ESE 498

Hypnolarm

By

Xiaoyang Ye, Daniel He, Pei Heng Zheng

Supervisor

Dr. Robert Morley

Submitted in Partial Fulfillment of the Requirement for the BSEE Degree,

Electrical and Systems Engineering Department, School of Engineering and Applied Science,

Washington University in St. Louis

May 2014
# Table of Contents

Table of Figures.......................................................................................................................... 2  
Student Statement....................................................................................................................... 3  
Abstract....................................................................................................................................... 3  
Acknowledgements..................................................................................................................... 3  
Problem Formulation.................................................................................................................... 4  
  Problem Statement...................................................................................................................... 4  
Ideal Project Specifications.......................................................................................................... 4  
Concept Synthesis......................................................................................................................... 9  
  Concept Generation.................................................................................................................... 9  
      EEG Electrode Setup ............................................................................................................. 9  
      Analog Filtering and Amplification of EEG signals .......................................................... 10  
Noise Analysis.............................................................................................................................. 12  
      Multiplexing and Digital Filtering of sampled EEG Signals............................................... 13  
Sleep Stage Detection Algorithm............................................................................................... 22  
Bluetooth Wireless Transmission Module..................................................................................... 27  
Bill of Materials........................................................................................................................... 30  
Cost Analysis............................................................................................................................... 31  
Conclusion................................................................................................................................. 32  
References.................................................................................................................................... 33  
Appendix....................................................................................................................................... 34
# Table of Figures

Figure 1: Optimize Sleep uses sleep tracking data to adjust the user's last sleep cycle to end at the set alarm time ........................................................................................................................................ 7
Figure 2: Maximize Productivity uses sleep tracking data to wake the user at the end of the last sleep cycle .................................................................................................................................... 8
Figure 3: Sensor Sheet with EEG sensor array ................................................................................................................................. 9
Figure 4: EEG sensor with leads and casing ................................................................................................................................. 10
Figure 5: 2nd order filter implemented with Sallen Key topology ............................................................................................................ 11
Figure 6: Frequency response of 2nd order low pass filter .................................................................................................................. 12
Figure 7: Multiplexer Function Operations Flowchart ...................................................................................................................... 16
Figure 8: Convolution Operations ....................................................................................................................................................... 19
Figure 9: Digital Filter State Chart ..................................................................................................................................................... 20
Figure 10: FIFO Circular Buffer ......................................................................................................................................................... 21
Figure 11: Sleep Stage Characterization by EEG Brain Wave Signals ........................................................................................................ 22
Figure 12: 3-D Plot of EEG Samples forming Sleep Stage Clusters .................................................................................................... 24
Figure 13: Sleep Detection Algorithm Flowchart ............................................................................................................................ 24
Figure 14 ............................................................................................................................................................................................. 27
Figure 15 ............................................................................................................................................................................................. 28
Figure 16 ............................................................................................................................................................................................. 29
Figure 17: 2nd order low pass filter ....................................................................................................................................................... 34
Figure 18: Summing circuit ................................................................................................................................................................. 34
Figure 19: Low pass filter frequency response data points ............................................................................................................... 35
Figure 20: Complete circuit frequency response .............................................................................................................................. 36
Figure 21: Complete circuit frequency response data points ........................................................................................................... 36
**Student Statement**

The authors have applied ethics during the design process and complied with the WUSTL Honor Code.

**Abstract**

Sleep is a necessary part of human life. In today’s world, the amount of time we spend sleeping can be regularly lower than the National Institutes of Health recommendation of 7 – 8 hours. Studies have shown that waking up during Rapid Eye Movement (REM) or Stage 1 sleep results in low sleep inertia. Sleep inertia is a state of lowered arousal that occurs upon waking from sleep and results in temporarily reduced performance. Similarly, studies show that waking during slow wave sleep (SWS) causes the most sleep inertia out of all the sleep stages. In addition, sleep inertia becomes worse with sleep deprivation.

The goal of our project is to create a device that improves lifestyle and productivity by targeting the sleep stage at waking as something that can be optimized. HypnoLarm is a pillow device that records and analyzes electroencephalogram (EEG) signals that characterize sleep stage. Using this information and user-input waking time range, HypnoLarm will be able to wake the user at the optimal time to minimize sleep inertia while maximizing sleep time. This semester we have been able to build analog circuitry to collect/filter 8 channels of EEG signal as well as develop algorithms to interpret the data.

**Acknowledgements**

The authors would like to thank Dr. Robert Morley for his guidance and expertise in design. Without his help we would not have made as much progress as we did.
Problem Formulation

Problem Statement
Sleep inertia is the phenomena of disorientation and reduced performance that occurs immediately after waking. The time duration of sleep inertia can last from one minute to four hours [1]. Literature has shown that waking during stage 1 or stage 2 sleep reduces the effects of sleep inertia. In this project, we aim to design a device that will optimize the waking stage of the user and his/her desired waking time. By use of electroencephalography (EEG), we aim to be more accurate in determining user sleep stage than current commercial products that enhance user sleep. The ultimate goal of our device is to reduce user sleep inertia in order to increase productivity.

Ideal Project Specifications

Sensor Sheet
The sensor sheet is composed of a layered sheet with a memory foam base for comfort. This sheet is embedded with EEG electrodes and localization sensors. The electrodes are spaced 1 inch apart, center-to-center in order to best replicate clinical EEG setting. The localization sensors are placed in a layout such that their data output can be used to determine the orientation and position of the user’s rested head. The layered sheet is composed of materials of different density in order to maximize both comfort of the sheet and electrode contact with the user.

Central Processing Circuit
The central processing circuit is composed of a processing board and auxiliary circuit components that allow piping of data produced by the sensor sheet to and from the processing board. Prior to sending sensor sheet data to the Mobile App, the central processing circuit must multiplex, amplify and filter the sensor sheet data to get both meaningful EEG and localization signals. These signals are then stored in memory as a data time series until sufficient data to produce a sample is generated. The sample is then transferred over Bluetooth to the Mobile App for sleep analytics processing. After transfer, a percentage of the buffer is flushed in order to ensure some amount of time overlap between samples.
The central processing circuit Bluetooth capability also includes handling incoming control signals from the Mobile App. These control signals include, but are not limited to, sensor calibration commands and alarm trigger commands.

*Sleeve Alarm*

The sleeve alarm is capable at vibrating at various frequencies and strengths in order to suit the needs of user-defined waking methods. In order to induce micro-awakenings, the sleeve alarm has a high frequency, low amplitude vibration mode. In order to ease the user awake, the sleeve alarm has a variable low frequency, increasing amplitude vibration mode. These modes are controlled by the alarm trigger commands that are sent from the Mobile App through Bluetooth communication.

**Functions of the Mobile App**

*Platforms and Administrative requirements*

The Mobile App can be used on both Java-based Android platform and C-based Windows and iOS platforms. Code has high cross portability through the use of multi-platform integrated extended Python languages (Jython and Cython). To process data during sleep, the Mobile App must be provided permissions to access clock, alarm and storage capabilities of the respective development platforms.

*Sample/Channel Selection and Storage*

Using the localization signal and the intrinsic quality of the EEG signal samples, the Mobile App selects which EEG samples are ideal for sleep stage prediction. The quality is determined by sample characteristics such as signal-to-noise ratio and the position of the user’s head determined by the localization signal. Regardless of the selected channel, all channels have their data stored for post-night processing and analytics, and are labeled with their appropriate physical positions (on the user’s head) using the localization signal.
Sleep Stage Classifier

A sleep stage classifier is implemented to deduce the sleep stage of given sample. The sleep stage classifier uses a supervised machine learning method in order to classify samples based on initial datasets. Due to the free-form nature of the sensor sheet, the initial training data for the machine learning method must be created from data extracted and manually scored from the sensor sheet.

Alarm Control Loop

The alarm control loop takes the predicted sleep stage and time stamp for a sample and stores it in memory. Using the history of sleep stages and user input wake time, the alarm control loop evaluates the current state of the alarm and communicates as an alarm trigger command to the Sleeve via Bluetooth.

Alarm Modes

The user can specify two different alarm modes to produce different degrees of sleep control and waking. In the “Optimize Sleep” alarm mode, the user is subjected to micro-awakenings through vibrations from the Sleeve Alarm. These micro-awakenings serve the purpose of regulating the sleep cycle such that the optimized waking time coincides with the user input wake time.
In the “maximize productivity” alarm mode, the alarm patterns the sleep cycles of the user and uses the pattern to find an optimal waking time within a half of a sleep cycle duration of the user’s input wake time. The sleeve alarm is then triggered to wake the user at the prescribed time. Due to the volatility of sleep cycles, these modes require the alarm control loop to dynamically solve the optimal wake time throughout the night.
Figure 2: Maximize Productivity uses sleep tracking data to wake the user at the end of the last sleep cycle

User interface

The Mobile App has a graphic user interface (GUI) that allows the user to specify wake times and alarm modes. The GUI can also access sleep analytics for stored sleep data that are presented as tables and plots. The code is loosely constructed to allow flexibility in adding additional graphics and modules.
Concept Synthesis

The HypnoLarm project is the brainchild of Forty Winks, LLC. Its inception was observed in the BME senior design class of fall 2012, where one of the company's founding members saw the potential in creating a reliable yet unobtrusive sleep optimization device. As a student in the engineering department, Piero Mendez realized how valuable sleep really is and just how much impact a product which scientifically improves our quality of sleep can have on society. Thus, the goal for developing a reliable and comfortable sleep optimization system was created. Since its beginning, HypnoLarm has encountered numerous setbacks and difficulties such as performing freeform EEG measuring. Complicated approaches such as using an integrated EEG/Pressure sensor array were proposed, tested and discarded. In order to push the dream of HypnoLarm into becoming reality, the team has decided to tackle the fundamentals of the task: by starting with designing the analog circuitry for a single EEG sensor and moving on from there.

Concept Generation

EEG Electrode Setup

The sensor sheet is composed of polyurethane foam, a material similar in conductive and physical properties to memory foam, with embedded EEG electrodes. The electrodes are arranged in an array such that they are 3 inches apart center-to-center. Our current prototype has seven recording electrodes and one ground electrode as shown in Figure 3 below.

Figure 3: Sensor Sheet with EEG sensor array
The polyurethane foam provides a comfortable surface for the user to lie on and has insulative properties to ensure the electrodes are only recording voltage data from the user’s scalp.

Figure 4 below depicts the EEG electrode setup used in this project, including the shielding wire and casing.

Figure 4: EEG sensor with leads and casing

The electrode used is an Ag-AgCl EL120 electrode. An audio cable containing internal static shielding was used to construct the electrode leads; acting like a Faraday’s cage and preventing effects from external voltage sources. The electrodes are housed within an insulative ABS plastic 3D printed electrode case, which facilitates contact between the electrodes and the audio wire.

The audio wire is first stripped on its end to expose the wire, which is then wrapped around the end of the electrode and held in place using the 3D printed case. The case is secured onto the electrode using super glue. The end of each sensor’s lead is then connected via soldering to an analog circuit which performs filtering and amplification of EEG signals.

**Analog Filtering and Amplification of EEG signals**

Our initial analog design was based on a do-it-yourself, EEG circuit [1]. It consisted of an instrumentation amplifier (IA), two 60 Hz notch filters, 7 Hz high pass filter, 31 Hz low pass filter and 1 Hz
high pass gain filter before reaching the A/D converter. Because this design is made to record awake EEG signals (which have higher frequency content than sleep EEG), it is not completely suitable for our needs. Our frequency range of interest is in the 1 – 30 Hz range. This range will allow us to capture data for all of the sleep stages and some waking frequencies.

We decided to drop the two 60 Hz notch filters since the 30 Hz low pass filter would attenuate some of the noise that occurs at 60 Hz. In addition, the remainder of the power at 60 Hz can be filtered out digitally and saves us from needing to use two operational amplifiers.

Since our frequency range of interest extends down to 1 Hz, we only utilize a 30 Hz low pass filter instead of high pass and low pass filter. Again, we can implement the 1 Hz high pass filter digitally while saving us the cost of a component. The low pass filter is second order filter designed using the Sallen Key topology (figure 5).

![Figure 5: 2nd order filter implemented with Sallen Key topology](image)

Using formulas found in [3], we derive \(R_1 = 8.192\, \text{M}\Omega, R_2 = 1.005\, \text{M}\Omega, C_1 = 4.22\, \text{nF}\) and \(C_2 = 806\, \text{pF}\). Appendix A shows our filter circuit implemented with a LM741CN operational amplifier. Figure 6 shows that we get unity gain below 30 Hz and our 3dB point is very close to 30 Hz.
Our instrumentation amplifier is an AD620AN and set to have a linear gain of 91 (39dB). For our initial prototype, we did not use analog multiplexers because the time constant introduced by the second order filter causes problems when we sample each channel in the multiplexer. Instead, since the Arduino has 8 analog inputs, we have 8 sets of the analog circuit collecting data and inputting into the Arduino.

The next problem we encountered was the operating voltage range of the Arduino. The Arduino operates from 0 – 5V while our EEG signal is centered around 0 and can take on negative values. To fix this, we implement a summing circuit that will shift the EEG signal up 2.5V so that it is in the operating range of the Arduino (Appendix).

**Noise Analysis**

We need to consider the noise coming from the circuit and from quantization of the signal at the A/D converter of the Arduino. The Arduino is a 10 bit A/D. With an operating voltage of 5V, we get a step size of $\frac{5V}{2^{10}} = 4.88 \text{ mV}$. Noise $V_{\text{RMS}} = 1.41 \text{ mV}$. 

![LPF Frequency Response](image)
Noise power = 1.986 uW in f, / 2 = 625 Hz.

The amount of power located in our range of interest is \( \frac{30}{625} \cdot noise\ power = 95.3\ nW \)

Noise \( V_{RMS} \) in our range of interest = 308.8 uV

Our IA has an input voltage noise of \( 9 \frac{mV}{\sqrt{Hz}} \)

Noise voltage introduced by IA: \( 9 \frac{mV}{\sqrt{Hz}} \cdot \sqrt{30\ Hz} \cdot 91 = 4.49\ uV \)

Noise power introduced by IA: 20.12 pW

(Assuming other stages introduce negligible noise contributions when compared to that of the IA)

Total noise power: 95.3 nW

Total noise \( V_{RMS} \): 308.8 uV

EEG signal amplitude can be expected to be in the microvolt range (~100 uV).

Signal voltage: 100 uV \cdot 91 = 9.1 mV

Signal-to-noise ratio: \( 20 \cdot \log \left( \frac{9.1\ mV}{308.8\ uV} \right) = 29\ dB \)

Multiplexing and Digital Filtering of sampled EEG Signals

**Multiplexer Function**

We chose to use the Arduino Pro mini in our prototype mostly for convenience and ease of use. Instead of designing a processing component from scratch, using the Arduino allowed us to focus on the other circuit section designs. When we move away from using pre-constructed microprocessors we will need to take the following requirements into account:
1. **Number of analog ports**

   The Arduino mini has 8 analog inputs which was enough to build a working prototype. When we expand the sensor array, a system of multiplexers will most likely need to be integrated into the circuit due to the increased number of electrodes being read. A microprocessor with 4-6 analog inputs should suffice.

2. **Analog to digital converter resolution**

   Since we are working with biological signals we require a high resolution in order to accurately analyze the measured EEG. The Arduino has an internal ADC resolution of 10 bits, which is lower than we would like. Increasing the resolution into the range of 12-16 bits would be ideal, but we realize that this will affect the processing requirements. At the moment, we based the gain of the instrumentation amplifier on the resolution of the Arduino ADC, which is about 5 mV.

3. **Operating voltage**

   The Arduino operates at 5 V, and can read values in the range of 0-5 V (0 – 1023). This is an issue we have had to work around by introducing a 2.5 V shift in the input signals. Ideally, the new processor would be able to read negative voltages in order to remove the need for the summing amplifier circuit.

4. **Clock speed**

   Up to this point, we have been working around the processing speed restrictions of the Arduino when writing the code for it. Increasing ADC resolution and the number of sensors would most likely be too much for the 16 MHz processor of the Arduino to handle.

The microprocessor’s in-built 8-channel, 10-bit analog-to-digital converter (ADC) operates using a 125 KHz clock signal, which is generated using the Arduino’s 16 MHz system clock at a pre-scaler of 128:

\[
 ADC \text{ clock frequency} = \left( \frac{16 \text{ MHz}}{128} \right) = 125 \text{ KHz}
\]
The 8 EEG sensors are connected to the analog input pins on the Arduino board. A simple multiplex-and-sample function for reading the 8 EEG sensors (channels) was designed and written in C++ language for the Arduino platform, and subsequently uploaded to the Arduino microprocessor. The multiplex function code can be found in section A of the Appendix.

The ADC on the Arduino pro mini V5 microprocessor takes a minimum of 13 clock cycles (using a 125 KHz clock) to perform one sample and convert. Thus, a minimum of 104μs is needed to read a single EEG sample. A significant constraint of the 10-bit ADC is its inaccuracy when rapidly switching between reading different analog inputs due to the Arduino’s sample-and-hold circuit’s tendency to retain a previously read value. This is even more significant when the analog inputs of the ADC are connected to sources with high impedance (such as the EEG sensors), which take a significant time to charge/discharge the sample-and-hold circuit’s capacitor. In order to prevent this, the ADC must be called twice to read another analog input channel:

1. An initial call triggers the ADC to select the new channel to be read.
2. A time delay then allows it to stabilize.
3. The next call to read the same channel will allow the ADC to perform an accurate conversion and return the correct value.

The multiplex-and-sample function was designed taking the sensing requirements and ADC considerations mentioned above. The result is a state machine implementation, which consecutively cycles through reading each of the 8 EEG inputs to the ADC once every 10 milliseconds, and with delays between switch-and-reads. The code utilizes various Arduino in-built functions such as `analogRead(pin)`, and `delayMicroseconds(time)` in order to perform its functions. Figure 7 below depicts the state diagram of the function.
The multiplexer function utilizes the Arduino timer function to sample all 8 EEG channels once every 5 milliseconds, maintaining a constant sample rate of 100Hz for every channel. During one sampling operation, the code iterates through the 8 channels; first selecting the channel, then delaying 150μs to allow the ADC to stabilize around the channel's voltage before reading it, followed by another 150μs delay. This gives a total time of 508μs per channel and 4 milliseconds for reading all 8 channels, which falls within the time constraint of 10 milliseconds for a 100 Hz sampling rate per channel.

After each sampling operation, the converted digital values of the 8 EEG channels are sent out via serial communication, as an 8-index array, where they undergo digital filtering.

The multiplexer code was tested for its ability to maintain a constant 100Hz sampling frequency for every channel. The Arduino “millis()” function was used to repeatedly read the time between successive sampling of one channel in milliseconds, which was then printed out. A precise time of 10ms was maintained throughout.

We then tested its accuracy by feeding 8 known voltage values to the Arduino’s analog input pins using the NI-Elvis prototyping board, and the ADC’s converted values were printed out and compared to the actual analog values. Table 1 below shows the results of this accuracy test:
### Table 1: Multiplexer Test Results

<table>
<thead>
<tr>
<th>Analog Input Pin\Voltage Input</th>
<th>ADC Digital Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A0\2.5V</td>
<td>512</td>
</tr>
<tr>
<td>A1\0V</td>
<td>0</td>
</tr>
<tr>
<td>A2\5V</td>
<td>1023</td>
</tr>
<tr>
<td>A3\0V</td>
<td>0</td>
</tr>
<tr>
<td>A4\2.5V</td>
<td>511</td>
</tr>
<tr>
<td>A5\5V</td>
<td>1022</td>
</tr>
<tr>
<td>A6\0V</td>
<td>1</td>
</tr>
<tr>
<td>A7\5V</td>
<td>1023</td>
</tr>
</tbody>
</table>

Thus, we see that the multiplexer code is able to accurately sample every input channel of the ADC while rapidly iterating through reading each channel at 100Hz. However when we tried to up the sample rate to 200Hz, the accuracy of the ADC fell visibly. We found that this is because the sample-and-convert operation of the ADC does not always precisely take $104\,\mu s$, and in order to sample 8 channels at 200Hz each, the $150\,\mu s$ delay has to be decreased to maintain a constant sample frequency. This leads to the ADC not having enough time to stabilize before reading a channel.
**Digital FIR Filter**

Given that the EEG signals collected by the sensor sheet have a spectrum of interest in the .5 Hz to 30 Hz interval, a digital low-pass filter with a frequency cutoff at 30Hz was implemented in C++ code on the Arduino. The digital filter was implemented using the convolution method, which is summarized and explained below.

The convolution of two functions $f$ and $g$ ($f*g$) is defined as:

$$\int_{-\infty}^{\infty} f(\tau) g(t - \tau) \, d\tau$$

Given a fixed sample rate, the index of each EEG data point can be represented as a step in the time domain. In other words, the convolution result, $y[n]$ can be expressed as:

$$y[n] = x[n]*h[n] = \sum_{k=0}^{i} x[k]h[n-k]$$

Where $n$ is the total number of samples passed in. Since the impulse response is finite, it only has a fixed number of data points which are not equal to 0. Given $i$ number of non-zero data points in the impulse response, there are only $i$ terms in the convolution result. Furthermore, $h[k]$ for $k<0$ and $k>i$ would be 0, thus the convolution equation’s boundaries change to: from $k = n - i$ to $k = n$.

Expanding the convolution equation:

$$y[n] = x[n-i]*h[i] + x[n-i-1]*h[i-1] + \ldots + x[n]*h[0]$$

Thus, convolution flips the order of the EEG dataset, and performs a multiply-and-addition with the impulse response terms. It is a First-in-First-Out (FIFO) operation, which does $i$ multiplies and
accumulates on the $i$ most recent EEG signal inputs, where $i$ is the number of impulse response terms. Its operations flowchart is depicted in Figure 8 below.

\[
x[n] \times h[0] \quad \rightarrow \quad y[n]
\]
\[
x[n] \times h[1] \quad \rightarrow \quad \vdots
\]
\[
x[n - i] \times h[i]
\]

**Figure 8: Convolution Operations**

The impulse response of the filter was generated using design tool at: [http://www.arc.id.au/FilterDesign.html](http://www.arc.id.au/FilterDesign.html). A cutoff frequency of 30 Hz was set for a low-pass filter with a sample rate of 200Hz, and a length of 59 impulse terms was chosen to minimize the delay introduced by the filter.
The digital filter was implemented in C++ as a state machine. It utilizes a FIFO circular buffer of size 59 to store incoming EEG data, and perform dot product with impulse response coefficients. An “end” pointer declares the index in the array where new EEG data will be stored, incrementing by one after the data is written in.
Figure 10 illustrates how the circular buffer operates.

![Circular Buffer Diagram](image)

Before data is written into array

After data is written into array

**Figure 10: FIFO Circular Buffer**

Each time new EEG data is read, the end pointer shifts to the next index. This process repeats until the end pointer reaches the largest index of the buffer array, at which point it resets to zero, and new data will replace data at index 0.

The digital filter first initializes all values stored in the circular buffer to 0, resetting itself. It then loads the impulse response coefficients as a header file for use during convolution calculations. Next, EEG data points are passed in sequentially one at a time and convolution is performed after each data is passed in. The results of the convolution operation give the time response of the signal to the filter.
Sleep Stage Detection Algorithm

Figure 11 below depicts how sleep can be broken down into its different stages through an analysis of different types of EEG sleep waves, each characterized by its frequency spectrum.

![Stages of sleep](image)

- **Stage 1:** transitional light sleep
  - 8-13 Hz (alpha waves)
  - to 4-7 Hz (theta waves)
- **Stage 2:** light sleep
  - 4-7 Hz, sleep spindles, K-complexes
- **Stage 3** (and 4): deep sleep
  - 0.5-3.5 Hz (delta waves), high-amplitude
- **REM:** rapid eye movement
  - 8-13Hz

*Figure 11: Sleep Stage Characterization by EEG Brain Wave Signals*

Our approach is to perform a frequency domain analysis on EEG data to find the dominant types of EEG sleep waves present at every point in time, and determine the current sleep stage which is characterized by the presence of those waves.

An algorithm was implemented and tested in Python, which decomposes and analyses 30 second EEG samples in the frequency domain in order to determine sleep stage. The Fourier analysis method was used to decompose each EEG sample and extract its attributes in each spectrum of interest in the frequency domain.

The algorithm uses a cluster learning approach on previously recorded and scored sleep EEG data from sleep research, to predict the sleep stages of an unscored set of data.
It is trained on input scored sleep EEG data, which were time annotated with sleep stages for every interval. Using the known scoring of the training data set, the Principal Component Analysis method was used to identify 3 eigenvector parameters from 9 extracted EEG signal features, which provided the best distinction between sleep stage clusters.

The 9 EEG signal features were chosen based on existing research literature on sleep scoring using EEG. They are:

1. 3 parameters of Hjorth: Activity, Mobility and Complexity – time domain parameters used as control parameters.
2. 3 harmonic parameters – frequency domain parameters measuring dominant frequency power band.
3. Frequency band power ratios: $\frac{\theta}{\alpha}$, $\frac{\delta}{\theta}$, and $\frac{\delta^2}{\theta \cdot \alpha}$.

The 3 eigenvectors were then calculated for each EEG sample, which is then plotted on a 3-D vector space and color coded for its corresponding sleep stage, with the 3 eigenvectors as the axis. As can be seen from Figure 12 below, visible clusters of points for each stage of sleep is formed, with clear boundaries between them.
The K-means distance approach was utilized in determining the sleep stage cluster a new sample point falls into. This approach calculates the average distance of an incoming data point from every point in a cluster, the cluster which has the smallest average distance is then determined to be the one the incoming sample falls into. Sleep stage can thus be identified. Figure 13 on the next page depicts the operational flowchart of the sleep detection algorithm:
The algorithm code in Python can be found in section “” of the appendix.

The sleep stage detection algorithm was tested using 15 sets of pre-scored EEG data obtained from http://physionet.org/physiobank/database/capslpdb/ sleep database, each measured over the course of a night’s sleep, as learning data.

The sleep stage annotations for the datasets were modified to combine sleep stages 1 and 2 (light sleep), and stages 3 and 4 (deep sleep). The reason being we only require distinguishing between light and deep sleep to optimize waking, and are not concerned with the type of light or deep sleep. Thus, 4 clusters were defined as follows:

1. REM.
2. Light Sleep.
4. Awake.

The algorithm was then tested using a pre-scored EEG dataset to calculate the sleep stage for each sample. The results were then compared to the scored sleep stages at the time interval for each sample to check for accuracy. The number of agreements between the sleep researchers’ and algorithm’s scored sleep stages were calculated, and an accuracy of 50% was achieved.

However, a large number of the discrepancies occurred between distinguishing REM and light sleep. This is unsurprising as REM sleep shares many common attributes to light sleep, such as high frequency brain activity. Despite the similarities between REM and light sleep, it is found that waking someone during REM sleep results in a high level of lethargy during the day.
We also tested the following machine learning algorithms (from the Scikit-learn python library: http://scikit-learn.org/stable/) to classify sleep stages on future EEG data sets: support vector machine, stochastic gradient descent, decision tree, random forest, Ada boost, and naive Bayes. However, we were not able to achieve any better accuracy. The results are shown in Table 2 below:

<table>
<thead>
<tr>
<th>Processing</th>
<th>Support vector machine</th>
<th>Stochastic gradient descent</th>
<th>KPCA</th>
<th>Decision trees</th>
<th>Random forest</th>
<th>Ada boost</th>
<th>Naive Bayes</th>
</tr>
</thead>
<tbody>
<tr>
<td>REM vs. all</td>
<td>0</td>
<td>1.171875</td>
<td>63.28125</td>
<td>60.9375</td>
<td>35.546875</td>
<td>84.765625</td>
<td>51.171875</td>
</tr>
<tr>
<td>REM+1 vs. 2</td>
<td>32.421875</td>
<td>0.78125</td>
<td>31.25</td>
<td>30.859375</td>
<td>38.671875</td>
<td>30.859375</td>
<td>33.984375</td>
</tr>
<tr>
<td>REM vs. 1+2</td>
<td>32.421875</td>
<td>0.78125</td>
<td>49.90625</td>
<td>33.203125</td>
<td>25</td>
<td>30.859375</td>
<td>33.984375</td>
</tr>
<tr>
<td>No post</td>
<td>processing</td>
<td>32.421875</td>
<td>0.78125</td>
<td>31.640625</td>
<td>21.484375</td>
<td>18.359375</td>
<td>30.859375</td>
</tr>
</tbody>
</table>

Thus, the algorithm is actually highly unreliable in optimizing sleep if it cannot distinguish REM from light sleep. We learned from EEG sleep scoring literature that secondary signals from Electrocardiography sensors are often required to distinguish the two stages of sleep. A future area of development for this project is the introduction of a second type of sensor which serves to measure a control signal to further increase the accuracy of detecting user sleep stages.
Bluetooth Wireless Transmission Module

Our Android Application is compiled with API 19: Android 4.4 (KitKat). The target SDK is API 19: Android 4.4 (KitKat) and the minimum required SDK is API 8: Android 2.2 (Froyo). Android SDK is the Android software development kit which uses XML to define layouts and Java to define logic. More information can be found on the Android developer’s website: http://developer.android.com/index.html.

We utilized source code found on the Android developers’ website: http://developer.android.com/guide/topics/connectivity/bluetooth.html. Particularity, we implemented source code for discovering Bluetooth devices, querying paired devices, initializing and managing the Bluetooth connection.

Our Android application references the GraphView 3.0 library. GraphView 3.0 has many features that are relevant to our programing need; notably GraphView 3.0 can draw multiple series of data in real-time. More information can be found on their website: http://android-graphview.org/.

Our Android application is composed of two activities and two layouts. Our main activity is launched on startup. The view is set to activity_main.xml, where our graphing interface is located. Our GraphView 3.0 associated variables are initialized in the init() method and our button associated variables are initialized with buttonInit().

Figure 14
Our buttons use the onClick(View v) method. The view is set to activity_bluetooth.xml. When the “Connect” button is pressed, the application starts the Bluetooth activity. Bluetooth activity checks to see if the Android device the application is being run on is a Bluetooth enabled device. The startDiscovery() and getPairedDevices() methods (defined by source code from the developer’s site) then populate the listview and check for paired devices respectively.

![Bluetooth activity](image)

**Figure 15**

When a paired device in the listview is clicked, the private class ConnectThread is called; ConnectThread is responsible for initializing the Bluetooth connection. Upon successfully connecting, it sends a message (tagged as SUCCESS_CONNECT) to the message handler (defined in MainActivity.java and imported into
Bluetooth.java) which in turn calls the private class ConnectedThread; ConnectedThread reads the inputStream and sends a message (tagged as MESSAGE_READ) to the same message handler which in turn parses the data, graphs it, and appends it to the config.txt file via the writeToFile(String s) method.

Unfortunately a few features were left out due to time pressure – in its current iteration the app merely exists to prove the concept. Future iterations will be able to parse multiple sets of incoming data and graph them simultaneously or separately. Also, the actual alarm algorithm (written in Python) has not been ported into the app yet but will be done with by implementing Jython.

Figure 16
## Bill of Materials

<table>
<thead>
<tr>
<th>Part</th>
<th>Supplier</th>
<th>Part #</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 kΩ resistor</td>
<td>Digikey</td>
<td>RNF 1/4 T9 5K 0.1% R</td>
<td><a href="http://www.digikey.com/product-detail/en/RNF%201%2F4%20T9%205K%200.1%25%20R/RNF1%2F4T95KBR-ND/1682715">http://www.digikey.com/product-detail/en/RNF%201%2F4%20T9%205K%200.1%25%20R/RNF1%2F4T95KBR-ND/1682715</a></td>
</tr>
<tr>
<td>800 pF capacitor</td>
<td>Mouser</td>
<td>661-EKY80ELL801M M25S</td>
<td><a href="http://www.mouser.com/ProductDetail/United-Chemi-Con/EKY-800ELL801MM25S/?qs=sGAEpiMZZMsh%252b1woXYuXj6XsRl28M3llet3CBBRwRDA%3d">http://www.mouser.com/ProductDetail/United-Chemi-Con/EKY-800ELL801MM25S/?qs=sGAEpiMZZMsh%252b1woXYuXj6XsRl28M3llet3CBBRwRDA%3d</a></td>
</tr>
</tbody>
</table>
## Cost Analysis

<table>
<thead>
<tr>
<th>Part</th>
<th>Unit price (US dollars)</th>
<th>Number needed</th>
<th>Cost (US dollars)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AD620AN</td>
<td>5.17</td>
<td>8</td>
<td>41.36</td>
</tr>
<tr>
<td>LM741CN</td>
<td>0.35</td>
<td>16</td>
<td>5.60</td>
</tr>
<tr>
<td>Electrode</td>
<td>15.00</td>
<td>8</td>
<td>120.00</td>
</tr>
<tr>
<td>Shielded wire</td>
<td>11.99</td>
<td>1</td>
<td>11.99</td>
</tr>
<tr>
<td>560 Ω resistor</td>
<td>0.00756</td>
<td>8</td>
<td>0.06</td>
</tr>
<tr>
<td>5 kΩ resistor</td>
<td>0.13808</td>
<td>32</td>
<td>4.42</td>
</tr>
<tr>
<td>1 kΩ resistor</td>
<td>0.00756</td>
<td>8</td>
<td>0.06</td>
</tr>
<tr>
<td>8.2 MΩ resistor</td>
<td>0.03290</td>
<td>8</td>
<td>2.63</td>
</tr>
<tr>
<td>1 MΩ resistor</td>
<td>0.00756</td>
<td>8</td>
<td>0.06</td>
</tr>
<tr>
<td>4.3 nF capacitor</td>
<td>0.22960</td>
<td>8</td>
<td>1.84</td>
</tr>
<tr>
<td>800 pF capacitor</td>
<td>0.697</td>
<td>8</td>
<td>5.58</td>
</tr>
<tr>
<td>Arduino Mini Pro</td>
<td>7.96</td>
<td>1</td>
<td>7.96</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td></td>
<td><strong>201.56</strong></td>
</tr>
</tbody>
</table>

*Unit Prices can be found in Bill of Materials URL*
Conclusion

To sum up, currently, FortyWinks is focused on completing product development and initiating the commercial release of the Hypnolarm system. To date, we have raised $20,000 in seed capital from private investors and are currently approaching angel investment groups, technology incubators, and venture capital firms to secure additional funds. This process can be expected to reach an initial goal by the end of 2014.

Future generations of the Hypnolarm system will be progressively more advanced. We plan on implementing deep sleep stimulation, which can enhance memory, cognition, and possibly productivity, in our next iteration. With time, Hypnolarm will evolve from a sleep optimization system into a complete wellness system with potential for medical, military, and other applications. For now, we remain completely focused on our initial commercial launch.

We have finished proof-of-concept, and within the next year our goal is to build a beta prototype and send it for product optimization. The beta prototype will have enhanced bio-signal acquisition, more accurate sleep stage determining, and include the accompanying sleep optimization app. We are seeking capital to progress from proof-of-concept to significant market penetration. With the increasing opportunistic wellness/wearables technology market, we will create a productivity niche in this booming market and set the wheels in motion for the success of FortyWinks, the lifestyle technology brand.

FortyWinks. Sleep less, do more.
References

Appendix

Figure 17: 2nd order low pass filter

Figure 18: Summing circuit
<table>
<thead>
<tr>
<th>Frequency (Hz)</th>
<th>Vin (mV)</th>
<th>Vo (mV)</th>
<th>Vo/Vin</th>
<th>Gain(dB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>480.92</td>
<td>474.15</td>
<td>0.985923</td>
<td>-0.12314</td>
</tr>
<tr>
<td>5</td>
<td>480.34</td>
<td>473.38</td>
<td>0.98551</td>
<td>-0.12678</td>
</tr>
<tr>
<td>10</td>
<td>481.92</td>
<td>469.32</td>
<td>0.973855</td>
<td>-0.23012</td>
</tr>
<tr>
<td>20</td>
<td>482.69</td>
<td>425.09</td>
<td>0.880669</td>
<td>-1.10375</td>
</tr>
<tr>
<td>30</td>
<td>482.4</td>
<td>324.05</td>
<td>0.671745</td>
<td>-3.45591</td>
</tr>
<tr>
<td>40</td>
<td>480.66</td>
<td>223.63</td>
<td>0.465256</td>
<td>-6.64616</td>
</tr>
<tr>
<td>50</td>
<td>480.4</td>
<td>155.77</td>
<td>0.324251</td>
<td>-9.78238</td>
</tr>
<tr>
<td>60</td>
<td>482.33</td>
<td>112.51</td>
<td>0.233264</td>
<td>-12.6431</td>
</tr>
<tr>
<td>70</td>
<td>483.43</td>
<td>84.57</td>
<td>0.174937</td>
<td>-15.1423</td>
</tr>
<tr>
<td>80</td>
<td>481.46</td>
<td>66.05</td>
<td>0.137187</td>
<td>-17.2537</td>
</tr>
<tr>
<td>90</td>
<td>481.08</td>
<td>52.72</td>
<td>0.109587</td>
<td>-19.2048</td>
</tr>
<tr>
<td>100</td>
<td>481.69</td>
<td>43.3</td>
<td>0.089892</td>
<td>-20.9256</td>
</tr>
<tr>
<td>150</td>
<td>482.24</td>
<td>20.24</td>
<td>0.041971</td>
<td>-27.5411</td>
</tr>
<tr>
<td>200</td>
<td>480.43</td>
<td>12.4</td>
<td>0.02581</td>
<td>-31.7642</td>
</tr>
<tr>
<td>500</td>
<td>481.5</td>
<td>4.06</td>
<td>0.008432</td>
<td>-41.4814</td>
</tr>
<tr>
<td>1000</td>
<td>479.31</td>
<td>2.96</td>
<td>0.006176</td>
<td>-44.1865</td>
</tr>
</tbody>
</table>

**Figure 19:** Low pass filter frequency response data points
**Figure 20:** Complete circuit frequency response

![Complete Circuit Frequency Response](image)

<table>
<thead>
<tr>
<th>Frequency (Hz)</th>
<th>Vin (mV)</th>
<th>Vo (mV)</th>
<th>Vo/Vin</th>
<th>Gain(dB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>17.84</td>
<td>1087</td>
<td>60.93049</td>
<td>35.69669</td>
</tr>
<tr>
<td>5</td>
<td>20.02</td>
<td>1061</td>
<td>52.997</td>
<td>34.48503</td>
</tr>
<tr>
<td>10</td>
<td>17.77</td>
<td>1036</td>
<td>58.30051</td>
<td>35.31345</td>
</tr>
<tr>
<td>20</td>
<td>18.49</td>
<td>937.91</td>
<td>50.72526</td>
<td>34.10449</td>
</tr>
<tr>
<td>30</td>
<td>17.93</td>
<td>729.72</td>
<td>40.69827</td>
<td>32.19152</td>
</tr>
<tr>
<td>40</td>
<td>16.9</td>
<td>504.14</td>
<td>29.83077</td>
<td>29.49329</td>
</tr>
<tr>
<td>50</td>
<td>17.57</td>
<td>354.07</td>
<td>20.15196</td>
<td>26.08635</td>
</tr>
<tr>
<td>60</td>
<td>18.8</td>
<td>270.34</td>
<td>14.37979</td>
<td>23.15505</td>
</tr>
<tr>
<td>70</td>
<td>17.5</td>
<td>195.79</td>
<td>11.188</td>
<td>20.97505</td>
</tr>
<tr>
<td>80</td>
<td>17.99</td>
<td>161.82</td>
<td>8.994997</td>
<td>19.08002</td>
</tr>
<tr>
<td>90</td>
<td>17.93</td>
<td>140.89</td>
<td>7.85778</td>
<td>17.906</td>
</tr>
<tr>
<td>100</td>
<td>17.59</td>
<td>103.05</td>
<td>5.858442</td>
<td>15.35564</td>
</tr>
<tr>
<td>150</td>
<td>16.87</td>
<td>53.78</td>
<td>3.187908</td>
<td>10.07011</td>
</tr>
<tr>
<td>200</td>
<td>18.68</td>
<td>48.95</td>
<td>2.62045</td>
<td>8.367516</td>
</tr>
<tr>
<td>500</td>
<td>18.15</td>
<td>33.17</td>
<td>1.827548</td>
<td>5.237377</td>
</tr>
<tr>
<td>1000</td>
<td>19.28</td>
<td>22.7</td>
<td>1.177386</td>
<td>1.418377</td>
</tr>
</tbody>
</table>

**Figure 21:** Complete circuit frequency response data points
AD620AN datasheet

LM741CN datasheet

Biopac EEG electrode specification sheet
http://www.biopac.com/Product_Spec_PDF/EL120.pdf

Arduino Mini Pro Schematic