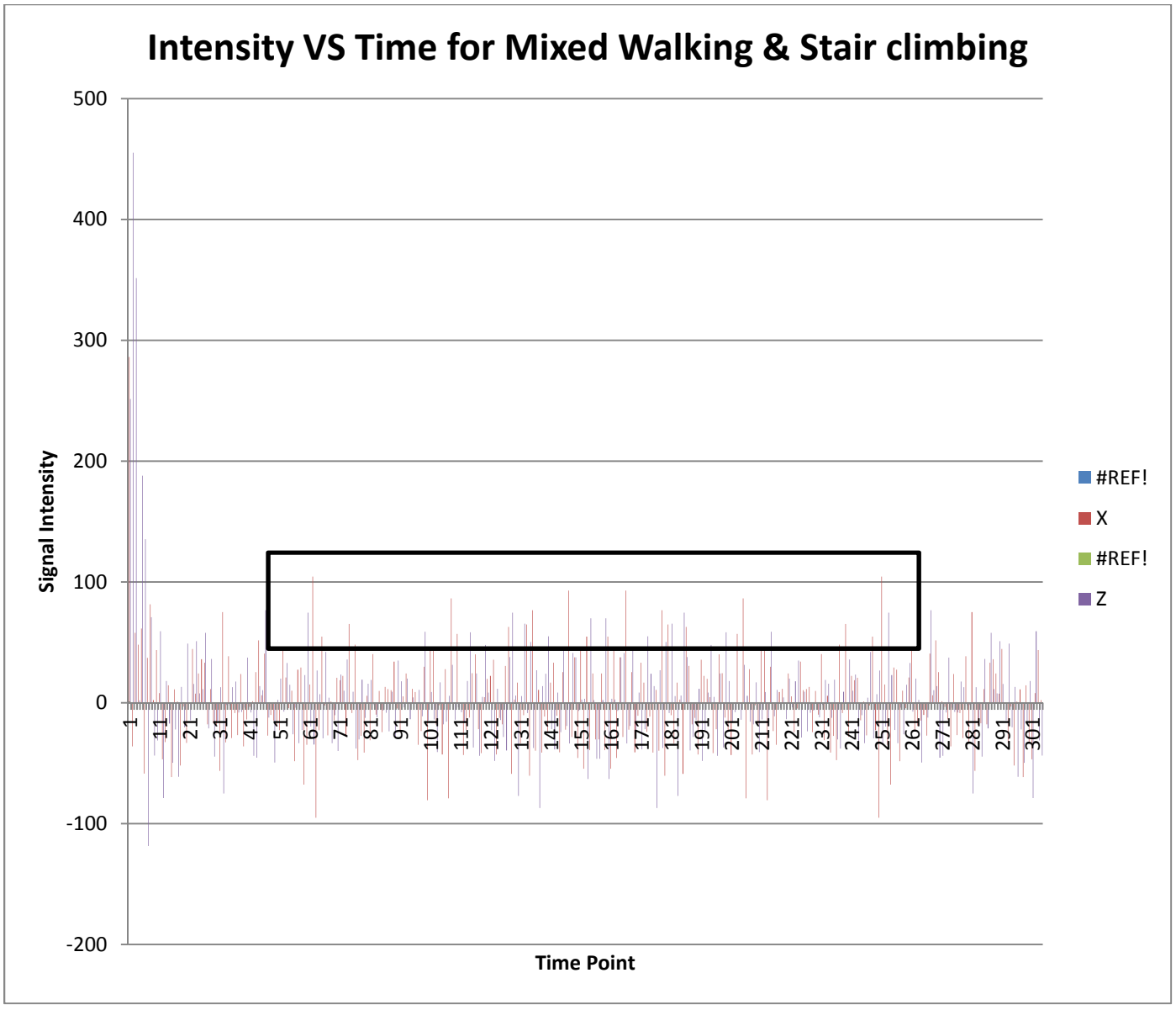

Figure 8: Mixed walking and stair climbing. Note, there are quite a few X peaks that are notable, and the data was collected for a 30 second time period. The time during which stair climbing occurred is boxed in black.

As stated before, the values were simply summed up, and then compared to previous values. The code saved the last stair climb that was used, and compared this current "stair climb". This took care of the situation where the user went up the stairs, and then back down the stairs. Otherwise, the values were compared to the last summations that were done, and if they were greater by a certain threshold, then we could determine if we walked up/down the stairs, and increment the z level accordingly. In all of the figures, it is worth noting that the first few data points, from 0 to about 7, are when I was taking the phone out of my pocket or placing it in. All

of the figures show a drastic increase in the accelerometer response, indicating that movement alone may have implications on the summation if not properly dealt with.

Figure 6 shows the intensity during a normal walk. Here we see that the accelerometer does not reach much in the x and z directions since it is mostly stationary. However, the y axis has the most movement during walking. This makes sense since the phone was placed sideways in my pocket, and based on figure 2, the y axis will have the most movement. Once I began stair climbing with the same method, the axes responses changed. The y axis remained relatively steady, but the x and z axes showed a response. The x axes showed a significant increase in the signal intensity, so the sum changed, almost doubling in most cases. This makes sense because as a human walks up a stair case, the leg must be lifted up faster than it is put down. So based on figure 2, the negative x axis is activated moreso than the positive x axis. The same logic applies to the z axis. Since the phone was horizontal, such that the z axis did not have much movement while walking, it suddenly changed its angle, and now the z axis has a greater response during stair climbing than originally in the strictly walking phase. Overall, the change in magnitude during walking vs stair climbing helped to differentiate between the two. This allowed for level change to be detected, incrementing the z level.

In order to decrement the z level, distinguishing between moving up vs down the staircase is required. Overall, the issue here was that the magnitude of both was approximately the same.

Walk up the stairs:
DFT sum result of X : -579.4062423706047
DFT sum result of Y : -1167.8052520751933
DFT sum result of Z : -542.6313400268543

Walk down the stairs:
DFT sum result of X : -613.882770538329
DFT sum result of Y : -1170.1036834716776
DFT sum result of Z : -595.4953193664549
Figure 9: The magnitude of each sum for walking up the stairs vs down the stairs.

As figure 9 shows, there is a very small difference amongst all axes for walking up vs down. In order to distinguish, I assumed that walking down will have higher magnitudes in comparison to walking up because human legs can sway and move more during the step down, increasing the magnitude of the signal produced by the accelerometer. In this case the y axis comes in handy as it also increases alongside all other axes. Overall, combining DFT with the accelerometer provided useful results in attempting to distinguish walking from stair climbing successfully.

Location & Relative Location

The next step is to calculate relative location. As stated before, this is simply a difference in longitude and latitude. In order to study this, I placed 1 phone in one part of the building, and kept the other phone with me. I was able to transfer the data file from one phone to the other successfully. Now, in order to parse the file, I read in all the lines into a List<String>, and then began scanning this list backwards so as to get the most recent location update. This successfully gave me the latest update, and I was able to compute the relative location between me and my peer. Since the location is simply a difference, I now had a directionality associated with it since I can treat this information as a vector, where the x component is latitude, and y component is longitude. I was not able to do this, but I could place both points on a map and draw an arrow between the two to show directionality and distance.

Along with 2D coordinates, I was able to extract 3D information as well. For my phone, I walked up stairs, paused the app, then walked up again, and was able to get my z level up to 2. I then had the other phone placed downstairs, so that the z level was 0. A 0 z level indicates that the phone has not changed position relative to its initial position upon application start. I then calculated relative positions, and found that my peer was at (-2.0) z level, indicating it is 2 floors below me. Knowing this, along with the latitude and longitude, I could create a 3D vector, adding the z level as the z component. Overall, this application was successful in using accelerometer data to track movement, location to get exact x,y coordinates, and finally using this information to construct a 3D representation of the relative location of each node.

Conclusion & Future Work

Overall, this application turned out to be a success. I was able to use accelerometer data and location information to determine relative location. The upside is the addition to do so in 3D. I have a vector containing latitude, longitude, and z level. Combining these three pieces allows me to construct 3D relative locations between nodes. This information has many potential applications. It could be used by builders to determine if anyone is in a building, or it could be used by Google to create a 3D structure to their current google maps so that they provide depth information along with the location of people inside a building in real time. This is simply a proof of concept showing that important information can be extracted from sensors found in everyday smartphones. The next step would be to triangulate exact coordinates in a building so that level information can be easily known, not just 2D location. Once that is known, a 3D map of the sensor nodes can easily be constructed without having a predetermined knowledge of placement. Also, automation of the peer connections would allow for multiple peers to be randomly selected as opposed to the user manually selecting the peers. Overall, this is a successful start to a project that could go a long way.

References

1. A beginner's guide to accelerometers.
   http://www.dimensionengineering.com/info/accelerometers

2. Discrete Fourier Transform.
   http://mathworld.wolfram.com/DiscreteFourierTransform.html

3. HTC One – Full phone specifications. http://www.gsmarena.com/htc_one-5313.php

4. Location Strategies | Android Developers.
   http://developer.android.com/guide/topics/location/strategies.html

5. Normalization – how should I normalize my accelerometer sensor data?
   http://stats.stackexchange.com/questions/40354/how-should-i-normalize-my-accelerometer-sensor-data

6. Samsung Galaxy S6 – Full phone specification.
   http://www.gsmarena.com/samsung_galaxy_s6-6849.php

7. Younis, Mohamed. Lecture 1. UMBC, CMSC684