A SMARTPHONE APPLICATION TO CHARACTERIZE ELECTROMAGNETIC FIELDS IN ELECTRON MICROSCOPY

A Thesis

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Master of Science

by

Arjun Shivanand Kannan

August 2013

ABSTRACT

Electron Microscopes are sensitive to noise of all kinds : Alternating Current (AC) Magnetic Fields, acoustic disturbances, temperature fluctuations and/or disturbances caused by rapid air flow in clean rooms. Therefore, existing and potential room locations need quick and reliable characterization methods for all these parameters. AC noise, in particular, is characterized today by either specialized instruments and handheld meters dedicated to the purpose. Our aim is to find a suitable substitute for these instruments by making an application that uses the in-built magnetometer in smartphones to measure electromagnetic fields.

Our research is motivated by the fact that smartphones with considerable computing ability have become ubiquitous enough that they become a viable alternative to some dedicated characterization instruments. Furthermore, a method to characterize these fields can also extend to other applications, such as finding and mapping electrical faults, and touchless 3-D interaction.

We will discuss the development process of an iPhone application for quick analysis and recording of electromagnetic noise in equipment rooms for the installation of sensitive equipment. This will cover the suitability of the hardware to the purpose, the sensitivity of the hardware to our measurement criteria, our methods for data analysis and the associated software visualization techniques, and comparison to existing methods. Finally, we cover the limitations and potential applications of the software developed.

BIOGRAPHICAL SKETCH

Arjun Shivanand Kannan earned his Bachelor of Engineering degree in Mechanical Engineering Engineering from PSG College of Technology, Anna University, India in 2011. Subsequent to that, he joined the Master of Science program in Applied Physics at Cornell University.

Arjun has been the recipient of numerous honors and awards including multiple Outstanding Student Awards from the Association of Mechanical Engineers, PSG College of Technology. He was also the recipient of the Award for Academic Excellence from the American Society of Mechanical Engineers (student chapter) for the graduating batch in 2011.

While pursuing his undergraduate degree, Arjun first worked as an intern for Sanees Alloys Pvt. Ltd., Coimbatore, India. This internship included a very successful project that Arjun spun off into his own company, Shree Sarabha Bronzes, Coimbatore, India. In addition, Arjun interned at Ivy Action Tank LLC, Ithaca, during his graduate study, where he was part of an 8-member team that gave rise to Yorango.com, winning numerous startup competition awards. After completion of his graduate degree, Arjun will join BlackRock Inc. as a full-time employee.

Arjun's thesis, entitled "A Smartphone Application to characterize Electromagnetic Fields in Electron Microscopy" was supervised by Dr. David Muller.

I would like to dedicate this thesis to my family, my friends and to my supervisor, Professor David Muller.

ACKNOWLEDGEMENTS

It would not have been possible to write this thesis without the help and support of the kind people around me, to only some of whom it is possible to give particular mention here.

Above all, I would like to thank my parents and my dear friends for their personal support and great patience at all times.They have given me their unequivocal support throughout, as always, for which my mere expression of thanks likewise does not suffice.

This thesis would not have been possible without the help, support and patience of my principal supervisor, Prof. David Muller. His good advice, support and friendship have been invaluable on both an academic and a personal level, for which I am extremely grateful.I would like to acknowledge the financial, academic and technical support of Cornell University and the Muller group. The library facilities and computer facilities of the University, have been indispensable. I also thank the School of Applied & Engineering Physics for their support and assistance, and to Prof. Manfred Lindau, my second committee member, for his guidance and support through the course.

I am most grateful to Robert Hovden, Paul Cueva, Pinshane Huang, Megan Holtz, Barnaby Levin and Quingyun Mao of the Muller Group for providing me with invaluable support and for many interesting discussions on a variety of topics.

Last, but by no means least, I thank my friends, both in Ithaca and elsewhere for their support and encouragement.

For any errors or inadequacies that may remain in this work, of course, the responsibility is entirely my own

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION & MOTIVATION

With the growth of electron microscopes in size and sensitivity, there has been a concurrent growth in the requirements for the laboratories that house them [1]. Though there is considerable expertise and knowledge in the construction industry in building quiet rooms and many microscope facilities have taken advantage of this knowledge, the impact of the residual noise sources such as AC fields is varying and requires rapid characterization at any given point in time. This need is currently served by dedicated handheld meters, which, though fairly widespread, may still not be the most convenient or most ubiquitous devices available on hand, so our motivation is to find a method that will allow a more convenient means of measuring magnetic fields using widespread technology. The motivation for our research is to find a convenient, cost-effective and easy-to-use alternative to the dedicated handheld field meters that are used to measure noise in microscope rooms. Electron Microscopes sense noise of all kinds [2] which means they need quick and reliable characterization methods.

Smartphones today possess considerable computing ability, and are ubiquitous enough that they become a plausible alternative to some dedicated scientific instruments. The focus of our research is to adapt iPhones as utilizable AC Field Meters using the Hall probes built into them for their magnetometers.

We attempt to develop such a method using an iPhone, which can then also be extended to other applications (A screenshot of our current application is

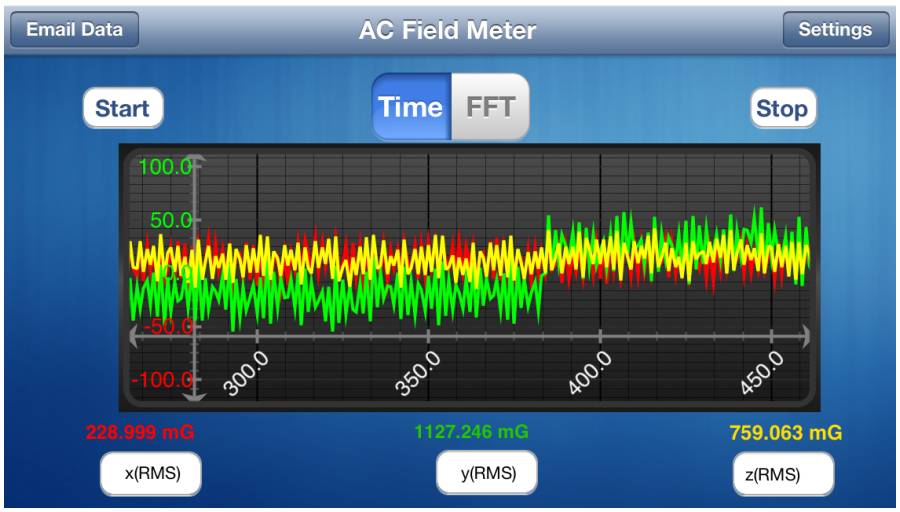

Figure 1.1: A snapshot of our iPhone application in operation. Each color of line corresponds to readings from one of the three axes - x, y and z - from the magnetometer.

shown in Figure 1.1).

We will cover the science of the signal processing behind the application in Chapter 2, where we will also discuss the framework of software that makes our application possible. Having established our motivation and scientific background, we shall cover our experimental process and the building of the application in Chapter 3, and finally discuss the outcomes of our process, the performance of the application and our results in Chapter 4, concluding with our summary in Chapter 5.

CHAPTER 2

THEORETICAL BACKGROUND

In this chapter, we shall cover some of the fundamentals behind the scientific methods we use in the application.

2.1 Why is magnetic noise important in electron microscopy ?

Electron microscopes are sensitive to electromagnetic noise. Electromagnetic interference can cause beam deflections in both the scanning system and the spectrometer [2]. The most common sources of AC electromagnetic interference are unbalanced electrical loads. These are illustrated by Ampere's law. Ampere's Law states that for any closed loop path, the sum of the length elements times the magnetic field in the direction of the length element is equal to the permeability times the electric current enclosed in the loop.

$$\oint_C B \cdot \mathrm{d}l = \mu_0 I \tag{2.1}$$

$$B \times 2\pi r = \mu_0 I \tag{2.2}$$

$$B = \left(\frac{\mu_0}{2\pi r}\right) I \tag{2.3}$$

Here, B is the electromagnetic field, I is the current in the conductor and r is the distance from the conductor. The sensitivity of electron microscopes is illustrated by the following ground rules : A 0.3 mG r.m.s. field can be detected in a 0.3-nm Scanning Transmission Electron Microscope (STEM) image. Less than 0.2 mG r.m.s is needed for clean 0.2-nm STEM images. A single metre of separation from a straight wire carrying just 500 μA of current, causes a mag-

netic field of about 1 mG. This is enough to degrade a 0.3-nm resolution STEM performance.

Ideally, this should not mean that flicking a switch in the room should degrade all STEM images as the fields from the supply and return currents should cancel each other. It is only when some of the return current finds another path to ground (for instance, through a wiring mistake in the circuit) that a net field will be generated. This is the case for a common 2-phase wiring mistake where the neutral and the ground lines are accidentally bonded at conduit junctions or at the load, instead of only at the source. These mistakes are easy to fix, but can be difficult to isolate. The same thing can occur as an analogous problem for three-phase electrical supplies.

2.1.1 The need for a handheld field meter

There are thousands of Scanning Electron Microscopes (SEMs) sold each year, and not each and every one of them is set up in a perfectly electromagnetically shielded environment. In addition, there may simply be factors that had not come under the room constructor's purview at the time of construction, and are influencing the electromagnetic fields in the room at the time of measurement [1]. As such, the microscopist can always find use for an instrument to characterize the field in the room at any given time.

The current de facto instruments used to characterize electromagnetic fields are handheld low frequency gauss meters with 30Hz - 300 Hz bandwidth and 0.1 mG r.m.s. sensitivity. These meters are widespread in use. However, as is the case with cameras - The best camera to capture any given moment is the

one you have on your person - so is the case with field meters. The number of users of field meters, while large, is still lower than the number of users of smartphones within the scientific community. This makes smartphones a viable option to turn into instruments of use to us. Although it is not designed to detect AC fields, with a little bit of clever signal processing and patience, we can turn a smartphone into a fairly capable field meter.

2.2 Hall probes

The magnetometer in an iPhone is a Hall probe [3]. One of the primary reasons for our selection of the iPhone as the first instrument to test our theory was that the hardware is fairly standardized across various models, and thus less divergent in results for the same environment. The working methodology of Hall probes and the specifications of the models in the iPhones are discussed forthwith.

2.2.1 Theoretical background

Hall effect sensors are transducers that vary their output voltage in response to a magnetic field. They are used for proximity switching, positioning, speed detection, and current sensing applications. Electricity carried through a conductor will produce a magnetic field that varies with current, and a Hall sensor can be used to measure the current without interrupting the circuit [4]. Typically, the sensor is integrated with a wound core or permanent magnet that surrounds the conductor to be measured. In its simplest form, the sensor operates as an analog

transducer, directly returning a voltage. With a known magnetic field, its distance from the Hall plate can be determined. Using groups of sensors, the relative position of the magnet can be deduced. The Hall effect is the production of a voltage difference (the Hall voltage) across an electrical conductor, transverse to an electric current in the conductor and a magnetic field perpendicular to the current, and it is seen when a conductor is passed through a uniform magnetic field [5].

2.2.2 Principle and Construction

When electrons (or holes) move in a conducting plate that is immersed in a magnetic field, they experience a Lorentz force.

$$F_{Lorentz} = q(E + v \times B) \tag{2.4}$$

Here, q is the charge, E is the electric field, v is the velocity, and B is the magnetic field. The second term is transverse to velocity and to the magnetic field. Consequently, if sensing electrodes are placed across the transverse dimension of the plate, a voltage, called the Hall voltage, will appear. Hall Effect Sensors consist basically of a thin piece of rectangular p-type semiconductor material such as gallium arsenide (GaAs), indium antimonide (InSb) or indium arsenide (InAs) passing a continuous current through itself [6]. A simple cartoon of the Hall Effect Sensor is shown in Figure 2.2.

When the device is placed within a magnetic field, the magnetic flux lines exert a force on the which deflects the charge carriers, electrons and holes, to either side of the semiconductor slab. This movement of charge carriers is a result of the magnetic force they experience passing through the semiconductor

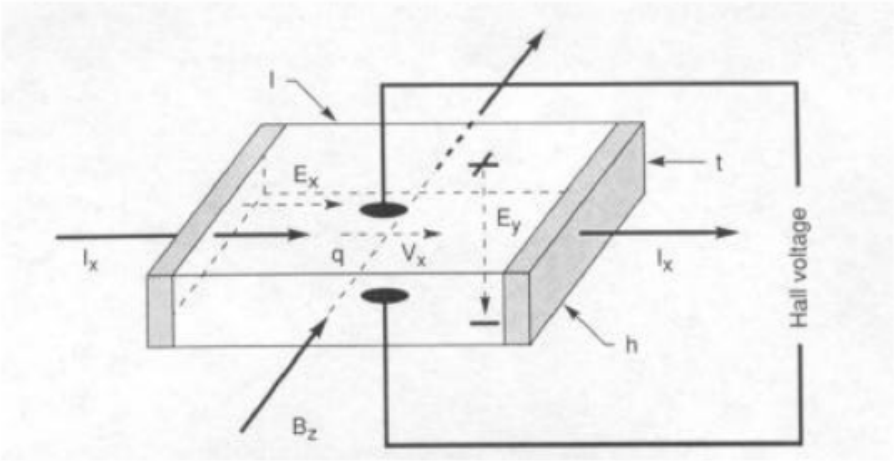

Figure 2.1: An illustration of the Hall effect. Image source : [6]

material. As these electrons and holes move side wards a potential difference is produced between the two sides of the semiconductor material by the build-up of these charge carriers. Then the movement of electrons through the semiconductor material depends on the strength of the external magnetic field which is at right angles to it. This effect is observed to be greater in a flat rectangular shaped material.

2.2.3 Applications of Hall sensors to sense magnetic heading in smartphones

Electronic compasses determine their magnetic heading by measuring the earths horizontal magnetic field. Maximum heading accuracy is achieved by keeping the two-axis modules approximately level. For applications where compass modules will not be level, a three-axis, tilt compensated compassing method is used. These three-axis compass modules perform an electronic gim-

baling function by adding the third magnetic axis and a tilt sensor for a gravity vector reference. Tilt sensors are made of either fluidic sensors or MEMS accelerometers. The quality of the tilt measurement contributes to the precision of the compass outputs.

2.2.4 Magnetometer hardware in the iPhone

The magnetometer is a magnetoresistive permalloy sensor found in all current models of the iPhone and iPad[3]. The iPhone 3GS uses the AN-203 integrated circuit produced by Honeywell, while the newer iPhones and iPads make use of the AKM8975 produced by AKM Semiconductor. The sensor is located towards the top right hand corner of the device, and measures fields within a 2 gauss (200 microtesla) range, and is sensitive to magnetic fields of less than 100 microgauss (0.01 microtesla).

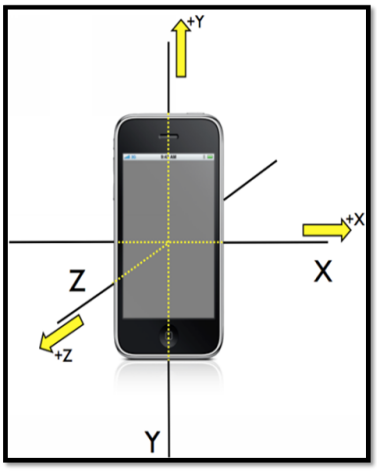

Figure 2.2: A simple schematic of the axes on the iPhone (Source : [3])

The magnetometer measures the strength of the magnetic field surrounding the device. In the absence of any strong local fields, these measurements will be

of the ambient magnetic field of the Earth, allowing the device to determine its heading with respect to the geomagnetic North Pole and act as a digital compass. The geomagnetic heading and true heading relative to the geographical North Pole can vary widely, by several tens of degrees depending on your location.

Further specifications for the iPhone magnetometer are detailed in the AKM8975 specification guide. Our main interest lies in the fact that the sampling rate is 128 samples per second, which is just above what we need to reliably and unambiguously detect and characterize AC field signals (which are at 60Hz in the United States and 50Hz elsewhere in the world).

2.2.5 Repurposing the Magnetometer as an AC Field meter

The basic premise of our iPhone application is that we are able to use the magnetometer to continuously take magnetic readings. We subsequently perform various signal analyses on the data thus generated to characterize and analyze the magnetic field strength in the area, and this is what we shall discuss further in this thesis.

2.3 Sampling from Fourier Transforms, Nyquist limits and Windowing

Our data analysis method extensively uses Fast Fourier Transforms [7][8]. The Fast Fourier Transform (FFT) is an algorithm for transforming data from the

time domain to the frequency domain as follows :

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i2\pi k \frac{n}{N}} \qquad k = 0, \ldots, N-1 \tag{2.5}$$

Here, X_k is the discrete Fourier transform and N is the number of data points with n being the point under consideration in the current calculation. This separates out the values (magnitude) at each frequency. Since this is exactly what we want our spectrum analyzer to do, it would seem straightforward to implement a Dynamic Signal Analyzer based on the FFT. Moreover, signal averaging in the frequency domain is also extremely effective. By averaging a set of identical measurements, the signal-to-noise ratio (SNR) will be improved (increased), ideally in proportion to the square root of the number of measurements.

The question of long FFTs (a large number of data points) versus short FFTs (fewer data points) also comes into play with this method of signal analysis. While long FFTs offer better frequency resolution (the more the number of data points within a specified window, the more precisely one can tell frequencies apart), we achieve a much better SNR by summing several short FFTs in the same time. If there are N FFTs summed over, then the SNR improves by a factor of $\sqrt{N}$ [9]. If we were able to measure for indefinite time periods, having a long FFT summed repeatedly would give us excellent SNR as well as resolution, but within a finite time-frame, summing over several short FFTs for a better SNR is the practical option for our application.

However, we will see that there are many factors which complicate this seemingly straightforward task. Note that we cannot now transform to the frequency domain in a continuous manner, but instead must sample and digitize the time domain input. This means that our algorithm transforms digitized samples from the time domain to samples in the frequency domain. Because

we have sampled, we no longer have an exact representation in either domain. However, a sampled representation can be as close to ideal as we desire by placing our samples closer together. There are also various limits inherent in sampling that must be dealt with in analyzing an FFT. One of the primary limitations posed on our data analysis from the iPhone readings is the Nyquist limit, a fundamental parameter of signal analysis and sampling theory, which limits the maximum frequency of field we can distinguish unambiguously using our hardware.

2.3.1 Nyquist limits - A quick recap of sampling theory

The signals we use in the real world, such as our voices, are called ”analog” signals. To process these signals in computers, we need to convert the signals to digital form. While an analog signal is continuous in both time and amplitude, a digital signal is discrete in both time and amplitude. In order to convert a signal from continuous time to discrete time, a process called sampling is used. The value of the signal is measured at certain intervals in time. Each measurement is referred to as a sample.

When the continuous analog signal is sampled at a frequency F, the resulting discrete signal has more frequency components than did the analog signal. In the discrete frequency response, the components of the original analog signal in each frequency are seen at their original position, and are also seen centered around +/- F, around +/- $2F$, and subsequent harmonic multiples.

If the signal contains high frequency components, we will need to sample at a higher rate to avoid losing information that is in the signal. In general, to

preserve the full information in the signal, it is necessary to sample at twice the maximum frequency of the signal. This is known as the Nyquist rate. The Sampling Theorem states that a signal can be exactly reproduced if it is sampled at a frequency $f_{Nyquist}$, where $f_{Nyquist}$ is greater than twice the maximum frequency in the signal f_{Signal}.

$$f_{Nyquist} \geq 2 \times f_{Signal} \tag{2.6}$$

Aliasing

What happens if we sample the signal at a frequency that is lower that the Nyquist rate ? When the signal is converted back into a continuous time signal, it will exhibit a phenomenon called aliasing. Aliasing is the presence of unwanted components in the reconstructed signal. These components were not present when the original signal was sampled. In addition, some of the frequencies in the original signal may be lost in the reconstructed signal. Aliasing occurs because signal frequencies can overlap if the sampling frequency is too low. Frequencies "fold" around half the sampling frequency - which is why this frequency is often referred to as the folding frequency.

2.3.2 Data windowing

There is a property of the Fast Fourier Transform (FFT) which affects its use in frequency domain analysis. We recall that the FFT computes the frequency spectrum from a block of samples of the input called a time record. In addition, the FFT algorithm is based upon the assumption that this time record is repeated throughout time. So, what happens when we are measuring a continuous signal

like a sine wave? If the time record contains an integral number of cycles of the input sine wave, then the input waveform is said to be periodic in the time record. However, the FFT algorithm is computed on the basis of the highly distorted waveform. We know that the actual sine wave input has a frequency spectrum of single line. This is, however, not always the case in the FFT of a continuous signal. It shows a smearing of energy throughout the frequency domains, a phenomenon known as leakage. We will see energy leak out of one resolution line of the FFT into all the other lines.

It is important to realize that leakage is due to the fact that we have taken a finite time record. For a sine wave to have a single line spectrum, it must exist for all time, from minus infinity to plus infinity. If we were to have an infinite time record, the FFT would compute the correct single line spectrum exactly. However, since we are not willing to wait forever to measure its spectrum, we only look at a finite time record of the sine wave. This can cause leakage if the continuous input is not periodic in the time record. The problem of leakage is severe enough to entirely mask small signals close to any periodic input. As such, the FFT would not be a very useful spectrum analyzer. The solution to this problem is known as windowing.

If the FFT could be made to ignore the ends and concentrate on the middle of the time record, we would expect to get much closer to the correct single line spectrum in the frequency domain. If we multiply the time record by a function that is zero at the ends of the time record and large in the middle, we would concentrate the FFT on the middle of the time record. Such functions are called window functions because they force us to look at data through a narrow window. Typically, we get vast improvement by windowing data that

is not periodic in the time record. However, it is important to realize that we have tampered with the input data and cannot expect perfect results. Also, the windowed data does not have as narrow a spectrum as an unwindowed function which is periodic in the time record. Windowing, however, is probably the best way in which we can get close to an ideal match for the signal in the frequency domain.

2.4 Apple's iOS Framework for the iPhone, and its use in development

No discussion on an iPhone application would be complete without referring to Apple's software framework for the applications. Our application was created using the Objective-C language within Apple's Cocoa Framework for iOS. We shall discuss very briefly the details and implementation of this framework in our application.

2.4.1 The iOS Development Framework

The iOS development framework is based on the same framework that Apple uses for its Macintosh line of personal computers. The language for development is Objective-C with support for Objective-C++, and the integrated development environment is XCode for Mac. The fundamental building block of all iPhone applications is the Cocoa Touch Framework[10] which controls all the standard input and output parameters of Apple's touchscreen devices.

2.4.2 Accessing Magnetometer Readings

The built-in interface to call the magnetometer within Apple's devices is known as Location Manager. This is the function we call within our application in order to operate and access data from the magnetometer. This interface contains customizable options for sampling rate and a built-in warning for environments which are extremely noisy electromagnetically. This latter functionality can cause some hindrances in the functioning of our application, as we will discuss later. This framework makes building the application possible.

2.4.3 Visualizing the data

Sampling data at such rapid rates can be quite overwhelming for the default graphing system on the iOS Cocoa Touch framework [11]. Therefore, in order to overcome this limitation, our application uses an open-source graphing interface called Core-Plot. Along with support for fast drawing rates, Core-Plot also brings several additional features such as touch manipulation, zooming and real-time plotting, which makes it ideal for our purposes.

CHAPTER 3

THE EXPERIMENTAL PROCESS

With the basic theoretical background for the application ready, it was time for the experimental process to begin. In this chapter, we shall detail the complete process of experimentation, the findings and setbacks encountered, and the methodology we followed, culminating in our results.

3.1 Building the application

To explain briefly our starting point for development, the software framework used to build the first working prototype of the iPhone application was a sample program provided by Apple on its website. This application, called the 'Teslameter' [12], plots the readings from the magnetometer in each of the three axes in a real-time graph. It also displays the values at each instance of measurement, and the total magnitude of the field that it measures. A screenshot of this application is shown in Figure 3.1.

3.2 Initial experiments at sampling in the time domain

The first attempts made at measuring fields were using a real space solution. This would be a quick & easy method to evaluate the application. This process comprised of simply sampling input data from magnetometer and writing code to analyze it in the time domain, where the application would measure the Root Mean Square (RMS) values and the standard deviation. The thinking behind

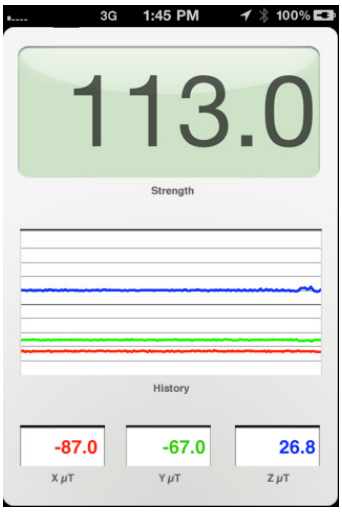

Figure 3.1: A screenshot of the demo Teslameter application provided by Apple. Each of the colored lines responds to an axis in the magnetometer. the readings are in microTesla (μT), and the large bold number indicates the total magnitude of the field ($\sqrt{x^2 + y^2 + z^2}$) in μT.

this method was that the mean of the data would be the DC field generated by the device, and due to the alternating nature of the field, the standard deviation should give us the AC field.

3.2.1 Limitations of sampling in the time domain

Our initial procedure method led to unexpected values in testing. In fact, the results were never anything outside 5mG to 10mG under any conditions, be it in low noise or in close proximity to a strong 60Hz electromagnetic field. This was due to various factors that had not yet been considered in the early stages of measurement. The inherent noise in the time domain is high due to both the nature of the chip and the fact that the screen of the phone updates itself at 60 frames a second, which makes it generate a 60Hz field of its own.

There is an inbuilt low pass filter in the magnetometer to filter out high-frequency fields. This is because the magnetometer is built mainly to serve as a compass and global positioning system, which means it must respond to the user's movement and not to other random noise around it. This leads to the device being made to block high-frequency signals. This low-pass filter would need to be compensated for. Secondly, there was an alternating field generated by continuously updating display, which was also influencing the measurements shown in the device. Thirdly, a very long sampling time would be needed for any reasonable output, even assuming the absolute correctness of our methods, simply due to the nature of the signal analysis methods. This made the measurement in time domain impractical. Therefore, we took further steps to find a method of compensating for these limitations, and finding a method of measurement that would be practical.

3.2.2 Finding alternatives - Measurement in the frequency domain

Subsequent to the results of our experiments in measuring the signal in the time domain, and the limitations produced therewith, we decided that measurement in the frequency domain would be a practical method of measurement. Firstly, it would make the mapping of frequencies of the signals easier, leading to easier identification of AC noise. Secondly, signal averaging is easier - summing several short FFTs is faster and more computationally efficient than taking one long FFT or data sample. In Fourier space, it would also be much easier to measure the response of the magnetometer in a quiet area. This would help us deduce the

low pass filter function and compensate accordingly in our application. Moreover, implementing an FFT for Apple devices is fairly straightforward, since Apple's own frameworks for digital signal processing are extremely fast and efficient. The combination of these factors made sampling in the frequency domain the most suitable solution for our endeavor.

3.3 Estimating the low pass filter built into the magnetometer chip

The iPhone's magnetometer chip, as previously mentioned, contains its own analog to digital converter to supply magnetic heading data to the phone. This filter was built in mainly to cater to the intended application of the device, which was to sense changes in the user's direction by determining the magnetic heading. This translates into a use case wherein the filter must place an emphasis on heading changes which are relatively low frequency (i.e., direction changes by the user), while filtering away any unwanted noise that might factor in the measurement. Typically, this noise is from electromagnetic fields of higher frequency, and typically from electronic devices and AC power sources, which are 50Hz in most regions in the world, and 60Hz in the United States. Therefore, the built-in filter in the device is designed to nullify the noise generated by these frequencies. This is counter-productive to our requirements, therefore, we needed to characterize this filter and compensate for it.

3.3.1 Measuring data in low-noise conditions

Fortunately, the Cornell campus has just the right kind of places in which to deduce the filter functions we wanted. Our reasoning was that, given an environment with very low electromagnetic noise across all the frequencies measured by the magnetometer, a measurement made by the magnetometer, when analyzed in the frequency spectrum, would exhibit the functional form of the filter itself.

To deduce this filter, each model of the iPhone available was set up in a zero electromagnetic noise shielded microscope room, and the corresponding data was recorded. This was then exported and analyzed using a Fourier Transform averaged over 10 cycles, with a sample length of 256 data points and the sampling rate equal to the the maximum sampling rate (128 Hz). The data thus recorded is shown in Figure 3.2.

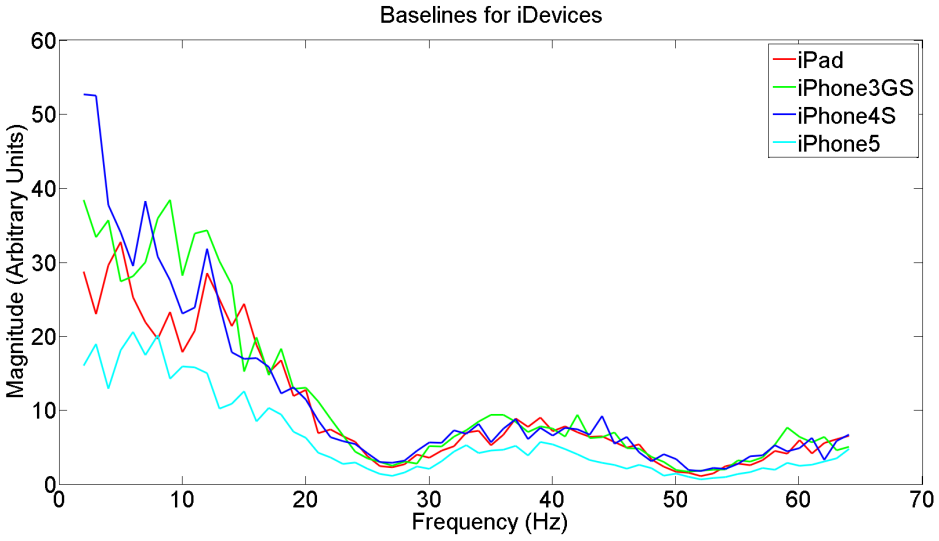

Figure 3.2: The consolidated baselines exhibited by the various devices under low-noise conditions

Each of the devices was found to correspond to a sinc function with a pe-

riodicity of 25Hz. This indicates that the magnetometer chip samples using a rectangular window with a sampling time interval corresponding to 25Hz, the Fourier transform of which is a sinc function. Using a Least-Squares fit, each of the sinc functions was found to correspond to the form

$$F(x) = a \times \left| \frac{\sin(\pi x/b)}{\pi x/b} \right| + c \tag{3.1}$$

Here, $F(x)$ is the low-pass filter function, a is the scale factor, b is the digital sampling frequency inherent to the chip and c is the offset. Our fit (illustrated for one of the devices in Figure 3.3) also allowed us to deduce the values of each of these parameters, as listed in Table 3.1.

Device model	**Scale Factor a (mG)**	**Digital Sampling Frequency b (Hz)**	**Offset c (mG)**
iPhone 3GS	40	25.91	0.0018
iPhone 4S	50	25.91	0.0009
iPhone 5	20	25.91	0.0035
iPad	30	25.91	0.0018

Table 3.1: Parameters of the low pass filter function for each device

Using these parameters, it was now possible to compensate for the filter in each device. Dividing by the appropriate filter function for each device should theoretically result in a flat curve in the frequency domain given a low-noise environment.

After adding this compensation in the code for the application, we returned to test the devices in the microscope room again, looking to see a reasonably flat response curve from the device (with exceptions at the minima of the sinc response since we were dividing by a small number at those points, which led to large noise values), and this was precisely the outcome. This compensated curve, though not without noise, formed the first step of our noise reduction measures. A sample of our compensated output is provided in Figure 3.3.

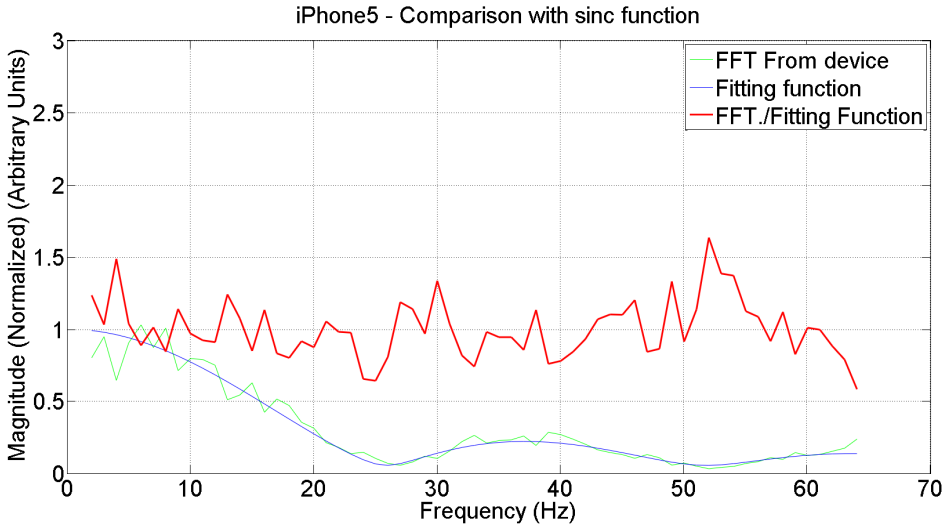

Figure 3.3: A comparison of the generated curves after compensating and normalizing for the filter function in an iPhone 5. The red curve shows the compensated curve. As we can see, there is noise around the minima in the compensated frequency spectrum since we compensate by dividing by a small number at these points. It is also evident that we do not encounter an exact zero at the 25Hz periodicity of the signal. This is due to the unavoidable presence of white noise in any environment.

Why is the low-pass filter in the form of a sinc function ?

The magnetometer in the iPhone was included in the hardware to sense changes in the user's direction by means of the the magnetic heading. This requires it to heavily weight low frequency changes, while simultaneously filtering higher frequencies so that the low-frequency signal is relatively noise-free.

In most use cases, the most likely source of noise is from electromagnetic fields, and electronic devices and AC power sources. These sources generate fields of frequency 50Hz in most of the world, and 60Hz in the United States. The low-pass filter, therefore, must place the least emphasis on these frequencies. The magnetometer unit in the iPhone is of Japanese make. Japan uses 50Hz AC power supplies, so this is taken as the reference frequency to blank in the

low-pass filter.

The sinc function behavior is realized, as we have discussed earlier, by the magnetometer chip performing its digital sampling using a rectangular window with a sampling time interval corresponding to 25Hz, the Fourier transform of which is a sinc function. Sinc functions are ideal low-pass filters since they fall quickly in magnitude and can be easily tuned in frequency to create zeros where required. In this case, the zero is set at 50Hz by tuning the sinc function's periodicity to 25Hz. This could also be achieved by setting it to 50Hz, but that would result in a much slower fall in amplitude, which is undesirable for the chip's primary function. Thus, the sinc function is the ideal filter for the magnetometer's intended use, and this explains why we see this behavior.

3.3.2 Enabling visualization of data in the frequency domain

Having established that frequency domain signal analysis was the way to go, we now needed an effective means of visualizing the data. Due to the iPhone's default graphing framework being limited in its capabilities and ease-of use for such high sampling rates, we turned to the open-source software community for answers. This led us to the utilization of Core Plot, a plotting framework created by Drew McCormack and Barry Wark [13], a custom plotting environment for Apple devices which enabled us to create touch-enabled, interactive and most importantly, fast-updating plots that we could use to visualize our data in the frequency domain, and this would prove significant in our moving forward with the application.

3.4 Real-world performance using measurements in the frequency domain

With the additional compensation for the low-pass filter incorporated into our application, we were ready to test it out again in a real-world setting. The first step was to validate the filter function itself, and confirm that the response curve was now flat instead of a sinc shape. This was achieved by incorporating the filter function into the iPhone and then taking it back to the quiet environment and observe the response of the system. Our methods were validated when we observed the response illustrated in Figure 3.4.

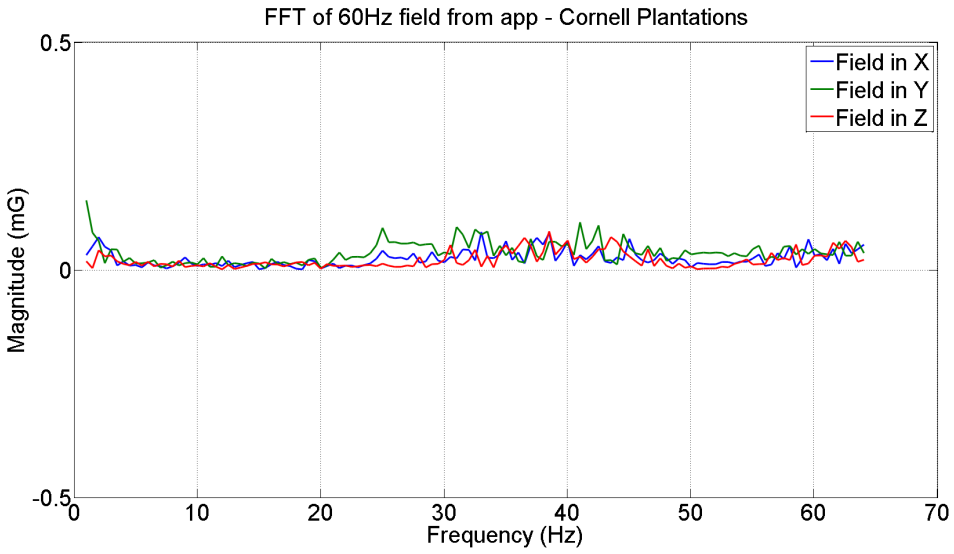

Figure 3.4: The flat response curve of the device in a quiet environment (Cornell Plantations) after the filter compensation, showing the different fields in each of the three axes

3.4.1 Comparison of our device performance with existing instruments

After verifying our calculations for the adjustment for the low-pass filter, the next step was to correlate the response of the application to that of standard measurement devices. Our standard devices were the Extech handheld meter, and for calibration, a Spicer Consulting Spectrum Analyzer unit (hereafter referred to as the Spicer unit) hooked up to a computer with the associated Frequency Spectrum Analyzer software.

With these units in hand, the testing of the application was made in two parts. The first part would be to qualitatively assess whether the application picked up a peak at the 60Hz frequency when within a strong field of 60Hz. If a peak were to be detected, another factor of interest would be how quickly the frequency spectrum would converge to show that peak after throwing away much of the noise. This would also allow us to understand the background noise and estimate a good background subtraction method for the application. With this objective in mind, the application was run with the iPhone next to a standard 60Hz source from the AC mains of the microscope chamber, an extremely strong electromagnetic field with no noise in the surrounding frequencies. The application quickly showed a peak at 60Hz which was very visible even above the noise generated, and within two runs of sampling and data acquisition (256 samples each), was able to produce a pronounced peak at 60Hz with a fair amount of noise reduction (illustrated in Figure 3.5). This showed that even with the inherent noise generated, it would be possible to identify strong 60Hz fields, and with good background subtraction, fields of lower magnitudes could be identified as well.

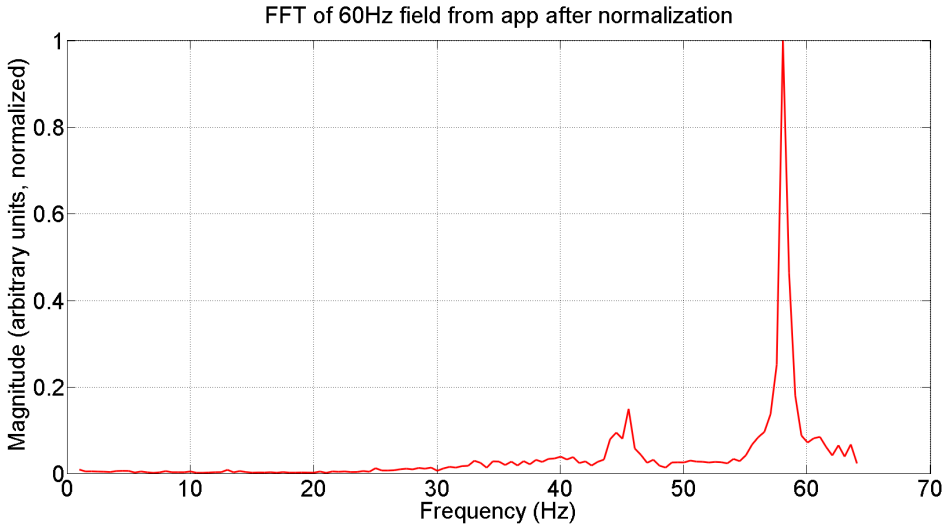

Figure 3.5: A plot of the response to a strong 60 Hz field from the iPhone 5 after the signal compensation. There is now a strong, clearly visible peak at the frequency of the strong field around 60Hz.

Our setup to calibrate the application was very simple. First, the strongest 60Hz source in the room was identified. Then, using graph paper glued on to a platform at the same height as the source, the instruments would be moved each time by the unit distance marked on the graph paper, from the point closest to the source to the farthest possible point from the source. This would give us a coherent picture of how the electric field would fall off with distance from the source, and by correlating the measurements from each of the three measuring devices (hand-held meter, Spicer unit and iPhone application), we would be able to understand if our application was giving us measurement that made sense physically. If the three devices followed the same general trend, then we would be able to find the scale factors that would allow us to convert the readings from the iPhone application into measurable values in milliGauss. A very brief review of our electromagnetism theory would say that since our source is a dipole, the potential is inversely proportional to the square of the distance from the source [14]. In essence, if $\phi(\mathbf{R})$ was the potential as a function of the distance

R from the source, then

$$\phi(\mathbf{R}) = \frac{1}{4\pi\varepsilon_0}\frac{\mathbf{p}\cdot\hat{\mathbf{R}}}{R^2} \tag{3.2}$$

Here, ε_0 is the permittivity of free space, **p** is the dipole moment and $\hat{\mathbf{R}}$ is the unit vector in the direction of **R**. The electric field **E** of the dipole is the negative gradient of the potential, which means we can derive it to be :

$$\mathbf{E} = \frac{3\mathbf{p}\cdot\hat{\mathbf{R}}}{4\pi\varepsilon_0 R^3}\hat{\mathbf{R}} - \frac{\mathbf{p}}{4\pi\varepsilon_0 R^3} \tag{3.3}$$

The important trend to note here is that the electric field for a dipole falls off as the cube of the distance i.e. $\mathbf{E} \propto \frac{1}{\mathbf{R^3}}$. This means that for our experiment to be valid, we first had to confirm this trend with the data collected from the field meter and the Spicer, and then verify that the iPhone application also followed the same trend, from which we could deduce the scaling factors.

As we computed the results of the experiments, we found that, as expected, the handheld meter and the Spicer did, indeed, fall off as expected. However, this was with a small variation : Since the magnetometer in each of them is not perfectly at the boundaries of the instruments, there was a small offset. This meant that our fit fell off as $\mathbf{E} \propto \frac{1}{(\mathbf{R}+\mathbf{a})^3}$ instead of $\mathbf{E} \propto \frac{1}{\mathbf{R^3}}$ where a is the offset of the magnetometer element in each of the devices. The constant of proportionality was deduced by using a Least-Squares curve fitting tool in the Matlab software package. As an illustration, the curve fit thus obtained for the hand-held meter is shown in Figure 3.6. The fit parameters and fitting function are detailed in Table 3.2 and Equation 3.4.

Once the hand-held meter and the Spicer were recorded and the data fitted to a function of this form, the iPhone data was plotted and found to correspond roughly with this form as well. However, the iPhone data was found to be besotted with noise. This was unsurprising, since we were already dividing by the

fitting function. In addition, we were also not compensating for the noise added at higher frequencies since the iPhone will pick up noise at the other harmonics of the signal as well, and measure them with ambiguity. In order to better separate the signal from the noise, a background subtraction technique was used, subtracting the values around the frequency to be monitored (60Hz). This gave us a much cleaner dataset that we could fit with higher confidence, as illustrated in Figure 3.7. Summing and background subtraction of the frequencies around the ones in question gives us a much better idea of the actual field picked up and alleviates some of the possible quantitative disparity present when we measure from only one data point at 60Hz.

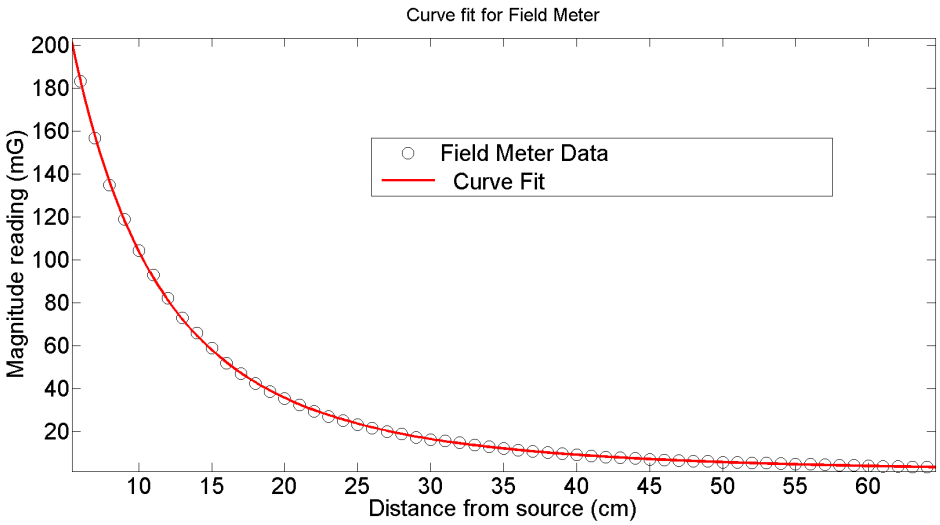

Figure 3.6: The comparison of the obtained data from the hand-held meter and the corresponding curve fit from Table 3.2 and Equation 3.4.

The functional form of **E** for each of the devices was found to be

$$\mathbf{E}(\mathbf{R}) = \frac{k}{(\mathbf{R} + a)^3} + C \tag{3.4}$$

Here, k is the constant of proportionality, a is the offset in distance and C is the offset which the field reading approaches as the device becomes distant from

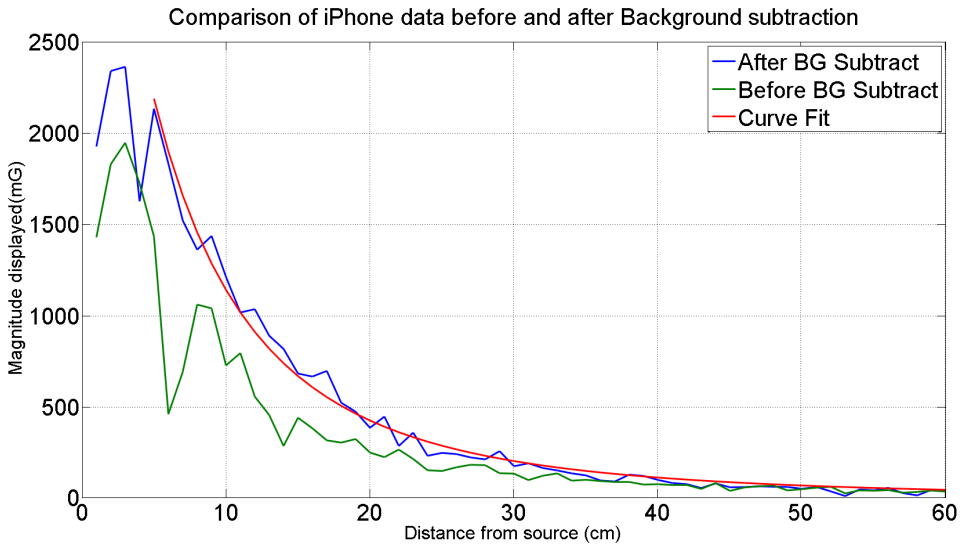

Figure 3.7: The comparison of the iPhone data before and after background subtraction. The background subtraction enables a better curve fit.

the source. The values of each of these factors deduced from the curve fitting are detailed in Table 3.2.

Device	**Scale Factor k (mG cm^3)**	**Constant Reading C (mG)**	**Offset a (cm)**
Hand-held meter	1.0×10^5	0.3341	12.00
Spicer unit	1.258×10^6	0.7656	13.02
iPhone	1.912×10^7	1.0105	15.6

Table 3.2: Parameters of the Field function for each device

Using these parameters, it was now possible to deduce the scaling factors to incorporate into our application, by means of which we would be able to achieve quantitative parity with the Spicer unit or the hand-held meter (depending on our choice of scaling factor), as illustrated in Figure 3.8. As we can see, the sensitivity is quite comparable at low fields, which is essential for our intended operation. As the magnitude of the the field increases, so does the noise, which is expected given the limitations of the device and our scaling methods. However, at such high field values, one would be quite unwise to be operating an electron microscope in any case, which means that this limitation

of our device will not impact our intended use. This left us only to characterize

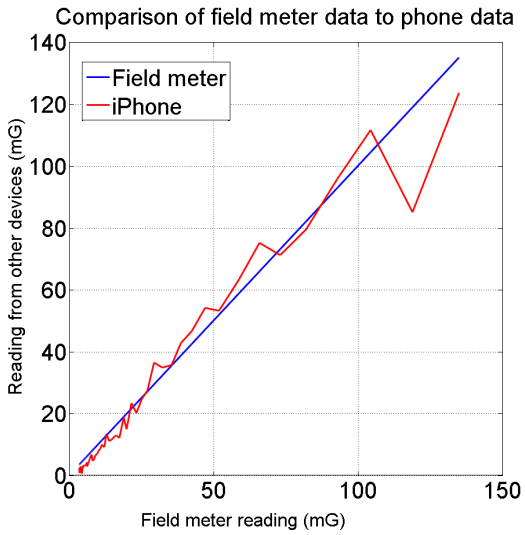

Figure 3.8: The comparison of the scaled data from the iPhone and the corresponding readings from the field meter.

the response time of our application and how long it took to achieve parity with the readings from the hand-held meter. In order to achieve this, we carried out a series of tests on the application in low fields (between 0.5mG and 10mG) in various environments around the Cornell campus, and it was deduced that with a sample size of 256 readings per FFT averaging cycle (the sampling rate remains fixed at 128Hz throughout), the reading on the iPhone converged close to the reading on the hand-held meter within three to five cycles (an effective time of six to ten seconds), similar to the time taken for a sample size of 512 readings per FFT averaging cycle. Thus, we were also able to reinforce confidence in our method, since summing several shorter FFTs faster gave us results equal to or better than larger FFTs summed together (The SNR scales as the square root of the number of data samples summed over). In fact, if one were willing to extend one's waiting time indefinitely, we would be able to gain a little more accuracy, but for all practical purposes, our application converges quickly (within 3 - 5

cycles) to a value accurate to within 5% - 15% of that given by a dedicated device. These results, when combined, brought us to realising our goal of creating an accessible, easy-to-use electromagnetic field meter embedded in our mobile devices, and we shall further discuss them in the next chapter.

CHAPTER 4

RESULTS AND PERFORMANCE OF THE APPLICATION

In this chapter, we shall discuss the results and the salient points that the experimental process brought to note about our application. To briefly recap our goals for the project, we set out to build a convenient and simple method of characterizing electromagnetic fields form our iPhones using some clever signal averaging techniques and the powerful software framework that made it possible to build such applications. To this end, we were able to build a prototype application to achieve this, a working screenshot of which is shown in Figure 4.1, so our broadest goal was achieved. Our main interests were to compare the results in terms of sensitivity and quickness of convergence.

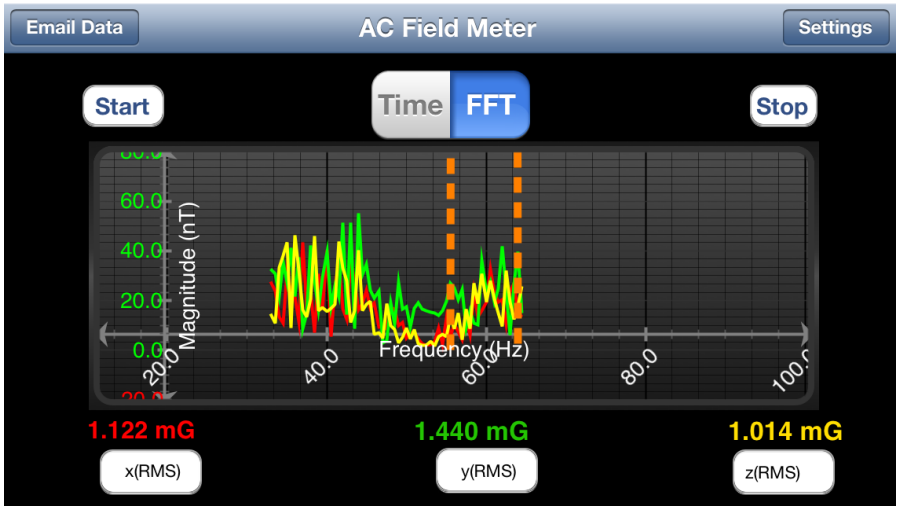

Figure 4.1: A screenshot of the application at work, showing measurement and data visualization in the frequency domain. Each of the colored lines corresponds to the readings from one of the the three axes (as labeled in the screenshot), while the vertical dotted lines are visual aids to indicate data between 55Hz and 64Hz - the region of greatest interest.

4.1 Quickness of convergence

In terms of quickness of convergence, readings from the hand-held meter were tested along with the application in low-field areas. With a fixed sampling rate of 128Hz, a sample size of 256 readings per FFT averaging cycle, the reading on the iPhone converged close to the reading on the hand-held meter within three to five cycles (an effective time of six to ten seconds), similar to the time taken for a sample size of 512 readings per FFT averaging cycle. The signal-to-noise ratio scales as the square root of the number of samples summed over, i.e., if N is the number of FFTs summed over, then

$$\text{SNR} \propto \sqrt{N} \tag{4.1}$$

If one were willing to extend one's waiting time indefinitely, we would be able to gain a little more accuracy, but for all practical purposes, the application converges quickly (within 3 - 5 cycles) to a value accurate to within 5% - 15% of that given by a dedicated device.

Given these results as an indicator of the performance of the application, the user can then modify the number of data points and use a longer or shorter FFT to achieve their desired resolution in frequency with some idea of the trade-offs involved in time and speed.

4.2 Sensitivity

In order to be useful for the characterization of magnetic fields within microscope rooms, a field meter would have to be sensitive to fields of the order of magnitude of a tenth of a milliGauss or less. For SEM, a sensitivity of 0.1 mG

would be quite useful (if there were fields of that magnitude in a TEM room, one would be in trouble). Our application shows a sensitivity on the order of 0.1 mG, especially for fields in the 30Hz - 60Hz bandwidth, which means we can use it with some confidence to characterize fields for SEM. Of course, there would be some associated error (an offset of between 0.5mG and 1mG) on the field, but we account for that by providing a background subtraction function which is user-adjustable inside the application itself, as shown in Figure 4.2, so that each individual user can calibrate it for their device with a standard field meter so they can trust the application's accuracy for themselves.

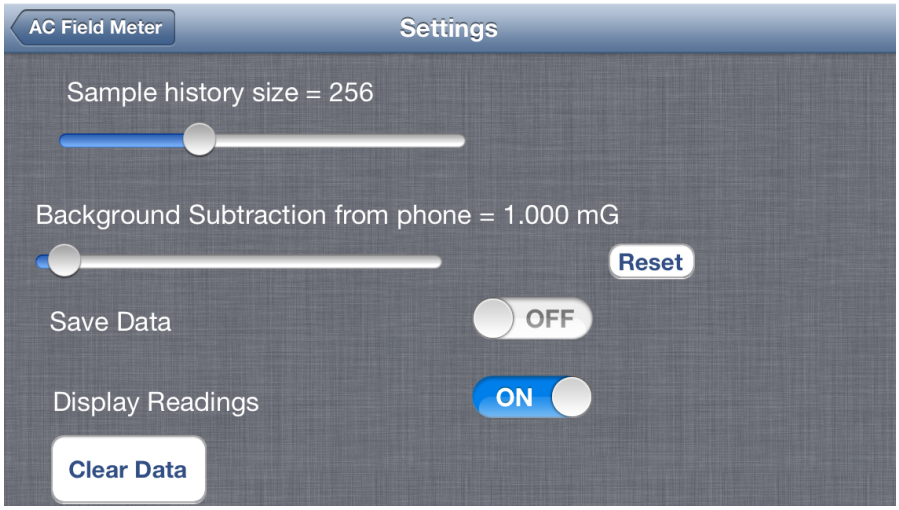

Figure 4.2: A screenshot of the application's settings page where the user can customize the background noise subtraction. This is meant for advanced users. In the typical use case, the offset for each device is obtained from our fitting function, and remains fixed for each device.

4.3 Performance and possible issues

Another consideration for our application was to use extremely fast and efficient computational techniques for all the calculations so as not to bottleneck the

speed at which the compass takes readings. A slow Fourier Transform routine, for instance, would stop the readings at every cycle of computation, thereby affecting our sampling rate and potentially even causing ambiguity in our measurements of frequency. This was continually checked at the addition of every computation to the program to ensure the application remained speedy. This was also enabled by Apple's FFT routine which has been consistently tested to be faster than FFTW, the fastest open-source FFT routine available [15]. This raises confidence in our sampling rate and consequently our signal analysis.

Lastly, one persistent issue with the application is a precautionary measure built into the iPhone by Apple itself. As discussed earlier in our theoretical background of the iOS framework and the Location framework, Apple has implemented a noise-checking routine that continually prompts the user to move the iPhone in a figure-of-eight motion whenever it runs into large or noisy fields. In our case, these may simply be the fields we want to measure, and this built-in routine can hinder our measurements. This can of course be negated by making the figure-of-eight motion that the prompt suggests and measuring the field again, however, future iterations of the application will look for ways to bypass this routine so that the user experience is consistent and reliably fast.

CHAPTER 5

CONCLUSIONS AND FUTURE WORK

With the results of our experiment, we were at a stage where we had proof-of-concept of our methods, as well as a fair performance indicator of where our application stood as a characterization method for electromagnetic fields in the context of electron microscopy. Our conclusions were that our application would be suitable for use for the desired purpose, especially in the context of SEM rooms (which is where the handheld meters are most highly used as well). Another important implication of our exercise was the effectiveness of our signal analysis methods, and how, in combination with good background subtraction and some effective signal averaging, even a fairly limited device could be purposed to obtain good data and the inherent trends, both qualitatively and quantitatively. All these factors, in combination, meant that our application was effectively deployable to the public, and could facilitate quick electromagnetic field characterization in applications from microscopy to garage laboratories, especially if made free and open, which is our aim.

5.1 Future work on this project

As covered in the previous chapter, the future work on our project would involve continuously improving our signal averaging methods as we receive user feedback, improving our accuracy even further. Another important concern is to bypass Apple's inbuilt signal compensation prompt when the device encounters noisy fields. Correspondence has been made to Apple about this, and we aim to take it forward with input from them.

Our plans for the project also include improvements to the user interface of the applications, such as easy switching of the parameters of the fit for the user, and a customized pinch-scaling/zoom system for the visualization of the FFT so that the user may better visualize trends in the signal, especially when the trends are at frequencies other than 60Hz. We also aim to implement a moving 'window' wherein we use our signal averaging techniques to quantitatively analyze any frequency measurable by the phone (i.e. $\leq$ 64Hz) , not just at 60Hz, so that the user can characterize any frequency they desire.

5.1.1 Application of our signal averaging techniques to other sensors

Another of the future aims of the project is to apply the same signal averaging and background subtraction techniques used in our application to signals from other sensors in the iPhone such as the accelerometer and the microphone. This would enable us to create effective characterization techniques for vibration and acoustic signals, allowing us to make a suite of characterization applications that could be used by scientists both experienced and budding, with access to just a smartphone. In fact, vibration and acoustic analysis would be even less constrained than electromagnetic field analysis simply because the range of frequencies that can be sampled by these sensors is much greater [3], rendering them even more effective.

5.2 Other applications of our software

In addition to scientific purposes, our software could be used for a number of other possible applications. The two most exciting possibilities are discussed forthwith.

5.2.1 Geolocation and wiring faults

Our software already uses the compass' subroutine to access data, so if we were able to add geolocation tagging to the data, we would, in effect, be able to create a 'heat map' of electrical fields within a room, or, given enough users to consolidate data from, even a locality. This data could then be transmitted in real-time to a data center for an electrical grid/power supply company, which could then monitor them to pinpoint points of possible failure in the electrical grid or wiring within an area, leading to a fair saving of resources as well as faster response. This would also be a simpler application since electrical faults give out large fields at 60Hz, which means that our application would detect them even quicker than it does low fields. In addition, all the software needs to decide is whether the field value is above a threshold value, in which case we could flag that point using a geolocation tag and send the data. This reduces the need for absolute quantitative accuracy, laying the emphasis more on sensitivity and quickness of response, which are strong points of our application.

5.2.2 Touchless 3-D Interaction

There are potential applications of the magnetometer to enable three-dimensional interaction with smartphone interfaces, rather than two-dimensional touch interaction. One such application is MagiMusic [16], which is a musical instrument simulator controlled using special magnetic gloves and the interpretation of the signal generated by using them on the magnetometer. Our signal averaging techniques would greatly increase the accuracy of such applications and increase the scope of touchless interaction with smartphones.

APPENDIX A

DEVICES USED IN THE MEASUREMENT PROCESS

This chapter offers a brief overview of the two devices used for the performance comparison of our iPhone application.

A.1 Spicer Consulting Field Measurement Unit

Spicer Consulting's SC11 unit used in measurement combines the measurements and analysis of magnetic fields, vibrations and acoustics into a compact, easy to operate system. is the simplest way to gain accurate usable data from site surveys for SEMs and similar sensitive equipment. Based on a laptop computer and operating under Windows XP/Vista/7 with accompanying Spectrum Analyser software, the Spicer unit produces results displayed graphically.

A.2 Extech 480822 Field Meter

The instrument used in our handheld device comparison was the Extech Instruments 480822 Electromagnetic Field Meter. It is a single axis meter which samples every 0.4 seconds. It has a range from 0.1 to 199.9 mG with an overrange indicator and measures ELF-EMF frequencies from 30 to 300 Hz with an accuracy of 2% at 50/60 Hz.

For further details on both these devices, it is recommended to visit the corresponding manufacturer's website.

APPENDIX B

SPECIFICATIONS OF THE MAGNETOMETER USED IN THE IPHONE 4S

The following page details the specification of the Asahi Kasei AK8973 compass chip used in the iPhone 4S. The AKM8975 chip's (used in the iPhone 5) specifications are proprietary and hence not available online.

=Preliminary= AK8973

3-axis Electronic Compass

1. Features

- 3-axis electronic compass IC
- Optimal built-in electronic compass for mobile phones and handy terminals
- High sensitivity Hall sensors are integrated.
- Functions
  - Built-in 8-bit ADC
  - Built-in amplifier for sensor signal amplification
  - Built-in 8-bit DAC for sensor signal offset compensation
  - Built-in EEPROM for storing individual adjustment values
  - Built-in temperature sensor
  - 8-bit digital output
  - Serial interface: I^2C bus interface (supporting the low-voltage specification)
  - Automatic power-down function
  - Interrupt function for measurement data ready
  - Built-in master clock oscillator
- Operating temperatures: -30°C to +85°C
- Operating supply voltage: +2.5V to +3.6V
- Low current consumption/measurement time:
  - Power-down: 0.2μA typ.
  - Magnetic sensor driving: 6.8mA/12.6ms
- Package: 16-pin QFN package: 4.0mm×4.0mm×0.7mm

APPENDIX C

SOURCE CODE

The following pages contain the source code of our application. Concessions have been made in the text to follow the LaTeXformatting methods. The aim of this section is to provide a brief overview of how the software is written. Further details and source code files can be obtained by emailing the author.

C.1 The application delegate

```
/*
 File: AppDelegate.h

 */

#import <UIKit/UIKit.h>
#
@class TeslameterViewController;

@interface AppDelegate : NSObject <UIApplicationDelegate> {
UIWindow *window; TeslameterViewController *viewController;
UINavigationController *navigationController;
}
}}
@property (nonatomic, strong) IBOutlet UIWindow *window;
@property (nonatomic, strong) IBOutlet TeslameterViewController
*viewController;

@end
```

```
/*
 File: AppDelegate.m */

#import "AppDelegate.h" import "TeslameterViewController.h"
#
//To remove error "implicit declaration of sysctlbyname"
#include <sys/types.h> include <sys/sysctl.h>
#
@implementation AppDelegate

@synthesize window; @synthesize viewController;

- (void)applicationDidFinishLaunching:(UIApplication
*)application {

    [self.window setRootViewController:self.viewController];

    viewController.view.frame = CGRectMake(0.0, 0.0, 480.0,
    320.0); [window addSubview:viewController.view]; [window
```

```
    makeKeyAndVisible];

    //To hide status bar - Change the property in the
    //info.plist, the code below is less effective [application
    //setStatusBarHidden:YES];
    //
    //To adapt to bigger screens
    //#warning Need to do more :
    //#http://stackoverflow.com/questions/12395200/how-to-
    //#develop-or-migrate-apps-for-iphone-5-screen-resolution
    //    [window setFrame:[[UIScreen mainScreen] bounds]];
    //
    //
    // }
    // }
    // }
    // }
-(BOOL) application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {

    [viewController setTitle:@"AC Field Meter"];
    navigationController = [[UINavigationController alloc]
    initWithRootViewController:viewController];

    //[navig pushViewController:viewController animated:YES];
    //[self.window addSubview:navigationController.view];
    //[self.window makeKeyAndVisible];
    //
    //This fixed the navigation issue
    self.window.rootViewController = navigationController;

    //Solving the auto-layout issue and unresponsiveness of
    //buttons
    //http://stackoverflow.com/questions/14345727/when-on-iphone
    //-5-landscape-mode-the-button-on-the-right-of-navigation-
    //bar-stops
    //
    NSLog(@"Device Model : %@", [self platformString]);
    [[NSUserDefaults standardUserDefaults] setObject:[self
    platformString] forKey:@"deviceModel"];

    return YES;
}
}
#ifdef IOS_OLDER_THAN_6
-
(BOOL)shouldAutorotateToInterfaceOrientation:(
UIInterfaceOrientation)toInterfaceOrientation{
    //[image_signature setImage:[self
    //resizeImage:image_signature.image]];
    return (toInterfaceOrientation ==
    UIInterfaceOrientationLandscapeLeft);
}
#endif ifdef IOS_NEWER_OR_EQUAL_TO_6
-(BOOL)shouldAutorotate { return YES;
}
- (NSUInteger)supportedInterfaceOrientations {
    //[image_signature setImage:[self
    //resizeImage:image_signature.image]];
    return UIInterfaceOrientationMaskLandscapeLeft;
}
#endif
#
```

```
- (NSString *) platform{ size_t size; sysctlbyname("hw.machine",
NULL, &size, NULL, 0); char *machine = malloc(size);
sysctlbyname("hw.machine", machine, &size, NULL, 0); NSString
*platform = [NSString stringWithUTF8String:machine];
free(machine); return platform;
}
}
- (NSString *) platformString{ NSString *platform = [self
platform]; if ([platform isEqualToString:@"iPhone1,1"])
{[[NSUserDefaults standardUserDefaults] setFloat:30.0
forKey:@"DeviceScaleFactor"];   return @"iPhone_1G";        } if
([platform isEqualToString:@"iPhone1,2"])    {[[NSUserDefaults
standardUserDefaults] setFloat:30.0
forKey:@"DeviceScaleFactor"];   return @"iPhone_3G";        } if
([platform isEqualToString:@"iPhone2,1"])    {[[NSUserDefaults
standardUserDefaults] setFloat:40.0
forKey:@"DeviceScaleFactor"];   return @"iPhone_3GS";       } if
([platform isEqualToString:@"iPhone3,1"])    {[[NSUserDefaults
standardUserDefaults] setFloat:30.0
forKey:@"DeviceScaleFactor"];   return @"iPhone_4";         } if
([platform isEqualToString:@"iPhone3,3"])    {[[NSUserDefaults
standardUserDefaults] setFloat:30.0
forKey:@"DeviceScaleFactor"];   return @"Verizon_iPhone 4"; } if
([platform isEqualToString:@"iPhone4,1"])    {[[NSUserDefaults
standardUserDefaults] setFloat:50.0
forKey:@"DeviceScaleFactor"];   return @"iPhone_4S";        } if
([platform isEqualToString:@"iPhone5,1"])    {[[NSUserDefaults
standardUserDefaults] setFloat:20.0
forKey:@"DeviceScaleFactor"];   return @"iPhone_5";         } if
([platform isEqualToString:@"iPod1,1"])      {[[NSUserDefaults
standardUserDefaults] setFloat:30.0
forKey:@"DeviceScaleFactor"];   return @"iPod_Touch_1G";    } if
([platform isEqualToString:@"iPod2,1"])      {[[NSUserDefaults
standardUserDefaults] setFloat:30.0
forKey:@"DeviceScaleFactor"];   return @"iPod_Touch_2G";    } if
([platform isEqualToString:@"iPod3,1"])      {[[NSUserDefaults
standardUserDefaults] setFloat:30.0
forKey:@"DeviceScaleFactor"];   return @"iPod_Touch_3G";    } if
([platform isEqualToString:@"iPod4,1"])      {[[NSUserDefaults
standardUserDefaults] setFloat:30.0
forKey:@"DeviceScaleFactor"];   return @"iPod_Touch_4G";    } if
([platform isEqualToString:@"iPad1,1"])      {[[NSUserDefaults
standardUserDefaults] setFloat:30.0
forKey:@"DeviceScaleFactor"];   return @"iPad";             } if
([platform isEqualToString:@"iPad2,1"])      {[[NSUserDefaults
standardUserDefaults] setFloat:30.0
forKey:@"DeviceScaleFactor"];   return @"iPad_2_(WiFi)";    } if
([platform isEqualToString:@"iPad2,2"])      {[[NSUserDefaults
standardUserDefaults] setFloat:30.0
forKey:@"DeviceScaleFactor"];   return @"iPad_2_(GSM)";     } if
([platform isEqualToString:@"iPad2,3"])      {[[NSUserDefaults
standardUserDefaults] setFloat:30.0
forKey:@"DeviceScaleFactor"];   return @"iPad_2_(CDMA)";    } if
([platform isEqualToString:@"i386"])         return
@"Simulator"; if ([platform isEqualToString:@"x86_64"])
return @"Simulator"; else
    return platform;
}
}
@end
```

C.2 The application controller

```
/*
 File: TeslameterViewController.h */

#import "AppDelegate.h" import <CoreLocation/CoreLocation.h>
#import <MessageUI/MessageUI.h> import <Accelerate/Accelerate.h>
#//For vDSP code to use the Apple-approved FFT import
#"CorePlot-CocoaTouch.h" import "GraphObject.h" import "iToast.h"
#
#import "Stack.h" import "Queue.h"
#
//This is the fudge factor for the sinc fitting function taken
//from the measured values, where the array size was 128 - We
//will scale this according to the current array size//
#define fudgeFactorForAraySize_128 25.9100 define
#scaleFactorToConvertToSpicerRMSValues 19.12
#
#
@class GraphObject;



@interface TeslameterViewController : UIViewController
<CLLocationManagerDelegate, MFMailComposeViewControllerDelegate>
{ UILabel *magnitudeLabel; UILabel *xLabel; UILabel *yLabel;
UILabel *zLabel;


    IBOutlet CPTGraphHostingView *graphHostingView; GraphObject
    *graphObject;

    CLLocationManager *locationManager; IBOutlet UIButton
    *startButton; IBOutlet UIBarButtonItem *settingsButton;
    IBOutlet UIBarButtonItem *sendMailButton; UISwitch
    *fileSaveSwitch; IBOutlet UIButton *stopButton;

}
}
@property(nonatomic, strong)GraphObject *graphObject;

// IBOutlets
@property (nonatomic, strong) IBOutlet UILabel *xLabel;
@property (nonatomic, strong) IBOutlet UILabel *yLabel;
@property (nonatomic, strong) IBOutlet UILabel *zLabel;


@property (strong, nonatomic) IBOutlet UIButton
*xReadingDisplayTitleButton; @property (strong, nonatomic)
IBOutlet UIButton *yReadingDisplayTitleButton; @property
(strong, nonatomic) IBOutlet UIButton
*zReadingDisplayTitleButton; @property (nonatomic, strong)
IBOutlet UISwitch *fileSaveSwitch; @property (strong, nonatomic)
IBOutlet UISwitch *showDisplaySwitch; @property (strong,
nonatomic) IBOutlet UISegmentedControl *graphDisplayControl;


- (IBAction)startButtonPressed:(id)sender; -
(IBAction)stopButtonPressed:(id)sender; -
(IBAction)settingsButtonPressed:(id)sender; -
(IBAction)sendMailButtonPressed:(id)sender; -
```

```
(IBAction)graphDisplaySwitch:(id)sender;

@property (nonatomic, strong) CLLocationManager
*locationManager;


@end



/*
 File: TeslameterViewController.m */

#import "TeslameterViewController.h" import "GraphObject.h"
#import "SettingsViewController.h" import "FFTAccelerate.h"
//#import "CodeTimestamps.h"
//#
//#
#include <sys/types.h> include <sys/sysctl.h> include
#<sys/time.h>
#
#include <stdio.h> include <math.h>
#
#import <mach/mach.h> import <mach/mach_time.h>
#
#include <stdlib.h> include <complex.h> include <math.h> include
#<algorithm>
#
#
@implementation TeslameterViewController{
    //Integers
    int                 counter; int
    fftCounterForGraph; int                 index; int
          arrayCapacity; int
    setNumberOfRunningUpdateValues; int
    setNumberOfFFTsToSumOver;

    //Conversion Factors
    float               conversionFactorMicroTeslaToMilliGauss;
    float
    conversionFactorNanosecondsToMilliseconds;

    float               initialX, initialY, initialZ; float
             sum_FFT_X, sum_FFT_Y, sum_FFT_Z;

    float               *arrayOfXData, *arrayOfYData,
    *arrayOfZData, *arrayOfMagnitudeData, *arrayOfForwardFFT_X,
    *arrayOfForwardFFT_Y, *arrayOfForwardFFT_Z,
    *arrayOfSummed_FFT_X, *arrayOfSummed_FFT_Y,
    *arrayOfSummed_FFT_Z;

    //long                *arrayOfTimeInMilliSeconds; long
    //          conversionFactorForMachAbsoluteTimeToNanoseconds;
    //
    float               deviceScaleFactor, fudgeFactor,
    sincOffset;


    UINavigationController  *navController;

    int           functionCounter;

    BOOL          saveFiles;
```

```
    BOOL          showDisplay;

    BOOL          checkStopButtonPressedConsecutively; //To make
    sure we don't dealloc unassigned memory for consecutive
    presses of the stop button

    int           RMSCounter;

    float         RMSAddX, RMSAddY, RMSAddZ;

    float         FFTXValue, FFTYValue, FFTZValue,
    baselineOffset;

    int           value58Hz, value64Hz;


}
}
@synthesize xLabel; @synthesize yLabel; @synthesize zLabel;
@synthesize graphObject = graphObject; @synthesize
xReadingDisplayTitleButton; @synthesize
yReadingDisplayTitleButton; @synthesize
zReadingDisplayTitleButton; @synthesize fileSaveSwitch;
@synthesize showDisplaySwitch; @synthesize graphDisplayControl;


@synthesize locationManager;

- (void)viewDidLoad { [super viewDidLoad];

    settingsButton = [[UIBarButtonItem
    alloc]initWithTitle:@"Settings"
    style:UIBarButtonItemStyleBordered target:self
    action:@selector(settings)]; [[self navigationItem]
    setRightBarButtonItem:settingsButton];

    settingsButton.target = self; settingsButton.action =
    @selector(settingsButtonPressed:);

    sendMailButton = [[UIBarButtonItem
    alloc]initWithTitle:@"Email Data"
    style:UIBarButtonItemStyleBordered target:self
    action:@selector(sendMail)]; [[self navigationItem]
    setLeftBarButtonItem:sendMailButton];

    sendMailButton.target = self; sendMailButton.action =
    @selector(sendMailButtonPressed:);

    [sendMailButton setEnabled:YES];

    //Get the scale factor for the device - This is not going to
    //be changed, so we'll just call it when the app is
    //initialized
    deviceScaleFactor = [[NSUserDefaults standardUserDefaults]
    floatForKey:@"DeviceScaleFactor"]; NSLog(@"The device scale
    factor = %g", deviceScaleFactor);


        // setup the location manager
        self.locationManager = [[CLLocationManager alloc] init];

    index = 0;
```

```
    counter = 0;

    fftCounterForGraph = 0;

    conversionFactorMicroTeslaToMilliGauss = 10.0;

    conversionFactorNanosecondsToMilliseconds = 1e-6;

    functionCounter = 1;

    graphObject = [[GraphObject alloc]
    initWithGraphHostingView:graphHostingView];
}
}
- (void)viewDidUnload { [self setGraphDisplayControl:nil];
graphHostingView = nil;
    //    graphView = nil;
    stopButton = nil; [self setShowDisplaySwitch:nil];
    startButton = nil; [self setZReadingDisplayTitleButton:nil];
    [self setYReadingDisplayTitleButton:nil]; [self
    setXReadingDisplayTitleButton:nil];
    //[self setStopButtonPressed:nil];
    //
    self.xLabel = nil; self.yLabel = nil; self.zLabel = nil;
    self.graphObject = nil;
}
}
- (void)dealloc {
    //Since ARC is now included in the project, this method
    //exists simply to stop the compass
        [locationManager stopUpdatingHeading];
}
}
- (IBAction)settingsButtonPressed:(id)sender {


    NSLog(@"Settings button pressed");

    [self.navigationController
    pushViewController:[[SettingsViewController alloc]
    init]animated:YES];

}
}
- (IBAction)sendMailButtonPressed:(id)sender {
    //NSLog(@"Send Mail button pressed");
    //
    NSArray *paths =
    NSSearchPathForDirectoriesInDomains(NSDocumentDirectory ,
    NSUserDomainMask, YES); NSString *documentsDir = [paths
    objectAtIndex:0]; NSFileManager *fileManager =
    [NSFileManager defaultManager]; NSError *error = nil;
    NSMutableArray *filePathArray = [[NSMutableArray alloc]
    init];

    NSArray * files = [[NSFileManager defaultManager]
    contentsOfDirectoryAtPath:documentsDir error:nil];

    if ([files count] == 0) {
        //There are no files to be emailed
        [[iToast makeText:NSLocalizedString(@"No files to
        email.", @"")] show]; return;
```

```
    }                                                                   180
    }                                                                   181
    if ([MFMailComposeViewController canSendMail]) {                    182
                                                                        183
        MFMailComposeViewController *composer =                         184
        [[MFMailComposeViewController alloc] init];                     185
        composer.mailComposeDelegate = self;                            186
                                                                        187
        [composer setSubject:[NSString                                  188
        stringWithFormat:NSLocalizedString(@"Readings", nil),           189
        @"ABC"]];                                                       190
                                                                        191
        int indexer = 0;                                                192
                                                                        193
        for (NSString *file in [fileManager                             194
        contentsOfDirectoryAtPath:documentsDir error:&error]) {         195
        NSString *filePath = [documentsDir                              196
        stringByAppendingPathComponent:file]; [filePathArray            197
        insertObject:filePath atIndex:indexer];                         198
                                                                        199
            NSData *fileData = [NSData                                  200
            dataWithContentsOfFile:[filePathArray                       201
            objectAtIndex:indexer]];                                    202
                                                                        203
            [composer addAttachmentData:fileData                        204
            mimeType:@"text/plain" fileName:[[filePathArray             205
            objectAtIndex:indexer]                                      206
            stringByReplacingOccurrencesOfString:documentsDir           207
            withString:@""]];                                           208
                                                                        209
                                                                        210
            NSLog(@"File : %@", filePath); indexer++;                   211
        }                                                               212
        }                                                               213
        NSLog(@"File Path Array = %@", filePathArray);                  214
                                                                        215
        // Fill out the email body text                                 216
        NSString *emailBody = [NSString                                 217
        stringWithFormat:NSLocalizedString(@"Readings from              218
        Date", filePath), @"ABCD", @"ABCDE"]; [composer                 219
        setMessageBody:emailBody isHTML:YES];                           220
                                                                        221
        [self presentViewController:composer animated:YES               222
        completion:NULL];                                               223
                                                                        224
                                                                        225
    }                                                                   226
    }}                                                                  227
    }}                                                                  228
    }}                                                                  229
- (IBAction)graphDisplaySwitch:(id)sender { int dummyCounter =         230
index;                                                                  231
    //NSLog(@"Dummy counter = %d", dummyCounter);                       232
    //                                                                  233
    if(dummyCounter > 0) { //The process is running, and so             234
    needs to be stopped                                                 235
                                                                        236
        [locationManager stopUpdatingHeading];                          237
                                                                        238
        counter = 0;                                                    239
                                                                        240
        fftCounterForGraph = 0;                                         241
                                                                        242
```

```
        index = 0;

    }
    UISegmentedControl *segmentedControl = (UISegmentedControl
    *) sender; NSInteger selectedSegment =
    segmentedControl.selectedSegmentIndex;

    if (selectedSegment == 0) { [[NSUserDefaults
    standardUserDefaults] setInteger:1
    forKey:@"SelectWhatToDisplay"];
        //NSLog(@"%ld", (long)[[NSUserDefaults
        //standardUserDefaults]
        //integerForKey:@"SelectWhatToDisplay"]);
        [graphObject setShowFFTTo:NO];
    }
    else{ [[NSUserDefaults standardUserDefaults] setInteger:2
    forKey:@"SelectWhatToDisplay"];
        //NSLog(@"%ld", (long)[[NSUserDefaults
        //standardUserDefaults]
        //integerForKey:@"SelectWhatToDisplay"]);
        [graphObject setShowFFTTo:YES];
    }
    }
    //Clear the graph before changing what to display
    [graphObject clearDrawing];

    if(dummyCounter > 0) { //The process was already happening,
    and so needs to be restarted after the selection has been
    made [self startButtonPressed:nil];
    }
    }
    }}
    }}
-
(void)mailComposeController:(MFMailComposeViewController*)
controller didFinishWithResult:(MFMailComposeResult)result
error:(NSError*)error {

    switch (result) { case MFMailComposeResultCancelled:
    NSLog(@"Result: Canceled"); break; case
    MFMailComposeResultSaved: NSLog(@"Result: Saved"); break;
    case MFMailComposeResultSent: NSLog(@"Result: Sent"); break;
    case MFMailComposeResultFailed: NSLog(@"Result: Failed");
    break; default: NSLog(@"Result: Not Sent"); break;
    }
    }
        [self becomeFirstResponder]; [self
        dismissModalViewControllerAnimated:YES];
}
}
#warning Sometimes this does not get an updated heading - This
#is causing problems
#
- (IBAction)startButtonPressed:(id)sender {

    [graphObject clearDrawing];

    //NSLog(@"Start button pressed");
    //
    [sendMailButton setEnabled:NO];


    if(index == 0){
```

```
    [startButton setTitle:@"Reset"
    forState:UIControlStateNormal];

}
else if(index != 0){ [locationManager stopUpdatingHeading];
index = 0; counter = 0; fftCounterForGraph = 0;
}
}
// check if the hardware has a compass
    if ([CLLocationManager headingAvailable] == NO) {
            // No compass is available. This application cannot
            // function without a compass,
    // so a dialog will be displayed and no magnetic data
    // will be measured.
    self.locationManager = nil; UIAlertView *noCompassAlert
    = [[UIAlertView alloc] initWithTitle:@"No Compass
    Available!" message:@"This device does not have the
    ability to measure magnetic fields." delegate:nil
    cancelButtonTitle:@"OK" otherButtonTitles:nil];
    [noCompassAlert show];

    }
else { //the hardware does have a compass

    //initialize the array with the specified capacity
    if ([[NSUserDefaults standardUserDefaults]
    boolForKey:@"arrayCapacity"] != 0) arrayCapacity =
    [[NSUserDefaults standardUserDefaults]
    integerForKey:@"arrayCapacity"]; else{ arrayCapacity =
    512; [[NSUserDefaults standardUserDefaults]
    setInteger:arrayCapacity forKey:@"arrayCapacity"];
    }
    }
    NSLog(@"Array capacity = %d", arrayCapacity);

    RMSAddX = 0.0; RMSAddY = 0.0; RMSAddZ = 0.0; RMSCounter
    = 1;


    arrayOfXData            = new float[arrayCapacity];
    arrayOfYData            = new float[arrayCapacity];
    arrayOfZData            = new float[arrayCapacity];
    arrayOfForwardFFT_X     = new float[arrayCapacity];
    arrayOfForwardFFT_Y     = new float[arrayCapacity];
    arrayOfForwardFFT_Z     = new float[arrayCapacity];
    arrayOfMagnitudeData    = new float[arrayCapacity];


    arrayOfSummed_FFT_X     = new float[arrayCapacity];
    arrayOfSummed_FFT_Y     = new float[arrayCapacity];
    arrayOfSummed_FFT_Z     = new float[arrayCapacity];

    arrayOfSummed_FFT_X[0] = 1.0; arrayOfSummed_FFT_Y[0] =
    1.0; arrayOfSummed_FFT_Z[0] = 1.0;

    for (int i =  1 ; i < arrayCapacity; i++) {
    arrayOfSummed_FFT_X[i] = 0.0; arrayOfSummed_FFT_Y[i] =
    0.0; arrayOfSummed_FFT_Z[i] = 0.0;

    }
    }
    }
```

```
if ([[NSUserDefaults standardUserDefaults]
boolForKey:@"numberOfSummations"] == 0) {

    setNumberOfFFTsToSumOver = 0; [[NSUserDefaults
    standardUserDefaults]
    setInteger:setNumberOfFFTsToSumOver
    forKey:@"numberOfSummations"];
}
}
else{ setNumberOfFFTsToSumOver = [[NSUserDefaults
standardUserDefaults]
integerForKey:@"numberOfSummations"];
}
}
NSLog(@"Number of FFTs to sum over = %d",
setNumberOfFFTsToSumOver);

if ([[NSUserDefaults standardUserDefaults]
boolForKey:@"baselineOffset"] == 0) {

    baselineOffset = 1.0; [[NSUserDefaults
    standardUserDefaults] setFloat:baselineOffset
    forKey:@"baselineOffset"];
}
}
else{ baselineOffset = [[NSUserDefaults
standardUserDefaults] floatForKey:@"baselineOffset"];
}
}
NSLog(@"Device baseline offset = %f mG",
baselineOffset);



if ([[NSUserDefaults standardUserDefaults]
boolForKey:@"FudgeFactor"] == 0) { [[NSUserDefaults
standardUserDefaults]
setFloat:(fudgeFactorForAraySize_128 * [[NSUserDefaults
standardUserDefaults] integerForKey:@"arrayCapacity"] /
128.0) forKey:@"FudgeFactor"]; fudgeFactor =
[[NSUserDefaults standardUserDefaults]
floatForKey:@"FudgeFactor"];
}
else { fudgeFactor = [[NSUserDefaults
standardUserDefaults] floatForKey:@"FudgeFactor"];
}
}
NSLog(@"Fudge Factor (for array size %d) = %g",
arrayCapacity,fudgeFactor);

if ([[NSUserDefaults standardUserDefaults]
boolForKey:@"SincOffset"] == 0) { [[NSUserDefaults
standardUserDefaults] setFloat:0.0018
forKey:@"SincOffset"]; sincOffset = [[NSUserDefaults
standardUserDefaults] floatForKey:@"SincOffset"];
}
else { sincOffset = [[NSUserDefaults
standardUserDefaults] floatForKey:@"SincOffset"];
}
}
NSLog(@"Sinc Offset = %g", sincOffset);

//set up showing display according to user preferences
```

```
        if ([[NSUserDefaults standardUserDefaults]
        boolForKey:@"showDisplaySwitchValue"] != 0) { switch
        ([[NSUserDefaults standardUserDefaults]
        integerForKey:@"showDisplaySwitchValue"]) { case 3:
        showDisplay = YES; break;

                case 2: showDisplay = NO; break;
            }
            }        }
            }        }
        else {  //default case showDisplay = YES;
        [[NSUserDefaults standardUserDefaults] setInteger:3
        forKey:@"showDisplaySwitchValue"];
        }
        }
        }
        //set up file saving as per user preferences
        if ([[NSUserDefaults standardUserDefaults]
        boolForKey:@"fileSaveSwitchValue"] != 0) { switch
        ([[NSUserDefaults standardUserDefaults]
        integerForKey:@"fileSaveSwitchValue"]) { case 3:
        saveFiles = YES; break;

                case 2: saveFiles = NO; break;
            }
            }        }
            }        }
        else {  //default case saveFiles = NO; [[NSUserDefaults
        standardUserDefaults] setInteger:2
        forKey:@"fileSaveSwitchValue"];
        }
        }
        // heading service configuration
        locationManager.headingFilter = kCLHeadingFilterNone;

        // setup delegate callbacks
        locationManager.delegate = self;

        // start the compass
        [locationManager startUpdatingHeading];

        [stopButton setEnabled:YES];

    }
    }
    }}
    }}
// This delegate method is invoked when the location manager has
// heading data.
- (void)locationManager:(CLLocationManager *)manager
didUpdateHeading:(CLHeading *)heading {

    //NSLog(@"Index = %d", index % arrayCapacity);
    index = index % arrayCapacity;


    //    if(counter == 0) { NSLog(@"Time at 0 samples =
    //    %lu",getMStime()); }
    if(index == 0){ initialX = (float)heading.x *
    conversionFactorMicroTeslaToMilliGauss; initialY =
    (float)heading.y * conversionFactorMicroTeslaToMilliGauss;
    initialZ = (float)heading.z *
    conversionFactorMicroTeslaToMilliGauss;
```

```
    //NSLog(@"initial X = %0.3f, Y = %0.3f, Z=%0.3f\n",
    //initialX, initialY, initialZ);
    //    }
    //    }
arrayOfXData[index] = ( (float)heading.x *
conversionFactorMicroTeslaToMilliGauss ); //- initialX;
arrayOfYData[index] = ( (float)heading.y *
conversionFactorMicroTeslaToMilliGauss ); //- initialY;
arrayOfZData[index] = ( (float)heading.z *
conversionFactorMicroTeslaToMilliGauss ); //- initialZ;

if (index == (arrayCapacity - 1)) {    //To average over the
whole of the array size


    //NSLog(@"\nIndex = %d\nCounter = %d\n", index, counter);
    //
    //Do the FFTs
    //
    [self FFT_of_an_ArrayWithTheInputArray:arrayOfXData
    theOutputArray:arrayOfForwardFFT_X
    andTheNumberOfSamples:arrayCapacity]; [self
    FFT_of_an_ArrayWithTheInputArray:arrayOfYData
    theOutputArray:arrayOfForwardFFT_Y
    andTheNumberOfSamples:arrayCapacity]; [self
    FFT_of_an_ArrayWithTheInputArray:arrayOfZData
    theOutputArray:arrayOfForwardFFT_Z
    andTheNumberOfSamples:arrayCapacity];

    //Fit the function
    [self
    DivideByFittingFunctionWithTheInputArray:
    arrayOfForwardFFT_X andTheNumberOfSamples:arrayCapacity
    andTheNumberOfFFTsToSumOver:setNumberOfFFTsToSumOver];
    [self
    DivideByFittingFunctionWithTheInputArray:
    arrayOfForwardFFT_Y andTheNumberOfSamples:arrayCapacity
    andTheNumberOfFFTsToSumOver:setNumberOfFFTsToSumOver];
    [self
    DivideByFittingFunctionWithTheInputArray:
    arrayOfForwardFFT_Z andTheNumberOfSamples:arrayCapacity
    andTheNumberOfFFTsToSumOver:setNumberOfFFTsToSumOver];



    //I'm already summing the FFT here, should cut out the
    //summing happening in the graph and save half the time.
    //Discard the second half and the very first element
    for (int i  = 1 ; i < arrayCapacity/2; i++) {
        //Here i : 0.....arrayCapacity, and j : fftCounter
        //for the ith row and jth column
        arrayOfSummed_FFT_X[i] =
        ((fabs(arrayOfSummed_FFT_X[i])*fftCounterForGraph +
        arrayOfForwardFFT_X[i])/(fftCounterForGraph +
        1))/scaleFactorToConvertToSpicerRMSValues;
        //NSLog(@"Element (%d,%d) => %g\t",i,j,
        //arrayOfSummed_FFT_X[i * (setNumberOfFFTsToSumOver
        //+ 1) + j]);
        arrayOfSummed_FFT_Y[i] =
        ((fabs(arrayOfSummed_FFT_Y[i])*fftCounterForGraph +
        arrayOfForwardFFT_Y[i])/(fftCounterForGraph +
        1))/scaleFactorToConvertToSpicerRMSValues;
```

```
            arrayOfSummed_FFT_Z[i] =
            ((fabs(arrayOfSummed_FFT_Z[i])*fftCounterForGraph +
            arrayOfForwardFFT_Z[i])/(fftCounterForGraph +
            1))/scaleFactorToConvertToSpicerRMSValues;
        }
        }
        if (graphDisplayControl.selectedSegmentIndex == 1) {
            //Display the FFT here. NSLog(@"fft counter for
            //graph = %d", fftCounterForGraph);
            [graphObject plotFFTwithX:arrayOfSummed_FFT_X
            andY:arrayOfSummed_FFT_Y andZ:arrayOfSummed_FFT_Z
            andCounter:fftCounterForGraph];


            value58Hz = ((arrayCapacity/2)/64 * 58) - 1;
            value64Hz = (arrayCapacity/2) - 1;

            //Update the value on the labels with the value at
            //60Hz
            FFTXValue = fabs((arrayOfSummed_FFT_X[value58Hz] +
            arrayOfSummed_FFT_X[value58Hz - 1] +
            arrayOfSummed_FFT_X[value58Hz + 1]) -
            (arrayOfSummed_FFT_X[value64Hz] +
            arrayOfSummed_FFT_X[value64Hz - 1] +
            arrayOfSummed_FFT_X[value64Hz - 2]) -
            baselineOffset);

            FFTYValue = fabs((arrayOfSummed_FFT_Y[value58Hz] +
            arrayOfSummed_FFT_Y[value58Hz - 1] +
            arrayOfSummed_FFT_Y[value58Hz + 1]) -
            (arrayOfSummed_FFT_Y[value64Hz] +
            arrayOfSummed_FFT_Y[value64Hz - 1] +
            arrayOfSummed_FFT_Y[value64Hz - 2]) -
            baselineOffset);

            FFTZValue = fabs((arrayOfSummed_FFT_Z[value58Hz] +
            arrayOfSummed_FFT_Z[value58Hz - 1] +
            arrayOfSummed_FFT_Z[value58Hz + 1]) -
            (arrayOfSummed_FFT_Z[value64Hz] +
            arrayOfSummed_FFT_Z[value64Hz - 1] +
            arrayOfSummed_FFT_Z[value64Hz - 2]) -
            baselineOffset);

            //Display after subtracting the background and the
            //average of the last three values
//            FFTXValue = fabs((arrayOfSummed_FFT_X[value58Hz])
//            - (arrayOfSummed_FFT_X[value64Hz] +
//            arrayOfSummed_FFT_X[value64Hz - 1] +
//            arrayOfSummed_FFT_X[value64Hz - 2])/3.0 -
//            baselineOffset);
// //
//            FFTYValue = fabs((arrayOfSummed_FFT_Y[value58Hz])
//            - (arrayOfSummed_FFT_Y[value64Hz] +
//            arrayOfSummed_FFT_Y[value64Hz - 1] +
//            arrayOfSummed_FFT_Y[value64Hz - 2])/3.0 -
//            baselineOffset);
// //
//            FFTZValue = fabs((arrayOfSummed_FFT_Z[value58Hz])
//            - (arrayOfSummed_FFT_Z[value64Hz] +
//            arrayOfSummed_FFT_Z[value64Hz - 1] +
//            arrayOfSummed_FFT_Z[value64Hz - 2])/3.0 -
//            baselineOffset);
//
```

```
//
//
            [xLabel setText:[NSString stringWithFormat:@"%0.3f
            mG", FFTXValue]]; [yLabel setText:[NSString
            stringWithFormat:@"%0.3f mG", FFTYValue]]; [zLabel
            setText:[NSString stringWithFormat:@"%0.3f mG",
            FFTZValue]];

        }
        }
        fftCounterForGraph++;

        //Add the FFTs to the sum
        //
        NSLog(@"The FFT counter = %d", fftCounterForGraph);

    }
    }
    if(showDisplay) {
        // Update the graph with the new magnetic reading.
        //
        if (graphDisplayControl.selectedSegmentIndex == 0) {
        //Plotting values in time

            [graphObject updateHistoryWithX:(arrayOfXData[index]
            - initialX) y:(arrayOfYData[index] - initialY)
            z:(arrayOfZData[index] - initialZ)];

            [xLabel setText:[NSString stringWithFormat:@"%0.3f
            mG", [self returnRMSforvalue:(arrayOfXData[index] -
            initialX) andIdentifier:0]]]; [yLabel
            setText:[NSString stringWithFormat:@"%0.3f mG",
            [self returnRMSforvalue:(arrayOfYData[index] -
            initialY) andIdentifier:1]]]; [zLabel
            setText:[NSString stringWithFormat:@"%0.3f mG",
            [self returnRMSforvalue:(arrayOfZData[index] -
            initialZ) andIdentifier:2]]];
        }
        }    }
        }    }
    index++; RMSCounter++;

}
}
}
- (IBAction)stopButtonPressed:(id)sender {

    //NSLog(@"%@", arrayOfData);
    //    while (index%arrayCapacity !=0 ) {
    //     //    }
    //     //    }
    NSLog(@"\nIndex = %d", index);

    if(index > 0){

        [locationManager stopUpdatingHeading];

        /*--------------------------
         //Test code for the circular shift function
         float *dummyTestArray; dummyTestArray  =
         (float*)malloc(arrayCapacity * sizeof(float)); for (int
         i = 0; i < arrayCapacity; i++) { dummyTestArray[i] =
         (float)(i+1.0);
```

```
        }                                                                   684
        for (int i = 0; i < arrayCapacity; i++) { NSLog(@"The               685
        dummy array before shift, element %d = %f", i,                      686
        dummyTestArray[i]);                                                 687
        }                                                                   688
        [self                                                               689
        circleShiftLeftTheElementsInTheFloatArray:                          690
        dummyTestArray byNumberOfElements:index                             691
        andArraySize:arrayCapacity]; for (int i = 0; i <                    692
        arrayCapacity; i++) { NSLog(@"The dummy array after                 693
        shift by %d, element %d = %f", index, i,                            694
        dummyTestArray[i]);                                                 695
        }                                                                   696
        free(dummyTestArray); ---------------------------*/                 697
                                                                            698
       [self                                                                699
       circleShiftLeftTheElementsInTheFloatArray:arrayOfXData               700
       byNumberOfElements:index andArraySize:arrayCapacity];                701
       [self                                                                702
       circleShiftLeftTheElementsInTheFloatArray:arrayOfYData               703
       byNumberOfElements:index andArraySize:arrayCapacity];                704
       [self                                                                705
       circleShiftLeftTheElementsInTheFloatArray:arrayOfZData               706
       byNumberOfElements:index andArraySize:arrayCapacity];                707
                                                                            708
       //        NSString *stringX = [[NSString alloc] init];               709
       //         //                                                        710
       //        for (int i=0; i<arrayCapacity; i++) {                      711
       //            //NSLog(@"index: %d, amp: %.2f",i,                     712
       //            //transformedArray[i]);                                713
       //            stringX = [stringX                                     714
       //            stringByAppendingString:[NSString                      715
       //            stringWithFormat:@"\n%0.2f",                           716
       //            arrayOfXData[i]]];                                     717
       //         //        }                                               718
       //        NSLog(@"%@", stringX); NSLog(@"---");                      719
       //                                                                   720
       //                                                                   721
       std::cout << "\nThe fitted arrays are \n"; for (int i =             722
       0 ; i < arrayCapacity; i++) { std::cout <<                           723
       arrayOfSummed_FFT_X[i] << "\t" << arrayOfSummed_FFT_Y[i]             724
       << "\t" << arrayOfSummed_FFT_Z[i] << "\t"; std::cout <<              725
       "\n";                                                                726
       }                                                                    727
       }                                                                    728
#warning Look at this again - Commenting it out for testing                 729
#purposes at the moment.                                                    730
       //Alert the user when the FFT is not being                           731
       //written/summed when the compass shouts out interference            732
//       if ((arrayOfSummed_FFT_X[arrayCapacity - 1] ==                      733
//       0)&&(arrayOfSummed_FFT_X[arrayCapacity - 2] ==                      734
//       0)&&(arrayOfSummed_FFT_X[arrayCapacity - 3] == 0)) {                735
//       [[iToast makeText:NSLocalizedString(@"The saved FFT                 736
//       array may be corrupted.", @"")] show];                              737
// //        }                                                               738
// //        }                                                               739
// //        }                                                               740
#warning Need to clearly document what happend in the text file             741
#that this writes - whether it is summing up FFTs or not, and               742
#what options the user has chosen                                           743
       if (saveFiles) { [self fileSaverforArrayX:arrayOfXData               744
       arrayY:arrayOfYData arrayZ:arrayOfZData                              745
       summedFFTArray_X:arrayOfSummed_FFT_X                                 746
```

```
        summedFFTArray_Y:arrayOfSummed_FFT_Y
        summedFFTArray_Z:arrayOfSummed_FFT_Z
        fftarrayX:arrayOfForwardFFT_X
        fftarrayY:arrayOfForwardFFT_Y
        fftarrayZ:arrayOfForwardFFT_Z];
        }
        }
        }
        }
        //NSLog(@"The element at arrayCapacity %d is %f",
        //arrayCapacity, arrayOfXData[arrayCapacity]);
        //
        counter = 0;

        fftCounterForGraph = 0;

        index = 0;

        //Free all arrays
        if (*arrayOfXData) delete [] arrayOfXData; if
        (*arrayOfYData) delete [] arrayOfYData; if
        (*arrayOfZData) delete [] arrayOfZData;

        if (*arrayOfMagnitudeData) delete []
        arrayOfMagnitudeData;

        if (*arrayOfForwardFFT_X) delete [] arrayOfForwardFFT_X;
        if (*arrayOfForwardFFT_Y) delete [] arrayOfForwardFFT_Y;
        if (*arrayOfForwardFFT_Z) delete [] arrayOfForwardFFT_Z;

        if (*arrayOfSummed_FFT_X) delete [] arrayOfSummed_FFT_X;
        if (*arrayOfSummed_FFT_Y) delete [] arrayOfSummed_FFT_Y;
        if (*arrayOfSummed_FFT_Z) delete [] arrayOfSummed_FFT_Z;


        NSLog(@"\nIndex = %d", index);

        [sendMailButton setEnabled:YES];

    }
    }
    [startButton setTitle:@"Start"
    forState:UIControlStateNormal]; [stopButton setEnabled:NO];

}
}
}
-(void)
circleShiftLeftTheElementsInTheFloatArray:(float*)inputArray
byNumberOfElements:(int)numberOfElements
andArraySize:(int)sizeOfTheArray{

    /*
     This algorithm shifts to the LEFT by numberOfElements

     Starting Array: 1 2 3 4 5 6 7

     After reversing the first three elements 3 2 1 4 5 6 7

     After reversing the remaining elements 3 2 1 7 6 5 4

     Finally reverse the entire array to get the final rotated
     array 4 5 6 7 1 2 3 */
```

```
    /*
     IF you want to shift right

     Rotating n elements to the left is the same as rotating
     (size - n) elements to the right, which is what we want

     So we set up the number of elements as sizeOfTheArray -
     numberOfElements

     */

    std::reverse(inputArray, inputArray + numberOfElements);
    std::reverse(inputArray + numberOfElements, inputArray +
    sizeOfTheArray); std::reverse(inputArray, inputArray +
    sizeOfTheArray);
}
}
// This delegate method is invoked when the location managed
// encounters an error condition.
- (void)locationManager:(CLLocationManager *)manager
didFailWithError:(NSError *)error{ if ([error code] ==
kCLErrorDenied) {
        // This error indicates that the user has denied the
        // application's request to use location services.
        [manager stopUpdatingHeading];
    } else if ([error code] == kCLErrorHeadingFailure) {
        // This error indicates that the heading could not be
        // determined, most likely because of strong magnetic
        // interference.
        //     }
        //     }}
        //     }}
//Return RMSValue so far
- (float)returnRMSforvalue:(float) valuePassed
andIdentifier:(int)identifier { float returnVal;

    switch (identifier) { case 0: RMSAddX +=
    valuePassed*valuePassed; returnVal =
    sqrt(RMSAddX/RMSCounter); break;

        case 1: RMSAddY += valuePassed*valuePassed; returnVal =
        sqrt(RMSAddY/RMSCounter); break;


        default: RMSAddZ += valuePassed*valuePassed; returnVal =
        sqrt(RMSAddZ/RMSCounter); break;
    }
    }
    }
    }
    return returnVal;
}
}
}
//-----To get time in milliseconds---//
//
static unsigned long getMStime(void) {

    struct timeval time; gettimeofday(&time, NULL); return
    (time.tv_sec * 1000) + (time.tv_usec / 1000);
}
//-----------------------------------//
```

```
//
//
//
//To account for compass interference by using Apple's standard
//figure-of-8 pattern cancellation, uncomment the following
//function
//
-(BOOL)locationManagerShouldDisplayHeadingCalibration:(
CLLocationManager *)manager{ return YES;
}
}
- (void)fileSaverforArrayX: (float *)arrayToBeSavedX arrayY:
(float *)arrayToBeSavedY arrayZ: (float *)arrayToBeSavedZ
summedFFTArray_X:(float *)summedFFTArrayX
summedFFTArray_Y:(float *)summedFFTArrayY
summedFFTArray_Z:(float *)summedFFTArrayZ fftarrayX: (float
*)FFTArrayToBeSavedX fftarrayY: (float *)FFTArrayToBeSavedY
fftarrayZ: (float *)FFTArrayToBeSavedZ {  //withTimeArray: (long
*)timeArray  {

    NSArray *paths =
    NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
    NSUserDomainMask, YES); NSString *basePath = ([paths count]
    > 0) ? [paths objectAtIndex:0] : nil;

    NSString *fileName = @"ReadingTest_";

    NSString *deviceModel = [[NSUserDefaults
    standardUserDefaults] objectForKey:@"deviceModel"];

    //Add device model to filename
    fileName = [fileName stringByAppendingString:deviceModel];

    // get the current date
    NSDate *date = [NSDate date];

    // format it
    NSDateFormatter *dateFormat = [[NSDateFormatter alloc]init];
    [dateFormat setDateFormat:@"_YYYY_MM_dd_HH_mm_ss_zzz"];

    // convert it to a string
    NSString *dateString = [dateFormat stringFromDate:date];

    // free up memory
    //
    fileName = [fileName stringByAppendingString:dateString];
    fileName = [fileName stringByAppendingString:@".txt"];

    NSString *filePath = [basePath
    stringByAppendingPathComponent:fileName];

    //NSLog(@"%@", filePath);
    //
    NSString *stringToWrite = [[NSString alloc] init];

    for (int i=0; i < arrayCapacity; i++)

        { stringToWrite=[stringToWrite
        stringByAppendingString:[NSString
        stringWithFormat:@"%g\t%g\t%g\t%g\t%g\t%g\n",
        arrayToBeSavedX[i], arrayToBeSavedY[i],
        arrayToBeSavedZ[i], summedFFTArrayX[i],
        summedFFTArrayY[i], summedFFTArrayZ[i]
```

```
                                                                    ]];
                                                                    ]]
                                                                    ]]
                                                                    ]]
    //    NSLog(@"The function counter = %d", functionCounter);
    //    functionCounter += 1;
    //
    //
    [stringToWrite writeToFile:filePath atomically:YES
    encoding:NSUTF8StringEncoding error:nil];
    //    NSLog(@"%@", stringToWrite);
    //
    // }
    // }
/*This method returns the forward FFT array (using the
/accelerate framework) of an input array with
 the number of samples being the size of the array*/

- (void) FFT_of_an_ArrayWithTheInputArray:(float *)inputArray
theOutputArray:(float*)outputArray andTheNumberOfSamples:(int)
numSamples {

    //Number of Samples must be a power of 2
    //
    //Output Array
    float *transformedArray = (float
    *)malloc(sizeof(float)*numSamples);

    FFTAccelerate *fftAccel = new FFTAccelerate(numSamples);
    fftAccel->doFFTReal(inputArray, transformedArray,
    numSamples);

    outputArray[0] = transformedArray[0] *
    (float)numSamples;//Scaling factor : The first element is
    scaled by N, the others are scaled by N/2

    for (int i=1; i<numSamples; i++) {
        //NSLog(@"index: %d, amp: %.2f",i, transformedArray[i]);
        outputArray[i] = transformedArray[i] *
        (float)numSamples/2.0;//Scaling factor : The first
        element is scaled by N, the others are scaled by N/2
    }
    }
    //    NSLog(@"%@", stringToWrite); NSLog(@"---");
    delete(fftAccel);
}
}
}
}
}
/* Fitting function for the device */
- (float) fittingFunctionForDevicesForElement : (float) x{

    float fittingFunctionValue;

    if (x == 0) { fittingFunctionValue = 1;
    }
    }
    else{ float sinc = (sin(M_PI * x/fudgeFactor))/(M_PI *
    x/fudgeFactor); fittingFunctionValue =  deviceScaleFactor *
    sqrt(sinc*sinc) + sincOffset;
    }
    }
```

```
    return fittingFunctionValue;
}
}
- (void) DivideByFittingFunctionWithTheInputArray:(float
*)inputArray andTheNumberOfSamples:(int) numSamples
andTheNumberOfFFTsToSumOver:(int)numberOfFFTsToSumOver {
    //#warning Check for the log values and see if it's
    //#necessary !
    //Divide by the fitting function
    for (int i=1; i<numSamples/2; i++) {
        //Account for the division by a small number at 25 and 50
        if ((i == ((numSamples/2)/64) * 50)||(i ==
        ((numSamples/2)/64) * 25)) { break;
        }
        }
        //Compensate for the sinc function
        inputArray[i] = inputArray[i]/([self
        fittingFunctionForDevicesForElement:((i * 1.0) + 1)]);
        //For individual arrays which will later be summed.

    }
    }}
    }}
    }}
//Device orientation
//
#ifdef IOS_OLDER_THAN_6
-
(BOOL)shouldAutorotateToInterfaceOrientation:(
UIInterfaceOrientation)toInterfaceOrientation{
    //[image_signature setImage:[self
    //resizeImage:image_signature.image]];
    return (toInterfaceOrientation ==
    UIInterfaceOrientationLandscapeLeft);
}
#endif ifdef IOS_NEWER_OR_EQUAL_TO_6
-(BOOL)shouldAutorotate { return YES;
}
- (NSUInteger)supportedInterfaceOrientations {
    //[image_signature setImage:[self
    //resizeImage:image_signature.image]];
    return UIInterfaceOrientationMaskLandscapeLeft;
}
#endif
#
@end
```

C.3 The graph subroutine

```
/*
 File: GraphView.h Abstract: A custom view for plotting history
 of x, y, and z magnetic values. Version: 1.2

 */

#import "TeslameterViewController.h" import
#"CorePlot-CocoaTouch.h" import "Stack.h" import "Queue.h"
```

```
#
@class TeslameterViewController;

@interface GraphObject :
NSObject<CPTPlotDataSource,CPTAxisDelegate,CPTPlotSpaceDelegate,
CPTScatterPlotDelegate> { NSUInteger nextIndex,
nextIndexCounter;

    CPTGraphHostingView *graphHostingView;

    CPTXYGraph *graph;

    NSMutableArray *xDataForPlot, *yDataForPlot, *zDataForPlot;

    NSMutableArray *xFFTPlot, *yFFTPlot, *zFFTPlot;

    NSString *xPlotIdentifier, *yPlotIdentifier,
    *zPlotIdentifier, *windowPlotIdentifier,
    *windowPlotIdentifier2, *backgroundLinePlotIdentifier;

    BOOL showFFT;

    int plotLength;

    TeslameterViewController *viewController;
}
}
@property (nonatomic, strong) CPTGraphHostingView
*graphHostingView; @property (nonatomic, strong) CPTXYGraph
*graph;

- (id)initWithGraphHostingView:(CPTGraphHostingView
*)hostingView;


- (void)updateHistoryWithX:(float)x y:(float)y z:(float)z;

- (void)plotFFTwithX:(float *)FFTx andY:(float *)FFTy
andZ:(float *)FFTz andCounter:(int)counter;

- (void)clearDrawing;

- (void)initPlot;

- (void)setShowFFTTo:(BOOL) yesOrNo;



@end


/*
 File: GraphView.m Abstract: A custom view for plotting history
 of x, y, and z magnetic values. Version: 1.2 */

#import "GraphObject.h"
#
@interface GraphObject ()

@end

const NSUInteger kMaxDataPoints = 200 + 1;
```

```
@implementation GraphObject

@synthesize graphHostingView = graphHostingView; @synthesize
graph  = graph;

-(id)initWithGraphHostingView:(CPTGraphHostingView
*)hostingView{

    self = [super init];

    if ( self != nil ) { graphHostingView = hostingView; graph =
    nil;
    }
    }
    xDataForPlot = [[NSMutableArray alloc]
    initWithCapacity:kMaxDataPoints]; yDataForPlot =
    [[NSMutableArray alloc] initWithCapacity:kMaxDataPoints];
    zDataForPlot = [[NSMutableArray alloc]
    initWithCapacity:kMaxDataPoints];

    viewController = [[TeslameterViewController alloc] init];

    [self initPlot];

    return self;
}
}
}
#pragma mark - Chart behavior
-(void)initPlot { [self configureHost]; [self setUpGraph]; [self
changePlotRange];
}
}
//Refer
//http://stackoverflow.com/questions/6533314/how-to-plot-a-graph
//-real-time-using-coreplot
//
-(void)configureHost {

    graphHostingView.allowPinchScaling = YES;
}
}
}
-(void)setUpGraph { CGRect frame = [graphHostingView bounds];
graph = [[CPTXYGraph alloc] initWithFrame:frame]; CPTTheme
*theme = [CPTTheme themeNamed:kCPTDarkGradientTheme]; [graph
applyTheme:theme]; graphHostingView.collapsesLayers = NO; //
Setting to YES reduces GPU memory usage, but can slow
drawing/scrolling graphHostingView.hostedGraph     = graph;

    graph.paddingLeft   = 3.0; graph.paddingTop    = 3.0;
    graph.paddingRight  = 3.0; graph.paddingBottom = 3.0;


    xPlotIdentifier = [NSString stringWithFormat:@"x plot"];
    yPlotIdentifier = [NSString stringWithFormat:@"y plot"];
    zPlotIdentifier = [NSString stringWithFormat:@"z plot"];

    windowPlotIdentifier = [NSString stringWithFormat:@"Window
    plot"]; windowPlotIdentifier2 = [NSString
    stringWithFormat:@"Window plot 2"];

    backgroundLinePlotIdentifier = [NSString
```

```
stringWithFormat:@"Background line plot"];



// Setup plot space
CPTXYPlotSpace *plotSpace = (CPTXYPlotSpace
*)graph.defaultPlotSpace; plotSpace.allowsUserInteraction =
YES;

// Grid line styles
CPTMutableLineStyle *gridLineStyle = [CPTMutableLineStyle
lineStyle];


CPTMutableLineStyle *majorGridLineStyle =
[CPTMutableLineStyle lineStyle];
majorGridLineStyle.lineWidth = 0.75;
majorGridLineStyle.lineColor = [[CPTColor
colorWithGenericGray:0.2] colorWithAlphaComponent:0.75];

CPTMutableLineStyle *minorGridLineStyle =
[CPTMutableLineStyle lineStyle];
minorGridLineStyle.lineWidth = 0.25;
//    minorGridLineStyle.lineColor = [[CPTColor whiteColor]
//    colorWithAlphaComponent:0.1];
//
CPTLineCap *lineCap = [CPTLineCap sweptArrowPlotLineCap];
lineCap.size = CGSizeMake(10.0, 10.0);

// Axes Label with an automatic label policy.
//
CPTXYAxisSet *axisSet = (CPTXYAxisSet *)graph.axisSet;

CPTXYAxis *x = axisSet.xAxis; x.labelingPolicy
= CPTAxisLabelingPolicyAutomatic; x.minorTicksPerInterval
   = 5; x.preferredNumberOfMajorTicks = 5;
x.majorGridLineStyle          = majorGridLineStyle;
x.minorGridLineStyle          = minorGridLineStyle;
x.axisConstraints             = [CPTConstraints
constraintWithLowerOffset:43.0]; x.labelOffset
  = -1.0; x.titleOffset                = -0.5;



lineCap.lineStyle = x.axisLineStyle; lineCap.fill      =
[CPTFill fillWithColor:lineCap.lineStyle.lineColor];
x.axisLineCapMax  = lineCap; x.axisLineCapMin  = lineCap;
// Rotate the labels by 45 degrees
x.labelRotation = M_PI * 0.25; x.majorGridLineStyle =
gridLineStyle;


CPTXYAxis *y = axisSet.yAxis; y.labelingPolicy
= CPTAxisLabelingPolicyAutomatic; y.minorTicksPerInterval
   = 5; y.preferredNumberOfMajorTicks = 5;
y.majorGridLineStyle          = majorGridLineStyle;
y.minorGridLineStyle          = minorGridLineStyle;
y.axisConstraints             = [CPTConstraints
constraintWithLowerOffset:43.0]; y.labelOffset
  = -1.0; y.title                      = [NSString
stringWithFormat:@"Magnitude (nT)"]; y.titleOffset
     = -25.0;
```

```
lineCap.lineStyle = y.axisLineStyle; lineCap.fill      =
[CPTFill fillWithColor:lineCap.lineStyle.lineColor];
y.axisLineCapMax  = lineCap; y.axisLineCapMin  = lineCap;
y.delegate            = self;

// Set axes
graph.axisSet.axes = [NSArray arrayWithObjects:x, y, nil];

// Create a red plot area for field value in x
CPTScatterPlot *xSourceLinePlot = [[CPTScatterPlot alloc]
init]; CPTMutableLineStyle *xlineStyle         =
[CPTMutableLineStyle lineStyle]; xlineStyle.lineWidth
      = 2.0f; xlineStyle.lineColor              = [CPTColor
redColor]; xSourceLinePlot.dataLineStyle = xlineStyle;
xSourceLinePlot.identifier    = xPlotIdentifier;
xSourceLinePlot.dataSource    = self; [graph
addPlot:xSourceLinePlot];


// Create a green plot area for field value in y
CPTScatterPlot *ySourceLinePlot = [[CPTScatterPlot alloc]
init]; CPTMutableLineStyle *ylineStyle         =
[CPTMutableLineStyle lineStyle]; ylineStyle.lineWidth
      = 2.0f; ylineStyle.lineColor              = [CPTColor
greenColor]; ySourceLinePlot.dataLineStyle = ylineStyle;
ySourceLinePlot.identifier    = yPlotIdentifier;
ySourceLinePlot.dataSource    = self; [graph
addPlot:ySourceLinePlot];

// Create a blue plot area for field in z
CPTScatterPlot *zSourceLinePlot = [[CPTScatterPlot alloc]
init]; CPTMutableLineStyle *zlineStyle         =
[CPTMutableLineStyle lineStyle]; zlineStyle.miterLimit
 = 1.0f; zlineStyle.lineWidth         = 2.0f;
zlineStyle.lineColor         = [CPTColor yellowColor];
zSourceLinePlot.dataLineStyle = zlineStyle;
zSourceLinePlot.identifier    = zPlotIdentifier;
zSourceLinePlot.dataSource    = self; [graph
addPlot:zSourceLinePlot];

// Window Lines
CPTScatterPlot *windowLinePlot                  =
[[CPTScatterPlot alloc] init]; windowLinePlot.identifier
                  = windowPlotIdentifier;
CPTMutableLineStyle *windowLineStyle            =
[CPTMutableLineStyle lineStyle]; windowLineStyle.lineWidth
                  = 5.0; windowLineStyle.lineColor
              = [CPTColor orangeColor];
windowLineStyle.dashPattern                     = [NSArray
arrayWithObjects:[NSNumber numberWithInteger:10], [NSNumber
numberWithInteger:6], nil]; windowLinePlot.dataLineStyle
               = windowLineStyle;

windowLinePlot.dataSource = self; [graph
addPlot:windowLinePlot];

CPTScatterPlot *windowLinePlot2                  =
[[CPTScatterPlot alloc] init]; windowLinePlot2.identifier
                   = windowPlotIdentifier2;
CPTMutableLineStyle *windowLineStyle2            =
[CPTMutableLineStyle lineStyle]; windowLineStyle2.lineWidth
                    = 5.0; windowLineStyle2.lineColor
                = [CPTColor orangeColor];
```

```
    windowLineStyle2.dashPattern                        = [NSArray
    arrayWithObjects:[NSNumber numberWithInteger:10], [NSNumber
    numberWithInteger:6], nil]; windowLinePlot2.dataLineStyle
                    = windowLineStyle2;

    windowLinePlot2.dataSource = self; [graph
    addPlot:windowLinePlot2];

    CPTScatterPlot *backgroundLinePlot                  =
    [[CPTScatterPlot alloc] init]; backgroundLinePlot.identifier
                      = backgroundLinePlotIdentifier;
    CPTMutableLineStyle *backGroundLinePlotStyle        =
    [CPTMutableLineStyle lineStyle];
    backGroundLinePlotStyle.lineWidth                   = 2.5;
    backGroundLinePlotStyle.lineColor                   = [CPTColor
    whiteColor]; backGroundLinePlotStyle.dashPattern
     = [NSArray arrayWithObjects:[NSNumber
    numberWithInteger:25], [NSNumber numberWithInteger:6], nil];
    backgroundLinePlot.dataLineStyle                    =
    backGroundLinePlotStyle;

    backgroundLinePlot.dataSource = self; [graph
    addPlot:backgroundLinePlot];


}
}
-(void)changePlotRange {
    // Setup plot space
    CPTXYPlotSpace *plotSpace = (CPTXYPlotSpace
    *)graph.defaultPlotSpace;

    if(showFFT) {
        //Arjun : This is where the graph is initially positioned
        plotSpace.xRange                = [CPTPlotRange
        plotRangeWithLocation:CPTDecimalFromFloat(30.0)
        length:CPTDecimalFromFloat(34.0)]; plotSpace.yRange
                    = [CPTPlotRange
        plotRangeWithLocation:CPTDecimalFromFloat(-30.0)
        length:CPTDecimalFromFloat(60.0)];
    }
    }
    else {
        //Arjun : This is where the graph is initially positioned
        plotSpace.xRange                = [CPTPlotRange
        plotRangeWithLocation:CPTDecimalFromFloat(-6.0)
        length:CPTDecimalFromFloat(16.0)]; plotSpace.yRange
                    = [CPTPlotRange
        plotRangeWithLocation:CPTDecimalFromFloat(-30.0)
        length:CPTDecimalFromFloat(60.0)];
    }
    }
    }
    }}
    }}
#pragma mark - pragma mark Plot Data Source Methods
#
-(NSUInteger)numberOfRecordsForPlot:(CPTPlot *)plot { if
(plot.identifier == windowPlotIdentifier) { return plotLength;
    }
    else if (plot.identifier == windowPlotIdentifier2) { return
    plotLength;
    }
```

```
    //NSLog(@"numberOfRecordsForPlot accessed");
    if (showFFT == NO) return [xDataForPlot count];

    //else
    return [xFFTPlot count];
}
}
-(NSNumber *)numberForPlot:(CPTPlot *)plot
field:(NSUInteger)fieldEnum recordIndex:(NSUInteger)index {
NSNumber *valueToBeReturned;

    int plotPoint55 = (plotLength/64)*55 - 1; int plotPoint56 =
    plotPoint55 + 1; int plotPoint63 = (plotLength/64)*63 - 1;
    int plotPoint64 = plotPoint63 + 1;

    if (showFFT == NO) { //Display the values, not the FFT
                         //NSLog(@"abcde");
        switch (fieldEnum) { case CPTScatterPlotFieldX:
        valueToBeReturned = [NSNumber
        numberWithUnsignedInteger:index + nextIndexCounter -
        xDataForPlot.count]; break;


            case CPTScatterPlotFieldY: if ([(NSString
            *)plot.identifier isEqualToString:xPlotIdentifier])
            { valueToBeReturned = [xDataForPlot
            objectAtIndex:index];
                }
                else if ([(NSString *)plot.identifier
                isEqualToString:yPlotIdentifier]) {
                valueToBeReturned = [yDataForPlot
                objectAtIndex:index];
                }
                else if ([(NSString *)plot.identifier
                isEqualToString:zPlotIdentifier]) {
                valueToBeReturned = [zDataForPlot
                objectAtIndex:index];
                }
                break;

            default: break;
        }
        }    }
        }    }
    if (showFFT == YES) { //Display the FFT switch (fieldEnum) {
    case CPTScatterPlotFieldX: valueToBeReturned = [NSNumber
    numberWithFloat:((64.0*index/plotLength) + 1)]; break;

            case CPTScatterPlotFieldY: if (index >=
            plotLength/2) { //Return only the values that we
            care about - between 32 & 64 Hz for now

                    if ([(NSString *)plot.identifier
                    isEqualToString:xPlotIdentifier]) {
                    valueToBeReturned = [xFFTPlot
                    objectAtIndex:index];
                    }
                    else if ([(NSString *)plot.identifier
                    isEqualToString:yPlotIdentifier]) {
                    valueToBeReturned = [yFFTPlot
                    objectAtIndex:index];
                    }
                    else if ([(NSString *)plot.identifier
```

```
                    isEqualToString:zPlotIdentifier]) {                 328
                    valueToBeReturned = [zFFTPlot                       329
                    objectAtIndex:index];                               330
                    }                                                   331
                    else if ([(NSString *)plot.identifier               332
                    isEqualToString:windowPlotIdentifier]) { if         333
                    (index == plotPoint55) { valueToBeReturned =        334
                    [NSNumber numberWithFloat:0.0]; break;              335
                        }                                               336
                        else if (index == plotPoint56) {                337
                        valueToBeReturned = [NSNumber                   338
                        numberWithFloat:1000]; break;                   339
                        }                                               340
                        }                                               341
                        //else                                          342
                        break;                                          343
                    }                                                   344
                    else if ([(NSString *)plot.identifier               345
                    isEqualToString:windowPlotIdentifier2]) { if        346
                    (index == plotPoint63) { valueToBeReturned =        347
                    [NSNumber numberWithFloat:1000]; break;             348
                        }                                               349
                        else if (index == plotPoint64) {                350
                        valueToBeReturned = [NSNumber                   351
                        numberWithFloat:0.0]; break;                    352
                        }                                               353
                        }                                               354
                        //else                                          355
                        break;                                          356
                    }                                                   357
                    }                                                   358
//                    else if ([(NSString *)plot.identifier             359
//                    isEqualToString:                                  360
//                    backgroundLinePlotIdentifier]) {                  361
// //                                                                    362
//                        if (index == 0) { valueToBeReturned =         363
//                        [NSNumber numberWithFloat:1000]; break;       364
// //                        }                                          365
//                        else if (index == plotPoint64) {              366
//                        valueToBeReturned = [NSNumber                 367
//                        numberWithFloat:0.0]; break;                  368
// //                        }                                          369
// //                        }//                                        370
//                        //else                                        371
//                        break;                                        372
// //                    }                                             373
// //                    }                                             374
// //                    }                 }                           375
// //                    }                 }                           376
            default: break;                                             377
        }                                                               378
        }    }                                                          379
        }    }                                                          380
    return valueToBeReturned;                                           381
}                                                                       382
}                                                                       383
}                                                                       384
- (void)updateHistoryWithX:(float)x y:(float)y z:(float)z {            385
                                                                        386
    //    NSLog(@"Before increment -> nextIndex = %d,                   387
    //    nextIndexCounter = %d", nextIndex, nextIndexCounter);         388
    CPTPlot *xPlot = [graph plotWithIdentifier:xPlotIdentifier];        389
    CPTPlot *yPlot = [graph plotWithIdentifier:yPlotIdentifier];        390
```

```
    CPTPlot *zPlot = [graph plotWithIdentifier:zPlotIdentifier];

    if ((xPlot != nil) && (yPlot != nil) && (zPlot != nil)) {
        //NSLog(@"Log from plot check");
        //
        if (xDataForPlot.count >= kMaxDataPoints) {
        [xDataForPlot removeObjectAtIndex:0]; [yDataForPlot
        removeObjectAtIndex:0]; [zDataForPlot
        removeObjectAtIndex:0];

            [xPlot deleteDataInIndexRange:NSMakeRange(0, 1)];
            [yPlot deleteDataInIndexRange:NSMakeRange(0, 1)];
            [zPlot deleteDataInIndexRange:NSMakeRange(0, 1)];

        }
        }
        CPTXYPlotSpace *plotSpace = (CPTXYPlotSpace
        *)graph.defaultPlotSpace; NSUInteger location       =
        (nextIndexCounter >= kMaxDataPoints ? nextIndexCounter -
        kMaxDataPoints + 1 : 0); plotSpace.xRange =
        [CPTPlotRange
        plotRangeWithLocation:CPTDecimalFromUnsignedInteger(
        location)
        length:CPTDecimalFromUnsignedInteger(kMaxDataPoints -
        1)];

        [xDataForPlot addObject:[NSNumber numberWithFloat:x]];
        [yDataForPlot addObject:[NSNumber numberWithFloat:y]];
        [zDataForPlot addObject:[NSNumber numberWithFloat:z]];

        // Advance the index counter.
        nextIndexCounter ++;

        [xPlot insertDataAtIndex:xDataForPlot.count - 1
        numberOfRecords:1]; [yPlot
        insertDataAtIndex:yDataForPlot.count - 1
        numberOfRecords:1]; [zPlot
        insertDataAtIndex:zDataForPlot.count - 1
        numberOfRecords:1];
    }
    }}
    }}
- (void)plotFFTwithX:(float *)FFTx andY:(float *)FFTy
andZ:(float *)FFTz andCounter:(int)counter {

    CPTPlot *xPlot = [graph plotWithIdentifier:xPlotIdentifier];
    CPTPlot *yPlot = [graph plotWithIdentifier:yPlotIdentifier];
    CPTPlot *zPlot = [graph plotWithIdentifier:zPlotIdentifier];

    if (counter == 0) { [xFFTPlot insertObject:[NSNumber
    numberWithFloat:0] atIndex:0]; NSLog(@"xFFTPlot(%d) = %@",
    0, [xFFTPlot objectAtIndex:0]); [yFFTPlot
    insertObject:[NSNumber numberWithFloat:0] atIndex:0];
    [zFFTPlot insertObject:[NSNumber numberWithFloat:0]
    atIndex:0]; for (int i = 1; i < plotLength; i++) {
            //NSLog(@"xFFTPlot(%d) = %@ + %f", i, [xFFTPlot
            //objectAtIndex:i], FFTx[i]);
            float xf = (100*FFTx[i]); float yf = (100*FFTy[i]);
            float zf = (100*FFTz[i]); [xFFTPlot
            insertObject:[NSNumber numberWithFloat:fabs(xf)]
            atIndex:i];
            //NSLog(@"xFFTPlot(%d) = %@", i, [xFFTPlot
            //objectAtIndex:i]);
```

```
            [yFFTPlot insertObject:[NSNumber                                    454
            numberWithFloat:fabs(yf)] atIndex:i]; [zFFTPlot                     455
            insertObject:[NSNumber numberWithFloat:fabs(zf)]                    456
            atIndex:i];                                                         457
                                                                                458
        }                                                                       459
        }                                                                       460
        [graph reloadData];                                                     461
                                                                                462
    }                                                                           463
    else {                                                                      464
                                                                                465
        NSLog(@"xFFTPlot(%d) = %@", 0, [xFFTPlot                                466
        objectAtIndex:0]); for (int i = 1; i < plotLength; i++) {               467
            //NSLog(@"xFFTPlot(%d) = (%@ + %f) / %d", i,                        468
            //[xFFTPlot objectAtIndex:i], FFTx[i], counter + 1);                469
#warning We might need to incorporate a log scale or scaling                    470
#factors of some sort                                                           471
            float xf = (100*FFTx[i]); float yf = (100*FFTy[i]);                 472
            float zf = (100*FFTz[i]); [xFFTPlot                                 473
            removeObjectAtIndex:i]; [yFFTPlot                                   474
            removeObjectAtIndex:i]; [zFFTPlot                                   475
            removeObjectAtIndex:i]; [xFFTPlot                                   476
            insertObject:[NSNumber numberWithFloat:xf]                          477
            atIndex:i];                                                         478
            //NSLog(@"xFFTPlot(%d) = %@", i, [xFFTPlot                          479
            //objectAtIndex:i]);                                                480
            [yFFTPlot insertObject:[NSNumber numberWithFloat:yf]                481
            atIndex:i]; [zFFTPlot insertObject:[NSNumber                        482
            numberWithFloat:zf] atIndex:i];                                     483
        }                                                                       484
        }                                                                       485
        }                                                                       486
        if ((xPlot != nil) && (yPlot != nil) && (zPlot != nil))                 487
        {                                                                       488
                                                                                489
            [graph reloadData];                                                 490
                                                                                491
            [xPlot deleteDataInIndexRange:NSMakeRange(0,                        492
            plotLength)]; [yPlot                                                493
            deleteDataInIndexRange:NSMakeRange(0, plotLength)];                 494
            [zPlot deleteDataInIndexRange:NSMakeRange(0,                        495
            plotLength)];                                                       496
                                                                                497
            [xPlot insertDataAtIndex:0                                          498
            numberOfRecords:plotLength]; [yPlot                                 499
            insertDataAtIndex:0 numberOfRecords:plotLength];                    500
            [zPlot insertDataAtIndex:0                                          501
            numberOfRecords:plotLength];                                        502
        }                                                                       503
        }    }                                                                  504
    //    [viewController.xLabel setText:[NSString                              505
    //    stringWithFormat:@"%f", [self                                         506
    //    returnRMSForArray:xFFTPlot]]]; [viewController.yLabel                 507
    //    setText:[NSString stringWithFormat:@"%f", [self                       508
    //    returnRMSForArray:yFFTPlot]]]; [viewController.zLabel                 509
    //    setText:[NSString stringWithFormat:@"%f", [self                       510
    //    returnRMSForArray:zFFTPlot]]];                                        511
    //[graph reloadData];                                                       512
    //}                                                                         513
    //}                                                                         514
    //}                                                                         515
    //}                                                                         516
```

```
//A function to clear everything such that the drawing is
//started from scratch again
-(void) clearDrawing { if ([xDataForPlot count] && [yDataForPlot
count] && [zDataForPlot count]) { [xDataForPlot
removeAllObjects]; [yDataForPlot removeAllObjects];
[zDataForPlot removeAllObjects];
    }
    }
    if ([xFFTPlot count] && [yFFTPlot count] && [zFFTPlot
    count]) { [xFFTPlot removeAllObjects]; [yFFTPlot
    removeAllObjects]; [zFFTPlot removeAllObjects];
    }
    }
    graphHostingView.hostedGraph = nil;
    graphHostingView.hostedGraph = graph;

    if (showFFT) { plotLength = [[NSUserDefaults
    standardUserDefaults] integerForKey:@"arrayCapacity"]/2;

        xFFTPlot = [[NSMutableArray alloc]
        initWithCapacity:plotLength]; yFFTPlot =
        [[NSMutableArray alloc] initWithCapacity:plotLength];
        zFFTPlot = [[NSMutableArray alloc]
        initWithCapacity:plotLength];

        for (int i = 0; i < [xFFTPlot count]; i++) { [xFFTPlot
        addObject:[NSNumber numberWithFloat:0]]; [yFFTPlot
        addObject:[NSNumber numberWithFloat:0]]; [zFFTPlot
        addObject:[NSNumber numberWithFloat:0]];
        }
        }
        //NSLog(@"Array after values = %@", xFFTPlot);
        //    }
        //    }
    [graph reloadData];

    nextIndex = 0; nextIndexCounter = 0;

}
}
- (void)setShowFFTTo:(BOOL)yesOrNo{ showFFT = yesOrNo;

    CPTXYAxisSet *axisSet = (CPTXYAxisSet *)graph.axisSet;

    CPTXYAxis *x = axisSet.xAxis;

    if (showFFT) { x.title = [NSString
    stringWithFormat:@"Frequency (Hz)"];
    }
    }
    else { x.title = [NSString stringWithFormat:@""];
    }
    }
    //    CPTXYPlotSpace *plotSpace = (CPTXYPlotSpace
    //    *)graph.defaultPlotSpace;
    //     //
    //    if (showFFT == YES) {
    //        //Set up a log scale
    //        plotSpace.xScaleType = CPTScaleTypeLinear; // this
    //        is the default plotSpace.yScaleType =
    //        CPTScaleTypeLog;
    //     //    }
    //     //    }    //
```

```
    //    else {
    //        //Set up a linear scale
    //        plotSpace.xScaleType = CPTScaleTypeLinear; // this
    //        is the default plotSpace.yScaleType =
    //        CPTScaleTypeLinear;
    //     //    }
    //     //    }
    //     //    }}
    //     //    }}
    //     //    }}
    //     //    }}
    //     //    }}
    //     //    }}
#warning Need to set this up
//Touch interaction on plot points -
//(void)scatterPlot:(CPTScatterPlot *)plot
//plotSymbolWasSelectedAtRecordIndex:(NSUInteger)index
//
//{
//    if ([(NSString *)plot.identifier
//    isEqualToString:kLinePlot])
// //
//    { touchPlotSelected = YES; [self
//    applyHighLightPlotColor:plot]; if ([delegate
//    respondsToSelector:@selector(linePlot:indexLocation:)])
//    [delegate linePlot:self indexLocation:index];
// //    }
// //    }//}
// //    }//}
#pragma mark - pragma mark Axis Delegate Methods
#
-(BOOL)axis:(CPTAxis *)axis
shouldUpdateAxisLabelsAtLocations:(NSSet *)locations { static
CPTTextStyle *positiveStyle = nil; static CPTTextStyle
*negativeStyle = nil;

    NSNumberFormatter *formatter = axis.labelFormatter; CGFloat
    labelOffset          = axis.labelOffset; NSDecimalNumber
    *zero        = [NSDecimalNumber zero];

    NSMutableSet *newLabels = [NSMutableSet set];

    for ( NSDecimalNumber *tickLocation in locations ) {
    CPTTextStyle *theLabelTextStyle;

        if ( [tickLocation isGreaterThanOrEqualTo:zero] ) { if (
        !positiveStyle ) { CPTMutableTextStyle *newStyle =
        [axis.labelTextStyle mutableCopy]; newStyle.color =
        [CPTColor greenColor]; positiveStyle  = newStyle;
            }
            theLabelTextStyle = positiveStyle;
        }
        else { if ( !negativeStyle ) { CPTMutableTextStyle
        *newStyle = [axis.labelTextStyle mutableCopy];
        newStyle.color = [CPTColor redColor]; negativeStyle  =
        newStyle;
            }
            theLabelTextStyle = negativeStyle;
        }
        }
        NSString *labelString       = [formatter
        stringForObjectValue:tickLocation]; CPTTextLayer
        *newLabelLayer = [[CPTTextLayer alloc]
```

```
        initWithText:labelString style:theLabelTextStyle];

        CPTAxisLabel *newLabel = [[CPTAxisLabel alloc]
        initWithContentLayer:newLabelLayer];
        newLabel.tickLocation = tickLocation.decimalValue;
        newLabel.offset       = labelOffset;

        [newLabels addObject:newLabel];

    }
    }
    axis.axisLabels = newLabels;

    return NO;
}
}
- (float)trapezoidalIntegrateDataFromArray:(NSMutableArray
*)theArray {

    int plotPoint55 = (plotLength/64)*55 - 1;
    //int plotPoint51 = plotPoint50 + 1;
    int plotPoint63 = (plotLength/64)*63 - 1; int plotPoint64 =
    plotPoint63 + 1; float retVal = 0.0;

    for (int i = plotPoint55 ; i <= plotPoint64; i++) { retVal =
    retVal + ([[theArray objectAtIndex:i] floatValue] -
    [[theArray objectAtIndex:i-1] floatValue]);
    }
    }
    }
    }
    return retVal*0.5;
}
}
- (float)returnRMSForArray: (NSMutableArray *)theArray {

    float retVal = 0.0;

    int i  = (plotLength/64)*55 - 1;

    while (i < plotLength) { retVal += [[theArray
    objectAtIndex:i] floatValue] * [[theArray objectAtIndex:i]
    floatValue];
    }
    }
    retVal = sqrtf(retVal/((plotLength -
    ((plotLength/64)*55))));

    return retVal;

}
}
}
}
}
@end
```

C.4 The settings page

```
//
//  SettingsViewController.h Teslameter
// //
//  Created by Arjun Shivanand Kannan on 11/9/12.
// //
//  Using the free framework In-App Settings Kit
//
//#import <UIKit/UIKit.h> import
//#"IASKAppSettingsViewController.h" import
//#"TeslameterViewController.h"
//#
@class TeslameterViewController;

@interface SettingsViewController : UIViewController{

    NSString *memorySizeLabelText;

    NSString *windowSizeLabelText;

    NSString *backgroundSubtractionLabelText;

    int numberOfSamples;

    IBOutlet UISlider *memorySizeSlider;


    IBOutlet UISlider *backgroundSubtractSlider;

    TeslameterViewController *teslameterViewController;
}
}
//Function to return the variables from the settings page to the
//main page + (SettingsViewController *) sharedInstance;
//
@property (strong, nonatomic) IBOutlet UISwitch *fileSaveSwitch;
@property (strong, nonatomic) IBOutlet UISwitch
*showDisplaySwitch; @property (strong, nonatomic) IBOutlet
UILabel *memorySizeLabel; @property (strong, nonatomic) IBOutlet
UILabel *backgroundSubtractionLabel; @property (strong,
nonatomic) TeslameterViewController *teslameterViewController;
@property (strong, nonatomic) IBOutlet UIScrollView
*settingsScrollView;

- (IBAction)clearDataButtonPressed:(id)sender; -
(IBAction)saveDataSwitchValueChanged:(id)sender; -
(IBAction)displayReadingsSwitchValueChanged:(id)sender; -
(IBAction)memorySizeSliderValueChanged:(id)sender; -
(IBAction)backgroundSubtractionSliderValueChanged:(id)sender; -
(IBAction)resetBackgroundSubtractButtonPressed:(id)sender;

@end
```

```
//
//  SettingsViewController.m Teslameter
// //
//  Created by Arjun Shivanand Kannan on 11/9/12.
// //
// ////
```

```
// ////
#import "SettingsViewController.h"
//#import "iToast.h" import "TeslameterViewController.h"
//#
//#
//#
@interface SettingsViewController ()

@end

@implementation SettingsViewController

@synthesize settingsScrollView;


- (id)initWithNibName:(NSString *)nibNameOrNil bundle:(NSBundle
*)nibBundleOrNil { self = [super initWithNibName:nibNameOrNil
bundle:nibBundleOrNil]; if (self) {
        // Custom initialization
        //     }
    return self;
}
}
@synthesize memorySizeLabel; @synthesize showDisplaySwitch;
@synthesize fileSaveSwitch; @synthesize
teslameterViewController; @synthesize
backgroundSubtractionLabel;


- (void)viewDidLoad { [super viewDidLoad];
    // Do any additional setup after loading the view from its
    // nib.
    self.title = @"Settings";

    //---Set the viewable frame of the scroll view---Adapted to
    //the various screens using
    //http://stackoverflow.com/questions/12645506/xcode-4-5-
    //iphone-5-breaks-my-uiscrollview
    settingsScrollView.frame = CGRectMake(0, 0, [[UIScreen
    mainScreen] bounds].size.height, [[UIScreen mainScreen]
    bounds].size.width);
    //interchanged height and width above for landscape
    //scrollview
    //
    //---set the content size of the scroll view---
    [settingsScrollView setContentSize:CGSizeMake([[UIScreen
    mainScreen] bounds].size.width, 700)];

    //Add the scroll view to the view
    [self.view addSubview:settingsScrollView];

    //    [[NSUserDefaults standardUserDefaults]
    //    setInteger:progressOfSlider
    //    forKey:@"memorySizeSliderPosition"];
    //
    //
    if ([[NSUserDefaults standardUserDefaults]
    boolForKey:@"memorySizeSliderPosition"] != 0)
    memorySizeSlider.value = [[NSUserDefaults
    standardUserDefaults]
    integerForKey:@"memorySizeSliderPosition"]; else{
    memorySizeSlider.value = 9; [[NSUserDefaults
    standardUserDefaults] setInteger:(int)memorySizeSlider.value
```

```
forKey:@"memorySizeSliderPosition"];
}
}
}
if ([[NSUserDefaults standardUserDefaults]
boolForKey:@"arrayCapacity"] == 0) {

    [[NSUserDefaults standardUserDefaults] setInteger:512
    forKey:@"arrayCapacity"];
}
}
memorySizeLabelText = [NSString stringWithFormat:@"Sample
history size = %d", [[NSUserDefaults standardUserDefaults]
integerForKey:@"arrayCapacity"]]; memorySizeLabel.text =
memorySizeLabelText;

if ([[NSUserDefaults standardUserDefaults]
boolForKey:@"baselineOffset"] == 0) {
backgroundSubtractSlider.value = 1.0; [[NSUserDefaults
standardUserDefaults]
setFloat:backgroundSubtractSlider.value
forKey:@"baselineOffset"];
}
}
else{ backgroundSubtractSlider.value = [[NSUserDefaults
standardUserDefaults] floatForKey:@"baselineOffset"];
}
}
backgroundSubtractionLabelText = [NSString
stringWithFormat:@"Background Subtraction from phone = %0.3f
mG", [[NSUserDefaults standardUserDefaults]
floatForKey:@"baselineOffset"]];
backgroundSubtractionLabel.text =
backgroundSubtractionLabelText;


// If the defaults for the switches are not set, set them up
// here Avoids checking for Boolean of Boolean Value = 2
// <--> Switch OFF, Value = 3 <--> Switch ON
//
if  ([[NSUserDefaults standardUserDefaults]
boolForKey:@"fileSaveSwitchValue"] == 0) { if
(fileSaveSwitch.on) [[NSUserDefaults standardUserDefaults]
setInteger:3 forKey:@"fileSaveSwitchValue"]; else
[[NSUserDefaults standardUserDefaults] setInteger:2
forKey:@"fileSaveSwitchValue"];
}
}
else { switch ([[NSUserDefaults standardUserDefaults]
integerForKey:@"fileSaveSwitchValue"]) { case 3:
fileSaveSwitch.on = YES; break;

        case 2: fileSaveSwitch.on = NO; break;
    }
    }    }
    }    }
if  ([[NSUserDefaults standardUserDefaults]
boolForKey:@"showDisplaySwitchValue"] == 0) { if
(showDisplaySwitch.on) [[NSUserDefaults
standardUserDefaults] setInteger:3
forKey:@"showDisplaySwitchValue"]; else [[NSUserDefaults
standardUserDefaults] setInteger:2
forKey:@"showDisplaySwitchValue"];
```

```
    }
    }
    else { switch ([[NSUserDefaults standardUserDefaults]
    integerForKey:@"showDisplaySwitchValue"]) { case 3:
    showDisplaySwitch.on = YES; break;

            case 2: showDisplaySwitch.on = NO; break;
        }
        }    }
        }    }}
        }    }}
- (void)didReceiveMemoryWarning { [super
didReceiveMemoryWarning];
    // Dispose of any resources that can be recreated.
    // }
    // }
//Function to delete the documents when the clear data button is
//pressed
- (IBAction)clearDataButtonPressed:(id)sender {

    NSArray *paths =
    NSSearchPathForDirectoriesInDomains(NSDocumentDirectory ,
    NSUserDomainMask, YES); NSString *documentsDir = [paths
    objectAtIndex:0]; NSFileManager *fileManager =
    [NSFileManager defaultManager]; NSError *error = nil;

    for (NSString *file in [fileManager
    contentsOfDirectoryAtPath:documentsDir error:&error]) {
    NSString *filePath = [documentsDir
    stringByAppendingPathComponent:file];

        BOOL fileDeleted = [fileManager
        removeItemAtPath:filePath error:&error];

        if (fileDeleted != YES || error != nil) {
            // Deal with the error...
            //             }
            //             }
            //             }        }
    [[iToast makeText:NSLocalizedString(@"All files have been
    cleared.", @"")] show];
    //Acknowledge iToast in settings Page
    //}
    //}
#warning This could be useful -->
#http://stackoverflow.com/questions/7341859/how-to-check-if-
#folder-is-empty-and-instantiate-file-names-inside-the-folder-in
#
- (IBAction)saveDataSwitchValueChanged:(id)sender { if
(fileSaveSwitch.on) [[NSUserDefaults standardUserDefaults]
setInteger:3 forKey:@"fileSaveSwitchValue"]; else
[[NSUserDefaults standardUserDefaults] setInteger:2
forKey:@"fileSaveSwitchValue"];
}
}
- (IBAction)displayReadingsSwitchValueChanged:(id)sender { if
(showDisplaySwitch.on) [[NSUserDefaults standardUserDefaults]
setInteger:3 forKey:@"showDisplaySwitchValue"]; else
[[NSUserDefaults standardUserDefaults] setInteger:2
forKey:@"showDisplaySwitchValue"];
}
}
- (IBAction)memorySizeSliderValueChanged:(id)sender {
```

```
    UISlider *localSliderDeclaration = (UISlider *)sender;
    //typecast slider progress
    int progressOfSlider = (int)(localSliderDeclaration.value +
    0.5f); [[NSUserDefaults standardUserDefaults]
    setInteger:progressOfSlider
    forKey:@"memorySizeSliderPosition"]; int
    memoryCapacitySetByThisAction = (int) pow(2.0,
    (double)[[NSUserDefaults standardUserDefaults]
    integerForKey:@"memorySizeSliderPosition"]);

    [[NSUserDefaults standardUserDefaults]
    setInteger:memoryCapacitySetByThisAction
    forKey:@"arrayCapacity"];

    memorySizeLabelText = [NSString stringWithFormat:@"Sample
    history size = %d", [[NSUserDefaults standardUserDefaults]
    integerForKey:@"arrayCapacity"]];

    memorySizeLabel.text = memorySizeLabelText;

    [[NSUserDefaults standardUserDefaults]
    setFloat:(fudgeFactorForAraySize_128 * [[NSUserDefaults
    standardUserDefaults] integerForKey:@"arrayCapacity"] /
    128.0) forKey:@"FudgeFactor"];
}
}
}
}
- (IBAction)backgroundSubtractionSliderValueChanged:(id)sender {

    UISlider *localSliderDeclaration = (UISlider *)sender;

    float progressOfSlider = localSliderDeclaration.value;

    [[NSUserDefaults standardUserDefaults]
    setFloat:progressOfSlider forKey:@"baselineOffset"];

    backgroundSubtractionLabelText = [NSString
    stringWithFormat:@"Background Subtraction from phone = %0.3f
    mG", [[NSUserDefaults standardUserDefaults]
    floatForKey:@"baselineOffset"]];

    backgroundSubtractionLabel.text =
    backgroundSubtractionLabelText;

}
}
- (IBAction)resetBackgroundSubtractButtonPressed:(id)sender {

    backgroundSubtractSlider.value = 1.0;

    [[NSUserDefaults standardUserDefaults]
    setFloat:backgroundSubtractSlider.value
    forKey:@"baselineOffset"];

    backgroundSubtractionLabelText = [NSString
    stringWithFormat:@"Background Subtraction from phone = %0.3f
    mG", [[NSUserDefaults standardUserDefaults]
    floatForKey:@"baselineOffset"]];
    backgroundSubtractionLabel.text =
    backgroundSubtractionLabelText;
```

```
}
}
- (void)viewDidUnload { [self setFileSaveSwitch:nil]; [self
setShowDisplaySwitch:nil]; [self setMemorySizeLabel:nil];
memorySizeSlider = nil; [self setSettingsScrollView:nil];
settingsScrollView = nil; [self
setBackgroundSubtractionLabel:nil]; backgroundSubtractSlider =
nil; [super viewDidUnload];
}
}
//Device orientation
//
#ifdef IOS_OLDER_THAN_6
-
(BOOL)shouldAutorotateToInterfaceOrientation:(
UIInterfaceOrientation)toInterfaceOrientation{
    //[image_signature setImage:[self
    //resizeImage:image_signature.image]];
    return (toInterfaceOrientation ==
    UIInterfaceOrientationLandscapeLeft);
}
#endif ifdef IOS_NEWER_OR_EQUAL_TO_6
-(BOOL)shouldAutorotate { return YES;
}
- (NSUInteger)supportedInterfaceOrientations {
    //[image_signature setImage:[self
    //resizeImage:image_signature.image]];
    return UIInterfaceOrientationMaskLandscapeLeft;
}
}
-(void)willAnimateRotationToInterfaceOrientation:(
UIInterfaceOrientation)toInterfaceOrientation
duration:(NSTimeInterval)duration { [super
willAnimateRotationToInterfaceOrientation:toInterfaceOrientation
duration:duration]; if (toInterfaceOrientation ==
UIInterfaceOrientationLandscapeLeft
        ||  toInterfaceOrientation ==
        ||  UIInterfaceOrientationLandscapeRight)
        { CGRect rect = self.view.frame; rect.size.width =
        self.view.frame.size.width+245; rect.size.height =
        self.view.frame.size.height+245;
        self.settingsScrollView.frame = rect;
        }
        }}
#endif
#
@end
```

BIBLIOGRAPHY

[1] David A Muller, Earl J Kirkland, Malcolm G Thomas, John L Grazul, Lena Fitting, and Matthew Weyland. Room design for high-performance electron microscopy. *Ultramicroscopy*, 106(11-12):1033–40, 2006.

[2] D A Muller and J Grazul. Optimizing the environment for sub-0.2 nm scanning transmission electron microscopy. *Journal of electron microscopy*, 50(3):219–26, January 2001.

[3] Alasdair Allan. *Basic Sensors in iOS*. O'Reilly, 1st edition, 2011.

[4] Edward Ramsden. *Hall-Effect Sensors: Theory and Application*, volume 2. Elsevier, 2nd edition, 2006.

[5] E.H. Hall. On a New Action of the Magnet on Electric Currents. *American Journal of Mathematics*, 2:287–292, 1879.

[6] R.S. Popovic. *Hall Effect Devices*. CRC Press, 2nd edition, 2010.

[7] Joyce Van de Vegte. *Fundamentals of Digital Signal Processing*. Prentice Hall, 3rd edition, 2001.

[8] Agilent Technologies. The Fundamentals of Signal Analysis. 2009.

[9] WH Press, SA Teukolsky, WT Vetterling, and BP Flannery. *Numerical recipes in C+: the art of scientific computing*. Cambridge University Press, 3rd edition, 2009.

[10] Apple. Object oriented programming with Objective-C. 2007.

[11] Joe Conway. *iOS Programming: The Big Nerd Ranch Guide*. 3rd edition, 2012.

[12] Apple. Teslameter, 2009.

[13] Drew McCormack, Barry Wark, and Brad Larson. Core Plot for iOS, 2012.

[14] David J. Griffiths. *Introduction to Electrodynamics*. Prentice Hall, 3rd edition, 1999.

[15] Apple. Using Fourier Transforms with the vDSP Package, 2012.

[16] Amin Haji Abolhassani Hamed Ketabdar, Amirhossein Jahanbekam, Kamer Ali Yüksel, Tobias Hirsch. MagiMusic : Using Embedded Compass (Magnetic) Sensor for Touch-less Gesture Based Interaction with Digital Music Instruments in Mobile Devices. In *Proceedings of the 5th International Conference on Tangible and Embedded Interaction 2011, Funchal, Madeira, Portugal*, pages 4–7, 2011.