5-2015

Mobile Health Sensing with Smart Phones

Abby Logan Wise
University of Arkansas, Fayetteville

Mobile Health Sensing with Smart Phones

An Undergraduate Honors College Thesis
in the

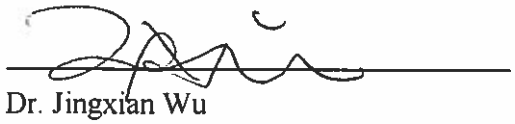
Department of Electrical Engineering
College of Engineering
University of Arkansas
Fayetteville, AR

by

Abby Logan Wise

This thesis is approved.

Thesis Advisor:



Dr. Jingxian Wu

Thesis Committee:

_____

_____

Abstract

Depression is a serious mental condition affecting many Americans and people across the world. Postpartum depression is a form of depression affecting women who have recently gone through childbirth. Postpartum depression can have serious symptoms that may affect not only the mother, but her newborn as well. Early detection of these symptoms is of critical importance to the welfare of the mother and her child. Many medical professionals express the need for postpartum screening being that treatment has proven to be very effective. This project aims to make early symptom detection convenient and reliable for mothers with newborns. This will be done be selecting measurable symptoms that can be detected using a variety of sensors contained in mobile "smart" phones, specifically Android-powered phones. An important aspect of this project is the selection of these symptoms and sensors. The focus of this project is motion detection and volume of conversations. These can be tracked using the microphone with a getMaxAmplitude filter function and the linear accelerometer. The Android Studio platform is the development tool used for this application. All data gathered by the phone sensors will be compiled using the application and sent to a server to be analyzed to provide warnings to the user about their potential for postpartum depression and urge them to consult a doctor.

ACKNOWLEDGEMENTS

I would like to thank my honors advisor, Dr. Jingxian Wu, for allowing me the opportunity to work on this project. It has been a great experience for me. I appreciate all the guidance and support he has provided.

I would also like to thank Connor Malpass and Joe Moquin for taking the time to explain the Android programming concepts that evaded me. Without their help, the project would not have been as successful as it is.

Table of Contents

List of Figures

1. Introduction

1.1 Problem

Postpartum depression is a serious mental illness affecting women everywhere. It can be difficult to diagnose in both early stages and later stages due to the symptoms' mental nature. Many mothers are not willing to admit to having feelings of depression for fear of the stigma connected to the disease. This makes early detection difficult because it may often take family or friends noticing symptoms to push the woman to consult a doctor. Even if the woman were to see a doctor, there is a high possibility for false reporting on the typical surveys for postpartum depression. It can be difficult to admit to having the feelings associated with the illness and can often lead to dishonesty when speaking to a doctor or completing a survey.

It is necessary to collect less subjective information about the woman's daily life. In order to do this sensors must be implemented that can track aspects related to the symptoms of postpartum depression.

1.2 Thesis Statement

The goal of this project is to create a mobile phone application that can help track potential symptoms of postpartum depression to achieve early diagnosis. This application is intended to be an easy to use system of sensors in order to encourage women to use the application despite the stress of having a newborn. It is also necessary to track the user without invading their privacy by capturing their conversations or location.

1.3 Approach

In order to create this application tracking points had to be selected based on potential postpartum depression symptoms. Once these were decided upon, it was

necessary to find how these are best tracked using Android smartphone sensors. Preliminary ideas included tracking the volume of conversations and activities around the user. Leading to consideration of the phone's microphone with a lowpass filter. The project later developed to include two tracking criteria, volume and motion.

After extensive research on available sensors, the *getMaxAmplitude* function related to voice recording and the linear accelerometer sensor were chosen to be implemented in the application. This involved heavy research into programming in both java and xml.

Once both sensors were implemented, data was taken as an example of the functionality of the application. It was graphed to represent a daily action of the user.

1.4 Potential Impact

Tracking certain aspects of the new mother's life in order to obtain the data necessary to suggest consulting a professional can help both issues of early detection and false reporting. The data collected could detect early potential for depression before it is noticed by friends and family. Also, a notification from the application could encourage women to be more honest about their symptoms because they can see the potential for a real problem.

1.5 Thesis Organization

This thesis consists of five main sections. The introduction section lays out the reason for the project and how it is to be completed. The background section contains information about postpartum depression, competing products, and using Android Studio. The research section details the sections of the project. This section discusses the selection of the volume and motion sensors based on postpartum depression surveys. It

also details setting up Android Studio software, programming the volume tracking, and programming the motion tracking. The data tracking section shows actual data taken by the device using the project code for both the volume and motion tracking. It also provides the data in graphs to show how the tracked data can be viewed for analysis. The conclusion section summarizes the research and discusses additions to be made to the program.

2. Background

2.1 Postpartum Depression

Around 14.8 million adults in the United States alone suffer from some form of serious depression [1]. This depression can often coincide with other illness or for about 14% of women who have children, childbirth [2]. Women suffering from depression related to childbirth are classified as having postpartum depression. Many women suffer from mood swings for a short time after giving birth, and it can be difficult to immediately realize the possibility of having a more serious mental illness. This can allow the illness to continue to worsen. As postpartum depression can lead to severe mood swings, insomnia, and thoughts of harming yourself or your child, early detection can be essential to the well-being of the mother and her newborn [3]. Early detection allows a woman to preventatively seek medical help rather than receiving treatment after a serious incident has occurred. The treatment of postpartum depression has proven to be effective in relieving mothers of their potentially harmful thoughts or feelings and should be implemented as early as possible.

2.2 Android Powered Smartphones

Several measurable aspects of postpartum depression exist. It is necessary to identify these signs and collect them in a readable fashion. Many sensors exist that can be used to detect the desired data. Mobile “smart” phones contain such programmable sensors for use in phone applications. Many Android-powered devices contain programmable sensors that detect motion, environmental factors (temperature, humidity, etc.), and physical position. These sensors may be either hardware or software based and can be accessed by an app developer [4]. The sensors available may depend on the

operating system version on the device. The most recent operating systems implemented are Android 4.0, Ice Cream Sandwich, Android 4.1, Jellybean, Android 4.4, KitKat, and Android 5.0, Lollipop [5]. Since mobile phones are used by a large percentage of the population and have built in sensors, application development is a good starting point in the collection and analysis of signs of postpartum depression.

2.2.1 Sound Tracking Sensors

App developers can access the phone's microphone for multiple uses including voice recording, visualizing sound, and assessing volume. The microphone has been programmed with several functions that allow the device to process sound multiple ways. The microphone is accessed through the public class MediaRecorder that houses these functions. Several critical functions include *prepare* in order to prepare the microphone to begin capturing data, *start* to actually capture data, *stop* to cease data collection, and *reset* to set the audio source back to its idle state. Another useful function is *getMaxAmplitude* that sends back the maximum amplitude of noises captured after each call to the method [6].

2.2.2 Motion Tracking Sensors

The Android platform houses several sensors for various aspects of motion detection. Some of these sensors of interest are TYPE_ACCELEROMETER, TYPE_GRAVITY, and TYPE_LINEAR_ACCELEROMETER. TYPE_ACCELEROMETER and TYPE_LINEAR_ACCELEROMETER both track acceleration along the x-axis, y-axis, and z-axis. The linear accelerometer differs from the accelerometer in that it removes the gravity on each axis. This could also be done manually by subtracting TYPE_GRAVITY from TYPE_ACCELEROMETER as the

TYPE_GRAVITY sensor only tracks the force of gravity along each axis. It is important to note that before Android 4.0, Ice Cream Sandwich the linear accelerometer was a software sensor, but has since been moved to a hardware sensor within the gyroscope. Not all phones have the gyroscope available [7].

2.3 OnePlus One Mobile Phone

The OnePlus One A0001 mobile phone runs the Android 4.4, KitKat operating system. It runs version 4.4.2 with the option of upgrading to version 4.4.4. The phones available sensors include the accelerometer, gyroscope, proximity sensor, and compass. The presence of the gyroscope in the KitKat operating system means that the linear accelerometer is accessible. All microphone functionality is also available on this device. The phone uses a USB cable to charge and connect to a computer to download an application program [8].

2.4 Using Android Studio

The Android Studio software development language contains both Java and xml. A project's main code is contained within the *MainActivity.java* that is programmed using Java. This is where most of the functional code is contained that performs the required tasks. Here the functions that access and use the sensors are defined and called. The *activity_main.xml* file is where the graphics information is generated. When a user tool (such as a push button, text box, etc.) is created on the virtual phone screen provided, the program code is generated in the *activity_main.xml* file. Permissions to access different sensors in the phone such as the microphone and speaker are programmed in the *AndroidManifest.xml* file. When the user downloads the application, the user agrees to the permissions set in this file, and these aspects of the phone may be accessed. [9] The

virtual phone screen provided can be used to physically place and organize the user interface. This can also be done in the .xml file by using the proper function calls for each user graphic and stating its relative location on the screen.

2.5 Similar Products on the Market

Several applications are already on the market for voice recording and motion tracking. The applications relating to postpartum depression specifically are survey based relating to the questions asked in the Edinburgh Postnatal Depression Scale. They do not focus on tracking real aspects of someone's daily life, which can lead to dishonest answers and biased results.

Applications for auto tuning voices and voice altering use similar filtering and modulation as is required to distort a conversation. These applications make use of the microphone and speaker and are widely available on the market. Applications related to motion tracking make use of several motion sensors to detect movement both movement and location. The gyroscope and linear accelerometer are used in running applications that track movement to determine distance. This may be used to track the amount of movement of a user to analyze if they have become less active. All these available sensors have been used widely to develop applications currently in existence.

3. Research

3.1 Selecting the Sensors

Two major surveys exist in order to aid in the diagnosis of postpartum depression. The Beck-Depression Inventory is a survey for general depression screening. The Edinburg Postnatal Depression Scale is a survey aimed specifically at diagnosing postpartum depression. This survey consists of ten questions related to happiness, anxiety, irritability, inexplicable crying, changes in sleeping habits, and feelings of self-harm. This is a trusted screening process and is widely used [10]. Despite the fact that the Beck-Depression inventory is a general depression-screening tool, it is used in the detection of postpartum depression as well. It is a 21 question survey that focuses on sadness, self-dislike, unexplained crying, agitation/irritability, changes in sleeping patterns, and feelings of self-harm. While not as commonly used as the Edinburgh Scale, it is found to be relatively reliable [10]. From these two screening surveys the typical signs of postpartum depression can be assessed. It can be seen that irritability, inexplicable crying, and activity level can be useful in the detection of postpartum depression. These can be analyzed by tracking volume of sounds along with motion.

3.2 Setting Up for Programming

The first step to developing the application was to download Android Studio and the SDK software from developer.android.com/sdk/index.html. These both have versions for all computer platforms. The proper one was selected for the computer running the program. The instructions for downloading were on this page, and the download prompted the user when the installation process required user input. Once the software was downloaded, a new application had to be created. The application had to be given a

name, company domain, and project location. The company domain generated a package name later used in the *MainActivity.java* file. On the next configuration page, both phone and tablet were selected to run the application and the proper operating system was selected. This involved ensuring the correct operating system version was selected that would work for the device(s) that would run the application. It is standard to select the minimum platform the device that is being used can run on. A blank activity was chosen and the new application was created. The theme design for the application was also chosen at this time. It was important to ensure that no errors occurred upon the start of the new application. Any initial errors had to be taken care of before any programming could begin otherwise no program will build or debug properly. The first line of the *MainActivity.java* file must be the package name for example *package com.example.user.programTitle();* where *user* is the name the developer set the program under and *programTitle* is the title of the project that the developer is saving the program under. This was essential to allowing the program to build properly. Once these tasks were properly completed, the program would allow building and debugging.

Before programming began, it was necessary to import specific libraries for sensors and functions that have already been created for Android Studio. These are imported based on their information types (e.g. widget, media, and hardware). The most important libraries for this program include *os.Bundle*, *view.View.OnClickListener*, *widget.Button*, *widget.Toast*, *util.Log*, *media.MediaRecorder*, *hardware.Sensor*, and *hardware.SensorEvent*. This is done by simply writing a line of code at the beginning of the program (e.g. *import android.os.Bundle;*). The uses of these libraries are shown throughout the program description.

In order to use a phone for application testing it must be set up for development. For operating system used by this phone (Android 4.4.2, KitKat) the developer go to *Settings->About Phone*. An option in this menu is *Build number*. This needs to be clicked seven times in order to bring up the *Developer options* tab. This turns on developer options and allows for different developer settings to be selected. Without completing this task, the device will not allow for code testing.

3.3 Tracking Volume

Since it is essential that only volume data be collected without recording details of the users conversations, the original idea was to implement a lowpass filter design using Matlab into the Android application program. A simple 2nd order Butterworth filter was created as shown below in Figure 1. The cutoff frequency for the filter is 50Hz since that is a range that would remove enough clarity from human voice while still gathering data about the volume. Figure 2 shows both the magnitude and phase of the filter's response.

```
%% Voice Recording Filter

% Lowpass Filter (Butterworth filter)

[b,a] = butter(3,0.1)
freqz(b,a)
input = randn(1000,1);
output = filter(b,a,input);
```

Figure 1: Butterworth filter created to filter conversations

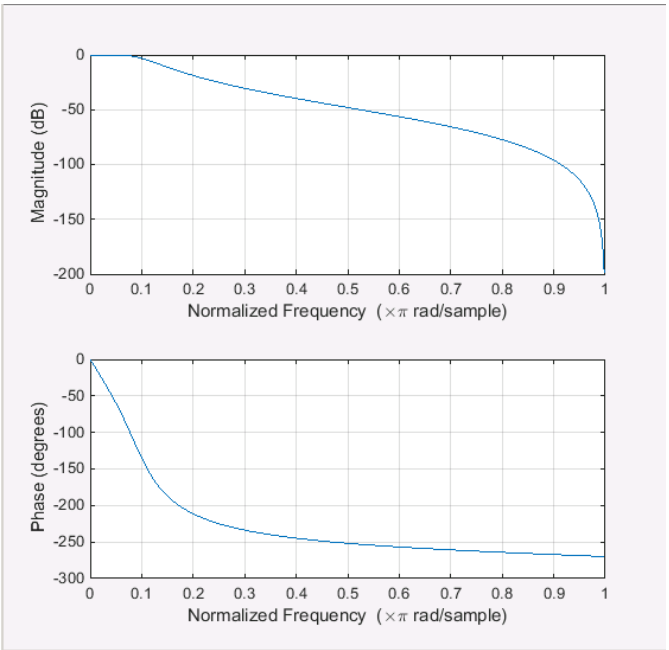


Figure 2: Plotted response of Butterworth filter

While researching and developing the early program code, the *getMaxAmplitude* function was discovered making the original filter design obsolete based on the complications with implementing a Matlab code into an Android application.

In order to track volume the microphone and *getMaxAmplitude* function were used. For an application to access the microphone, the program must include permissions for both recording audio (RECORD_AUDIO) and writing external storage (WRITE_EXTERNAL_STORAGE). These are coded into the manifest xml file. When the user downloads the application, they are agreeing to the permissions declared in the manifest file.

The actual functions to start recording, call the *getMaxAmplitude* function, and stop recording were all placed in the *MainActivity.java* file. Here, it is necessary to call *setAudioSource*, *setOutputFormat*, *setAudioEncoder*, *setOutputFile*, *prepare*, and *start*. All of the set functions set up the microphone for use. The *prepare* function then tests to

see if the microphone can be started. If it cannot, it sends a programmed message to the logcat that is tracked by the developer in the Android Studio software. Once that check is successful, the start function allows the microphone to begin taking audio data. This information is placed within the *onStart* function and utilizes the built-in *onStart* command. This *onStart* command allows for an activity, in this case the *MediaRecord* sensors, to be called by user input. The function contains all the microphone setup calls and is located in public class *MainActivity* that houses the majority of the code in this file. This function contains a call to the logcat to print out "Start Record Called" to ensure that this function is called properly. The *onStart* function also calls the *getAmplitude* function.

getAmplitude calls the built-in *getMaxAmplitude* function that is used to capture only information about the volume, not recording sounds and conversations around the user. This captures an integer value in the range from 0 to 32767. Within this function is the following equation.

$$double\ max\ =\ 20 * Math.log(mRecorder.getMaxAmplitude()/2700) \quad (3.1)$$

Equation 3.1 takes the integer value and converts it to a decibel value. This equation translates to equation 3.2 below.

$$new\ max = 20 * \log_{10}(maximumAmplitude/2700) \quad (3.2)$$

Due to this equation a value that is too small is documented as –Infinity. The *getAmplitude* function contains two calls to the logcat, one to print out a confirmation message "getAmplitudeFunction Called" and the other to print out the maximum amplitude values found. These are constantly streaming to the logcat when the *onStart* function is called. To get the constantly streaming data, the getMaxAmplitude function must have also been located within the public void *onSensorChanged* function. This

causes the data to print out data on any sensor event. This means that the logcat will print a value of 0.0 for the maximum amplitude until the start button is pushed and will return a value of 0.0 again after the stop button is pushed. While the start is running, the maximum amplitude will print as expected. The *onStart* function is called within the public boolean *onCreateOptionsMenu*. This is called within the *button* function aimed to start tracking.

In order to stop tracking sound the *MediaRecorder stop* and *reset* are to be called. These are both called within the *onStop* function. *onStop* contains the built-in *onStop* command and a call to the logcat that prints out "Stop Record Called", so the developer may see the function has been properly called. A second button, *button2*, is used to call *onStop*. This is contained within the *onCreateOptionsMenu*. Figure 3 below shows a general flowchart for general layout of the volume tracking program.

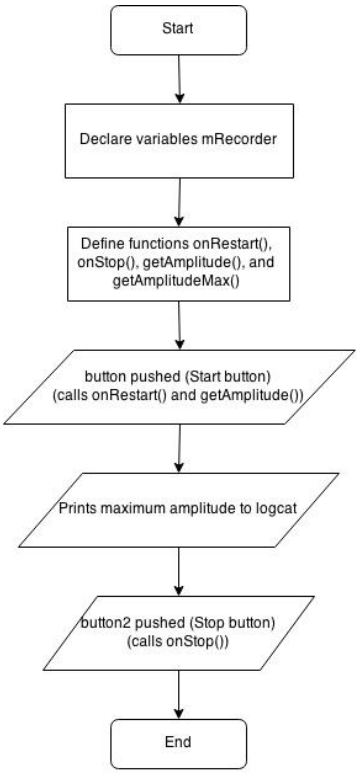


Figure 3: Flowchart for volume tracking

3.4 Tracking Motion

After researching each available motion sensor, the linear accelerometer was chosen to be the best option for tracking an individual's daily motion. All the functions used to implement the linear accelerometer are in the *MainActivity.java* file. Since this function requires quite a bit of math it was necessary to declare several variables. These variables are x, y, z, xDelta, yDelta, zDelta, vX, vY, vZ, and a. These are to be called later in the program to obtain the acceleration in each direction, the velocity and each direction, and the acceleration of the vector.

The linear accelerometer currently starts as soon as the application is opened due to a programming error. The stop button must be selected when the application opens to allow for normal operation. In order to restart the linear accelerometer the defined sensor manager must register the sensor listener. This task is called using the built-in *onResume* command placed in the function *onResume*. A message stating "Motion Resume Called" appears in the logcat if the sensor listener is registered. The sensor listener is created within the protected void *onCreate* function that defines the sensor as TYPE_LINEAR_ACCELEROMETER. This runs a check to ensure the linear accelerometer is called and prints "Linear Accelerometer Called" to the logcat if successful or "Linear Accelerometer Failed" if it has failed.

The *onStart* function allows for all information in the public void *onSensorChanged* function to register the change and run. This is where the linear acceleration values are called upon. The original values captured by the linear accelerometer are sent to x, y, and z. The values that are actually returned are xDelta, yDelta, and zDelta. These values are the result of equations 3.3-3.5 below.

$$xDelta = Math.abs(x - event.values[0]) \quad (3.3)$$

$$yDelta = Math.abs(y - event.values[1]) \quad (3.4)$$

$$zDelta = Math.abs(z - event.values[2]) \quad (3.5)$$

The values[0], values[1], and values[2] are the acceleration minus Gx, Gy, and Gz respectively. This is used to get the appropriate values to be displayed. xDelta, yDelta, and zDelta are the acceleration values for each axis presented in meters/second^2. The velocity it each direction is also tracked by using equation for the x-axis below.

$$vX = vX + xDelta * 0.001 \quad (3.6)$$

This equation simply takes the initial velocity and adds it to the new acceleration multiplied by 1 millisecond. The velocity values are all given by vX, vY, and vZ. The acceleration of the vector is also calculated as seen below.

$$a = sqrt(xDelta * xDelta + yDelta * yDelta + zDelta * zDelta) \quad (3.7)$$

This is translated as the simple acceleration vector equation shown below.

$$a = \sqrt{x^2 + y^2 + z^2} \quad (3.8)$$

All of these values are continuously captured and sent to the logcat to be monitored by the developer.

It is important to note how the coordinate plane is laid out within the mobile phone in order to understand how the data is tracked. Figure 4 below displays the x, y, z axis from the front view of a mobile phone where the x-axis comes from the side of the phone, the y-axis comes from the top, and the z-axis comes from the middle.

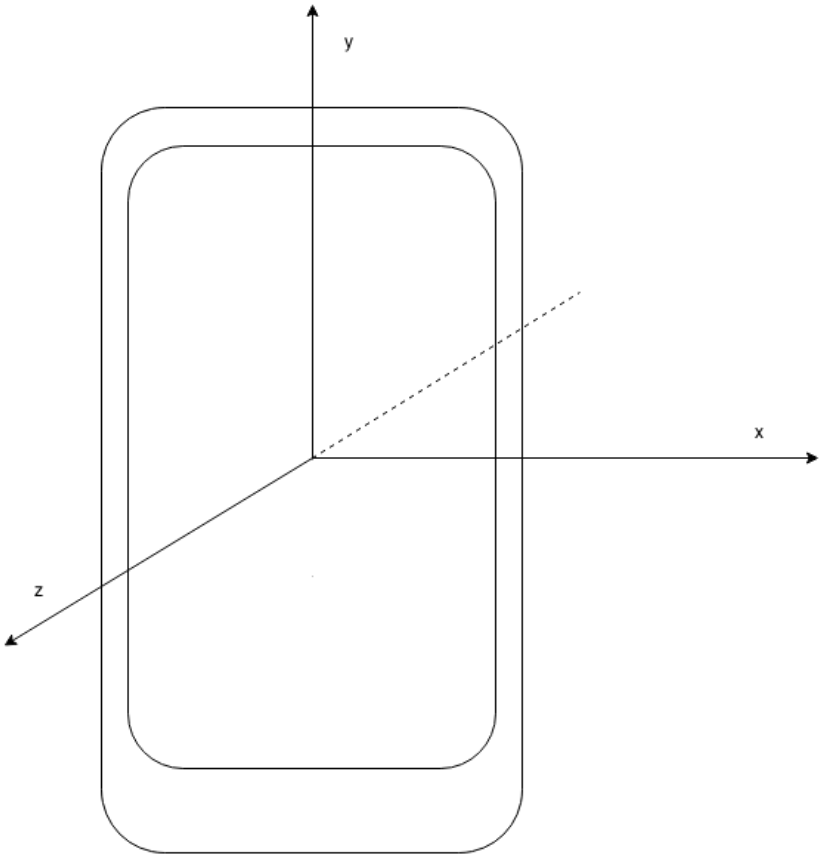


Figure 4: Diagram of phone x, y, and z axis

In order to stop tracking motion the sensor listener must unregister. This is called within the *onPause* function. *onPause* contains the built-in *onPause* command and a call to the logcat that prints out "Motion Pause Called", so the developer may see the function has been properly called. The *onPause* is designed to be a temporary stop. A second button, *button2*, is used to call the *onPause* function. This is contained within the *onCreateOptionsMenu*. Figure 3 below shows a general flowchart for general layout of the motion tracking program.

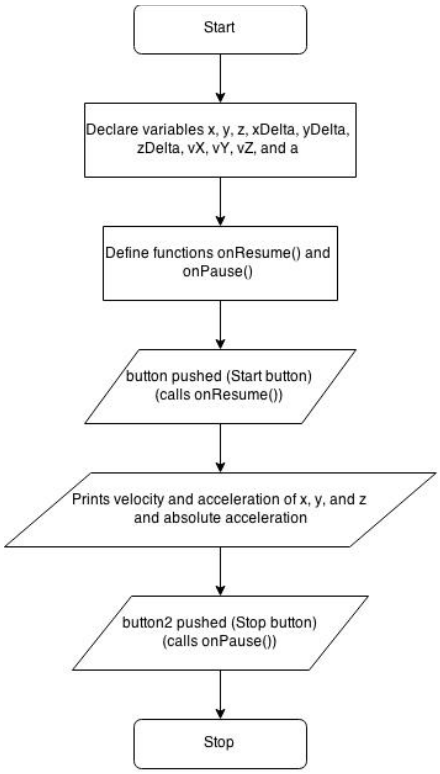


Figure 5: Flowchart for motion tracking

3.5 User Interface

Aside from the calls to the buttons in the *MainActivity.java* file, the user interface is built entirely within the *activity_main.xml* file. Within the "Design" tab of this file the text boxes were both placed which describes each button to the user. These generate all necessary xml code pertaining to the text and location in the "Text" tab. The next step was to add the actual buttons that the users press to begin and end tracking. These were placed onto the simulated screen within the "Design" tab and the label text was set. It was necessary to set these buttons as "clickable". The actual user interface designed is shown below in Figure 6.

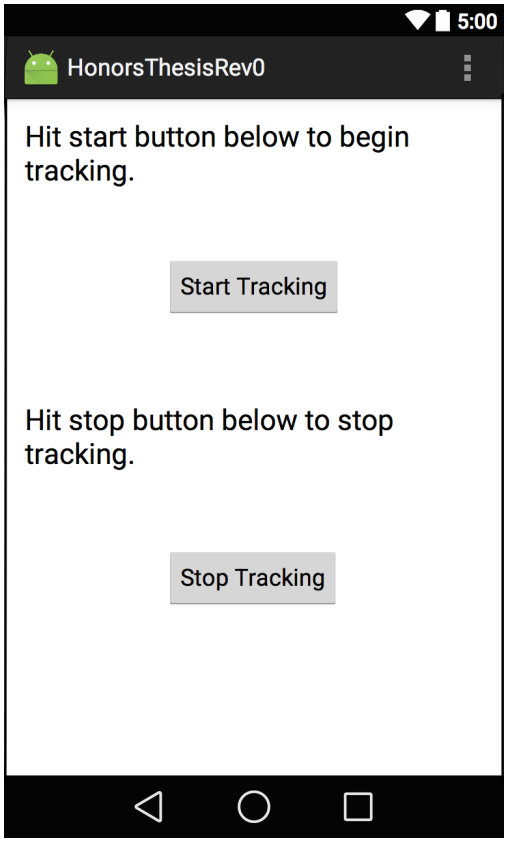


Figure 6: User interface for the application

The "Start Tracking" button is labeled *button* in the program. This is the button that calls the *onRestart* and *onResume* functions to start the volume and motion tracking. The *button* function in the *MainActivity.java* file contains a toast call that prints the message "Tracking" momentarily on the screen to notify the user the command has sent. The "Stop Tracking" button is labeled *button2* and calls the *onStop* and *onPause* functions to stop the volume and motion tracking. The *button2* function in the *MainActivity.java* file also has a toast call, but this message reads "Tracking Stopped". This gives the user notice that their data is no longer being tracked.

4. Data Tracking

4.1 Viewing Data

As has been mentioned within the programming documentation, the data is viewed in the logcat. There is a terminal within Android Studio that accesses the program files. It was necessary to enter the file location into the terminal to access the platform-tools for this specific project. From this location the logcat was called using the following command *./adb -s d573e9cd logcat -s "DataPointCollection"* where *d573e9cd* was the device name and *"DataPointCollection"* was the developer defined log name within the program. The logcat continuously tracks the maximum volume amplitude and the velocity and acceleration of each axis based on the logcat calls built into the program code. A data point for each function is shown approximately every 3/8ths of a second. Each line of information appears in the terminal next to a number that shows that all lines with that value were collected within the same run. Each logcat command from the program prints on its own line.

4.2 Volume Data

Two sets of volume data were taken in order to demonstrate the function of the volume tracking portion of the application. The first was a control. This was done by recording a single voice repeating a single word while maintaining the same tone and volume. The results of this recording were placed in an Excel file to be graphed. This graph can be seen below in Figure 7.

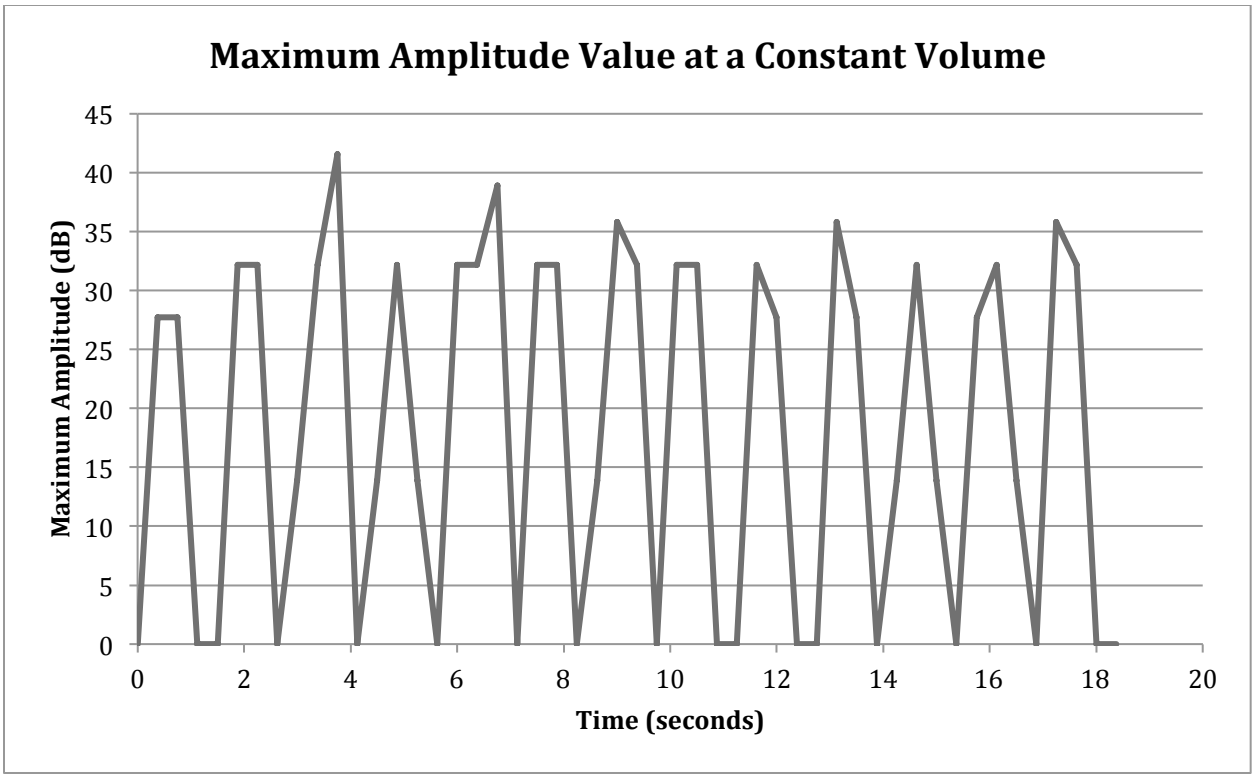


Figure 7: Maximum amplitude value at a constant volume graph

This graph shows that the volume tracking was fairly accurate. The high points for each spike are within a close range of one another demonstrating when the word was being spoken it was at approximately the same volume level. The constant drops in amplitude to zero represents the short pause between repetitions of the word. The slight variation between the high points can be attributed to inconsistency in the actual volume of the voice being recorded.

The second set of volume data taken was based on a single voice that got continuously louder and then continuously quieter. This data was also moved into an Excel file where it was graphed. This graph is shown below in Figure 8.

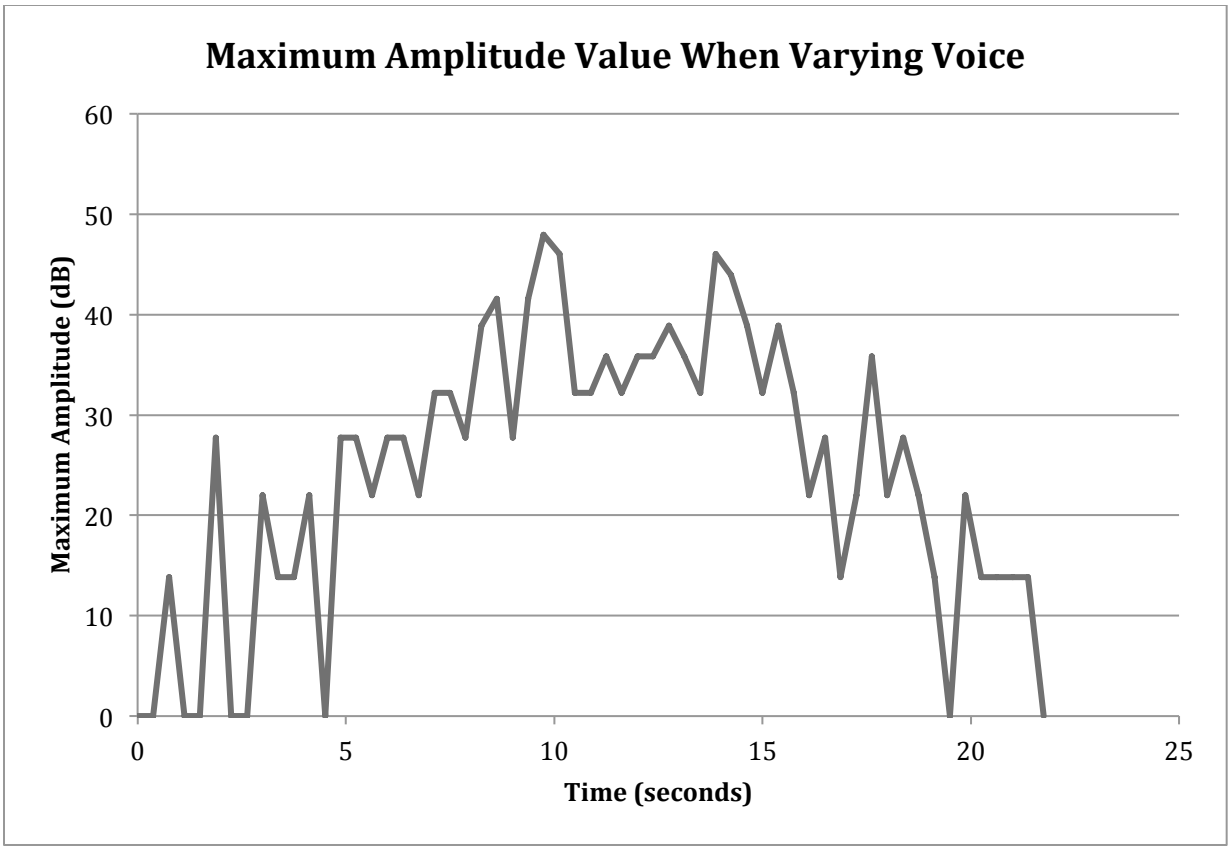


Figure 8: Maximum amplitude value when varying voice graph

This graph also shows good results for the volume tracking functionality. The recording was started when there was no sound and continued to record as the voice volume increased. In between 10 and 15 seconds the voice volume began to decrease until stopped completely. This is accurately depicted in the upward trend of the first half of the graph until it reaches a peak and then follows a downward trend until the recording is ended. Low spikes during the recording can be attributed to natural pauses in speech due to breaks between words and breathing.

4.3 Motion Data

Two sets of motion data for each axis were taken in order to demonstrate the functionality of the motion tracking side of the program. All three sets of data were taken in similar fashion. This was done by orienting the phone based on the axis that was being tested and walking in one direction at an increasing speed, maintaining that speed for a few seconds, and then decreasing speed until reaching a stop. In order to orient the phone in the proper direction for each axis, the phone was held such that the axis in question was pointing forward (see Figure 4 for the phone's axis orientation). Both the acceleration and velocity for each axis were tracked and graphed using Excel. Figures 9, 10, and 11 below show the acceleration graphs for x, y, and z respectively.

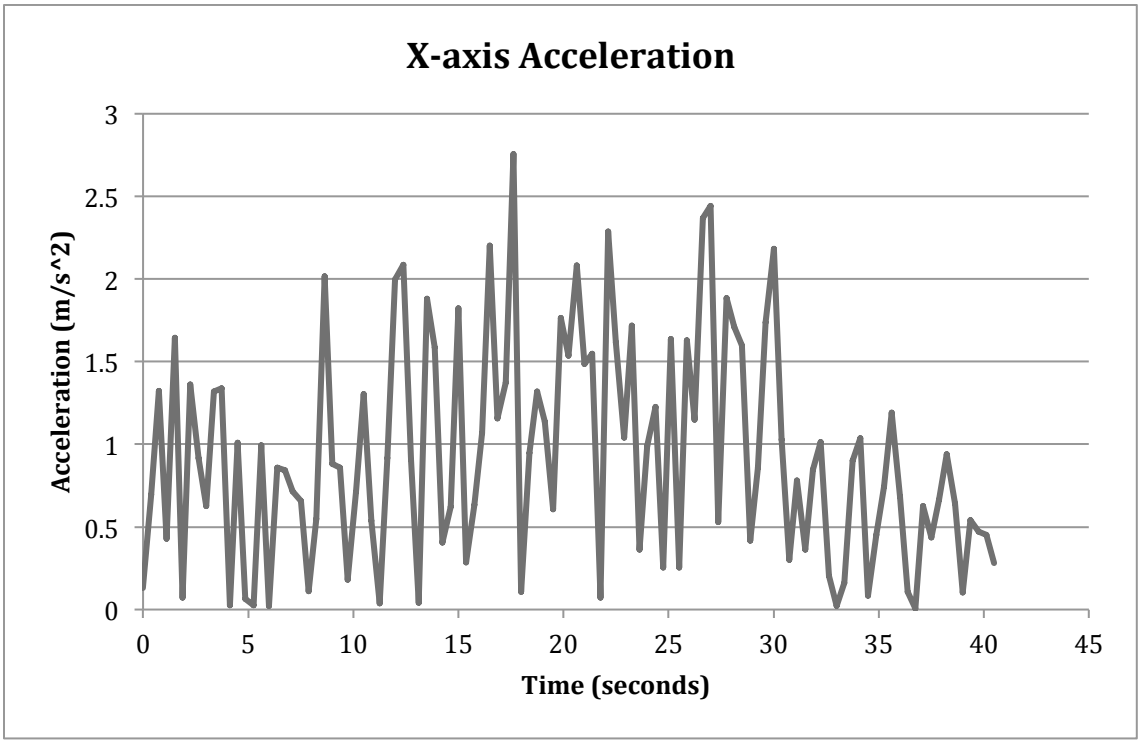


Figure 9: X-axis acceleration graph

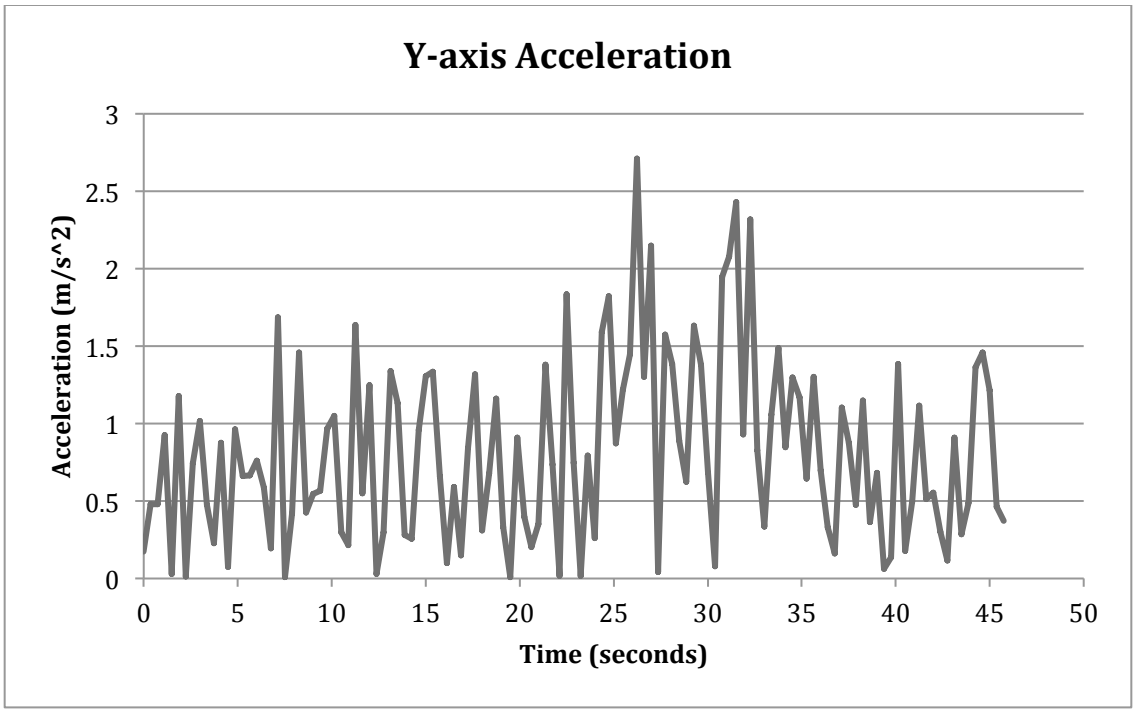


Figure 10: Y-axis acceleration graph

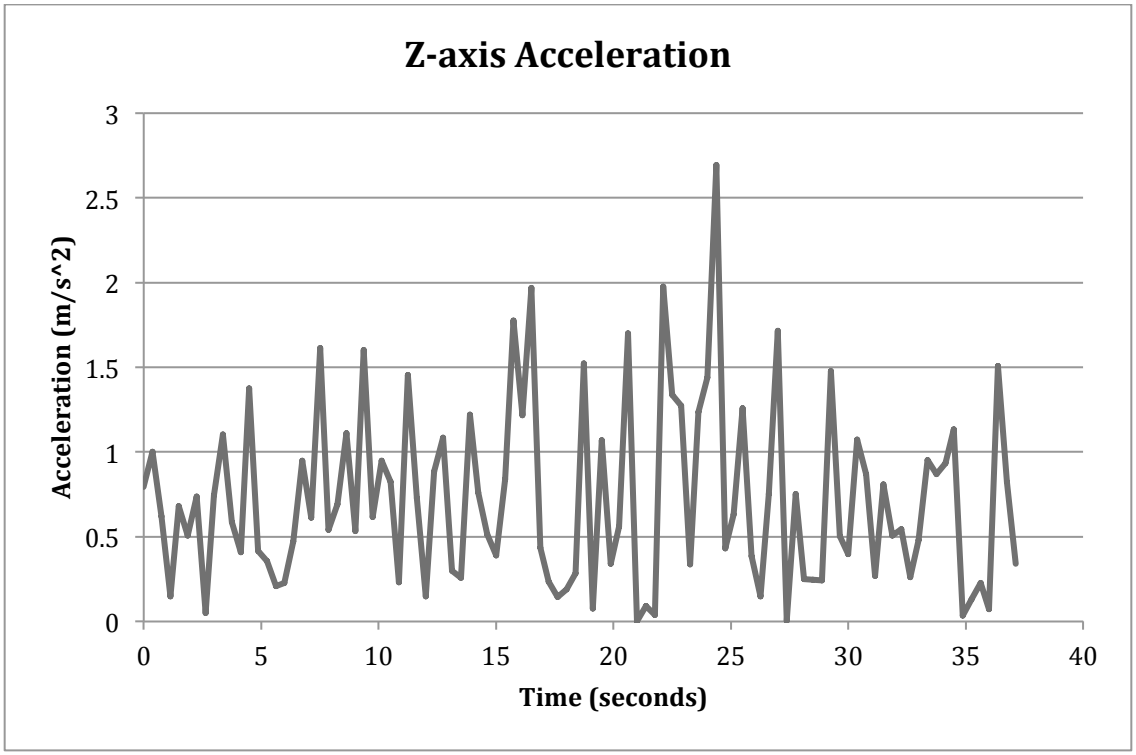


Figure 11: Z-axis acceleration graph

All three graphs display similar information. The peaks of the acceleration spikes increased as the phone was moved forward faster and begins to decrease as the phone was moved forward slower. The graph drops at points that the acceleration was not increasing; therefore there was no acceleration. This displays promising results for the acceleration tracking.

The velocity information for each axis was taken at the same type as the acceleration, so it is a representation of the velocity of the acceleration curves shown. The graphs for the velocity information can be seen in Figures 12, 13, and 14 below.

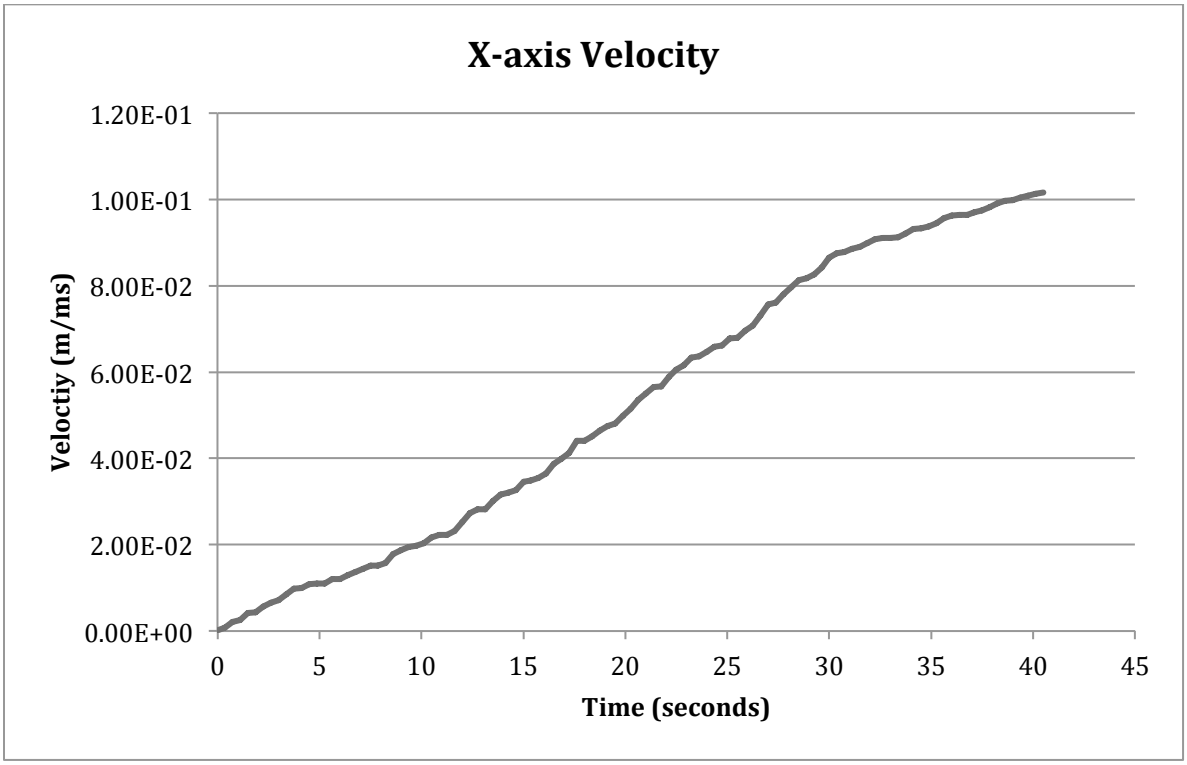


Figure 12: X-axis velocity graph

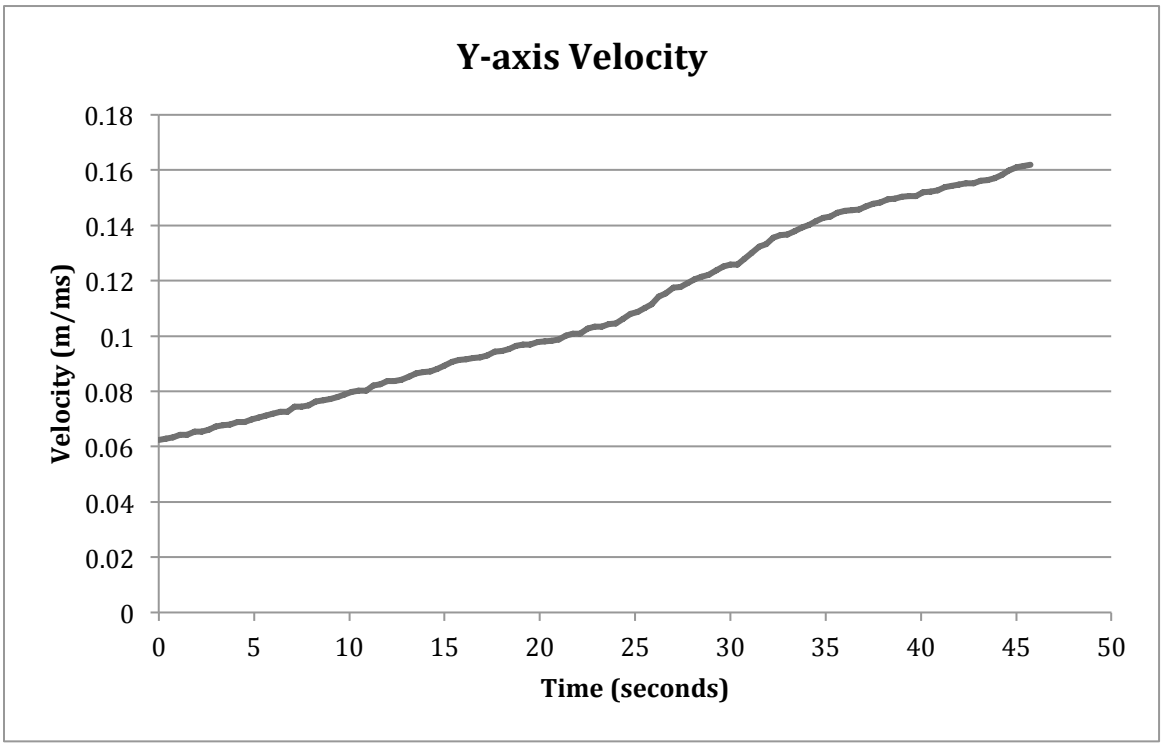


Figure 13: Y-axis velocity graph

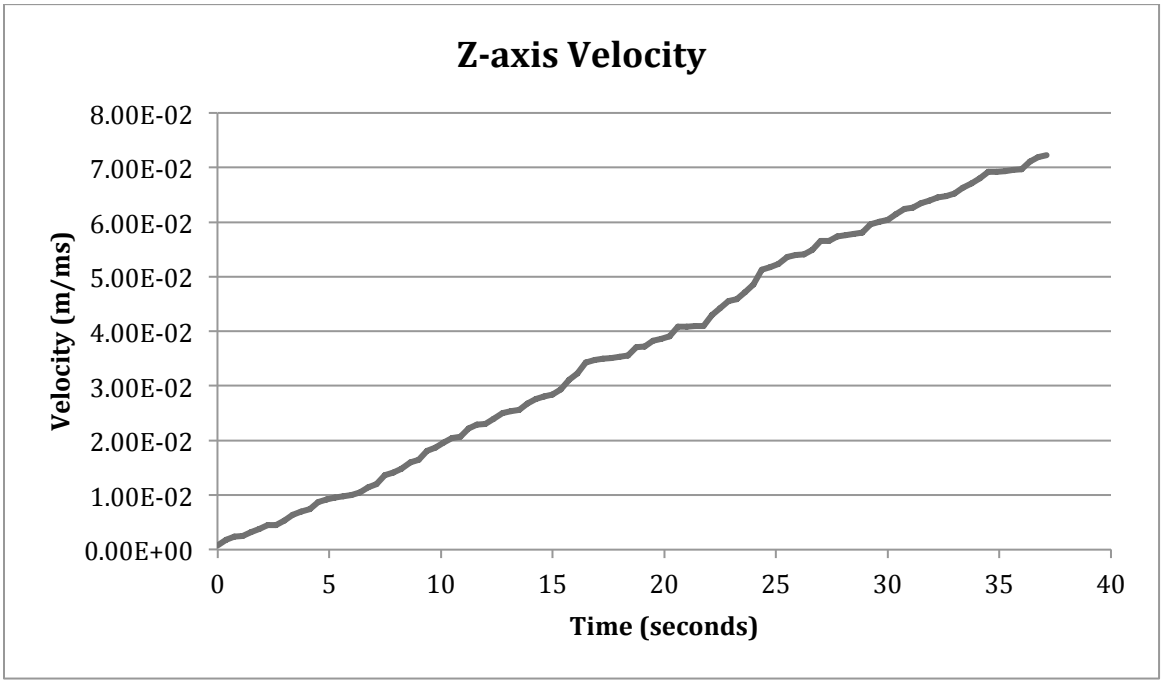


Figure 12: Z-axis velocity graph

Each of these three graphs provides a similar result. The velocity is seen as constantly increasing despite the speed of the phone being increased and then decreased. This error is most likely the result of equation 3.6 used to calculate the velocity, which adds the previous velocity to the new acceleration multiplied by 1 millisecond. This equation is based on the standard velocity equation where the previous velocity is the initial velocity. Knowing this, it is clear that the new velocity is being added to each previous velocity and will always be larger than the previous. This creates the consistent incline of the three velocity curves shown above. The error in this equation most likely stems from the sensitivity of the device and the gyroscope. Nearly imperceptible human movements can be tracked by the gyroscope. This means that the acceleration for any axis is rarely truly zero as can be seen in the acceleration graphs. The linear accelerometer function also appears to not provide negative acceleration values. This means that a value, even if small, is constantly being added to the previous velocity value, so it will never show a decrease in velocity. This may be remedied by the used of a filter within the program code that only allows significant acceleration values to pass to calculate the velocity.

5. Conclusions

5.1 Future Program Additions

In order to bring this program closer to helping women with early detection of postpartum depression, some program improvements and corrections are necessary. The most important edits to the program are the corrections for the current design errors. These errors include inaccurate velocity values, issues with the application running the motion tracking without user input, and the application not continuing to track the user when the application closes. These are all program errors that need to be researched and edited. The inaccurate velocity values when collecting the motion tracking data can most likely be fixed using an if statement that only passes acceleration values above a certain level. This would filter out the noise that is caused by imperceptible movements. When the acceleration is below this predetermined value, a value of zero would pass to the velocity equation. The application also has two issues with motion tracking starting without proper user input. The motion tracking currently starts as soon as the application is opened and when the application screen is rotated when the phone is turned to its side. This is a bit of a privacy issue, because the user should know and choose when they are being tracked. This could potentially be corrected by placing the code that collects and prints the tracking data within an if statement that only runs if the *onResume* function has been called. The final issue is that the application stops tracking when the user closes it. It is essential that the volume and motion data be taken constantly in order to get an accurate and consistent sample. This would involve studying the *onRestart*, *onStop*, *onResume*, and *onStop* functions as well as the button functions. The "Start Button" needs to hold its action even when the application is closed.

Two improvements should also be made to the program to allow for better use. The first is to replace the current use of the linear accelerometer to the use of the accelerometer minus gravity. Since the linear accelerometer has been moved to a hardware sensor in later operating systems, older devices cannot access it. The linear accelerometer is already defined as the accelerometer minus gravity in all vectors. By using the accelerometer and removing gravity, the application becomes more accessible to all users. The second improvement is to be made when a server is created to collect and analyze the data streamed by the phone. This would be to save the data in a large array that would empty into the server when a wireless Internet connection is made by the device. The basic start to this code has already been placed in the current code, but has been commented out with comments above it to describe its potential use. The code is written as such, *public static List<String> MaxValue = new ArrayList()*. This sets up the data to be placed within an array. This addition will eventually be essential for data analysis and early diagnosis.

5.2 Summary

This thesis provided an example of a program that could potentially help with early detection of postpartum depression. This program was made to extrapolate useful data from the user's day-to-day lives that are indicative of postpartum depression, such as motion and environment noise. The program was developed as an application for Android-powered devices to allow easy access to users.

Two sensors were used within the mobile device. The first was the microphone with a maximum amplitude filter that allows for easy tracking of volume information around the user. The implementation of this sensor proved to track motion with relative

accuracy. The second sensor was the gyroscope with an emphasis on the linear accelerometer. This tracks the acceleration on the x-axis, y-axis, and z-axis of the person when carrying the device. The acceleration data showed accuracy in its results, but the velocity equation implemented for additional data collection proved erroneous.

Overall, the application program proved to be a solid base for tracking these two aspects of daily life. It has proven that the sensors for tracking of this nature exist and data collection is available. With the revisions recommended for the program in place, it stands that proper data tracking can be achieved for analysis.

Citations

[1] “Depression and Bipolar Support Alliance.” [Online]. Available: http://www.dbsalliance.org/site/PageServer?pagename=education_statistics_depression. [Accessed: 18-Oct-2014].

[2] “National Public Radio.” [Online]. Available: http://www.npr.org/blogs/health/2013/03/13/174214166/postpartum-depression-affects-1-in-7-women. [Accessed: 18-Oct-2014].

[3] “Mayo Clinic.” [Online]. Available: http://www.mayoclinic.org/diseases-conditions/postpartum-depression/basics/symptoms/con-20029130. [Accessed: 18-Oct-2014].

[4] “Android Developer.” [Online]. Available: http://developer.android.com/guide/topics/sensors/sensors_overview.html. [Accessed: 18-Oct-2014].

[5] “Android.” [Online]. Available: https://www.android.com/intl/en_us/history/. [Accessed: 13-February-2015].

[6] “Android Developer.” [Online]. Available: http://developer.android.com/reference/android/media/MediaRecorder.html. [Accessed: 15-March-2015].

[7] “Android Developer.” [Online]. Available: http://developer.android.com/reference/android/media/MediaRecorder.html. [Accessed: 25-March-2015].

[8] “GSM Arena.” [Online]. Available: www.gsmarena.com/oneplus_one-6327.php. [Accessed: 10-April-2015].

[9] “Android Developer.” [Online]. Available: http://developer.android.com/guide/topics/sensors/sensors_motion.html. [Accessed: 10-Mar-2015].

[10] “The American Congress of Obstetricians and Gynecologists.” [Online]. Available: http://www.acog.org/Resources-And-Publications/Committee-Opinions/Committee-on-Obstetric-Practice/Screening-for-Depression-During-and-After-Pregnancy. [Accessed: 19-Oct-2014].

Appendix

A. Android Manifest.xml Code

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.abbywise.honorsthesisrev0" >

    <!--Gives permission to save information-->
    <permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

    <!--Gives permission to record audio-->
    <permission android:name="android.permission.RECORD_AUDIO" />
    <uses-permission android:name="android.permission.RECORD_AUDIO" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category
android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

B. Android activity_main.xml Code

```
<RelativeLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
    android:layout_height="match_parent"
android:paddingLeft="@dimen/activity_horizontal_margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity_vertical_margin"
    android:paddingBottom="@dimen/activity_vertical_margin"
tools:context=".MainActivity"
    android:clickable="true">


    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:text="Hit start button below to begin tracking."
        android:id="@+id/textView"
        android:layout_alignParentTop="true"
        android:layout_alignParentRight="true"
        android:layout_alignParentEnd="true" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Start Tracking"
        android:id="@+id/button"
        android:clickable="true"
        android:layout_below="@+id/textView"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="52dp" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Stop Tracking"
        android:id="@+id/button2"
        android:layout_below="@+id/textView2"
        android:layout_alignLeft="@+id/button"
        android:layout_alignStart="@+id/button"
        android:layout_marginTop="58dp" />

    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textAppearance="?android:attr/textAppearanceLarge"
        android:text="Hit stop button below to stop tracking."
        android:id="@+id/textView2"
        android:layout_centerVertical="true"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true" />

</RelativeLayout>
```

C. Android MainActivity.java Code

```
package com.example.abbywise.honorsthesisrev0;

// Project libraries used in this program
// Project libraries that are grayed out are not currently used by
// this program, but may be useful for future additions to the code.

import android.support.v7.app.ActionBarActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.Toast;
import android.util.Log;
import android.media.MediaRecorder;
import android.hardware.SensorManager;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.app.Activity;
import android.content.Context;
import android.widget.TextView;
import android.widget.LinearLayout;
import android.os.Bundle;
import android.os.Environment;
import android.view.ViewGroup;
import android.widget.CheckBox;
import android.content.Intent;
import android.widget.EditText;
import android.media.AudioFormat;
import android.media.AudioRecord;

import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

import static android.util.FloatMath.sqrt;


public class MainActivity extends Activity implements
SensorEventListener{

    // Initialize variables for programming

    // Initialize LOG_TAG to call in logcat to track data collected
    // The live streaming data from both the volume sensing and motion
detecting
    // can be seen in the logcat
    private static final String LOG_TAG = "DataPointCollection";
    // Initializes the MediaRecorder as mRecorder for volume tracking
    private MediaRecorder mRecorder = null;
    // Creates list of maximum amplitude values collected
```

```
    // Would be used to send data to server
    public static List<String> MaxValue = new ArrayList();
    // Initializes the SensorManager and Sensor for motion sensing
    private SensorManager mSensorManager;
    private Sensor mAccelerometer;
    // Initializes variables for each axis
    private float x;
    private float y;
    private float z;
    // Initializes acceleration values
    private float xDelta = 0;
    private float yDelta = 0;
    private float zDelta = 0;
    // Initializes velocity values
    private double vX = 0;
    private double vY = 0;
    private double vZ = 0;
    // Initializes absolute value acceleration value
    private float a = 0;
    // Creates list of velocity values collected
    // Would be used to send data to a server
    public static List<String> MotionDataX = new ArrayList();
    public static List<String> MotionDataY = new ArrayList();
    public static List<String> MotionDataZ = new ArrayList();


    // Start recording function
    protected void onRestart() {
        super.onRestart();
        if (mRecorder == null) {
            mRecorder = new MediaRecorder();
            mRecorder.setAudioSource(MediaRecorder.AudioSource.MIC);

mRecorder.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);

mRecorder.setAudioEncoder(MediaRecorder.AudioEncoder.AMR_NB);
            mRecorder.setOutputFile("/dev/null");
            try {
                mRecorder.prepare();
            } catch (IOException e) {
                Log.e(LOG_TAG, "prepare() failed");
            }
            mRecorder.start();

            // Calls getAmplitude to use getMaxAmplitude filter
            getAmplitude();
        }
        // Shows start record function has been called in logcat
        Log.e(LOG_TAG, "Start Record Called");
    }

    // Stop recording function
    protected void onStop(){
        super.onStop();
        if (mRecorder != null){
            mRecorder.stop();
            mRecorder.reset();
```

```
            mRecorder = null;
        }
        // Shows stop record function has been called in logcat
        Log.e(LOG_TAG, "Stop Record Called");
    }

    // Function to implement getMaxAmplitude filter
    private double getAmplitude() {
        if (mRecorder != null) {
            // Shows getAmplitude function has been called in logcat
            Log.e(LOG_TAG, "getAmplitude Function Called");
            // Calls getMaxAmplitude function
            // Equation is used to get the maximum amplitude in
decibels
            double max = 20*Math.log(mRecorder.getMaxAmplitude()/2700);
            return (max);
        } else {

            return 0;
        }
    }

    // Function to collect maximum amplitude data
    public double getAmplitudeMax() {
        double amp = getAmplitude();
        // Converts double amp to string max
        // This MaxValue would be used to create an array to send to
server
        String max = String.valueOf(amp);
        MaxValue.add(max);
        // Shows value of maximum amplitude in logcat
        Log.e(LOG_TAG, "Max Amplitude " +amp);
        return amp;
    }

    // Registers the linear accelerometer for listening the events
    protected void onResume() {
        super.onResume();

        mSensorManager.registerListener(this, mAccelerometer,
SensorManager.SENSOR_DELAY_NORMAL);
        // Shows onResume function has been called in logcat
        Log.e(LOG_TAG, "Motion Resume Called");
    }

    // Unregisters the linear accelerometer for stop listening the
events
    protected void onPause() {
        super.onPause();

        mSensorManager.unregisterListener(this);
        // Shows onPause function has been called in logcat
        Log.e(LOG_TAG, "Motion Pause Called");
    }


    @Override
```

```
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        //initializeViews();

        // Starts linear accelerometer
        mSensorManager = (SensorManager)
getSystemService(Context.SENSOR_SERVICE);
        if
(mSensorManager.getDefaultSensor(Sensor.TYPE_LINEAR_ACCELERATION)!=null
)
        {
            // Shows linear accelerometer has been called in logcat
            Log.e(LOG_TAG, "Linear Accelerometer Called");

            mAccelerometer =
mSensorManager.getDefaultSensor(Sensor.TYPE_LINEAR_ACCELERATION);
            mSensorManager.registerListener(this, mAccelerometer,
SensorManager.SENSOR_DELAY_NORMAL);
        }
        else {
            // Shows linear accelerometer failed to be called in logcat
            Log.e(LOG_TAG, "Linear Accelerometer Failed");
        }
    }


    @Override
    public boolean onCreateOptionsMenu(Menu menu) {

        // Inflate the menu; this adds items to the action bar if it is
present.
        getMenuInflater().inflate(R.menu.menu_main, menu);


        // Reaction to Start button (button) pushed
            final Button button = (Button) findViewById(R.id.button);
            button.setOnClickListener(new

            OnClickListener() {

                @Override public void onClick (View view){
                    // Calls start recording function on Start button
(button) push
                    onRestart();
                    // Calls start linear accelerometer function on
Start button (button) push
                    onResume();

                    // Notifies the user tracking has begun
                    Toast.makeText(MainActivity.this, "Tracking",
Toast.LENGTH_SHORT).show();
                }
            }
    );
```

```
        // Reaction to Stop button (button2) pushed
        Button button2 = (Button) findViewById(R.id.button2);
        button2.setOnClickListener(new OnClickListener() {

            @Override
            public void onClick(View view) {
                // Calls stop recording function on Stop button
(button2) push
                onStop();
                // Calls stop linear accelerometer function on Stop
button (button2) push
                onPause();

                // Notifies the user tracking has stopped
                Toast.makeText(MainActivity.this, "Tracking Stopped",
Toast.LENGTH_SHORT).show();
            }

        });

        return true;
    }



    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        // Handle action bar item clicks here. The action bar will
        // automatically handle clicks on the Home/Up button, so long
        // as you specify a parent activity in AndroidManifest.xml.
        int id = item.getItemId();

        //noinspection SimplifiableIfStatement
        if (id == R.id.action_settings) {
            return true;
        }

        return super.onOptionsItemSelected(item);
    }

    @Override
    public void onSensorChanged(SensorEvent event) {

        // Get the change of the x,y,z values of the accelerometer
        xDelta = Math.abs(x - event.values[0]);
        vX = vX+xDelta * 0.001;
        yDelta = Math.abs(y - event.values[1]);
        vY = vY+yDelta * 0.001;
        zDelta = Math.abs(z - event.values[2]);
        vZ = vZ+zDelta * 0.001;

        // Gets the absolute value of the acceleration
        a = sqrt(xDelta * xDelta + yDelta * yDelta + zDelta * zDelta);
        String acc = String.valueOf(a);
        // Shows absolute value of acceleration in logcat
        Log.e(LOG_TAG, "a "+ acc);
```

```
        // Displays constant stream of x-axis acceleration data
        String xD = String.valueOf(xDelta);
        Log.e(LOG_TAG, "x " + xD);
        // This MotionDataX would be used to create an array to send to
server
        String xx = String.valueOf(vX);
        MotionDataX.add(xx);
        Log.e(LOG_TAG, "vX " + xx);
        // Displays constant stream of y-axis acceleration data
        String yD = String.valueOf(yDelta);
        Log.e(LOG_TAG,"y " + yD);
        // This MotionDataY would be used to create an array to send to
server
        String yy = String.valueOf(vY);
        MotionDataY.add(yy);
        Log.e(LOG_TAG, "vY " + yy);
        // Displays constant stream of z-axis acceleration data
        String zD = String.valueOf(zDelta);
        Log.e(LOG_TAG,"z " + zD);
        // This MotionDataZ would be used to create an array to send to
server
        String zz = String.valueOf(vZ);
        MotionDataZ.add(zz);
        Log.e(LOG_TAG, "vZ " + zz);

        // Constantly calls getAmplitude function
        double amp = getAmplitude();
        //String max = String.valueOf(amp);
        //MaxValue.add(max);
        // Shows constant stream of maximum amplitude values in logcat
        Log.e(LOG_TAG, "Max Amplitude " +amp);
    }

    @Override
    public void onAccuracyChanged(Sensor sensor, int i) {

    }
}
```