4-1-2013

Real-time Stage 1 Sleep Detection and Warning System Using a Low-Cost EEG Headset

Bryan Van Hal
Grand Valley State University

Real-time Stage 1 Sleep Detection and Warning System Using a Low-Cost EEG headset

Bryan Van Hal

A Thesis Submitted to the Graduate Faculty of

GRAND VALLEY STATE UNIVERSITY

In

Partial Fulfillment of the Requirements

For the Degree of

Masters of Science in Engineering

Department of Engineering

April 2013

Acknowledgements

I would like to thank my committee chair Dr. Samhita Rhodes for her assistance and direction in my research and thesis. I would also like to thank Dr. Rob Bossemeyer, Dr. Bruce Dunne, and Dr. Paul Fishback for being willing to participate in my thesis committee. All of their input helped improve the depth of my research and quality this thesis.

I would also like to thank my wife, Jen Van Hal, for giving me the time to devote to this thesis and for helping and supporting me every step of the way.

Abstract

The goal of this thesis is to design and test a real-time Stage 1 sleep detection and warning system using a low-cost single dry-sensor EEG headset. Such a system would allow aircraft pilots or truck drivers to receive an auditory warning when they are beginning to fall asleep. The device designed in this study records a single EEG signal and filters it into low Alpha (7.5 - 9.25 Hz), high Alpha (10 - 11.75 Hz), low Beta (13 - 16.75 Hz), and high Beta (18 - 29.75 Hz) frequency bands. When the EEG transitions to match that of Stage 1 sleep for a short period of time, the device produces an audible alarm.

The system proved 81% effective at detecting sleep in a small sample group. All failures were due to false alarms. Compared to tradition sleep scoring, this device predicted and responded to the onset of drowsiness preceding stage 1 sleep.

Keywords: single-channel EEG, sleep, dry sensor, pilot, driver, low cost, monitor, drowsiness

Table of Contents

Table of Figures

I. Introduction

According to the National Highway Traffic Safety Administration, there are 100,000 police-reported crashes due to driver drowsiness each year [National Sleep Foundation, 2012a]. These have resulted in 1,550 deaths, 71,000 injuries, and $12.5 billion in monetary losses. The actual numbers may be higher, because the NHTSA statistics rely on self-reporting. Drowsiness and fatigue are of concern to airline pilots as well. In the 2009 crash of Colgan Air flight 3407, "the pilots' performance was likely impaired because of fatigue" [National Transportation Safety Board, 2010]. According to the National Sleep Foundation, "One in five pilots (20%) admit that they have made a serious error and one in six train operators (18%) and truck drivers (14%) say that they have had a "near miss" due to sleepiness" [National Sleep Foundation, 2012b]. Some of these accidents could have been avoided if drivers or pilots were alerted well in advance of the onset of fatigue and associated drowsiness.

Currently, there are two major ways of driver drowsiness detection: driving pattern based detection, and eye closure detection. Driving pattern based detection is reliable, but focuses on the effects of falling asleep at the wheel. By the time these effects become noticeable, it may already be too late to prevent an accident. Eye closure detection depends on a vision system being able to track the eyes of a driver to determine if they are closed. This detection system allows for earlier detection of driver drowsiness than driving pattern detection, but it is limited by the accuracy of the vision system used to determine eye closure.

The drawbacks of poor accuracy and insufficient reaction time in current driver drowsiness detection systems have led to the exploration of new techniques based on changes in body

physiology as a function of fatigue. One promising method is the use of signals recorded from scalp electrodes that measure patterns of changing electrical activity in the brain as someone goes from a state of complete alertness to fatigue and drowsiness. The process of recording these signals is known as Electroencephalography (EEG).

EEG has been used to detect the stages of sleep since the 1930s [Loomis, Harvey, & Hobart, 1937]. It has also been clinically used to monitor driver and pilot drowsiness [Lal & Craig, 2002; Caldwell, Mallis, Caldwell, Paul, Miller, & Neri, 2009]. However, these medical grade EEG devices are impractical for everyday driver drowsiness detection since they require the use of expensive equipment and complicated skin preparation with conductive gel for effective monitoring.

Within the last few years, toys, such as the Star Wars Force Trainer and the Mind Flex, that detect simple EEG signals, have been designed for entertainment purposes. Both of these devices use a single dry-electrode to record EEG signal power in select frequency bands. This differs from medical grade EEG which uses multiple electrodes, attached with conductive gel and adhesive, to record EEG signal power at multiple locations along the subject's scalp. Along with EEG toys like the Mind Flex, manufacturers also provide low cost EEG headsets that are comfortable and compact. These toys offer very limited EEG recording capability but are simple and cheap, requiring minimal specialized training. This makes them an attractive target for research efforts into effective driver drowsiness detection systems using EEGs. One such low cost EEG headset is the NeuroSky Mindset.

The NeuroSky Mindset (see Figure 1), used in this thesis, is a Bluetooth audio headset with a single dry-sensor electrode (electrode circled in figure 1) [Neurosky, 2009]. These devices do not have the accuracy of a clinical EEG, however they can detect general patterns in brain activity. These devices are small and inexpensive. Their size and general use of dry sensors makes them more practical than clinical EEG equipment for driver drowsiness detection.

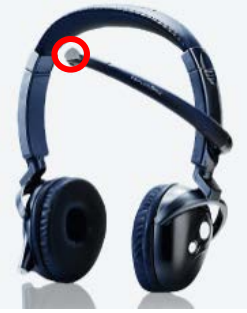

Figure 1: The Neurosky Mindset.

Picture taken from Neurosky Brain Computer Interface Technologies (*Neurosky Mindset)*. Electrode location circled in red.

II. Background/Review of Literature

The Brain During Sleep

Sleep is typically broken into four sleep stages proceeded by Rapid Eye Movement (REM) sleep. Stage 1 sleep is the transition from wakefulness to sleep. At this stage, a person can be woken easily, and may not be aware that they were sleeping. During stage 1 sleep, EEG signals are low amplitude and low frequency. An example of these signals can be seen in Figure 2. During stage 2 sleep, body temperature decreases and the heart rate slows. In stage 2 sleep, alpha waves are periodically interrupted by alpha spindles or sleep spindles. Alpha spindles are 12-14 Hz bursts of brain activity that last at least half a second [Rechtschaffen & Kales, 1968]. These periods of alpha spindle activity are sometimes called alpha spindle epochs. Stages 3 and 4 are deeper sleep, with stage 4 being deeper than stage 3. REM sleep follows stage 4 sleep. REM sleep is most readily identified by rapid eye movement. During REM sleep, dreaming occurs and brain activity increases. Each of these stages continue to cycle from stage 1 through REM sleep throughout the sleeping period.

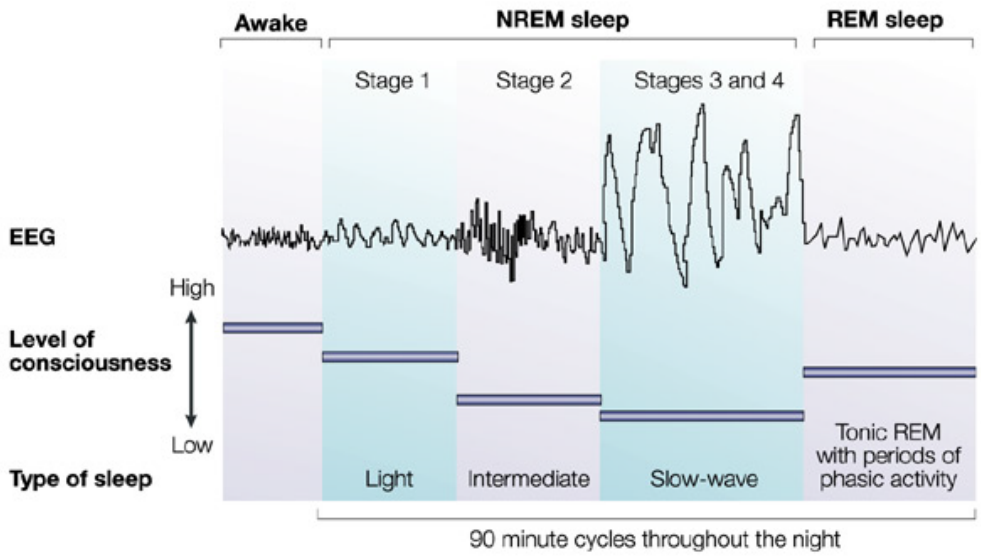

Figure 2: Stages of Sleep in Humans.

Picture taken from Bryant et al. [2004].

The majority of brain activity during the transition from wakefulness to sleep occurs in the frontal and occipital lobes (see Figure 3). During the transition from wakefulness to sleep, alpha activity transitions from the occipital lobe to the frontal lobe. High occipital lobe activity is associated with relaxed wakefulness. During stages 2-4, delta activity in the frontal lobe increases and theta activity in the occipital lobe increases [Finelli, Borbély, & Achermann, 2001]. Methods for sleep detection can be created by observing these features in the brain during the transition from wakefulness to sleep.

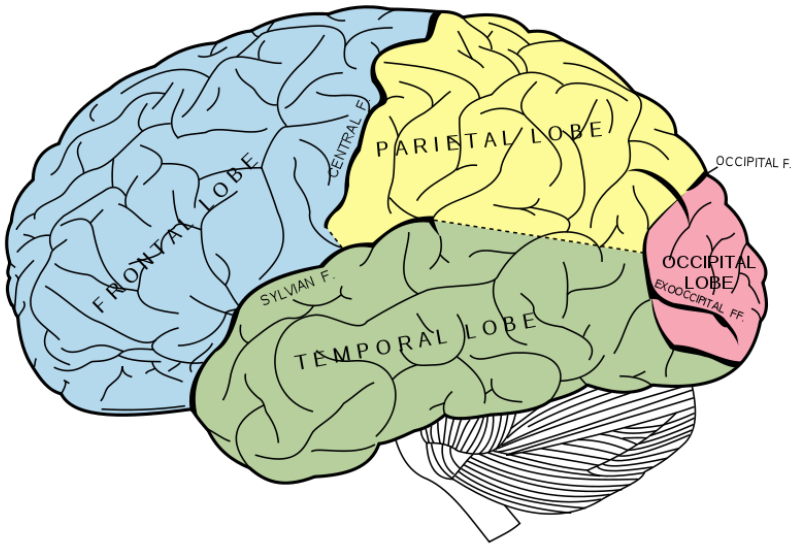

Figure 3: Four Lobes of the Cerebral Cortex of the Brain.

Picture from Wikipedia (*Cerebral Cortex*).

EEG Drowsiness and Sleep Detection

Morrow & Casey [1986] outlined a method for the real-time detection of sleep by focusing on three critical parameters in EEG recordings: waveform amplitude, waveform frequency, and duration of synchronization of the waveform. This last parameter is critical in that the waveform amplitude may meet a predefined voltage threshold for a frequency band for short periods of time, but this does not necessarily indicate sleep unless it meets the threshold for a given duration. Morrow et al. used a 50 μV predefined voltage threshold. The frequencies of focus were 8-12 Hz (Alpha) and 11.5-15 Hz (low Beta). Two counters were used to detect EEG threshold crossing with one counter for the number of sequential pattern matches indicative of sleep, and the other counter for the number of sequential non-matches. When a frequency and amplitude matched the focus frequencies and thresholds the matching counter was incremented. When it did not match, a non-matching counter was incremented. When the match counter

reached 3, sleep was indicated. When the non-matching counter reached 8, wakefulness was indicated. Morrow et al. was able to reach a reliability of 97.9% for 48 of 49 alpha-spindle epochs. In addition, the detector was able to detect about 12.2% more epochs than visual scoring. This methodology may be applicable to Stage 1 sleep detection, but the presence of alpha-spindle epochs indicates a deep stage of sleep.

Merica & Fortune [2004] found that the transition between sleep and wakefulness was a gradual transition. When comparing both delta waves and beta waves to the overall power of the EEG signal, a gradual increase in delta waves was seen along with a gradual increase of beta waves. Merica commented that sleep was, “more one of withdrawal from the waking state rather than of invasion of the sleeping state.” In this way, it may be easier to detect the progression toward sleep rather than its onset. In general, frequencies from 1 Hz to 16 Hz increase while approaching sleep, while frequencies 17 Hz and higher decrease. This pattern continues after the onset of sleep, except that power in the 8-11 Hz range begins to increase as well. The change in the 8-11 Hz signal is especially noticeable in the occipital region where the signal diminishes and transitions to the frontal lobe.

With this gradual transition toward sleep in mind, Lal, Craig, Boord, Kirkup, & Nguyen [2007] documented the creation of an algorithm for detection of drowsiness in drivers. The algorithm classified drowsiness/fatigue EEG signals into transitional (early fatigue phase), transitional–posttransitional (medium fatigue phase), posttransitional (extreme fatigue phase and early Stage 1 of sleep), and arousal phases (emergence from drowsiness). The signal was separated into delta, theta, alpha, and beta waves. An EEG baseline was recorded before the subject was

drowsy. From this baseline, the mean and standard deviation for each of the frequency bands was computed. The algorithm included coefficients to allow for fine-tuning a threshold for each frequency band. A maximum threshold was also hard coded to remove outliers. Data were analyzed in blocks of thirty seconds and this algorithm demonstrated a 10% error rate in sleep detection.

Driver Fatigue Detection

There are many commercial systems that can detect changing driving patterns of fatigued drivers, such as the Ford Driver Alert, the Mercedes Attention Assist, the Volkswagen Driver Alert System, or Volvo Driver Alert Control and Lane Departure Warning [Ford Motor Company, 2011; Taylor, 2008; Volkswagen, 2012; ZerCustoms, 2007]. Each of these systems monitor changes in driving displayed by drowsy drivers like jerky steering movements or drifting out of lanes. When this occurs, an audible and visual warning is produced. Ford's Driver Alert system even includes scenarios in which if the warning is ignored for too long, it can only be discontinued by stopping and exiting the car. The Danish "Anti Sleep Pilot" not only monitors driving patterns, but it requires that the driver push a button on the device as quickly as possible when indicated to verify that the driver's response time is adequate [Coxworth, 2011].

Other systems like the Fraunhofer's Eyetracker or the Toyota Driver Monitoring System monitor the driver's eyes to confirm that they are watching the road [Coxworth, 2010; Williams, 2008]. The driver can be warned if they are not watching the road when an obstacle is ahead. The driver is also warned if their eyes close for a period of time.

Each of these systems has its pros and cons. The erratic driving detection systems can monitor all forms of erratic driving, which includes distracted driving. Their disadvantage is a slow response time so that by the time driving has become erratic; it may be too late to stop an accident from occurring. The driver monitoring systems have the potential to detect driver drowsiness before an accident occurs, but they rely on being able to monitor a driver's eyes to determine if they are open. According to Barr, Howarth, Popkin, & Carroll [2006], "For real-world, in-vehicle applications, sunlight can interfere with IR illumination, reflections from eyeglasses can create confounding bright spots near the eyes, and sunglasses tend to disturb the IR light and make the pupils appear very weak." An EEG based device may be able to detect the onset of sleep before erratic driving has begun, and it has the added benefit of not requiring an unhindered view of the driver's eyes.

Low-Cost EEG devices

Low cost EEG devices have not been on the market for long and relatively little research has been done on them. Most of the limited literature involves detecting the attentiveness of the user. In most cases, these low cost EEG devices proved to be adequate substitutes for medical grade EEG devices for certain low-level signal analysis. Since they usually have fewer sensors than medical grade equipment, they do not do a good job of locating specific signals within the brain.

In one test with a low cost EEG device, Crowley, Sliney, & Murphy [2010] conducted two psychological tests (Stroop's color-word inference test [1935] and the Tower of Hanoi test [Hinz, 1989]), to induce stress and correlate the results to the measured attention and meditation signals recorded from the Neurosky Mindset. By relying on signal trends, they were able to detect when

a subject's emotions changed. In another test, Haapalainen, Kim, Forlizzi, & Dey [2010], collected data from multiple bio-sensors, including a Neurosky Mindset, and compared their ability to assess cognitive load. The attention signal from the device was then given a ranking of third best predictor of high cognitive load after heat flux and electrocardiogram (ECG) across the cognitive experiments conducted.

Tan [2012] used a Neurosky Mindset to record EEG signals for adults and children reading easy or difficult sections of text to determine whether a user was having difficulty reading the text. They were able to distinguish easy/hard text pairings with an accuracy of 70% for adults and 64% for children. The thesis commented that the Neurosky Mindset was shown to be "capable of collecting quality information."

Choi, College and Schwartz [2012] wrote a short article about several low cost EEG devices, including a few Neurosky dry-sensor EEG devices, and their ability to detect drowsiness. They found that these devices made "suitable candidates for further research in the detection of vigilance states." They did not perform real-time analysis of the data or produce a warning when drowsiness was detected.

In the study by Choi et al. [2012], the main area of focus was the frequency domain analysis of EEG collected from the Neurosky dry-sensor EEG device. The frequency content of the signals were divided into clinically relevant frequency bands Alpha (8-13 Hz), Beta (14-30 Hz), and Theta (4-7 Hz) waves. It was expected that, as in clinical studies, alpha and beta waves would decrease when drowsy and theta waves would increase in Stage 1 sleep. In the study using EEG

signals obtained from the Neurosky device, alpha and beta waves did decrease when drowsy, but theta waves remained constant. These results suggest that EEG signals obtained from low-cost EEG devices like the Neurosky may prove a useful target for drowsiness detection schemes.

Goals and Contributions

This thesis proposes the use of the Neurosky Mindset[Neurosky Brain Computer Interface Technologies, 2009], a low-cost EEG device using a single dry-sensor electrode on the forehead at position Fp1 (see figure 4) and grounded with three electrodes on the ear (see figure 5), along with supporting portable hardware and software, to design and validate a standalone system to detect the onset of stage 1 sleep in real-time. Such a system could be used by truck drivers or aircraft pilots to detect and warn for driver drowsiness and inattention with a very quick response time. This system would not need to monitor driving patterns, nor would it require an unhampered view of the driver's eyes. This system will record a single EEG signal via a commercially available dry sensor and apply filters to split the signal into delta (0.5 - 2.75Hz), theta (3.5 - 6.75Hz), low alpha (7.5 - 9.25Hz), high alpha (10 - 11.75Hz), low beta (13 - 16.75Hz), high beta (18 - 29.75Hz), low gamma (31 - 39.75Hz), and mid gamma (41 - 49.75Hz) frequencies. Time-frequency analysis will be implemented to monitor changes in these frequencies over time. Stage 1 sleep is indicated when the amplitude of the raw signal is low, and signal power in higher frequencies has been attenuated. When the EEG transitions resemble that of Stage 1 sleep, the device will produce an auditory alarm.

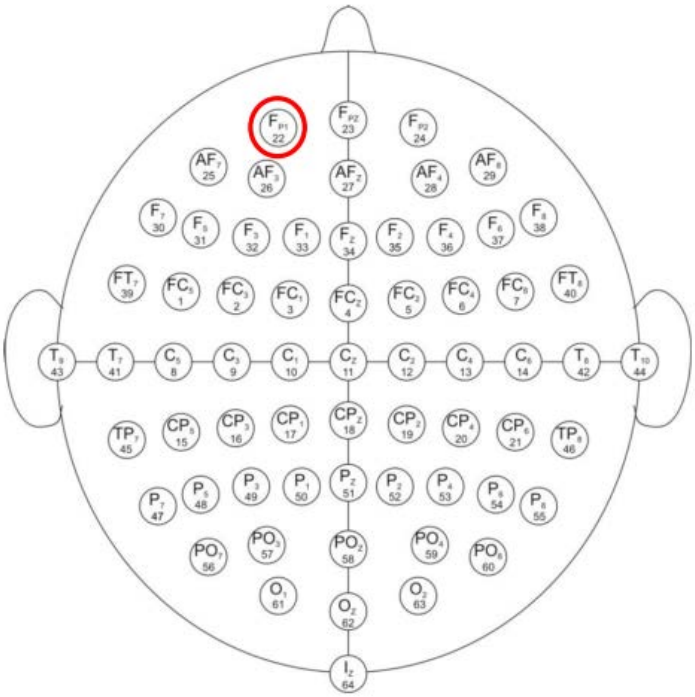

Figure 4: Placement for a 64-electrode System using the International 10-20 standard.

Original Picture taken from Sharbrough, Chatrian, Lesser, Luders, Nuwer, & Picton [1991].

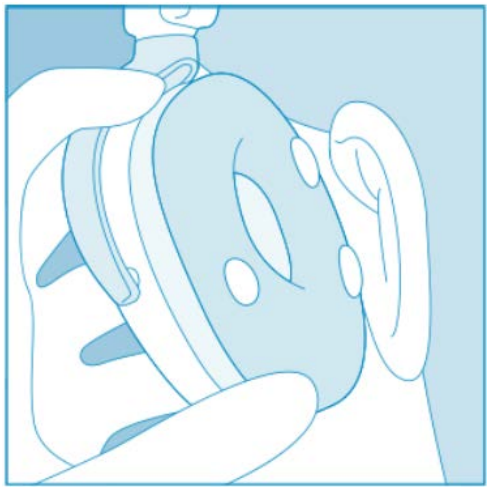

Figure 5: Placement of the Three Ground Electrodes on the Ear.

Picture taken from Neurosky Brain Computer Interface Technologies [Neurosky, 2009].

Of the background literature reviewed, the algorithm by Lal et al [2007] offered the ability to detect each stage of sleep including stage 1 sleep. The algorithm provided by Morrow & Casey [1986] was able to detect alpha epochs with a high degree of accuracy, but alpha epochs occur during stage 2 sleep. To reduce the number of false positives, counters, like those used by Morrow & Casey [1986] should be employed. Combining these methods allows for an algorithm capable of detecting stage 1 sleep while reducing the likelihood of false positives.

Specific Aims

In order to design and test the above described device, the following specific aims will be implemented:

1. Data will be sampled from the Neurosky Mindset at 512 Hz along with estimates of power in the low alpha, high alpha, low beta, and high beta frequency bands.
2. The data will be input to a Bluetooth-enabled microprocessor board.
3. Data will be collected at the following time points: at least 30 seconds of baseline, awake data, and then continuously while a subject is falling asleep.
4. Data in the frequency bands will be analyzed for changes to note differences in measures of mean power and variance.
5. These measures will be used to set thresholds for differentiating between awake and stage 1 sleep.
6. A counter for awake or stage 1 sleep will increment when a power threshold for a specific frequency band has been reached.
7. An auditory alarm will be produced when the maximum counter value is reached.
8. Design criterion for the device are as follows:

a. Audio alarm within 30 seconds of sleep onset
b. Volume of audio alarm to be set at least 70 dB
c. Device size to be less than 125 cubic inches

Challenges in design

There are many problems which can affect the success of a stage 1 sleep detection system. One problem, discussed by Balkin, Horrey, Graeber, Czeisler, & Dinges [2011], is false alarms. In one study to detect fatigue in drivers using EEG, a system had a 14% false alarm rate. This high false alarm rate is unsustainable for a standalone device to detect driver drowsiness as it would prevent consumer acceptance of said device. It will likely affect a user's trust of the device and be a distraction. Care should be taken to balance rapid response with accurate readings, free of false alarms. The device will maximize both sensitivity and specificity, and will additionally provide an auditory and visual warning in the case of unexpected failure such as weak signal or loss of power.

The biggest challenge foreseen with this device is finding the right balance between high sensitivity with high specificity to reduce the potential for false alarms. If the system cannot reliably determine that a person is falling asleep, it will not be a practical solution. Arguably, the more samples that can be gathered and analyzed, the better the results. A baseline signal should be gathered while the subject is awake so that each subject can serve as his or her control against which to compare the subsequent real-time data. The number of samples required must be balanced with the speed of response. The device will not be useful if it can only identify that a

person has fallen asleep several minutes after they the onset of stage 1 sleep. By that point, it would presumably be too late for a driver or pilot.

III. Methods

The Neurosky Mindset

The Neurosky Mindset uses Bluetooth technology to send data to the hardware host for analysis. Since the device uses a dry-sensor, it requires no saline or gel in order to ensure proper connectivity with the surface of the forehead and noise-free EEG signals. Contact with the dry-sensor electrode is achieved by the pressure of the electrode against the subject's forehead and held in place by the headset. It has proprietary filters to eliminate noise from muscle movement and electrical interference. It also includes a notch filter to eliminate 60 Hz noise from a power source. Since there are no wires attaching the electrode to an analysis device, interference due to electrode wire length is greatly reduced.

The Neurosky Mindset can sample data at up to 512 samples per second. In addition to the raw EEG data, the Mindset can output calculated delta (0.5 - 2.75Hz), theta (3.5 - 6.75Hz), low alpha (7.5 - 9.25Hz), high alpha (10 - 11.75Hz), low beta (13 - 16.75Hz), high beta (18 - 29.75Hz), low gamma (31 - 39.75Hz), and mid gamma (41 - 49.75Hz) waves as well as blink strength. It also outputs Neurosky proprietary attention and meditation signals that are meant to identify when a subject is paying attention or is relaxed. The attention and meditation signals will not be used for this research. These attention and meditation signals are created using data from the other frequency bands (e.g. alpha, beta, gamma, etc). These signals are not standard EEG signals, and they do not represent specific frequency bands.

Software

The approach for creating the detection algorithm is based on the work by Lal et al [2007]. The foundation of this analysis is a 30 second baseline signal while the subject is awake. From this 30 second recording (15360 samples), a mean and standard deviation of each frequency band is calculated. With this baseline data, a threshold is established for each frequency band in the form:

$$A_{LST} = a_{LS1}A_{LM} + a_{LS2}A_{LSD}$$

Where A_{LST} is the threshold of low alpha at which a trend suggesting sleep is indicated, A_{LM} and A_{LSD} are the baseline low alpha mean and low alpha standard deviation, and a_{LS1} and a_{LS2} are proportionality constants that define the weighting associated with the baseline mean and standard deviation. These proportionality constants are derived experimentally by comparing the baseline signal to the signal amplitude when sleep is indicated by visual scoring. To visually score an EEG, the data was reviewed for the point in which trend line of the amplitude is near its lowest value, and the slope of the trend line is nearly zero for a sustained period of more than ten seconds. The proportionality constants are determined by relating the mean and standard deviation of a baseline signal to the signal amplitude when sleep is indicated by visual scoring. These proportionality constants are fine-tuned during debugging to improve response time while limiting false positives. After these proportionality constant values were determined, they were use for each subject without modification. In other words, these values needed to be fine-tuned for the specific hardware and software setup, but they did not require additional fine-tuning afterward even if a different subject was tested. Figure 6 describes the process flow for determining the proportionality constants using b_{HS1} as an example.

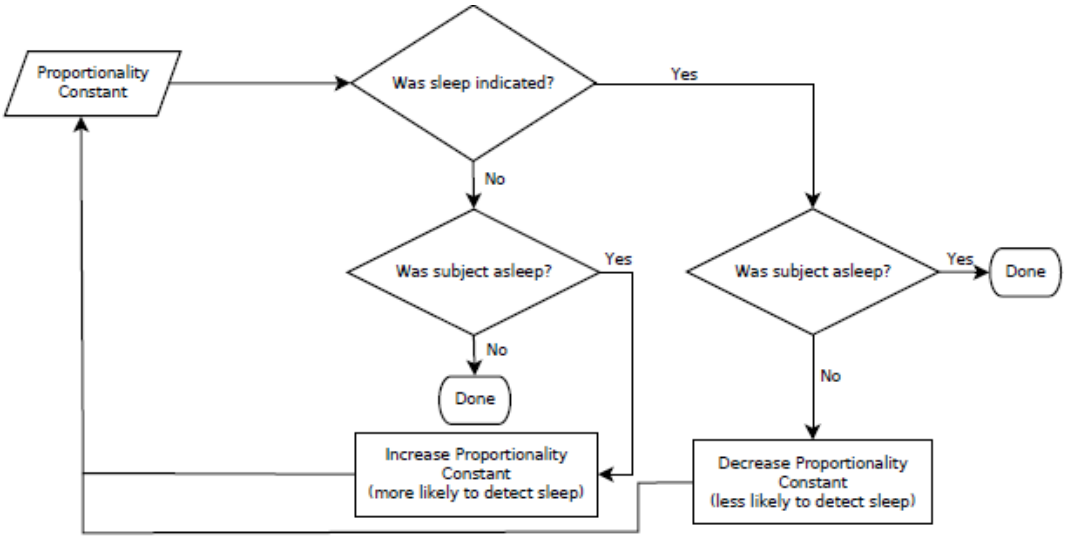

Figure 6: Process for Determining $\mathbf{b_{HS1}}$

Experimental data was used was used to establish initial proportionality constants, identify trends, and determine the correlation to stage 1 sleep for each signal. All of the formulas and the variable definition for the algorithm can be seen in Figure 7 and Figure 8. Using the defined increment and decrement values, sleep was indicated when the sleep counter reached 35. This value was chosen because it was high enough to limit false alarms due to transient signal drops, but it was not so high that it negatively affected responsiveness of the detection. A functional diagram of the software behavior can be seen in Figure 9.

$$A_{LST} = a_{LS1}A_{LM} + a_{LS2}A_{LSD}\text{; If } A_L < A_{LST} \text{ Sleep Counter incremented by 1}$$
$$A_{LWT} = a_{LW1}A_{LM} + a_{LW2}A_{LSD}\text{; If } A_L > A_{LWT} \text{ Sleep Counter decremented by 1}$$
$$A_{HST} = a_{HS1}A_{HM} + a_{HS2}A_{HSD}\text{; If } A_H < A_{HST} \text{ Sleep Counter incremented by 1}$$
$$A_{HWT} = a_{HW1}A_{HM} + a_{HW2}A_{HSD}\text{; If } A_H > A_{HWT} \text{ Sleep Counter decremented by 1}$$
$$B_{LST} = b_{LS1}B_{LM} + b_{LS2}B_{LSD}\text{; If } B_L < B_{LST} \text{ Sleep Counter incremented by 2}$$
$$B_{LWT} = b_{LW1}B_{LM} + b_{LW2}B_{LSD}\text{; If } B_L > B_{LWT} \text{ Sleep Counter decremented by 2}$$
$$B_{HST} = b_{HS1}B_{HM} + b_{HS2}B_{HSD}\text{; If } B_H < B_{HST} \text{ Sleep Counter incremented by 2}$$
$$B_{HWT} = b_{HW1}B_{HM} + b_{HW2}B_{HSD}\text{; If } B_H > B_{HWT} \text{ Sleep Counter decremented by 2}$$
$$R_{WT} = r_{W1}R_M + r_{W2}R_{SD}\text{; If } R > R_{WT} \text{ Sleep Counter decremented by 7}$$
$$R_{STM} = r_{S1}R\text{; If } R < R_{STM} \text{ Sleep Counter incremented by 7}$$

Figure 7: Formulas Used in the Sleep Algorithm.

Each threshold has a different effect on the sleep counter. For example, if the Alpha Low Sleep Threshold is crossed, the sleep counter increments by 1. Each term affects the sleep counter simultaneously. If Beta Low Wake Threshold was also crossed, the counter would be decremented by 2 resulting in a net counter change of -1.

Awake/Asleep Thresholds	**Baseline Values**
A_{LST} ≡ Low Alpha Sleep Threshold	A_{LM} ≡ Low Alpha Mean
A_{LWT} ≡ Low Alpha Wake Threshold	A_{LSD} ≡ Low Standard Deviation
A_{HST} ≡ High Alpha Sleep Threshold	A_{HM} ≡ High Alpha Mean
A_{HWT} ≡ High Alpha Wake Threshold	A_{HSD} ≡ High Standard Deviation
B_{LST} ≡ Low Beta Sleep Threshold	B_{LM} ≡ Low Beta Mean
B_{LWT} ≡ Low Beta Wake Threshold	B_{LSD} ≡ Low Beta Standard Deviation
B_{HST} ≡ High Beta Sleep Threshold	B_{HM} ≡ High Beta Mean
B_{HWT} ≡ High Beta Wake Threshold	B_{HSD} ≡ High Beta Standard Deviation
R_{WT} ≡ Raw Wake Threshold	R_M ≡ Raw Mean
R_{STM} ≡ Raw Sleep Threshold Mean	R_{SD} ≡ Raw Standard Deviation
Proportionality Constants	
a_{LS1} ≡ Low Alpha Sleep Proportionality Constant 1	b_{LW1} ≡ Low Beta Wake Proportionality Constant 1
a_{LS2} ≡ Low Alpha Sleep Proportionality Constant 2	b_{LW2} ≡ Low Beta Wake Proportionality Constant 2
a_{LW1} ≡ Low Alpha Wake Proportionality Constant 1	b_{HS1} ≡ High Beta Sleep Proportionality Constant 1
a_{LW2} ≡ Low Alpha Wake Proportionality Constant 2	b_{HS2} ≡ High Beta Sleep Proportionality Constant 2
a_{HS1} ≡ High Alpha Sleep Proportionality Constant 1	b_{HW1} ≡ High Beta Wake Proportionality Constant 1
a_{HS2} ≡ High Alpha Sleep Proportionality Constant 2	b_{HW2} ≡ High Beta Wake Proportionality Constant 2
a_{HW1} ≡ High Alpha Wake Proportionality Constant 1	r_{W1} ≡ Raw Wake Proportionality Constant 1

$a_{HW2} \equiv$ High Alpha Wake Proportionality Constant 2	$r_{W2} \equiv$ Raw Wake Proportionality Constant 2
$b_{LS1} \equiv$ Low Beta Sleep Proportionality Constant 1	$r_{S1} \equiv$ Raw Sleep Proportionality Constant 1
$b_{LS2} \equiv$ Low Beta Sleep Proportionality Constant 2	
Signal Values	
$A_H \equiv$ High Alpha Signal	$B_L \equiv$ High Alpha Signal
$A_L \equiv$ High Alpha Signal	$R \equiv$ Raw Signal
$B_H \equiv$ High Alpha Signal	

Figure 8: Definition of Formula Terms

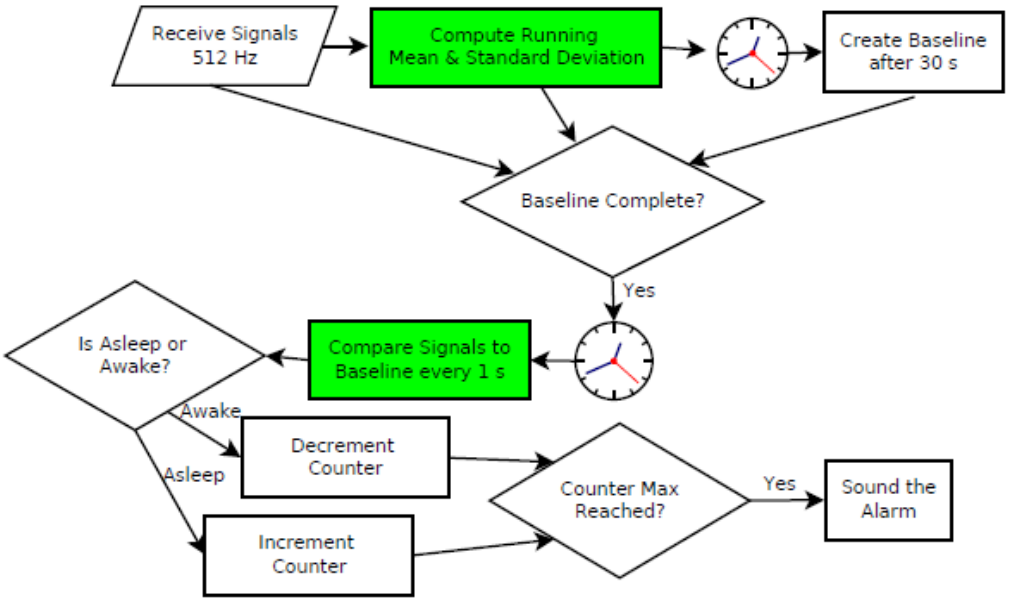

Figure 9: Functional Diagram of Software Behavior.

Elements in green are based on the work of Lal et al [2007] and J.D. Cook [Cook, 2008]

Since much of this algorithm is based on the baseline calculation, great care needs to be taken to ensure that the baseline is a good representative of standard awake brain activity. If the signal strength is too low due to poor skin contact, or too high due to abnormal activity, the baseline could be skewed. The Neurosky Mindset has a built-in detection system to flag noisy signals that can be used to ensure that only noise-free signals are used to create a baseline. Lal et al [2007]

used a maximum limit to throw out signals that were of an unusually high magnitude. Both a poor signal monitor and maximum threshold were employed to reduce the amount of bad data used to create the baseline.

Initial testing of the algorithm was done using a PC based model with a Bluetooth serial interface reading data from the Neurosky Mindset. All data were recorded to a .CSV file for later analysis. After about 30 seconds worth of samples (about 15360 samples), the baseline values were recorded. After the baseline had been recorded, the low alpha, high alpha, low beta, and high beta magnitude calculated by the Neurosky Mindset were sampled once per second. Raw EEG was sampled every second, and the mean raw EEG for the last second was calculated. All of this data was used as an input to the sleep algorithm which ran once each second.

Hardware

The Arduino Uno was chosen as the main processing board for the data retrieval and sleep algorithm analysis. The Arduino has been used with the Neurosky Mindset before, and the interface is well documented [Neurosky 2010a, 2010b]. The BlueSMiRF module was used to retrieve the Bluetooth signals from the Neurosky Mindset and send them to the Arduino. The onboard USB port on the Arduino was used for debugging purposes. The Arduino was powered by a 5V USB power adaptor, which could be used in a car. A wiring diagram showing each component can be seen in figure 10.

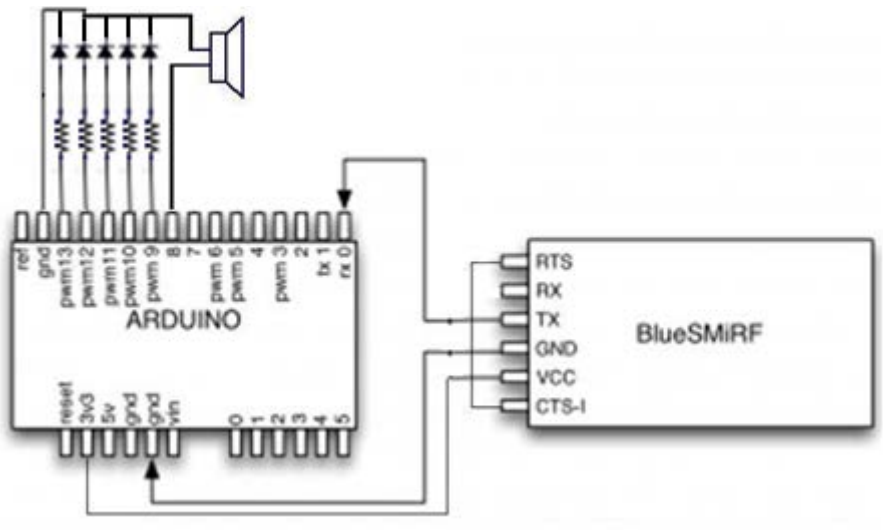

Figure 10: Arduino with BlueSMiRF

Once the device is powered, the piezoelectric buzzer provides the audio indication when sleep has been detected. The piezoelectric buzzer produces high pitched 4000 Hz sound at 70 dBA measured at 12 inches from the device enclosure. The LEDs provide indication when connection to the Neurosky Mindset has been established. A functional diagram of the hardware can be seen in figure 11. After the connection has been established, the LEDs indicate the progression toward sleep as determined by the sleep counter. If a poor connection to the Neurosky Mindset Bluetooth headset or poor contact with the scalp is sensed after the baseline has been recorded, the alarm is activated to alert the user that the device is no longer monitoring drowsiness. The fully assembled unit is 17.77 cubic inches in its enclosure. Additional details about the hardware used can be found in appendix A. A picture of the finished hardware can be seen in Figure 12.

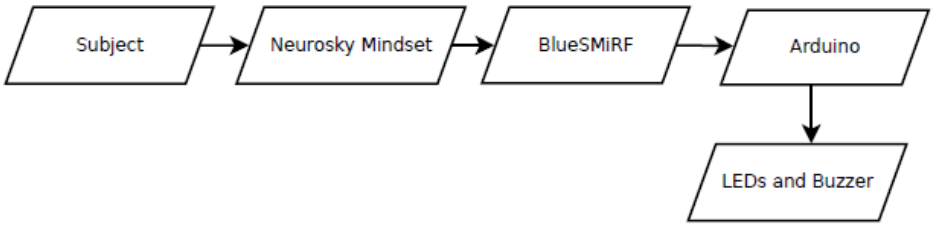

Figure 11: Functional Diagram of the Hardware

Figure 12: Photograph of the Sleep Detection System

Integration

Once the hardware was assembled, hardware and software integration took place. The USB port on the Arduino was used for debugging and fine-tuning the algorithm. The main difference between the PC model and the Arduino model was memory management. Since the Arduino has far less RAM than a PC, values for mean and standard deviation could not be calculated in a typical manner. Typically, each value is stored in an array and then the mean and standard deviation is computed from that array. Because of memory limitations, a running mean and

standard deviation were used based on the calculations by J.D. Cook [Cook, 2008]. This allowed for the mean and standard deviation calculation to be made without requiring storage of long durations of EEG signals.

The algorithm for calculating the running mean and standard deviation were as follows, where M is the mean and s^2 is the variance:

$$M_1 = x_1 \text{ and } S_1 = 0$$

For subsequent x's, use the recurrence formulas

$$M_k = M_{k-1} + (x_k - M_{k-1})/k$$
$$S_k = S_{k-1} + (x_k - M_{k-1})*(x_k - M_k).$$

For $2 \leq k \leq n$, the k^{th} estimate of the variance is $s^2 = S_k/(k - 1)$.

To aid in fine tuning the software, raw, raw mean, low alpha, high alpha, low beta, and high beta measurements were sent over the USB port at about once per second – the same rate as sleep detection was being conducted. The raw signal is was the raw eeg signal magnitude at the moment the signal was read. The raw mean measurement was the mean raw signal magnitude for the last 512 samples (one second). The low alpha, high alpha, low beta, and high beta measurements were the magnitude of each frequency band as calculated by the Neurosky Mindset. The baseline values for raw mean, raw standard deviation, low alpha mean, low alpha standard deviation, high alpha mean, high alpha standard deviation, low beta mean, low beta standard deviation, high beta mean, and high beta standard deviation were also sent over the USB port after the 30 second baseline. Once the sleep algorithm was running, the sleep counter value was sent over USB as well as information about which factors added or subtracted to the

sleep counter within the last second, as seen in Figure 7. Using this information, the sleep counter values could be correlated to the signals to determine when the alarm must go off. These values were fed back into the algorithm to fine tune the performance. These values needed to be fine-tuned for the specific hardware and software setup, but they did not require additional fine-tuning afterward even if a different subject was tested. Similar to Lal et al [2007], a maximum limit was used to throw out signals that were of an unusually high magnitude. The final values were as shown in figure 13.

Proportionality Constant Values and Maximum Signal Limit Value	
a_{LS1} = 0.7	b_{LW1} = 0.9
a_{LS2} = 0	b_{LW2} = 0
a_{LW1} = 0.8	b_{HS1} = 1
a_{LW2} = 0	b_{HS2} = -0.53
a_{HS1} = 0.6	b_{HW1} = 0.9
a_{HS2} = 0	b_{HW2} = 0
a_{HW1} = 0.7	r_{W1} = 3
a_{HW2} = 0	r_{W2} = 0
b_{LS1} = 1	r_{S1} = 0.6
b_{LS2} = -0.57	Maximum Limit = 300

Figure 13: Values Used in the Sleep Algorithm

From initial testing with the PC model, it was noted that low alpha and high alpha were only weakly correlated to sleep. Because of this, sleep was indicated if the alpha signals were sufficiently lower than the mean value. Values for low beta and high beta more strongly correlated with sleep at about half a standard deviation below the mean value. A sudden change of three times the mean raw EEG signal value was given as an indication of wakefulness. To avoid transient signal drop from setting off the raw signal sleep detection, the mean of 512 samples (sampled at 512 Hz) was used. If this mean value was less than 60% of the baseline mean, sleep was indicated. It can be noted that since all of the signals decrease in magnitude as approaching sleep, the thresholds were all set to indicate sleep when the value is below the sleep

threshold and awake when the value is above the awake threshold. All values were fine-tuned during integration testing to avoid false positives while still ensuring a fast response time. Figure 14 shows how the algorithm operated for an example low beta signal. For more information about the software, see appendix B.

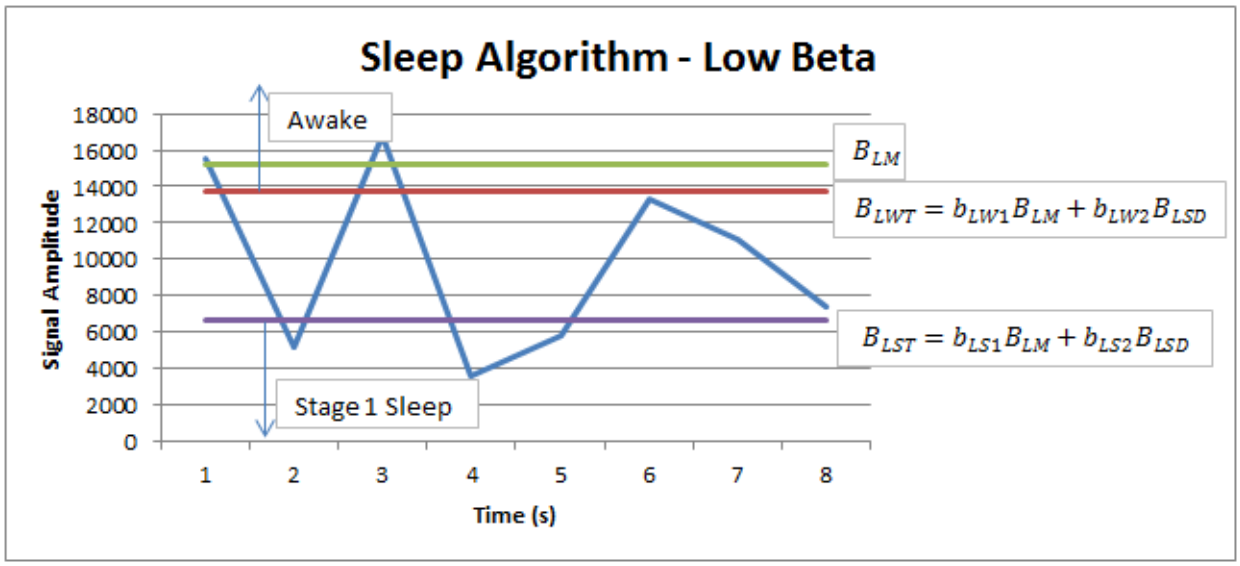

Figure 14: Example of algorithm operating on a Low Beta signal.
When the signal rises above the $\mathbf{B_{LWT}}$ value, the counter decrements. When the signal dips below the $\mathbf{B_{LST}}$ value, the counter increments.

Testing

Tests were performed on three test subjects in two different sleep positions, lying down and seated. Only tests which concluded in a false positive or actual sleep were considered. If the subject failed to sleep and the alarm failed to activate, the test was discarded. The subject was encouraged to get comfortable in a position where they thought sleep was possible. They were informed that they needed to stay awake for a few minutes to record baseline values. Once the baseline was recorded, they could attempt to sleep if they wished. During some tests, the subject immediately tried to sleep. During other tests, the subject continued to read or intentionally stay

awake for a period of time, after which they went to sleep. In three out of sixteen of the tests, the drowsiness alarm went off before the subject was attempting to sleep.

IV. Results

Sample Data

Initial sleep data was gathered from the Neurosky Mindset in order to assess the viability of the sleep data and to start to determine coefficients of the sleep detection algorithm. This data was captured and analyzed for trends preceding and during the early stages of sleep. An example of this data can be seen in Figure 15. This data shows the changes in activity in the each frequency band and the raw EEG signal during the onset of stage 1 sleep. Asleep and awake are marked in each graph.

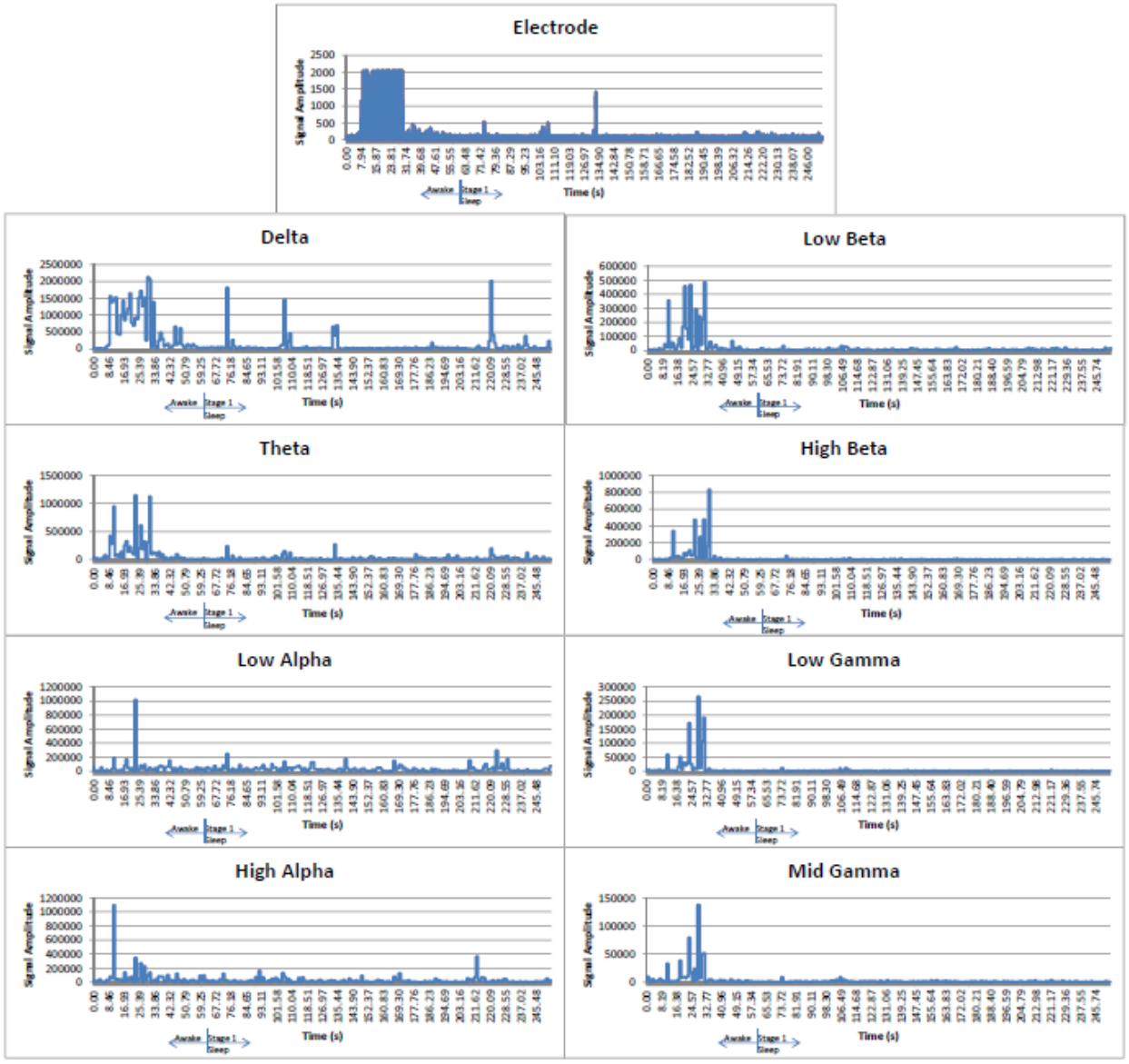

Figure 15: Example Sleep Data Gathered from the Neurosky Mindset

This data showed that the majority of the changes occurred in the low alpha, high alpha, low beta, high beta, and overall raw signal. This is consistent with stage 1 sleep. Stage 1 sleep is indicated when the amplitude of the raw signal is low, and the higher frequencies have dropped off. In figure 15, the transition from awake to asleep is estimated to be at about 60 seconds due to the drop in power in the raw and high and low beta signals. This graph shows that the signal

power in each band drops off during the transition to sleep. This fits with a trend of decreased power compared to the baseline for the high alpha, low beta, high beta, and the raw signal. This drop is most noticeable in the raw and beta signals. The low alpha signal can spike high while awake, but it also periodically spikes after sleep has been reached. This is likely caused by alpha spindle epochs. Since the purpose is to detect the onset of sleep, and not deeper sleep, these alpha spindle epochs do not help detect stage 1 sleep. The data in figure 15 was used to establish initial proportionality constants. The raw signal was most highly correlated to sleep, so it was given the highest weighting with respect to the sleep counter increment or decrement value. Low beta and high beta had the second highest correlation. Low alpha and high alpha were weakly correlated to stage 1 sleep.

Algorithm Effectiveness

In 16 tests, drowsiness was detected in 13 or appropriately 81% of the cases. In 10 cases, the sleep algorithm indicated sleep an average of about 8.4 seconds after stage 1 sleep was identified with estimated visual scoring. In two cases, the sleep algorithm indicated sleep an average of about 38 seconds before stage 1 sleep was identified with estimated visual scoring. In another case, the sleep algorithm indicated sleep about 13 minutes into a 33 minute session in which stage 1 sleep was never clearly identified with estimated visual scoring. This may have been an indication of extreme drowsiness (as the subject was attempting to sleep at this point), or it may have been a false positive. In 3 cases, the sleep algorithm indicated sleep before the subject had begun to attempt to sleep. These cases were considered false positives.

Figure 16 shows the sleep counter and its correlation to each signal. Using estimated visual scoring, stage 1 sleep occurred around second 641.The sleep algorithm indicated sleep at second 647, six seconds later. Asleep and awake are marked in each graph.

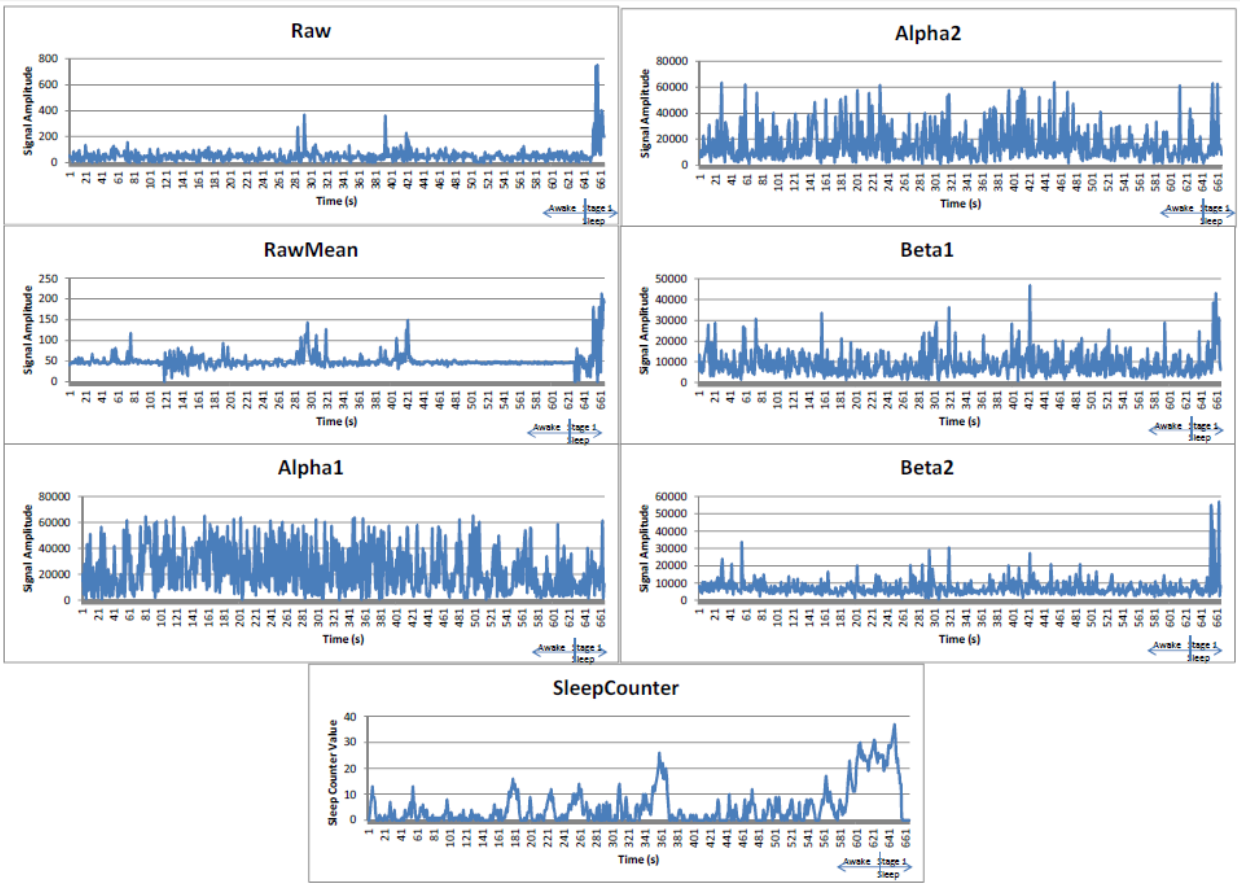

Figure 16: Data Correlated to the Sleep Counter Values

When false alarms occurred using the algorithm, they occurred almost immediately after the baseline was recorded. Figure 17 shows the raw and raw mean correlated to the sleep counter value for a false alarm result. This figure shows the raw signal dropping to less than one sixth of peak power in a few seconds. This tripped a false indication that the subject was asleep. In every case when a false alarm was indicated, the signal started out relatively high immediately after the baseline, and it dropped suddenly afterward while the subject was still awake. Also, in each of

these cases, the baseline mean values and standard deviation values were greater than the cases where there was not a false alarm.

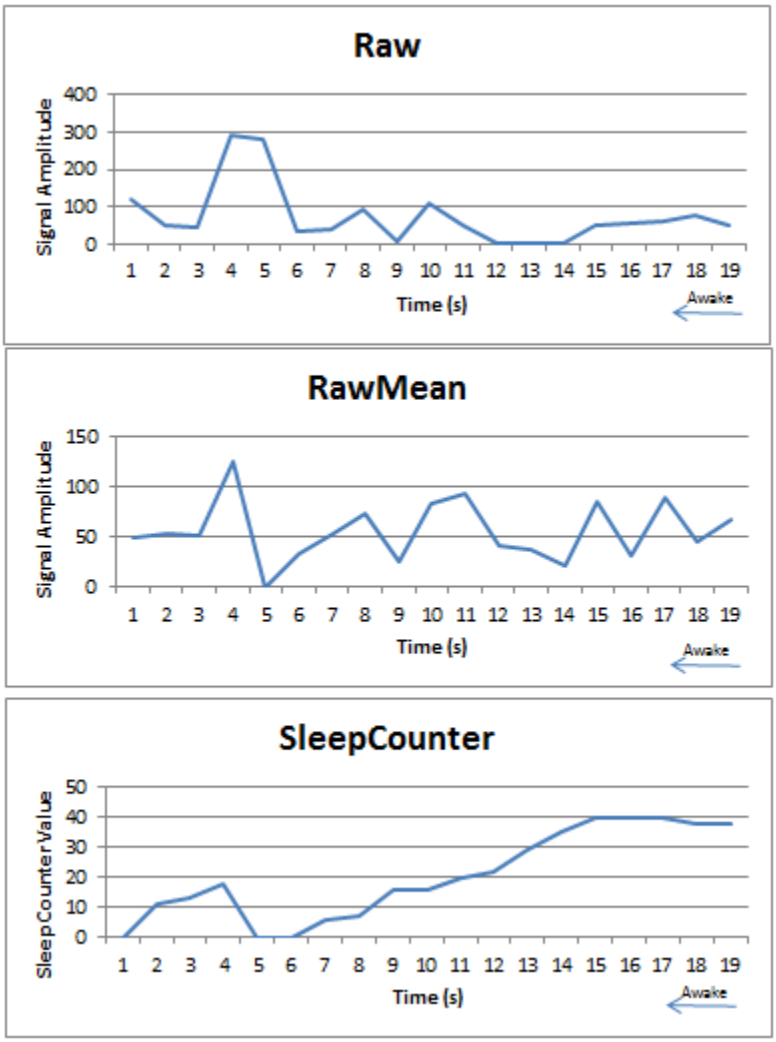

Figure 17: Raw and Raw Mean Correlated to the Sleep Counter for a False Alarm

In some cases the subject reported to have not been quite asleep. Since a major indicator of stage 1 sleep is that a subject does not know they had been asleep, it is difficult to separate micro-sleep sessions from brain activity indicating drowsiness. Although there may have been some

premature alarms, they went off only when the subject was attempting to sleep and indicated that they were drowsy.

To validate the algorithm, three datasets of sleep data from an electrode at position Fp0 from physionet.org was used [Goldberger et al, 2010]. This sleep study data had been reviewed and each sleep stage had been marked. This data was filtered into low alpha, high alpha, low beta, and high beta frequency bands and run through the sleep algorithm. These results mirrored the results from integration testing. The algorithm is tuned to detect the onset of drowsiness and may occasionally produce false alarms. Figure 18 shows a comparison of one of the clinical results versus the algorithm results. The algorithm does not miss the beginning of stage 1 sleep. Instead it warns of the drowsiness at 516 seconds for five seconds at and at 968 seconds for most the rest of the sample duration. According to clinical visual scoring, stage 1 sleep occurred at about 1226 seconds. The sleep algorithm indicated sleep or extreme drowsiness 11.8 and 4.3 minutes before clinical visual scoring identified stage 1 sleep.

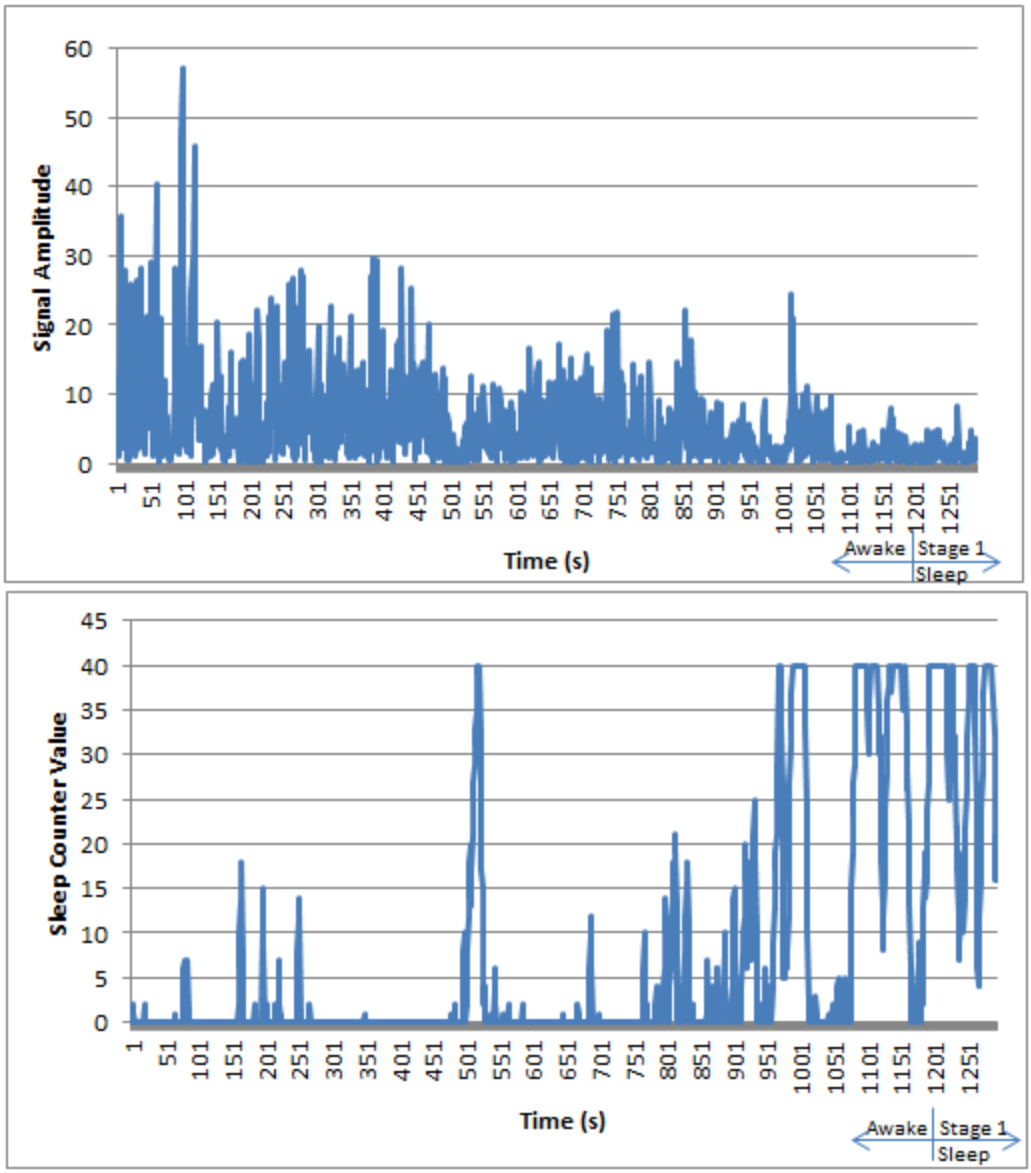

Figure 18: An Example of the Algorithm Results Compared to Clinical Results

For each of the three validated datasets used, sleep was detected. In each case, the algorithm indicated sleep once several minutes before the clinical results indicated sleep, and again one to three minutes before clinical results indicated sleep. On average, the algorithm indicated sleep approximately 120 seconds before clinical results indicated sleep. Figure 19 shows the response time for each of the three datasets that were used for validation.

	Start of Stage 1 Sleep (in seconds)		
	Algorithm	Clinical	Difference
DataSet 1	968	1226	258
DataSet 2	1046	1106	60
DataSet 3	724	766	42

Figure 19: Algorithm Results Compared to Clinical Results

In reviewing the baseline values for each recorded dataset, the baseline values for datasets which resulted in a false alarm were higher than for a successful sleep detection case. It was also interesting to note that the baselines which resulted in accurate sleep detection varied very little from each other. An average of all of the baseline values was calculated, and the majority of the signals were within 8% of these aggregated mean values.

V. Discussion and Conclusions

Discussion

The greatest challenge in the design of this device was the obtaining an accurate baseline for when the subject was awake. This baseline varied more than expected depending on the task being performed during the baseline, the posture of the subject (sitting versus lying down), and the initial drowsiness of the subject. A maximum signal limit was added to remove some of the larger spikes from things like sneezing during the baseline. If the signal was skewed higher because of excess movement or some other transient activity during the baseline recording period, but still below the maximum limit, the sleep algorithm would indicate sleep prematurely. Conversely, if the subject was already greatly fatigued when the baseline was recorded, it would affect the responsiveness of the algorithm, and potentially prevent the algorithm from ever detecting sleep.

On reviewing the cases where sleep was accurately detected and comparing them to the false alarms, it was noted that the baselines which resulted in accurate sleep detection varied very little compared to each other. When the baseline mean values across all tests were averaged per person, it was found that the best response was when the baseline values were closer to this aggregated baseline. In other words, the longer the duration of recorded baseline activity, the better the estimates of mean and standard deviation. These improved estimates directly improved the detection of sleep onset.

If baseline data does not vary much over time, and longer baselines appear to be more accurate, it may be beneficial in future work to eliminate the 30 second baseline that occurs before using the device. Instead, a learning mode should be added which takes a baseline over several minutes. This baseline should be conducted under the normal use case. In other words, if the device is to be used during driving, the baseline should record several minutes of standard driving activity. This baseline would be stored to flash rather than RAM so that it could be saved after the device has been powered down. When the user later uses the device for drowsiness detection, the prerecorded baseline from the learning mode would be used rather than recording a new baseline.

Saving the baseline values would have additional benefits. This would make the device a bit easier to use, and it would reduce some of the false positives. It would have the added benefit of being able to detect drowsiness as soon as the device is turned on and collecting data. If the driver is already drowsy when they activate the device, the device could indicate that they are

impaired immediately. Right now, it is assumed that the driver is not drowsy when the baseline is recorded.

Since the baseline is recorded over several minutes rather than thirty seconds, the maximum limit filter may not be necessary. Over a long enough time, outlier signals (like sneezing) will not influence the detection of sleep. This would place a great emphasis on the learning mode calibration. If unusually high or low brain activity were occurring during the baseline, every use of the device afterward would be suspect. For example, if the baseline was taken while the subject was nearly asleep, it would be very difficult to detect drowsiness.

Future work needs to focus on testing. More tests on the system should be performed, especially using simulated driving. In all of the tests performed for this thesis, the subject attempted to sleep. There was not adequate time to perform testing on sleep deprived subjects who were trying to stay awake, but if this device or one like it is to be used in an automotive or aerospace setting, testing must be conducted on its ability to detect this type of uncontrolled sleep and drowsiness.

Conclusion

Low cost EEG devices such as the Neurosky Mindset can be used to detect driver drowsiness. Even with the limited hardware capability and a single electrode, the device used in this study proved 81% effective at detecting drowsiness in a small sample group. In 62% of the cases, the sleep algorithm indicated stage 1 sleep an average of 8.4 seconds after stage 1 sleep was indicated with estimated visual scoring. In 19% of the cases, the sleep algorithm indicated sleep 30 seconds to 20 minutes before stage 1 sleep was indicated with estimated visual scoring. The

remaining 19% of the cases were false positives in which the sleep algorithm indicated sleep, but the subject was not yet attempting to sleep. With a few modifications, such a system would be suitable for aircraft pilots or truck drivers to receive an auditory warning when they are beginning to become drowsy. Difficulties arise in comparing the current values to a baseline values. Great care should be taken in recording baseline values from which to compare current brain activity. Future research should focus on getting better, more permanent baseline values over a longer time. This should improve accuracy and response, and reduce the number of false alarms.

Bibliography

Balkin, T. J., Horrey, W. J., Graeber, R. C., Czeisler, C. a, & Dinges, D. F. (2011). The challenges and opportunities of technological approaches to fatigue management. *Accident; analysis and prevention*, *43*(2), 565–72. doi:10.1016/j.aap.2009.12.006

Barr, L., Howarth, H., Popkin, S., & Carroll, R. J. (2006). A Review and Evaluation of Emerging Driver Fatigue, 1–27.

Bryant, P. a, Trinder, J., & Curtis, N. (2004). Sick and tired: Does sleep have a vital role in the immune system? *Nature reviews. Immunology*, *4*(6), 457–67. doi:10.1038/nri1369

Caldwell, J. a., Mallis, M. M., Caldwell, J. L., Paul, M. a., Miller, J. C., & Neri, D. F. (2009). Fatigue Countermeasures in Aviation. *Aviation, Space, and Environmental Medicine*, *80*(1), 29–59. doi:10.3357/ASEM.2435.2009

Cerebral Cortex. In *Wikipedia.*Retreived April 24, 2013, from http://en.wikipedia.org/wiki/Cerebral_cortex

Choi, H. S., College, S. M., & Schwartz, G. (2012). Using Brain-Computer Interfaces to Analyze EEG Data for Safety Improvement.

Cook, J. D. (2008) Accurately computing running variance. Retrieved February 5, 2013, from http://www.johndcook.com/standard_deviation.html

Coxworth, B. (2010). Eyetracker watches drivers' eyes for signs of drowsiness. *gizmag*. Retrieved November 12, 2012, from http://www.gizmag.com/fraunhofer-eyetracker-driver-monitoring-system/16643/

Coxworth, B. (2011). Anti Sleep Pilot detects drowsy drivers. *gizmag*. Retrieved November 12, 2012, from http://www.gizmag.com/anti-sleep-pilot-monitors-driver-fatigue/17439/

Crowley, K., Sliney, A. and Murphy, I.P.D. (2010). Evaluating a Brain-Computer Interface to Categorise Human Emotional Response. In Proceedings of the 10th IEEE International Conference on Advanced Learning Technologies, 5-7 July, 2010, 276-278.

Finelli, L. a, Borbély, a a, & Achermann, P. (2001). Functional topography of the human nonREM sleep electroencephalogram. *The European journal of neuroscience*, *13*(12), 2282–90. Retrieved from http://www.ncbi.nlm.nih.gov/pubmed/11454032

Ford Motor Company (2011). Ford's Wake-Up Call for Europe's Sleepy Drivers. Retrieved November 12, 2012, from http://media.ford.com/article_display.cfm?article_id=34562

Goldberger, AL, Amaral, LAN, Glass, L, Hausdorff, JM, Ivanov, PCh, Mark, RG, Mietus, JE, Moody, GB, Peng, C-K, Stanley, HE. PhysioBank, PhysioToolkit, and PhysioNet:

Components of a New Research Resource for Complex Physiologic Signals. Circulation 101(23):e215-e220 [Circulation Electronic Pages; http://circ.ahajournals.org/cgi/content/full/101/23/e215]; 2000 (June 13).

Haapalainen, E., Kim, S., Forlizzi, J.F. and Dey, A.K. (2010). Psycho-physiological measures for assessing cognitive load. In Proceedings of the 12th ACM International Conference on Ubiquitous Computing, 2010, 301-310.

Hinz, A. (1989). The Tower of Hanoi. Enseignement Mathematique 35, 239-321.

Lal, S. K. L., & Craig, A. (2002). Driver fatigue: electroencephalography and psychological assessment. *Psychophysiology*, *39*(3), 313–21. doi:10.1017.S0048577201393095

Lal, S. K. L., Craig, A., Boord, P., Kirkup, L., & Nguyen, H. (2007). Development of an algorithm for an EEG-based driver fatigue countermeasure, *34*(2003), 321–328. doi:10.1016/S0022-4375(03)00027-6

Loomis, A. L., Harvey, E. N., & Hobart, Garret A, I. (1937). Cerebral States During Sleep, as Studied by Human Brain Potentials. *Journal of Experimental Psychology*, *21*(2), 127–144.

Merica, H., & Fortune, R. D. (2004). State transitions between wake and sleep, and within the ultradian cycle, with focus on the link to neuronal activity. *Sleep medicine reviews*, *8*(6), 473–85. doi:10.1016/j.smrv.2004.06.006

Morrow, T. J., & Casey, K. L. (1986). A microprocessor device for the real-time detection of synchronized alpha and spindle activity in the EEG. *Brain research bulletin*, *16*(3), 439–42. Retrieved from http://www.ncbi.nlm.nih.gov/pubmed/2939925

National Sleep Foundation. (2012a). Sleepy Pilots Train Operators and Drivers. Retrieved November 11, 2012, from drivingdrowsy.org

National Sleep Foundation. (2012b). Facts and Stats. Retrieved November 11, 2012, from drivingdrowsy.org

National Transportation Safety Board. (2010). Loss of Control on Approach Colgan Air, Inc. Operating as Continental Connection Flight 3407 Bombardier DHC-8-400, N200WQ Clarence Center, New York February 12, 2009. *Aircraft Accident Report*.

Neurosky Brain Computer Interface Technologies. *Neurosky Mindset* [Photograph]. Retrieved April 24, 2013 from http://www.neurosky.com/Images/LocalizedImage?PageName=/PRODUCTS/MINDSET.ASPX&ImageName=Left2

Neurosky Brain Computer Interface Technologies (2009). *NeuroSky MindSet Instruction Manual*.

Neurosky Brain Computer Interface Technologies. (2010a) *Interfacing the MindSet with Arduino*.

Neurosky Brain Computer Interface Technologies. (2010b). *MindSet Communications Protocol*.

Rechtschaffen A, Kales A. (1968). A manual of standardized terminology, techniques and scoring system for sleep stages of human subjects. Los Angeles: UCLA Brain Information Service. Brain Research Institute.

Sharbrough, F., Chatrian, C., Lesser, R., Luders, H., Nuwer, M. and Picton, T. (1991). American Electroencephalographic Society guidelines for standard electrode position nomenclature. Clinical Neurophysiology 8, 200–202.

Stroop, J.R. (1935). Studies of Interference in Serial Verbal Reactions. Journal of Experimental Psychology 18(6), 643-662.

Tan, B. H. (2012). Using a Low-cost EEG Sensor to Detect Mental States, (August).

Taylor, M. (2008). No Doze: Mercedes E-Class alerts drowsy drivers. *Autoweek*. Retrieved November 12, 2012, from http://www.autoweek.com/article/20081224/FREE/812249991

Volkswagen. Driver Alert System. Retrieved November 12, 2012, from http://www.volkswagen.co.uk/new/passat-alltrack-gp/explore/experience/driver-assistance/driver-alert-system

Williams, M. (2008). Toyota Cars to Monitor Driver's Eyes for Safety. *PCWorld*. Retrieved November 12, 2012, from http://www.pcworld.com/article/141603/article.html

ZerCustoms (2007). *Volvo-Driver-Alert-Control-and-Lane-Departure-Warning*. Retrieved November 12, 2012, from http://www.zercustoms.com/news/Volvo-Driver-Alert-Control-and-Lane-Departure-Warning.html

Appendix A – Hardware

Below is a bill of materials used in this design:

1 Radio Shack 273-0059 75 dB Piezo Buzzer

5 Radio Shack 276-1622 5mm LEDs (2 yellow, 2 red, 1 green)

5 Radio Shack 276-0079 Panel-Mount 5mm LED Holders.

5 150 Ohm resistors

1 SparkFun WRL-10269 BlueSMiRF Silver Bluetooth Modem

1 SparkFun DEV-11021 Arduino Uno R3

1 SparkFun PRT-10088 - Arduino Project Enclosure

Appendix B – Software

```
#define LED9 9
#define LED10 10
#define LED11 11
#define LED12 12
#define LED13 13
#define BAUDRATE 57600
#define DEBUGOUTPUT 0
#define BLUESMIRFON 2

// neuro data variables
byte errorRate = 200;
byte attention = 0;
byte meditation = 0;
byte blinkstrength = 0;
bool NewData = false;
short raw;
unsigned int rawpwr;
unsigned int delta;
unsigned int theta;
unsigned int alpha1;
unsigned int alpha2;
unsigned int beta1;
unsigned int beta2;
unsigned int gamma1;
unsigned int gamma2;

// system variables
unsigned long lastReceivedPacket = micros();
unsigned long totalTime = 0;
boolean newRawData = false;
boolean bigPacket = false;

const int MaxValue = 40;
const int MinValue = 0;
const int Threshold = 35;
const int RawLimit = 300;
bool IsBaseline = true;
bool FirstRun = true;
bool IsAsleep = false;
const int BaselineSize = 15360;
const int BufferSize = 512;
const int FlushLimit = 1024;
int bufffill = 0;
int SleepCounter = 0;
int HeartBeatCounter = 0;
const int HeartBeatPeriod = 512;

int BaselineRawMean;
int BaselineRawStdDev;
int BaselineAlpha1Mean;
int BaselineAlpha1StdDev;
int BaselineAlpha2Mean;
int BaselineAlpha2StdDev;
int BaselineBeta1Mean;
int BaselineBeta1StdDev;
```

```
int BaselineBeta2Mean;
int BaselineBeta2StdDev;

class RunningStat
    {
    public:
        RunningStat() : m_n(0) {}

        void Clear()
        {
            m_n = 0;
        }

        void Push(unsigned int x)
        {
            m_n++;
            // See Knuth TAOCP vol 2, 3rd edition, page 232
            if (m_n == 1)
            {
                m_oldM = m_newM = double(x);
                m_oldS = 0.0;
            }
            else
            {
                m_newM = m_oldM + (x - m_oldM)/m_n;
                m_newS = m_oldS + (x - m_oldM)*(x - m_newM);

                // set up for next iteration
                m_oldM = m_newM;
                m_oldS = m_newS;
            }
        }

        int NumDataValues() const
        {
            return m_n;
        }

        double Mean() const
        {
            return (m_n > 0) ? m_newM : 0.0;
        }

        double Variance() const
        {
            return ( (m_n > 1) ? m_newS/(m_n - 1) : 0.0 );
        }

        double StandardDeviation() const
        {
            return sqrt( Variance() );
        }

    private:
        int m_n;
        double m_oldM, m_newM, m_oldS, m_newS;
    };
```

```
RunningStat RawStats;
RunningStat Alpha1Stats;
RunningStat Alpha2Stats;
RunningStat Beta1Stats;
RunningStat Beta2Stats;

//////////////////////////
// Microprocessor Setup //
//////////////////////////
void setup() {

  pinMode(LED9, OUTPUT);
  pinMode(LED10, OUTPUT);
  pinMode(LED11, OUTPUT);
  pinMode(LED12, OUTPUT);
  pinMode(LED13, OUTPUT);
  pinMode(BLUESMIRFON, OUTPUT);
  digitalWrite(BLUESMIRFON, HIGH);
  Serial.begin(BAUDRATE);           // USB
  Serial.println("Boot Complete.");
}


/////////////
//MAIN LOOP//
/////////////
void loop() {
       ReadData();

#if !DEBUGOUTPUT

  // *** Add your code here ***

  if(bigPacket) {
    bigPacket = false;
  }

#endif
  if (NewData)
  {
    AnalyzeBrainWaves();

    if (HeartBeatCounter == HeartBeatPeriod)
    {
           HeartBeat();
           HeartBeatCounter = 0;
    }
       HeartBeatCounter++;
  }
}

void SerialSleepNumber()
{
       Serial.println(SleepCounter, DEC);
}
void SerialPrintBaseline()
{
        Serial.println("Baseline:");
```

```
Serial.println("RawMean,RawStdDev,Alpha1Mean,Alpha1StdDev,Alpha2Mean,Alpha2StdDev,Beta1Me
an,Beta1StdDev,Beta2Mean,Beta2StdDev");
         Serial.print(BaselineRawMean, DEC);
      Serial.print(",");
      Serial.print(BaselineRawStdDev, DEC);
      Serial.print(",");
      Serial.print(BaselineAlpha1Mean, DEC);
      Serial.print(",");
      Serial.print(BaselineAlpha1StdDev, DEC);
      Serial.print(",");
         Serial.print(BaselineAlpha2Mean, DEC);
      Serial.print(",");
         Serial.print(BaselineAlpha2StdDev, DEC);
      Serial.print(",");
         Serial.print(BaselineBeta1Mean, DEC);
      Serial.print(",");
         Serial.print(BaselineBeta1StdDev, DEC);
      Serial.print(",");
         Serial.print(BaselineBeta2Mean, DEC);
      Serial.print(",");
         Serial.println(BaselineBeta2StdDev, DEC);
}

bool IsHeaderPrinted = false;
String debugtext = "";
void SerialPrintData()
{
         if (!IsHeaderPrinted)
         {
              Serial.println("Raw,RawMean,Alpha1,Alpha2,Beta1,Beta2,SleepCounter");
              IsHeaderPrinted = true;
         }
         Serial.print(",");
         Serial.print(rawpwr, DEC);
      Serial.print(",");
         Serial.print(int(RawStats.Mean()), DEC);
      Serial.print(",");
      Serial.print(alpha1, DEC);
      Serial.print(",");
         Serial.print(alpha2, DEC);
      Serial.print(",");
         Serial.print(beta1, DEC);
      Serial.print(",");
         Serial.print(beta2, DEC);
      Serial.print(",");
         Serial.println(SleepCounter, DEC);
}


////////////////////////////////
// Read data from Serial UART //
////////////////////////////////
byte ReadOneByte() {
  int ByteRead;

  while(!Serial.available());
  ByteRead = Serial.read();
```

```
#if DEBUGOUTPUT
  Serial.print((char)ByteRead);   // echo the same byte out the USB serial (for debug
purposes)
#endif

  return ByteRead;
}

////////////////////////////////////////
// Reading data from MindSet bluetooth //
////////////////////////////////////////
byte lastchecksum = 0;
void ReadData() {
  static unsigned char payloadData[256];
  byte generatedChecksum;
  byte checksum;
  byte vLength;
  int payloadLength;
  int powerLength = 3; // defined in MindSet Communications Protocol
  int k;


  // Look for sync bytes
  if(ReadOneByte() == 170) {
    if(ReadOneByte() == 170) {

      do { payloadLength = ReadOneByte(); }
      while (payloadLength == 170);

      if(payloadLength > 170) {    //Payload length can not be greater than 170
         return;
      }

      generatedChecksum = 0;
      for(int i = 0; i < payloadLength; i++) {
        payloadData[i] = ReadOneByte();            //Read payload into memory
        generatedChecksum += payloadData[i];
      }

      checksum = ReadOneByte();                      //Read checksum byte from stream
         generatedChecksum = 255 - generatedChecksum;   //Take one's compliment of
generated checksum

      if(checksum != generatedChecksum) {
        // checksum error
      } else {

        for(int i = 0; i < payloadLength; i++) {    // Parse the payload
          switch (payloadData[i]) {
          case 2:
            bigPacket = true;
            i++;
            errorRate = payloadData[i];
            break;
          case 4:
            i++;
            attention = payloadData[i];
```

```
            break;
          case 5:
            i++;
            meditation = payloadData[i];
            break;
                case 0x16:
            i++;
            blinkstrength = payloadData[i];
            break;
          case 0x80: // raw data
            newRawData = true;
            i++;
            vLength = payloadData[i];
            raw = 0;
            for (int j=0; j<vLength; j++) {
              raw = raw | ( payloadData[i+vLength-j]<<(8*j) ); // bit-shift little-endian
            }
            i += vLength;
            rawpwr=abs(raw);
                     break;
          case 0x83:  // power data
            i++;
            vLength = payloadData[i];
            k = 0;

            // parse power data starting at the last byte
            gamma2 = 0; // mid-gamma (41 - 49.75Hz)
            for (int j=0; j<powerLength; j++) {
              gamma2 = gamma2 | ( payloadData[i+vLength-k]<<(8*j) ); // bit-shift little-
endian
              k++;
            }
            gamma1 = 0; // low-gamma (31 - 39.75Hz)
            for (int j=0; j<powerLength; j++) {
              gamma1 = gamma1 | ( payloadData[i+vLength-k]<<(8*j) ); // bit-shift little-
endian
              k++;
            }
            beta2 = 0; // high-beta (18 - 29.75Hz)
            for (int j=0; j<powerLength; j++) {
              beta2 = beta2 | ( payloadData[i+vLength-k]<<(8*j) ); // bit-shift little-
endian
              k++;
            }
            beta1 = 0; // low-beta (13 - 16.75Hz)
            for (int j=0; j<powerLength; j++) {
              beta1 = beta1 | ( payloadData[i+vLength-k]<<(8*j) ); // bit-shift little-
endian
              k++;
            }
            alpha2 = 0; // high-alpha (10 - 11.75Hz)
            for (int j=0; j<powerLength; j++) {
              alpha2 = alpha2 | ( payloadData[i+vLength-k]<<(8*j) ); // bit-shift little-
endian
              k++;
            }
            alpha1 = 0; // low-alpha (7.5 - 9.25Hz)
            for (int j=0; j<powerLength; j++) {
```

```
              alpha1 = alpha1 | ( payloadData[i+vLength-k]<<(8*j) ); // bit-shift little-
endian
              k++;
            }
            theta = 0; // theta (3.5 - 6.75Hz)
            for (int j=0; j<powerLength; j++) {
              theta = theta | ( payloadData[i+vLength-k]<<(8*j) ); // bit-shift little-
endian
              k++;
            }
            delta = 0; // delta (0.5 - 2.75Hz)
            for (int j=0; j<powerLength; j++) {
              delta = delta | ( payloadData[i+vLength-k]<<(8*j) ); // bit-shift little-
endian
              k++;
            }

            i += vLength;
            break;
          default:
            break;
          } // switch
        } // for loop
      } // checksum success
    } // sync 2
  } // sync 1

  //Check if data read is actually new.
  if (checksum !=lastchecksum)
  {
         NewData=true;
         lastchecksum=checksum;
  }
  else
  {
         NewData=false;
  }
} // ReadData

void HeartBeat()
{
       if (attention == 0)
       {

              LEDWrite(errorRate, 200, true);
               if (!FirstRun & !IsBaseline)
               {
                      SoundTheAlarm();
               }
              Serial.print("P/S:");
              Serial.println(errorRate,DEC);
       }
       else
       {
              LEDWrite(SleepCounter, Threshold, false);
              SerialPrintData();
              debugtext = "";
       }
```

```
}

void LEDWrite(int value, int max, bool countdown)
{
         if (countdown)
           {
          switch(value / (max / 5) ) {
          case 0:
            digitalWrite(LED13, HIGH);
            digitalWrite(LED12, HIGH);
            digitalWrite(LED11, HIGH);
            digitalWrite(LED10, HIGH);
            digitalWrite(LED9, HIGH);
            break;
          case 1:
            digitalWrite(LED13, HIGH);
            digitalWrite(LED12, HIGH);
            digitalWrite(LED11, HIGH);
            digitalWrite(LED10, HIGH);
            digitalWrite(LED9, LOW);
            break;
          case 2:
            digitalWrite(LED13, HIGH);
            digitalWrite(LED12, HIGH);
            digitalWrite(LED11, HIGH);
            digitalWrite(LED10, LOW);
            digitalWrite(LED9, LOW);
            break;
          case 3:
            digitalWrite(LED13, HIGH);
            digitalWrite(LED12, HIGH);
            digitalWrite(LED11, LOW);
            digitalWrite(LED10, LOW);
            digitalWrite(LED9, LOW);
            break;
          case 4:
            digitalWrite(LED13, HIGH);
            digitalWrite(LED12, LOW);
            digitalWrite(LED11, LOW);
            digitalWrite(LED10, LOW);
            digitalWrite(LED9, LOW);
            break;
          case 5:
            digitalWrite(LED13, HIGH);
            digitalWrite(LED12, LOW);
            digitalWrite(LED11, LOW);
            digitalWrite(LED10, LOW);
            digitalWrite(LED9, LOW);
            break;
          }
           }
         else
           {
          switch(value / ((max / 5) - 1) ) {
          case 0:
            digitalWrite(LED13, LOW);
            digitalWrite(LED12, LOW);
            digitalWrite(LED11, LOW);
```

```
            digitalWrite(LED10, LOW);
            digitalWrite(LED9, LOW);
            break;
          case 1:
            digitalWrite(LED13, LOW);
            digitalWrite(LED12, LOW);
            digitalWrite(LED11, LOW);
            digitalWrite(LED10, LOW);
            digitalWrite(LED9, HIGH);
            break;
          case 2:
            digitalWrite(LED13, LOW);
            digitalWrite(LED12, LOW);
            digitalWrite(LED11, LOW);
            digitalWrite(LED10, HIGH);
            digitalWrite(LED9, HIGH);
            break;
          case 3:
            digitalWrite(LED13, LOW);
            digitalWrite(LED12, LOW);
            digitalWrite(LED11, HIGH);
            digitalWrite(LED10, HIGH);
            digitalWrite(LED9, HIGH);
            break;
          case 4:
            digitalWrite(LED13, LOW);
            digitalWrite(LED12, HIGH);
            digitalWrite(LED11, HIGH);
            digitalWrite(LED10, HIGH);
            digitalWrite(LED9, HIGH);
            break;
          case 5:
            digitalWrite(LED13, HIGH);
            digitalWrite(LED12, HIGH);
            digitalWrite(LED11, HIGH);
            digitalWrite(LED10, HIGH);
            digitalWrite(LED9, HIGH);
            break;
          }
           }
}

void AsleepAlgorithm()
{
       if ((beta1 < (BaselineBeta1Mean - (0.57 * BaselineBeta1StdDev)))) {
              SleepCounter = (SleepCounter < (MaxValue - 1)) ? SleepCounter + 2 :
MaxValue;
              Serial.print("b1+");
       } else if (beta1 > (0.9 * BaselineBeta1Mean)) {
              SleepCounter = (SleepCounter > (MinValue + 1)) ? SleepCounter - 2 :
MinValue;
              Serial.print("b1-");
       }
       if ((beta2 < (BaselineBeta2Mean - (0.53 * BaselineBeta2StdDev)))) {
              SleepCounter = (SleepCounter < (MaxValue - 1)) ? SleepCounter + 2 :
MaxValue;
              Serial.print("b2+");
       } else if (beta2 > (0.9 * BaselineBeta2Mean)) {
```

```
                SleepCounter = (SleepCounter > (MinValue + 1)) ? SleepCounter - 2 :
MinValue;
                Serial.print("b2-");
        }
        if ((alpha1 < (0.7 * BaselineAlpha1Mean))) {
                SleepCounter = (SleepCounter < (MaxValue)) ? SleepCounter + 1 : MaxValue;
                Serial.print("a1+");
        } else if (alpha1 > (0.8 * BaselineAlpha1Mean)) {
                SleepCounter = (SleepCounter > (MinValue)) ? SleepCounter - 1 : MinValue;
                Serial.print("a1-");
        }
        if ((alpha2 < (0.6 * BaselineAlpha2Mean))) {
                SleepCounter = (SleepCounter < (MaxValue)) ? SleepCounter + 1 : MaxValue;
                Serial.print("a2+");
        } else if (alpha2 > (0.7 * BaselineAlpha2Mean)) {
                SleepCounter = (SleepCounter > (MinValue)) ? SleepCounter - 1 : MinValue;
                Serial.print("a2-");
        }
        if ((rawpwr > (3 * BaselineRawMean))) {
                SleepCounter = (SleepCounter > (MinValue + 6)) ? SleepCounter - 7 :
MinValue;
                Serial.print("r-");
        }else if ((RawStats.Mean() < (0.6 * BaselineRawMean))) {
                SleepCounter = (SleepCounter < (MaxValue - 6)) ? SleepCounter + 7 :
MaxValue;
                Serial.print("r+");
        }

        if ((SleepCounter > Threshold)) {
                IsAsleep = true;
        } else {
                IsAsleep = false;
        }
}

void AnalyzeBrainWaves()
{

        if (attention == 0 & meditation == 0) {
                if (bufffill != 0)
                        bufffill = bufffill - 1;
        }

        if (!FirstRun & (attention > 0 | meditation > 0) & (rawpwr < RawLimit) ) {
                RawStats.Push(rawpwr);
                Alpha1Stats.Push(alpha1);
                Alpha2Stats.Push(alpha2);
                Beta1Stats.Push(beta1);
                Beta2Stats.Push(beta2);
        }

        if (bufffill == BufferSize - 1 & !FirstRun & !IsBaseline) {
                bufffill = 0;
                if (attention > 0 | meditation > 0) {
                        AsleepAlgorithm();
                }
                ClearAll();
                if ((IsAsleep)) {
```

```
                        //Sound the alarm
                        SoundTheAlarm();
                }
                else
                {
                        QuietTheAlarm();
                }
        }
        if (bufffill == BaselineSize - 1 & !FirstRun & IsBaseline) {
                IsBaseline = false;
                bufffill = 0;
                CreateBaseline();
                SerialPrintBaseline();
                ClearAll();
        }
        if (bufffill == FlushLimit - 1 & FirstRun) {
                FirstRun = false;
                bufffill = 0;
                ClearAll();
        }

        bufffill = bufffill + 1;
}

void ClearAll()
{
         Alpha1Stats.Clear();
         Alpha2Stats.Clear();
         Beta1Stats.Clear();
         Beta2Stats.Clear();
         RawStats.Clear();
}


void CreateBaseline()
{
  BaselineRawMean = RawStats.Mean();
  BaselineRawStdDev = RawStats.StandardDeviation();

  BaselineAlpha1Mean = Alpha1Stats.Mean();
  BaselineAlpha1StdDev = Alpha1Stats.StandardDeviation();

  BaselineAlpha2Mean = Alpha2Stats.Mean();
  BaselineAlpha2StdDev = Alpha2Stats.StandardDeviation();

  BaselineBeta1Mean = Beta1Stats.Mean();
  BaselineBeta1StdDev = Beta1Stats.StandardDeviation();

  BaselineBeta2Mean = Beta2Stats.Mean();
  BaselineBeta2StdDev = Beta2Stats.StandardDeviation();
}


void SoundTheAlarm()
{
    tone(8, 4000);
}
```

```
void QuietTheAlarm()
{
    noTone(8);
}
```