2014

ANALYZING THE EFFECTIVENESS OF THE ACTIVITY ANALYZER FOR GUIDED INDEPENDENT LIVING ENVIRONMENTS

Gabriel Ausfresser
University of Rhode Island, gausfresser@my.uri.edu

ANALYZING THE EFFECTIVENESS OF THE ACTIVITY ANALYZER FOR GUIDED INDEPENDENT LIVING ENVIRONMENTS

BY

GABRIEL AUSFRESSER

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE

REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

IN

ELECTRICAL ENGINEERING

UNIVERSITY OF RHODE ISLAND

2014

MASTER OF SCIENCE THESIS

OF

GABRIEL AUSFRESSER

APPROVED:

Thesis Committee:

Major Professor Ying Sun

John DiCecco

Patricia Burbank

Nasser H. Zawia
DEAN OF THE GRADUATE SCHOOL

UNIVERSITY OF RHODE ISLAND
2014

ABSTRACT

Exercise is the key to maintaining a healthy lifestyle. However, it is becoming more of a challenge for people to break from their busy lives to take the time to do some physical activity. This is a problem that is experienced by all different age groups, for various different reasons. For younger people, it is difficult to find motivation to take time out of the day for exercise. For the elderly population, it is difficult to find the energy to exercise. One way to facilitate physical activity for people is to increase motivation to do so. This study is designed to test a specific device that specializes in personalized motivational messages, called the Activity Analyzer for Guided Independent Living Environments, or AAGILE for short.

The AAGILE is an exercise monitoring device worn by a person throughout the day. At predetermined times, personalized, prerecorded messages play from the AAGILE to encourage exercise to the person wearing it. As the wearer exercises and creates motion, the AAGILE captures the motion data and applies a scoring scheme to the data. The scheme is on a scale from one to ten, with one being the lowest exercise score and ten being the highest exercise score. Then, at the end of the day, the user can export the data to a PC and inspect the exercise response to the messages throughout the day.

For this study, the AAGILE was worn by ten subjects between the ages of 18 and 30 who are in good health. The device is worn for at least six hours, and monitors the subject's activity during that period. Three messages are programmed to be played at preselected times unknown to the wearer. The three messages are personalized to the wearer. At the message play times, the personalized message is played which

encourages exercise. At the end of the six hour period, the device is returned and the exercise score data is captured on a PC using the AAGILE Windows® application graphical user interface. These ten sets of score data are analyzed and presented.

All ten subjects had a significant increase of exercise score data immediately after a personalized message was played. On average, scores increased by 3.5 points after a message was played. The increase in exercise demonstrates the overall effectiveness of the AAGILE, in terms of providing encouragement and motivation for the user to exercise, and thus, promote healthy lifestyles.

ACKNOWLEDGMENTS

Completing this thesis took motivation and encouragement from a number of people in my life. Without them, I would not have been able to complete this study.

First, I'd like to thank some of the excellent faculty at URI. Thank you to electrical engineering professors Dr. Swaszek, Dr. Boudreaux-Bartels, Dr. Fischer, and Dr. Alan Davis. Thank you to biomedical engineering professors Dr. Besio, Dr. Helen Huang, Dr. Vetter, and Dr. John DiCecco. Without the knowledge learned from these wonderful people, I would be unable to establish my career as an engineer. I would also like to thank Dr. Patricia Burbank from the Nursing Department at URI. I have been blessed to have met such a caring professor, who not only cares about her students, but also cares about the wellbeing of pretty much everyone. Thanks for working with me on this study.

I also would like to thank other people who assisted me with understanding the field of electrical engineering, including TAs Andrew O'Shea, Xiaorong Zhang, Andrew Burke, and of course, Fu Luo. I'd like to thank my classmates who helped me with homework and study for tests. Some include Shane Walton, Erica Duncanson, Andrew Dunne, Tonya Ruggieri, and many others. Thanks guys.

A special thanks to all of the students who have worked on the development of the AAGILE over the years. I'd especially like to thank Kyle Rafferty and Tim Alberg, who put all of the hard work into bringing up the AAGILE from paper to prototype. Without them, understanding the intricate design of the AAGILE would be near impossible, which would have prolonged my graduation even further. I am thankful that they are still great friends of mine, and I hope to remain close to them for as long as possible.

Completing this study has proven difficult for me ever since I started working as an electrical engineer full time. However, with the motivation of my thesis professor, Dr. Ying Sun, I have been able to finally complete it. Dr. Sun was the first professor I met at URI, and the first to tell me about the biomedical engineering program. Years later, I stuck with the biomedical engineering program and graduated with my bachelors in 2011. Thank you so much Dr. Sun for being there for me every step of the way throughout my education at URI. Thank you for calling me every week to keep me on my toes, and keep me moving forward with the study. I couldn't have done it without you.

I'd like to thank dad for always being there for me and always believing in me. I know it's been an interesting journey, and I know that the journey does not stop here.

Finally, I'd like to thank my mom. If it wasn't for my mom's enthusiasm for college searching, I don't know where I would have ended up. Thanks for bringing me on all of those college tours all around the country. Thanks for continuing to push me to finish this study, and please continue pushing me to be the best person I can be. And thanks for being a great mom and giving me so much love.

Thanks guys.

PREFACE

This thesis is divided into seven chapters. Chapter one is an introduction to the study. Chapters two through four go into detail on the design of the AAGILE system, starting with the electronics, then firmware, then software. Chapter five explains the study methodology. Chapter six shows the results of the study and analysis of the data. Finally, there is a conclusion at the end of the thesis.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

“You need to exercise more.” These five words are repeated continuously by health care physicians to thousands of Americans each day. Some reasons for the repetitive instruction is because people who are physically active generally live longer and have lower risk for heart disease, stroke, type 2 diabetes, depression, and some types of cancer (CDC 3). In fact, exercise has been linked to facilitate the prevention of up to 35 chronic conditions (Booth 1143). According to the Center for Disease Control’s *2008 Physical Activity Guidelines for Americans*, adults should participate in at least 2 hours and 30 minutes a week of moderate-intensity aerobic physical activity to obtain the necessary health benefits to reduce the risk of the diseases mentioned above (CDC 4). While the benefits of exercise have been studied and reported throughout the past number of years, still only about half of adults and less than a third of youth meet aerobic physical activity guidelines.

Other than chronic illnesses, regular exercise is a healthy course of action for overweight or obese patients, since obese individuals experience improved insulin sensitivity, lipid and lipoprotein profile, and blood pressure (Bouchard 1). Regular exercise has been shown to be one of the best predictors of successful weight maintenance (McInnis 111).

Exercising has psychological benefits as well. Exercising regularly has been linked to protecting against depression in Alzheimer’s disease (Regan 1). People who exercise regularly admit that they feel that they have a better quality of life, and exercising may be a useful intervention strategy for people who are less willing to change (Laforge). Studies show that aerobic exercise training is useful for reducing the severity and duration of depressive reactions following a stressful life change (Roth). In general, findings from research indicate that exercise is associated with improvements in mental health, including mood state and self-esteem (Raglin).

If exercise is so important, and the benefits are widely known, then why do Americans still struggle to meet the exercise requirements laid out in published guidelines by the CDC? The simple answer is that people are finding other ways to use their time. Some people, for example, choose to play videogames for countless hours instead of exercising. In fact, videogames are now present in 72% of all American homes ("Steinberg 1"). According to the Entertainment Software Rating Board, as of 2010, the average gamer spends 8 hours a week playing videogames ("Videogame statistics"). The increase of gaming time consumption is just one reason for lack of exercise. Another reason for lack of exercise is the time consumption of work duties (St. Marie). After a long day of work, many people have little energy left to do any exercise. Aside from symptoms of ill-health, long work hours have been associated with numerous poor lifestyle habits, with lack of exercise included (Sparks 391). The overall fatigue due to long work hours have been associated with lack of exercise, as well as other negative life-style mechanisms (Hulst 171).

According to Geoffrey St. Marie, one of the biggest reasons that people neglect to exercise enough is that they are not properly motivated, whether that person is a young person or an elderly person. In some cases, such as in older adults, the concern for their own health is often strong motivation to increase exercise (Newson 1). For others, motivation other than self-motivation is helpful. According to the Virginia Anderson and Brunilda Nazario, one effective way for a person to find the motivation to exercise is for that person to find friends, family, co-workers, and neighbors that can encourage him to take the time from the busy day and do some physical activity (Anderson and Nazario). According to the article, *Promoting Exercise and Behavior Change in Older Adults: Interventions and Transtheoretical Model,* "The capability for a friend or family member to record personalized caring messages to help motivate the person to exercise brings not only the motivational message and instruction, but uses the TTM and Motivational Interviewing interventions to increase participation in physical activity (Burbank and Riebe)." Unfortunately, oftentimes, people are too busy to even encourage someone else to exercise. This is the motivation that lead to the invention of the Activity Analyzer for Guided Independent Living Environments, or AAGILE.

The AAGILE is a device that monitors physical motion of a person throughout the day (Rafferty et al., Greene et al). Using an accelerometer, the device can measure the overall physical activity of the person wearing the device, which can be translated into measuring the amount of exercise the wearer does throughout the day. The AAGILE also has built in message recording and playback features, which can be used to provide the motivation that has been shown to facilitate the increase amount of exercise for individuals. For example, a child can record a motivational message to be played at different intervals throughout the day to encourage a parent or grandparent to exercise for ten minutes or so to assist with maintaining a healthy lifestyle.

Furthermore, physical activity has been shown to be important for fall prevention, health promotion and maintenance, and reversal of frailty among older adults (Burbank and Sun 2). For further motivation, the processor inside the AAGILE overlays the raw accelerometer data onto a scoring system from one to ten, which can be reviewed after the day is complete. Using a motivational message and a scoring system, the AAGILE device provides both peer-motivation and self-motivation for the encouragement to do some exercise activity (Wang et al).

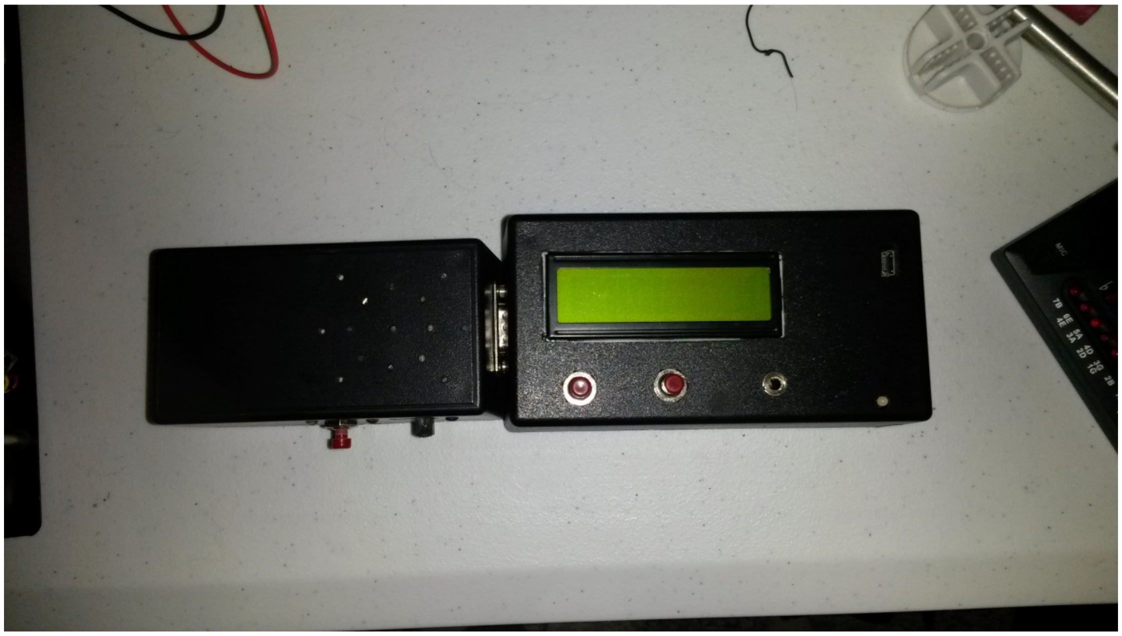


Figure 1. Picture of AAGILE System. Mobile unit is on left. Docking station is on right.

In this study, the effectiveness of AAGILE for the motivational capability to encourage exercise in individuals is determined by applying the AAGILE to ten different subjects for six hours. Throughout the six hour period, three messages are played at predetermined times to attempt to encourage the wearer to exercise for some time. The exercise of the subject is monitored with the AAGILE's onboard accelerometer, and the processor extracts a score from the raw data. After the six hour period, the scores are then transferred to a PC and plotted using Microsoft Excel. The goal of the study is to determine the effectiveness of the AAGILE in encouraging exercise of the wearer by attempting to find a correlation in the message playtimes and the subsequent exercise score following the messages.

If a strong correlation between message playtimes and exercise level exists, then it can be determined that the motivation for exercise of the AAGILE is effective enough to encourage exercise in individuals. Using the Excel data extrapolated from the AAGILE, the increase in exercise score can be used to quantify the effectiveness of the AAGILE. If the AAGILE can successfully motivate people to exercise, then it can be considered a practical device that can efficiently help prevent against many of the chronic conditions mentioned at the beginning of this chapter.

Before reporting the results of the study, an overview of the AAGILE device is discussed, which talks about the hardware and firmware design of the device and how the design meets the requirements. Then, the PC software is discussed and explained how it interfaces with the AAGILE.

CHAPTER 2

AAGILE ELECTRONICS

The AAGILE system has a handful of basic requirements to fulfill in order to successfully conduct the essential performance of the device. These basic hardware requirements are listed in the table below:

Requirement Number	**Requirement**	**Rationale**
1	The electronics SHALL have a 3-axis accelerometer.	Used to detect motion of the device in three directions.
2	The electronics SHALL have means to record and playback voice.	Used to record the voice of a loved one, and played back to the patient to encourage exercise.
3	The electronics SHALL support a User Interface consisting of a character LCD and various pushbuttons.	Allows the practitioner to program message playing times, as well as monitor activity levels of the wearer.
4	The electronics SHOULD support a PC interface	Allows for further data analysis of the activity

		scores of the wearer.
5	The electronics SHALL be packaged in a wearable box.	The device needs to be portable enough to be worn in a fanny-pack.
6	The electronics SHALL be powered with a 9V battery.	9V batteries are compact and have sufficient battery life for the portable system.

Table 1. Basic Hardware Requirements for the AAGILE

The electronics used to satisfy the requirements listed above is broken down into three main components. One component is the mobile unit, which comprises of the electronics responsible for monitoring the activity and playing back stored messages. The electronics in the mobile unit consists of the microcontroller, the voice record IC, the speaker, and the accelerometer. The mobile unit is also the component that is worn by the user.

The next component of the device is the docking station electronics. The electronics in the docking station are responsible for providing the user interface of the system. The UI consists of two pushbuttons, a character LCD, a microphone jack, and a mini USB jack. The docking station is to be used by the practitioner, who is responsible for recording messages, setting message play times, setting the clock time, and connecting to the PC application. The docking station connects to the mobile unit during the programming portion of the procedure. Once the message times are squared away, etc., the mobile unit is removed from the docking station, and the docking station is left behind.

The final component of the AAGILE system is the PC interface. Once the mobile unit is inserted into the docking station, a USB interface is provided by the docking station to the PC application. The PC application is responsible for receiving activity scoring data from the mobile unit, and then displays that data onto a PC via a graphical user interface. The PC application is a Windows® application, and is described in more detail in Chapter 4 of this thesis.

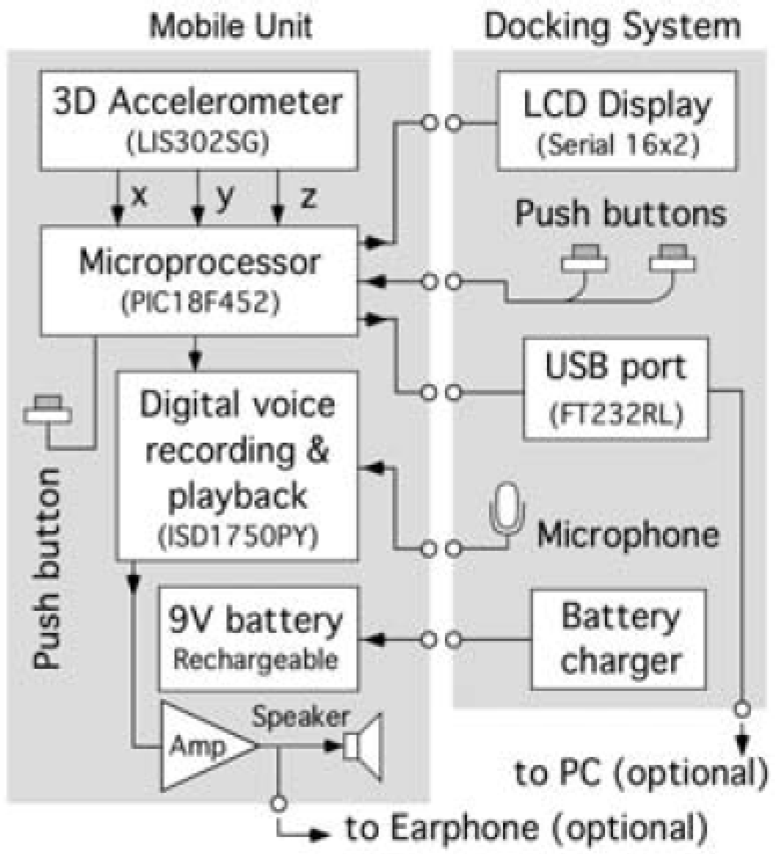


Figure 2. Block Diagram for AAGILE Hardware

The mobile unit of the AAGILE system houses the microcontroller of the device, which is analogous to the brains of the system. The processor is responsible for interfacing with all of the peripherals of the system, including the voice record chip, the accelerometer, as well as various components found in the docking station. The processor utilizes various IO to control and monitor components of the system, as well as contains non-volatile memory to store information even after a power cycle.

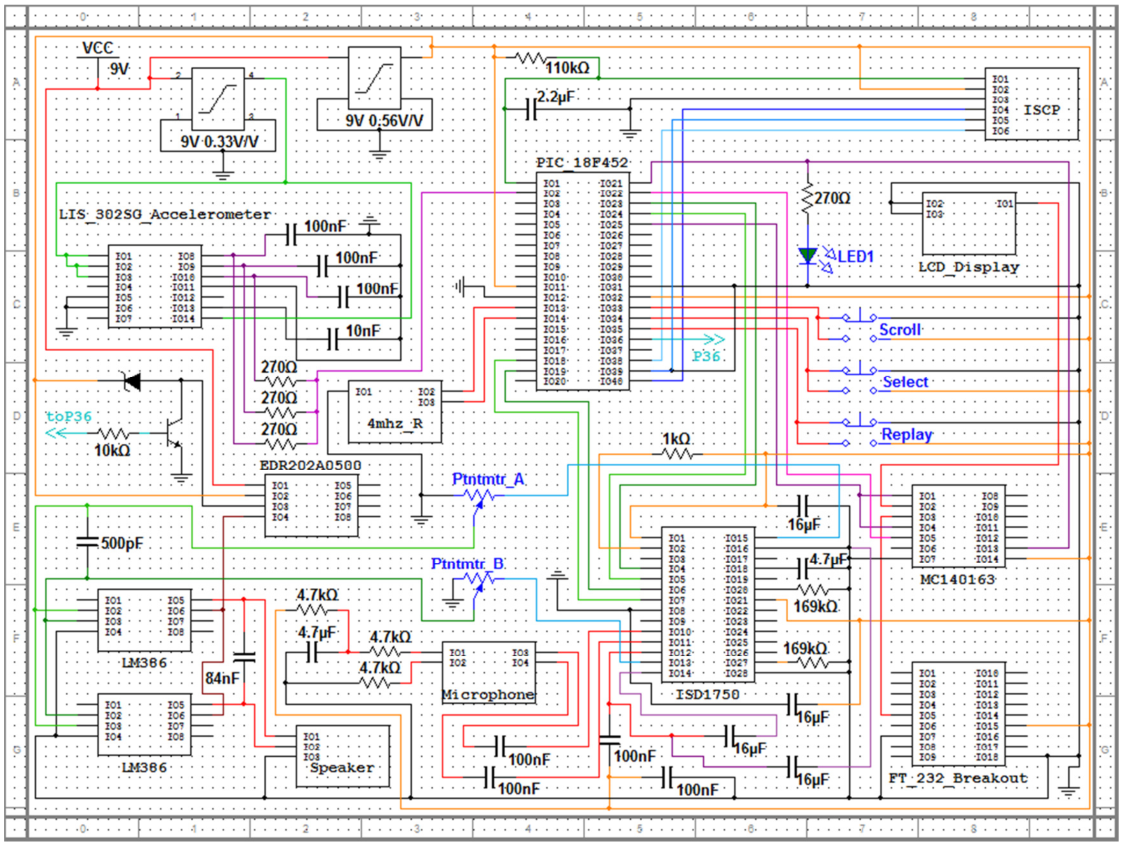


Figure 3. Schematic for the AAGILE hardware.

The microcontroller implemented in the AAGILE system is the Microchip PIC18F452 High-Performance, Enhanced Flash Microcontroller with 10-bit A/D. The PIC processor has 32 kilobytes of on-chip flash memory, 1536 bytes of on-chip RAM, and 256 bytes of nonvolatile EEPROM ("PIC18FXX2"). The processor has 40 pins and operates off of a 4 MHz crystal oscillator. The processor has various timer modules, which are used to keep time as well as provide a timing basis for other parts of the system. The processor also has an 8-channel 10-bit analog to digital converter, which is used to read in data from the accelerometer. The PIC controller has a SPI bus, used to interface with the voice record chip, and a UART module, used to communicate to the PC application and the LCD screen via RS232. Finally, the processor has numerous GPIO pins for controlling the speaker relay, the digital switch, detecting button presses, as well as other components of the system. More information on the processor control interface with the peripherals of the system is detailed in Chapter 3 of this thesis.

To monitor activity of the AAGILE wearer, the ST Microelectronics LIS302SG 3-axis accelerometer is used. The LIS302SD accelerometer is a 3-axis accelerometer that detects accelerations within a range of ± 2g ("LIS302SG"). The accelerometer has a built in electronics to provide three distinct analog outputs, one for each axis, that all have a ratiometric output with sensitivity of 0.145*Vdd V/g. Since the accelerometer is powered with 3.3V supply, the sensitivity of the sensor is 0.4785 V/g. The AAGILE's scoring system is not dependent on the acceleration of a specific axis, so the three analog outputs of the sensor are averaged together using three 270 Ω resistors. Each of the three signals passes through one 270Ω resistor, then wired in parallel and connected to a single A/D pin on the processor. By connecting to the processor in this fashion, a single A/D pin is used to measure averaged accelerometer data from 3 axes down to a signal analog voltage. This method conserves pins on the processor, reduces the number of connections required, as well as implements a simple averaging of the accelerometer data to sufficiently quantify the amount of activity of the wearer.

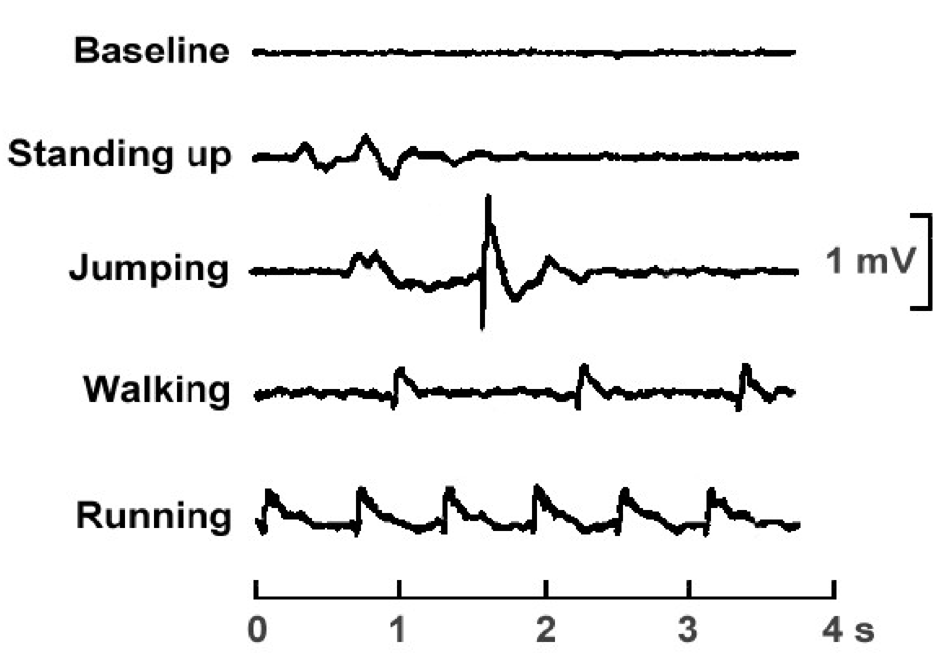


Figure 4. Resulting motion signal shown for various types of activities.

In order to record, store, and playback messages to the AAGILE wearer, the Nuvoton ISD1750 Multi-Message Voice Record and Playback IC is implemented. The voice record chip has built in nonvolatile flash memory for message storage, and interfaces with the processor via SPI bus (“ISD1700”). With the SPI interface, the user can store and play various messages stored in the chip’s memory, as well as other features. Using an external biasing 160 kΩ resistor, the selectable sampling rate is set to 4 kHz, allowing for the maximum 100 second message storage time. This allows for the AAGILE system to store multiple messages at varying message times, which can be used to encourage exercise using different messages during wear. The voice chip has built in Automatic Gain Control, which maximizes voice recording quality, as well as a smoothing filter and preamplifier, for providing clean voice output to the speaker.

The microphone’s output is connected to the voice record chip through 100 nF capacitors, which are used to block any DC signal through the microphone. The voice record chip provides the AGC to promote a good quality recording. The output of the voice record chip goes through two potentiometers to two different audio amplifiers. The audio amplifiers used are the Texas Instruments LM386 Low Voltage Audio Power Amplifier. The amplifier has an internal gain set to 20, but can be set to any gain between 20 and 200 using an external resistor-capacitor network ("LM386"). The amplifier inputs are ground referenced, while the output automatically biases to one half of the supply voltage. The potentiometers are used to set the input voltage to the non-inverting input of the amplifier. By altering the potentiometers, the voice output from the speakers can be set with different volume levels and different sound quality levels. The gain is set to 20 in the AAGILE, and the outputs of the amplifiers go to the input of the speaker. The audio amplifier gain stage allows for the flexibility to tweak the speaker output to select a voice signal that has a good sound quality and sufficient volume.

The speaker and supporting circuitry are powered with 9V. To avoid wasting battery life when not in use, a single pole, single throw relay is used to allow the processor to connect the 9V supply to the speaker. The Mobicon EDR202A0500 relay is selected, which is a relay with a coil voltage rating of 5V and a contact rating of 1A continuous ("EDR Series"). The current rating is sufficient for powering the speaker and amplifiers, and is simple to implement, only requiring an additional flyback diode to prevent damage to the relay and other circuitry.

The voice record circuitry allows for virtually any microphone, since the AAGILE docking station contains a simple 1/8 inch audio jack for the microphone input. Furthermore, the voice record circuitry allows for the use of a small size computer speaker, which has decent audio quality that comes in a small package.

While the docking station is responsible for housing the user interface components, such as the pushbuttons, LCD, and RS232 output, the docking station acts as a dummy terminal. In other words, the components in the docking station cannot work standalone, and requires direct control from the processor. There are two pushbuttons embedded into the docking station enclosure, where both are used for interacting with the user interface menu system. One pushbutton functions as the cycle options button, and the other button functions as the select button. Together, these two buttons can successfully navigate throughout the entire menu interface.

The main component that provides feedback to the user during menu cycling is the character LCD. The LCD used is from Sparkfun, with part number ADM1602U. The LCD uses 5V TTL signals with a Serial UART interface ("SerLCD v2.5"). The PIC processor has a UART module that can interface with the LCD without the need to write low level driver code. The PIC's UART library code can sufficiently communicate with the LCD with ease. Due to the serial communication feature of the LCD, only three wires are required to control the LCD, which is beneficial when building prototypes due to the lower amount of connections. The LCD provides a good backlight which adequately illuminates the display for excellent readability.

The component that is used to communicate with the PC application via RS232 is the FTDI FT232R USB-UART breakout board. The IC is responsible for converting the RS232 signals to USB signals without the need to implement a USB protocol ("FT232R"). The IC is directly connected to the processor's UART module. The processor simply writes characters through the UART module, and the FTDI chip outputs a signal compatible with mini USB protocol. This allows for a user to read data from the processor's EEPROM using the PC application. The user simply connects a USB cable between the FTDI board found in the docking station to a PC, and runs the application. The USB support is especially convenient for most PC users, since the RS232 DB9 port is being phased out of most laptops and modern PCs, but the USB input is still prevalent in most new computers and tablets.

The final component that is directly related to the user interface is the digital switch, MC14016B, from On Semiconductor. The digital switch is used to control one signal line with another ("MC14016B"). In the AAGILE application, the switch is controlled by the processor, and is used to determine whether the LCD or the FTDI chip is communicating with the processor. Since the PIC18F452 processor only has one UART module, but the LCD and FTDI both communicate via UART, the UART bus is shared between the two. The processor outputs a GPIO signal to the control of the digital switch, which determines whether the UART bus is connected to the LCD or to the FTDI chip. Thus, the processor can communicate to two UART devices with only one UART module.

The last components that make up the AAGILE electronics are the components related to power management. The AAGILE unit is powered with a rechargeable 9V battery. The 9V is regulated down to 5V for the logic level circuitry related to the processor, and is also regulated down to 3.3V for the accelerometer. The 5V regulator is the ST Micro L78L05 linear dropout regulator. The regulator is a basic three terminal LDO with output current limit of 100 mA ("L78L"). The regulator is also thermal protected, where if the part heats up past 150°C, the regulator will automatically disable the output. The IC also has short-circuit protection, where the regulator will also automatically shut down if the output voltage is shorted to ground. The 3.3V regulator is also an LDO three pin regulator with part number MCP1702-3302E, from Microchip. The regulator can output up to 250mA of current with an input voltage up to 13.2V ("MCP1702"). The regulator has low quiescent current of only 2µA, which makes it ideal for the battery operated AAGILE system. The current output is more than enough to power the accelerometer and supporting circuitry. The regulator also has short circuit and thermal protection, and requires only an input and output decoupling capacitor.

While the design was successfully implemented in the AAGILE prototype system, there are plenty of areas where the electronics can be improved. First, a full system power budget should be created to ensure that the 100 mA output of the 5V regulator is sufficient enough to power the system when the system is in full operation. If the regulator is being overdrawn, the regulator could start to generate heat, or it could cause a brown-out situation where the system is running with not enough current or voltage. Next, the audio output can be improved significantly. During the testing for the study, just about all ten subjects indicated that the AAGILE messages were difficult to understand. To improve the audio output, better audio amplifiers can be used. Furthermore, the voice record chip can be replaced with a high definition voice codec chip that has much better signal-to-noise ratio. There are also audio processor ICs that can read and play MP3 audio files, which would also output a message that is clearer to hear. The next area that can be improved is the removal of the audio circuitry 9V relay. Relays that only switch low contact voltages tend to waste more power to energize the coil than if another switching method was implemented. Instead of using a processor controlled relay to switch 9V, a simple MOSFET can be used, which can have a very low ON resistance. A lower ON resistance will allow for the same switching capability, but without much of the wasted power loss.

A final improvement to the electronics assembly would be to use a printed circuit board. The current prototype unit uses through hole components and is placed on proto-board. The connections are made with point-to-point soldering, which takes a lot of time to complete, and is also extremely difficult to debug. For example, if there was a cold solder joint on a single connection, it could take days to find where the solder problem exists. The cost of PCBs has gone down over the years due to improved technology. Now, there are PCB houses that provide instant quotes as well as very quick turn times, and have low cost solutions for 2 and 4 layer boards. Using surface mount parts on a custom PCB for the AAGILE would greatly reduce the probability of human error for board assembly, and also decreases the amount of time and effort that it takes to debug the circuity. Last, switching to a PCB with surface mounted components could significantly reduce the board size. Since the board is meant to be portable and worn by the user, a smaller PCB would be ideal.

Despite the areas of improvement that exist, the AAGILE prototype has successfully fulfilled the basic requirements, and has supported the ten-subject study without too many challenges along the way. And, as with most projects, the AAGILE system is in the prototype stage and will evolve with improvements from revision to revision.

CHAPTER 3

AAGILE FIRMWARE

The AAGILE firmware consists of the C code that is programmed onto the PIC18F452 processor. The firmware is designed to interface the processor control and monitoring with the AAGILE electronics to fulfill the basic system requirements. The code is compiled using the B Knudsen CC8E compiler, which is a compiler that supports Microchip PIC18 devices. The firmware is developed with the MPLab integrated development environment (IDE). Lastly, the firmware HEX file is loaded onto the processor using the Microchip ICD3 in-circuit debugger, which is a device that connects the processor to the PC via a USB port.

The AAGILE firmware architecture consists of three basic parts. Initially, a summary of each part is described. Then, the parts are followed by a more detailed description. First, there is the initialization code that takes place immediately after the PIC comes out of RESET. The initialization code is responsible for declaring and initializing the variables, setting up the PIC's peripheral modules, configuring various processor registers, display the splash text screen on the LCD, prepare the EEPROM memory locations for data storage, and set the initial clock time. The next part of the architecture is the main loop state machine. The state machine consists of all of the top-level code that is navigated via the docking station user interface or via the scoring sequence event posted by the lower level timing code. The state machine provides the code to allow the user to scroll through the various menus and select various settings of the system. Finally, the last part of the architecture is the lower level code contained in the interrupt routine. The interrupt routine runs in the background of the state machine, and is responsible for updating variables to be read by the higher level code. The interrupt code responds to various inputs either from the user or the processor registers, and then posts events which cause for a change of state in the state machine. A detailed description of each portion of code is documented below.

The first part of the initialization code configures the processor registers to function as needed to run the program. The registers include the Timer0 Control register, the Interrupt Control register, the Tristate registers for Port B and Port D, External interrupt trigger register, and ADC register and channel select. The T0CON register is initialized to 0x88, which enables timer 0, configures timer 0 as a 16 bit timer, sets the timer source clock to the internal instruction cycle clock, and disables any prescaler to timer 0.

The interrupt control register is initialized with a 0xA0. This configures the INTCON register to enable global interrupts, enables timer 0 overflow interrupt, and disables all other interrupt sources. The code sets the TRISTATE of port B to 0x07, which sets all pins of port B to output except bits 2, 1, and 0. TRISTATE for port D is set to 0x00, which sets all pins of Port D to output. Finally, interrupts for Timer 0, 1 and 2 are all enabled, since the timers drive the lower level code under the state machine.

The last initialization to occur is the configuration of the ADC. The ADC is set via a function called Setup_adc(char channel). This function is called on initialization with setting the ADC channel to 0. The function sets the TRISTATE of port A to all inputs, enables the A/D converter module in the processor, and sets the channel to read in to channel 0. Referencing the schematic shows that channel 0 is connected to port A bit 0, which is wired to the output of the accelerometer. In fact, the only ADC channel used in the AAGILE system is channel 0, since the accelerometer data is the only analog signal read in by the processor. For this reason, Setup_adc() is only called once in the initialization.

At this point in the initialization code, all of the necessary registers are initialized to allow the state machine to start in the first state properly. It also ensures that the lower level code is ready to execute properly. More details on the purpose of the initialization of registers are mentioned in the lower level driver code section.

After initializing the PIC's registers, many variables are declared with various data types. The function and description of the variables will be covered in the following sections of this chapter. The final part of the initialization code specifies the location of the EEPROM where the scores/message played data will continue from. This way, the user can power off the unit and be able to power on again and have the scoring data be stored in the next memory location that it would have stored to if the unit did not power off. This continuity will allow for the user to operate the device across power cycles without losing or overwriting previously stored data. The specify EEPROM algorithm works by first reading the last byte of memory in the EEPROM. This location holds the last memory location that had a score written to. Now that the PIC knows where the last written location was, it checks the first byte and second to last byte of the EEPROM. With these three data points, the PIC then does a check on these three locations. If the data present in all three of these locations is 0xFF, then the processor determines that the EEPROM has no data stored into it. If this is true, the processor resets the EEPROM starting location to the first byte. If not, then the processor sets the EEPROM starting location to continue storing data where it last left off on the previous power down.

After initialization, the main loop state machine starts to run. The state machine code lies inside an infinite loop, where the actions to occur in the loop are determined by the current state that the system is in. The first state of the state machine is the scoring state. In the scoring state, the analog input from the accelerometer is measured for sixteen data samples and stored in a sixteen character array called "array", and then is averaged over the samples. After the average is computed and stored in a 32-bit unsigned long variable called "average," it is compared to the threshold. If the average is greater than the threshold plus three, the motion counter is incremented by one.

If the code is in the scoring state, and the AAGILE time clock is a multiple of five minutes, the motion counter is used to calculate the score for that five minute period. Five minute periods were chosen due to the lack of memory. The PIC has 256 bytes of EEPROM, so to maximize the data collection time before the memory is full, a five minute scoring resolution is used. If the clock is of a five minute interval, the motion counter is compared to a basic scoring system where the count is mapped to a score between one and ten. The sensitivity mode of the AAGILE determines the motion count criteria for each score. The scoring table is as such:

Sensitivity Setting	**Motion Count Range**	**Score**
<u>High</u>	0 to 30	1
	31 to 60	2
	61 to 90	3
	91 to 120	4
	121 to 150	5
	151 to 180	6
	181 to 210	7
	211 to 240	8
	241 to 270	9
	Greater than 270	10
<u>Medium</u>	0 to 45	1
	46 to 90	2
	91 to135	3
	136 to 180	4
	181 to 225	5
	226 to 270	6
	271 to 315	7
	315 to 360	8
	361 to 405	9
	Greater than 405	10
<u>Low</u>	0 to 60	1
	61 to 120	2
	121 to 180	3
	181 to 240	4
	241 to 300	5
	301 to 360	6
	361 to 420	7
	421 to 480	8
	481 to 540	9
	Greater than 540	10

Table 2. Scoring System Based on Sensitivity

After the score is calculated, the timestamp and score are stored in the EEPROM at the specified location. The last step of the scoring state is the processor resetting the motion counter to zero.

The next state of the state machine is the clock sequence state. This state is the first state that is displayed on the LCD. The clock sequence state writes the AAGILE clock time to the LCD, with hour, minute, second, and AM or PM designation. As each second goes by in this state, the clock changes accordingly and the LCD is updated with the new time. The state machine will not leave this state unless it is time to get an exercise score or if the operator presses either the scroll button or the select button. If either of these buttons is pressed, the state machine progresses to the main menu sequence state.

The main menu state is responsible for providing the user interface to the operator, with the LCD being the feedback and the pushbuttons acting as the input. As the user presses the scroll button, the LCD updates the text to indicate which menu item can be selected. There are seven menu options that are selectable. They include setting the AAGILE clock, editing the message play times, editing the messages themselves, and uploading the AAGILE score data via USB to the PC application, clearing the EEPROM memory, changing the exercise sensitivity scale, and exiting the main menu and returning to the clock state. Once the LCD displays the menu item that the operator wishes to select, the user then presses the select button and the state machine enters the corresponding menu item state.

In the set clock state, the user can use the scroll button and select button to set the AAGILE clock time. First, the user selects whether the clock is operating in AM or PM time. Once the operator uses the scroll button to the desired AM/PM setting, the select button is pressed to move onto setting the hours. This is done in a similar fashion, where the operator uses the scroll button and select button to finalize the hour of the clock. The code repeats for minutes and seconds. The code is responsible for handling the LCD update, ensuring that the seconds and minutes count wraps back to “00” after “59”, and handling the two pushbuttons.

In the set play time state, the user can set different parameters relating to the playing of the messages. First, the user can edit the play times. This corresponds to what time on the AAGILE clock will cause the selected message to play. If this submenu is selected, the user must then scroll through which of the messages to edit the play time for. The user can choose to set the play times for messages one through nine. Setting the play times for each message calls the same Set_time() function that is used by the set clock state. After all of the message play times are selected, the message itself has to be selected to play. This is done by navigating through the submenu for the edit play time state. After the option to set the play time is the option to select the message that was edited. The user simply presses the select button on this option and the corresponding message is enabled to be played when the AAGILE clock matches the play time set previously by the user.

The next state in the menu sequence is the edit message state. In the edit messages state, the operator can cycle through the messages and choose a message to rerecord. The operator can also playback a message, or delete a message altogether. The operator uses the scroll button to select a message. After a message is chosen, one of the three actions is chosen for the selected message. Finally, the operator presses the select button to make the desired action choice. The firmware code handles the various message slots, including the recording times available for each message.

The next state in the state machine is the check for play time state. This state simply compares the AAGILE clock time with the preselected message play time. In order to play a message, the processor code checks for a flag indicating which message has been selected to be played. If a message has been selected and the time for that message matches the AAGILE clock exactly, then the firmware will utilize its lower level drivers to turn on the voice record chip and speaker and play the message.

The last state of the state machine is the sensitivity setting state. In this state, the operator uses the scroll button to switch between which sensitivity to set the AAGILE to. Once the correct sensitivity is scrolled to, the operator presses the select button to switch the sensitivity of the AAGILE. The sensitivity of the AAGILE determines the scoring system as mentioned earlier in this chapter.

The final pieces of the code are the lower level driver code and interrupt code. These final pieces are responsible for linking the higher level state machine code with the respective drivers needed to interface with the AAGILE hardware. The interrupt code has a higher priority than the main space state machine code. This ensures that the time sensitive code is executed on time, despite where the code is in the state machine. In the interrupt code for the AAGILE firmware lays the timer handles and the button handles. When an interrupt occurs, an interrupt flag is set by the processor. In the interrupt service routine (ISR), the code will check the interrupt flag and determine which interrupt service to execute.

One of the interrupt services is initiated by timer0. Timer0 interrupt is a 16-bit counter that is set using the high byte and low byte of the TMR0 register. The timer is set to 0xFC36, which causes the timer to interrupt every one millisecond due to the 4

MHz clock applied to the PIC. This one millisecond timer is the only interrupt driven timer in the system. Thus, the time base of all of the code in the AAGILE is one millisecond. After a millisecond elapses, the ISR first reads the accelerometer data via ADC. Next, the one millisecond software timer is incremented by one millisecond. Then, the button debouncing is handled. Last, the AAGILE clock is incremented with a one millisecond resolution.

The next interrupt handled in the ISR is the scroll button. When the scroll button is pressed, the main loop code will be interrupted by the ISR, which will start the button debouncing process. If the button has been successfully pressed with a debouncer, the code will set a flag indicating that the scroll button has been pressed. Another interrupt handled in the ISR is the select button, which works identically as the scroll button does, but with a different interrupt pin and interrupt flag.

Underneath the three ISR code functions lays the driver code required to interface with the AAGILE hardware. Many of these drivers involve setting up the PIC's peripheral registers and writing or reading data to and from peripheral devices. The driver functions are outlined in the table below:

Driver Function Name	**Peripheral Used**	**Input Parameters**	**Output Parameters**	**Description**
Setup_adc()	ADC	ADC channel to be used	Void	Sets the ADC control registers in the PIC to select which ADC channel will be read from.

Read_adc()	ADC	Void	Char adc_value	Sets the ADC conversion GO bit and waits for the ADC to finish the conversion. The function returns the ADC value.
Setup_SPI()	SPI	Void	Void	Sets up the SPI register to the correct timing parameters to interface with the voice record chip.
SPI_read_byte()	SPI	Void	Char value	Reads a byte from the SPI bus from the voice record chip, and returns the character.
SPI_write_byte()	SPI	Char value	Void	Writes the inputted character to the SPI bus to the voice record chip.
SPI_write_2bytes()	SPI	Unsigned long value	Void	Writes a 16 bit value on the SPI bus to the

				voice record chip.
Setup_USART()	UART	Void	Void	Sets up the control registers for the UART protocol which interfaces with the LCD and the USB FTDI chip. This sets up the baud rate and other configuration parameters.
Transmit()	UART	Char value	Void	Sends a character across the UART bus to either the LCD or the FTDI USB chip.
Clear_screen()	LCD	Void	Void	Sends a 0xFE to the LCD driver via UART, which corresponds to the first LCD location. Then, a 0x01 is sent, which is the command for

				clearing the LCD screen.
Backlight()	LCD	Char state	Void	Sends a 0x7C to the LCD driver, and then sends another character which represents the desired state of the backlight (On/Off).
Set_position()	LCD	Char position	Void	Sends a command to the LCD driver to put the character cursor to the specified location.
Print_line()	LCD	String pointer, char num_chars	Void	Prints the specified string of characters to the LCD, up to the number of characters inputted into the function.
readEEPROM()	EEPROM	Void	Char EEDATA	Reads the EEPROM character at the previously specified

				location and returns the value.
writeEEPROM()	EEPROM	Char value	Void	Writes the input value to the previously specified location.
INCEEPROM()	EEPROM	Void	Char EEADR	Increments one EEPROM location and returns the selected address.
SpecifyEEPROM()	EEPROM	Char address	Char value	Sets the memory location in EEPROM to be accessed.
Print_2dig_num()	LCD	Char value, char position	Void	Prints a two digit number to the LCD screen at the specified location.
Delay_ms()	Software Timer	Char delay time	Void	Uses the interrupt driven 1 ms timer to delay the AAGILE system for the specified amount of milliseconds.

Delay_sec()	Software Timer	Char delay time	Void	Uses the millisecond timer interrupt to delay the AAGILE system by the specified number of seconds.

Table 3. Firmware Driver Functions

The PIC code is written to implement the AAGILE functions in a timely and sufficient manner. However, there are some areas of improvement for the AAGILE code, including facilitating making changes to the code, reading and understanding the code, and optimizing the code for code space preservation. One example that does this is modularizing the code. All of the AAGILE code is compiled into one source file, resulting in over two thousand lines of code. One way to mitigate this complication is to modularize the code by giving the main state machine loop its own source file and the driver code its own source file. Going even further, some handler functions can be divided into source files with other functions that are related. For example, the LCD code can be compiled into one source file, the voice record functions can be compiled into a single file, etc.

Another way to improve the code is to make the code more like an event driven state machine, as opposed to using many flags. The AAGILE code has many individual flags that have similar names, which can be confusing to someone trying to understand the code. Making an event driven state machine means that while the system is in a state, only events related to that state will be acted on, and all others will be ignored. For example, when the select button is pressed, the code posts a “button press” event. Then, the event is put into a queue. Finally, when in a particular state, the state machine will look at the posted event in the queue and check its event table. If the event is in the event table, it will act on it accordingly as instructed in the event table.

While the firmware code may not be perfect, it still is able to execute the AAGILE functionality with an interrupt driven program. Using a one millisecond time base allows for all of the controls and functions to be handled in an organized manner with one millisecond timing resolution. By interfacing the processor with the hardware, the AAGILE system can successfully monitor the user’s exercise activity and play a message when programmed to do so. The code successfully provides a user interface to the operator to setup various play times and record new messages.

CHAPTER 4

AAGILE SOFTWARE

The AAGILE software consists of the application code written for a Windows® PC. The application allows the AAGILE score data to be imported to a PC and displayed in an embedded graph. Furthermore, the data can be directly exported into an Excel file, or the graph can be saved as an image file. The software code is developed as a Windows® Form Application in the Microsoft Visual C# 2010 Express development environment. The development environment allows the application to be run in debug mode, where the programmer can tweak parts of code and step through the lines of code as desired. The application can also be made into a Windows® executable, where the user simply needs to open the program as any other Windows® application would be opened. The application then runs standalone.

The application is built using the Windows® form tool, which allows for quick development of a graphical user interface. The application uses various forms to provide the interface to the user. These forms include a rich text box, a data grid, a chart, a label, three buttons, a menu strip, a save file dialog, and a numeric up/down box. These interface tools are brought into the Windows® form application, and code is implemented to determine what functions are executed with each form.

The rich text box form is used to display application information to the user, including the raw data from the AAGILE and the portioned time and score data. Also, the COM port information is displayed. The data grid is used to compile the time and score information in a table. The table is then matched with the chart form. The chart form binds to the data grid and plots the data on a line graph. The label form is used to display the AAGILE title on the application graphical user interface, or GUI. The three buttons are used to allow the user to control the application, including the ability to connect to the AAGILE via the COM port, stop the COM port communications, and import the data from the AAGILE. The menu strip form allows the menu bar to appear in the application. The menu bar for the AAGILE application includes a file menu and a help selection. The help selection provides a pop up window to appear which describes how to use the application. The File menu has two choices, including saving the graph as an image, or export the AAGILE data to Excel. The save file dialog allows the user to save the data with the standard Windows® navigation window. Finally, the numeric up/down box is used to allow the user to select the COM port for the AAGILE communication.

The AAGILE application is not driven by any timers or by any state machine. The application simply waits for user input via the GUI, and responds to the input with the coded action. The only code that executes without user input is the initialization code. The initialization code first draws the GUI form, then creates a serial COM port to interface with the AAGILE, then sets up the data table, and declares various global variables.

After initialization, the GUI application waits for user input. When the user clicks the Ready To Connect button, the application code first displays the COM port information in the rich text box form. Next, the application tries to open the COM port at the preset baud rate of 9800 and with COM port number equal to the number displayed in the numeric up/down box. If the application is able to connect to the AAGILE, then the rich text box displays that the application is connected. If not, then the application displays in the rich text box that the connection failed, and displays an error message provided by Windows® that explains why the connection failed. If connection with the AAGILE is successful, the application then waits for user input to continue.

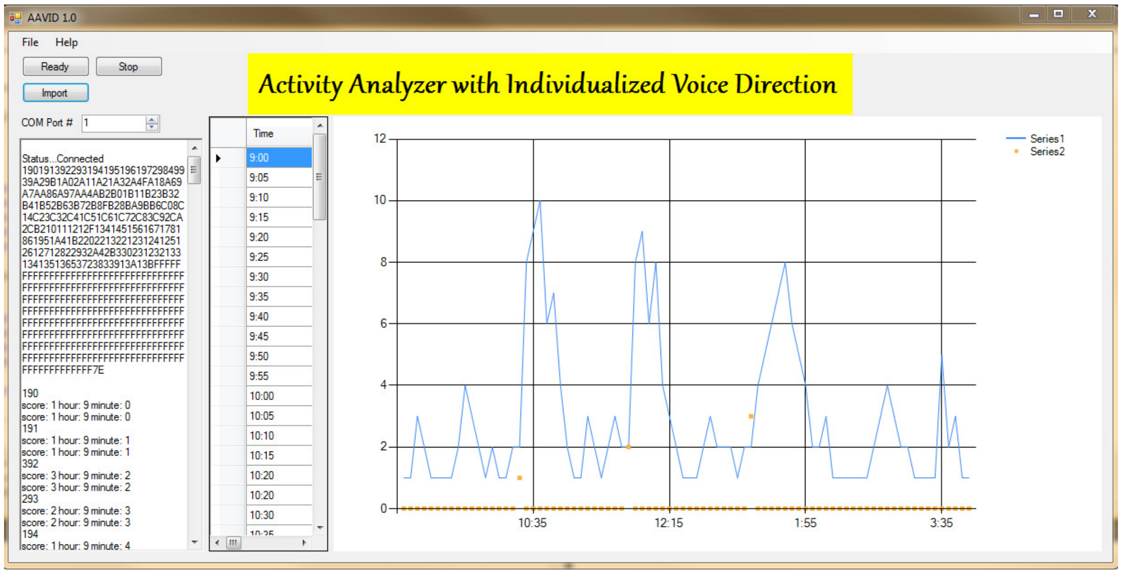
Next, the user clicks the Import button. When this happens the application tries to read 256 bytes of data from the COM port. The operator must first navigate the AAGILE menu on the docking station and select data USB upload. If this has been done, then when the user clicks the Import button, the 256 bytes of data from the AAGILE's EEPROM will be sent across the COM port. If the data read is successful, the application code then proceeds to parse the data. First, the application instantiates an Excel file to be used to store the data. Next, the data table and chart tools are updated with the proper fields including the title fields. Then, the 256 bytes are converted from binary data to a string representation of the hex values. The hex values are then displayed in the rich text box. Next, the string of hex data is broken up into substrings that represent three nibbles of the data. The first nibble represents the score, the second represents the hour, and the last nibble represents the minute. The score/hour/minute data is displayed in the rich text box.

After the data is broken up into the score, hour, and minute partitions, the data is converted back from a sting to an integer. If the minute data represents zero minutes, then the minutes value is converted to a string of double zeros, as traditional time is displayed. If the minutes value is equal to five minutes, then the data is converted to a string of “05”, also as traditional time is displayed. Otherwise, the data is ready to be stored into the data table as integers.

If a message is played, the first nibble of the threesome is F. The code handles this case, and stores the message played in the chart instead of the score. If the data nibbles are FFF, then that means that the data stream has ended, and the AAGILE EEPROM has cleared data at that point. If the application sees FFF, then it completes the parsing of data, completes the data table form, adds heading to the chart, and binds the chart to the data table. The binding of the chart causes the GUI to display the data on the chart. If the importing of AAGILE data fails, then the application displays that the importing failed, and also displays the Windows® provided error message for the cause of failure. Last, the application closes the serial port.

If the user clicks the help menu item, the application displays a pop up window that explains how the AAGILE application is used. If the user navigates the file menu bar, and selects save graph, the application opens the Save As dialog window to assist the user to choose a location and filename to save the graph to. If the user navigates the file menu bar and selects to export data to Excel, the application opens a Save As dialog window to assist the user to choose a filename and location of the generated Excel file. The user can then open the Excel file at his or her leisure at a later time and utilize the data as desired.

If the user clicks the Stop button, then the application will close the serial port. Similarly, if the user closes the application, the code will also close the serial port to prevent it from being unintentionally left open upon exit.



```
Status...Connected
190191392293194195196197298499
39A29B1A02A11A21A32A4FA18A69
A7AA86A97AA4AB2B01B11B23B32
B41B52B63B72B8FB28BA9BB6C08C
14C23C32C41C51C61C72C83C92CA
2CB210111212F134145156167178 1
861951A41B220221322123124125 1
261271282293 2A42B330231232133
134135136537238339 13A13BFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFF7E

190
score: 1 hour: 9 minute: 0
score: 1 hour: 9 minute: 0
191
score: 1 hour: 9 minute: 1
score: 1 hour: 9 minute: 1
392
score: 3 hour: 9 minute: 2
score: 3 hour: 9 minute: 2
293
score: 2 hour: 9 minute: 3
score: 2 hour: 9 minute: 3
194
score: 1 hour: 9 minute: 4
```

Figure 5. AAGILE PC Application Screenshot.

The application is fully functional, but has areas of possible improvement. For example, the application requires that the operator clicks the buttons in a required sequence that is not intuitive in order to properly read the data from the AAGILE. First, the user needs to click the connect button. Then, the user select the USB data upload option on the AAGILE's docking station. Next, the user clicks the Import button in the application. Last, the user must close the serial port with either the stop button or exiting the application properly. If the operator does not execute this sequence properly, the AAGILE data may not import properly, parse the data properly, or close the serial port properly, which could cause the computer to work inefficiently. One way to fix this would be to integrate all of the functions into the Connect button so that once the user connects to the AAGILE, the rest of the data handling is done internally without user intervention. For example, the user would click the Connect button once, then press the USB data upload selection on the docking station. Then, the application would immediately and automatically import the data from the data stream and parse the data, then close the serial port. The application would act more like a dumb terminal in that once the COM port is open, the application will sit and wait for data on its own.

Another area of improvement is the correction of the lack of persistency of the application. If any of the bytes that come across the COM port are not in proper sequence or the correct data format, then the application tries to plot the bad data anyway, or simply gives up and throws an error message. The application should be made more persistent with parsing the data, in that the code should realize that the data is bad, and prevent the displaying of bad data. Furthermore, the application should try to skip over the bad data and continue parsing on the next good data byte. This way, if the AAGILE transmission gets corrupted for any reason, the application is robust enough to handle the situation and still retain some good AAGILE score data without losing the day’s work.

Despite some inefficiency, the AAGILE Windows® application GUI successfully provides an easy way for the operator to review the AAGILE data with just a few clicks of some buttons. This relieves the operator from manually reading in the EEPROM data and parsing the data, as well as plotting the data in a graph. In fact, this application has been used for the AAGILE study reported in this thesis, which was a crucial timesaver and an easy and quick way to obtain, display, and analyze the exercise data.

CHAPTER 5

STUDY METHODOLOGY

To demonstrate the effectiveness of the AAGILE, a study is conducted on ten human subjects. The study is approved by the Institutional Review Board, with an IRB approval number of HU1112-025. The ten subjects are within the age range of eighteen to thirty years old. Each of the ten subjects is assigned to wear the AAGILE mobile unit for at least a six hour period. During the six hours, three prerecorded personalized messages are played at intervals unknown by the wearer. The personalized messages encourage the wearer to exercise for some time. Although the exact play times are not expressed to the subject, the messages times are selected to work around the subject's schedule to ensure not to disrupt the subject from any important meeting, class, etc.

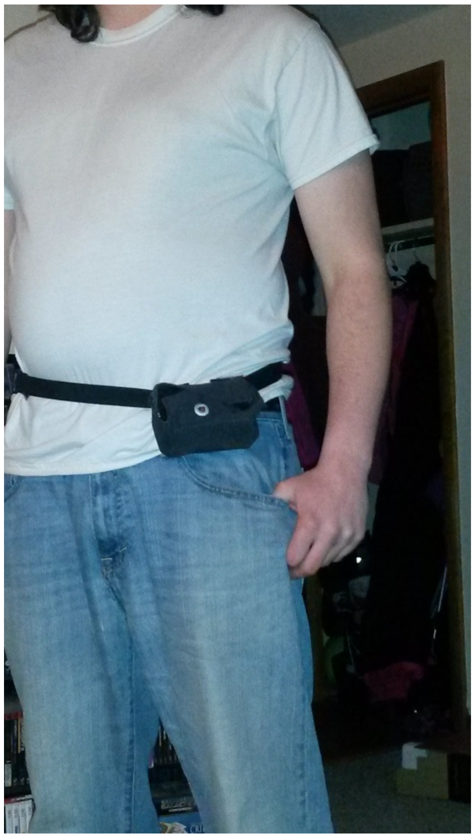


Figure 6. Wearable AAGILE mobile unit.

To setup the study, the experimenter first powers up the AAGILE and clears the EEPROM memory. Next, the AAGILE mobile unit is connected to the docking station, and the person running the study will use the user interface to set the time, record messages, and set the message play times. Once the AAGILE mobile unit is programmed, it is put into a fanny pack and handed to the subject. The subject then puts on the AAGILE mobile unit and then goes about the day as usual.

The hypothesis is that the personalized messages of encouragement will result in an immediate increase of physical activity to be monitored by the AAGILE's accelerometer. The scoring system to quantify the amount of exercise should then see an increase in score as well. This hypothesis is tested three times by three different messages during the six hour period. Another method is also used on the subject to determine the effectiveness of the AAGILE. Although more subjective than a discrete exercise score, a survey is given to the subject after the six hours expires to be filled out honestly and completely. The survey aims to determine the overall experience of the subject during the study, the aesthetics of the AAILGE for the user, and the ability to motivate the subject to exercise. The survey is pasted in the Appendix of this thesis.

When the subject completes the six hour study, the device is turned off and handed back to the one running the study. Then, the AAGILE is rebooted while being connected to the docking station. The AAGILE is then connected to a PC via the USB port, and the data is exported to the AAGILE Windows® application. A copy of the raw EEPROM data is stored in case it is needed later on. A copy of the AAGILE application graph is saved as a picture file. Finally, the data is exported to an Excel file to be processed later on if desired. All three sets of data are used to create the completed graph of the exercise score of the subject from the six hour test, and the results are reported in the Study Results section.

Using the data extracted from the AAGILE, some statistical analysis is conducted to quantify the effectiveness of the AAGILE in encouraging exercise of the subject. First, a baseline for each subject is calculated by averaging the score on the times when no message has played. Then, the difference in score value from the baseline to the score after a message is played is determined, and then this difference is averaged for the three instances the messages were played. The difference averages are then compared between the ten subjects and the overall effectiveness of the AAGILE is determined.

As a final set of data points, the survey responses are evaluated for each subject. The responses in the survey are used to report the subjective feelings of the subjects on how effective they thought the AAGILE was at encouraging them to exercise. Furthermore, any recommendations for improvement of the AAGILE device dictated by the subjects are recorded.

The results of the study are reported in the next chapter.

CHAPTER 6

STUDY RESULTS

The results of the study are analyzed by first calculating the baseline for each subject. This is done by computing the average score during the period of time where no messages were played. The baseline score is enumerated with the variable, *SBL*. Next, the average is computed for scores in the period immediately following the message played, and is enumerated with the variable, *SMP*. The SMP value is averaged from the first non-1 score after a message played to when three 1's are recorded in a row. In other words, it is assumed that if no exercise has occurred after fifteen minutes elapses after a message is played, then the exercise period for that message is considered to be over. Finally, the difference of the score after message played and the baseline score is calculated and enumerated with the variable, ΔS. Listed below are the results for each subject, as well as the exercise score plot extracted from the Excel file generated by the AAGILE Windows® application. The data is represented with a dual series line graph, where the x-axis is the time of day, and the y-axis represents either the score (blue line) or the message number that was played (red line). The title of the graph, S1, S2, etc., indicates the subject number.

Subject #	SBL	SMP	ΔS.
1	1.233	4.777	+3.544

Table 4. Results for Subject 1.

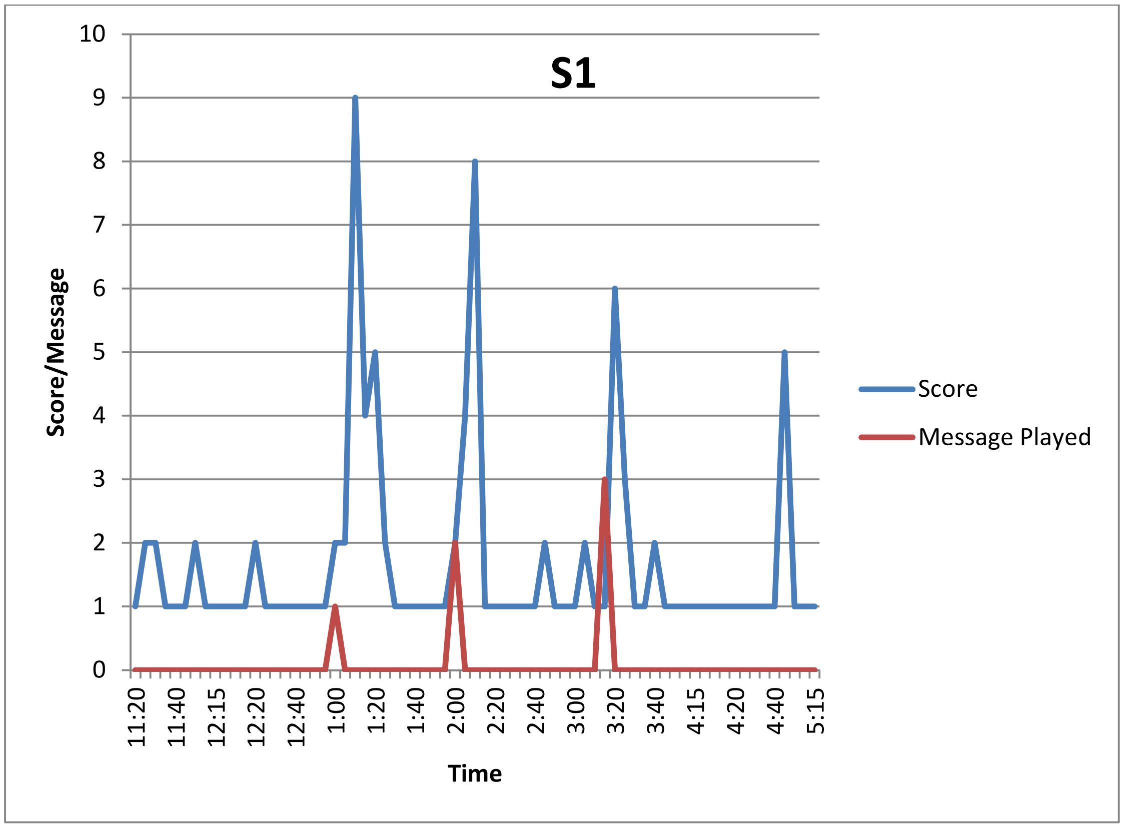


Figure 7. Score data graph for Subject 1.

Subject #	SBL	SMP	Δ*S*.
2	1.031	4.100	+3.069

Table 5. Results for Subject 2.

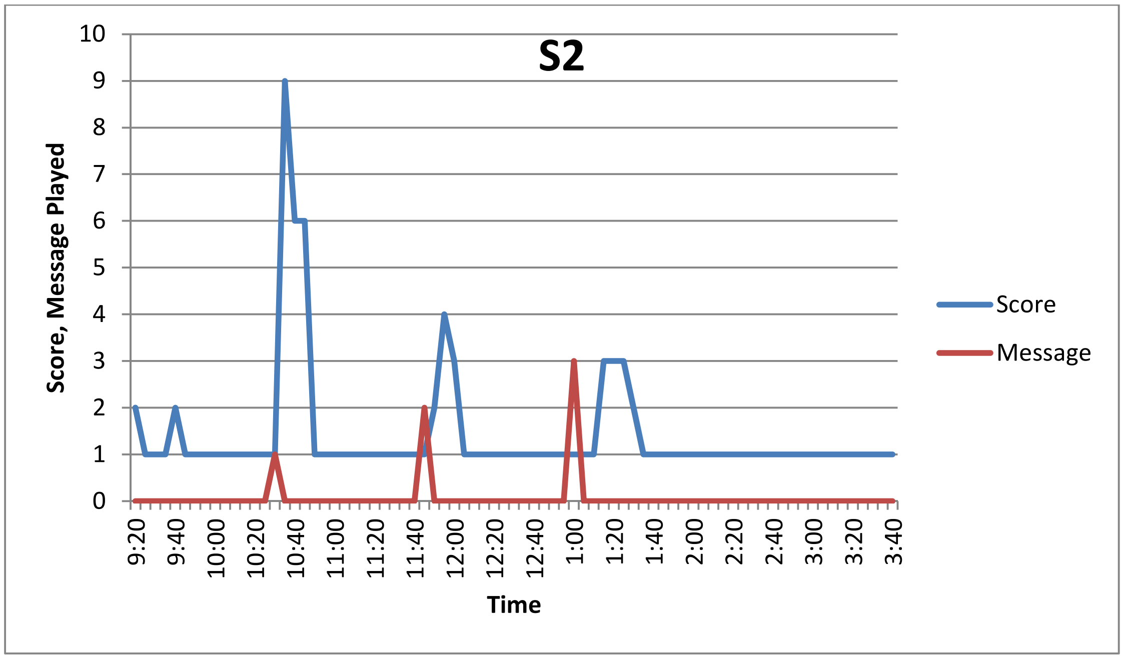


Figure 8. Score data graph for Subject 2.

Subject #	SBL	SMP	Δ*S*.
3	1.010	6.000	+4.990

Table 6. Results for Subject 3.

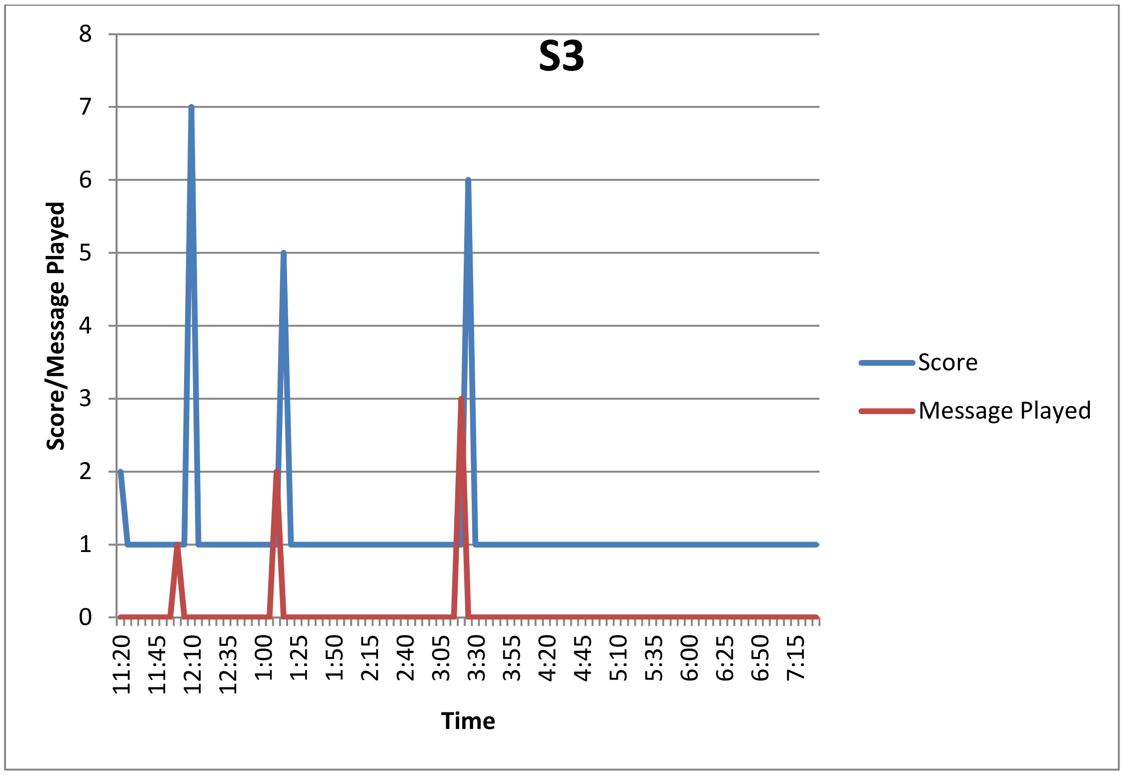


Figure 9. Score data graph for Subject 3.

Subject #	SBL	SMP	Δ*S*.
4	1.000	4.667	+3.667

Table 7. Results for Subject 4.

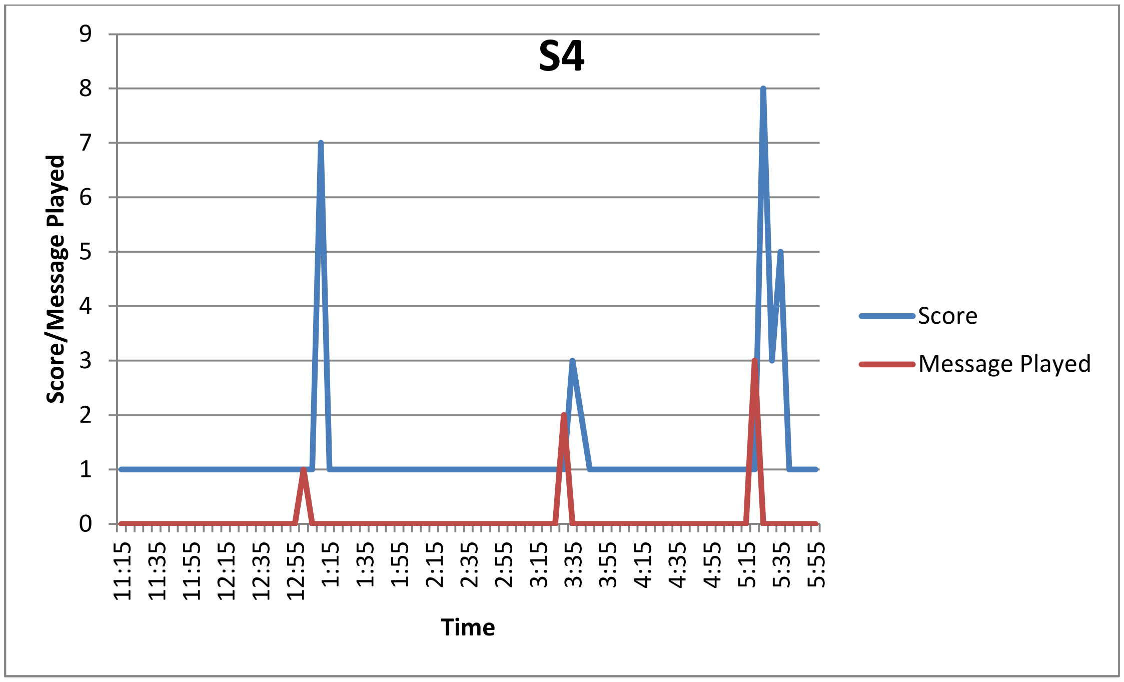


Figure 10. Score data graph for Subject 4.

Subject #	SBL	SMP	ΔS.
5	1.000	3.833	+2.833

Table 8. Results for Subject 5.

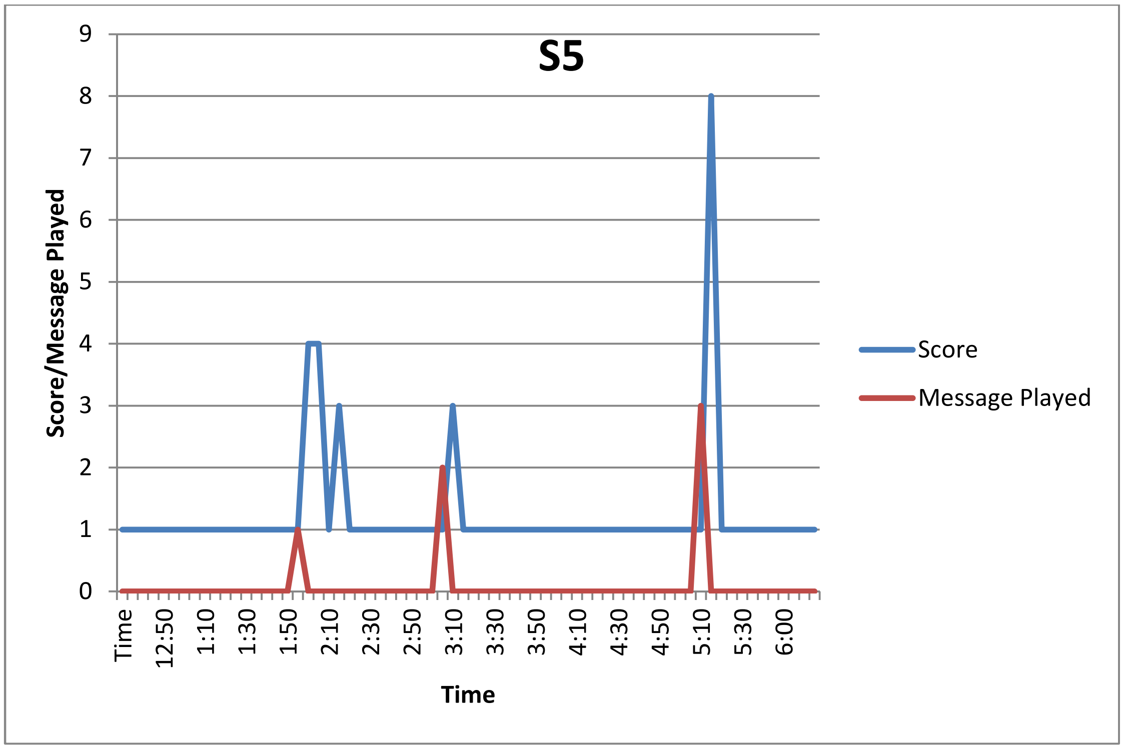


Figure 11. Score data graph for Subject 5.

Subject #	SBL	SMP	Δ*S*.
6	1.000	3.857	+2.857

Table 9. Results for Subject 6.

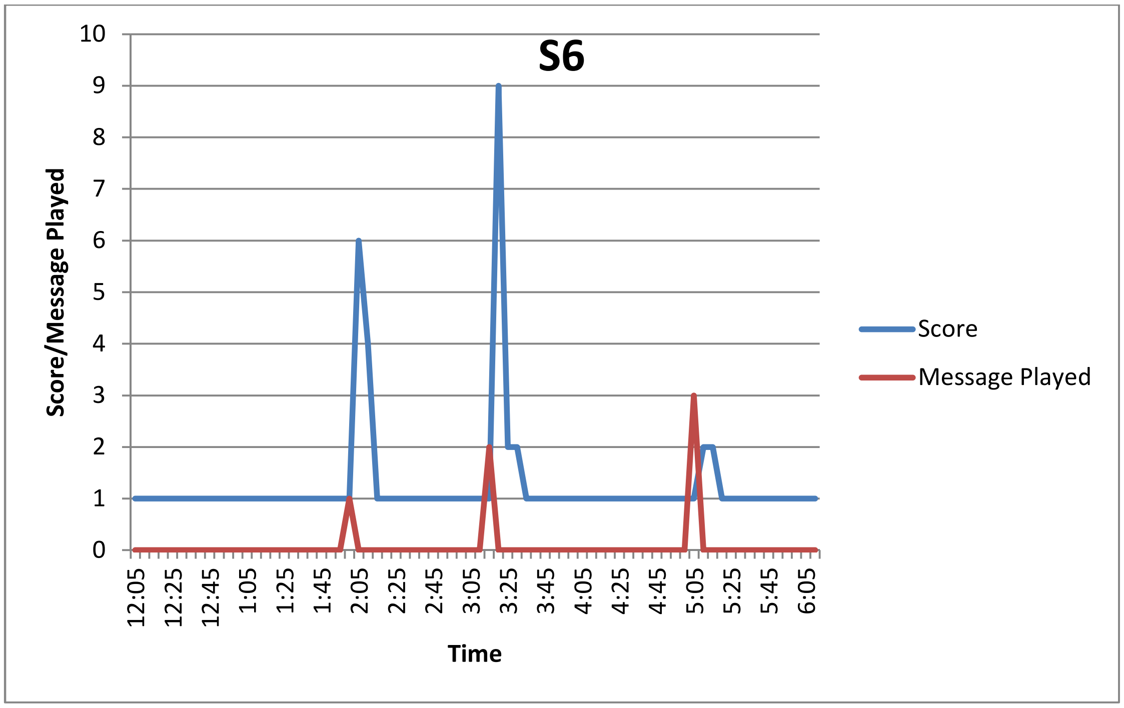


Figure 12. Score data graph for Subject 6.

Subject #	SBL	SMP	Δ*S*.
7	1.043	6.500	+5.457

Table 10. Results for Subject 7.

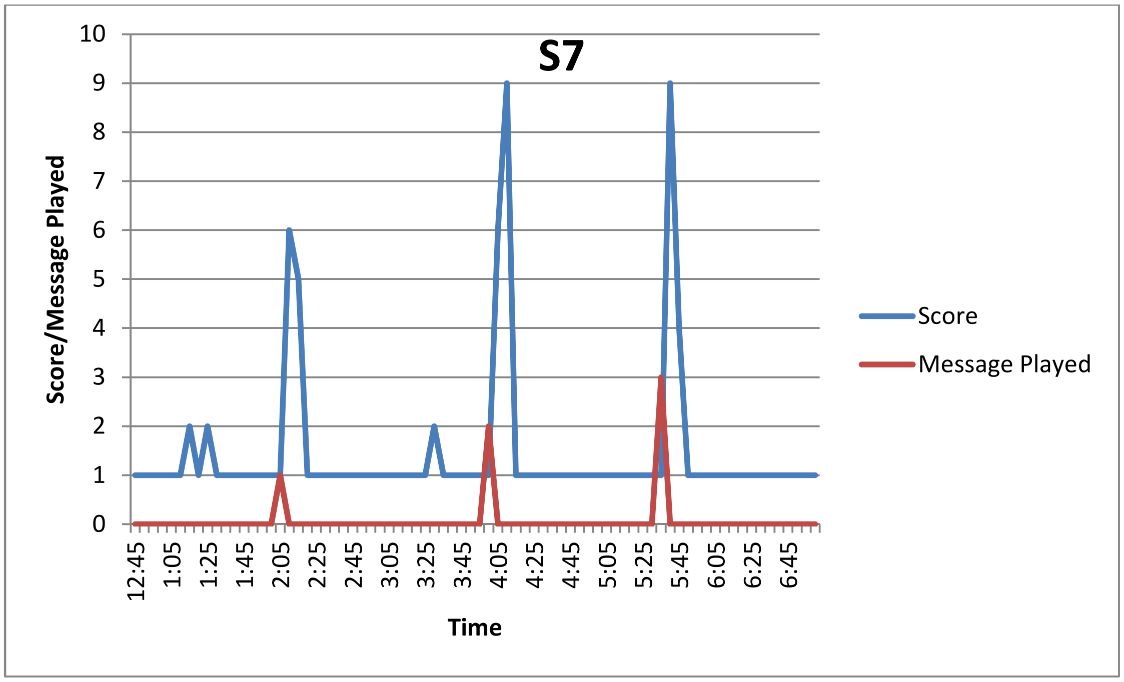


Figure 13. Score data graph for Subject 7.

Subject #	SBL	SMP	Δ*S*.
8	1.043	6.500	+5.457

Table 11. Results for Subject 8.

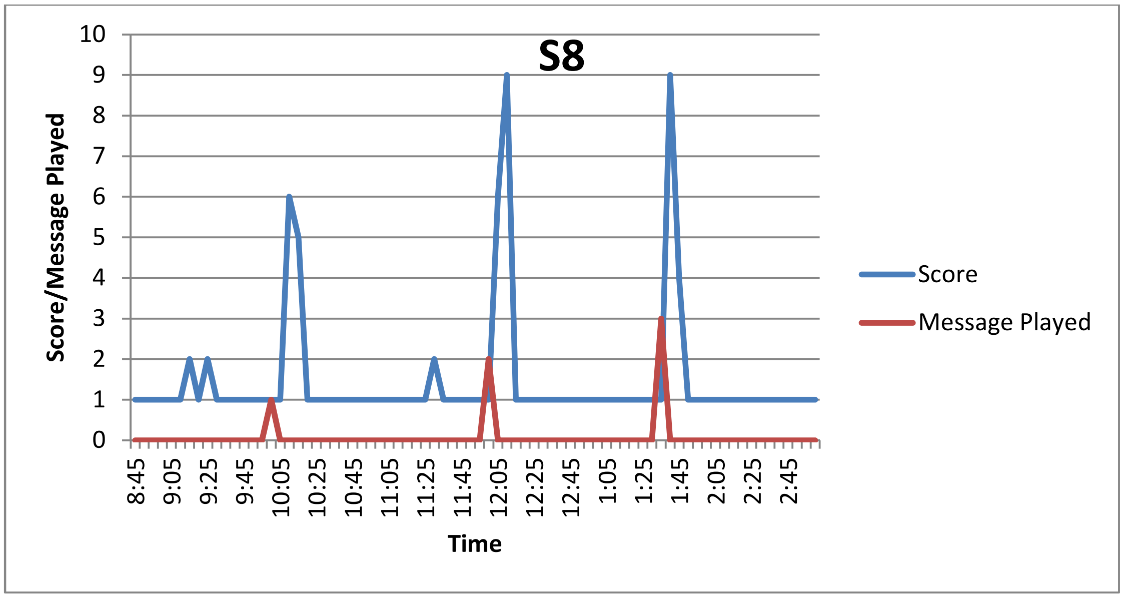


Figure 14. Score data graph for Subject 8.

Subject #	SBL	SMP	Δ*S*.
9	1.053	3.625	+2.572

Table 12. Results for Subject 9.

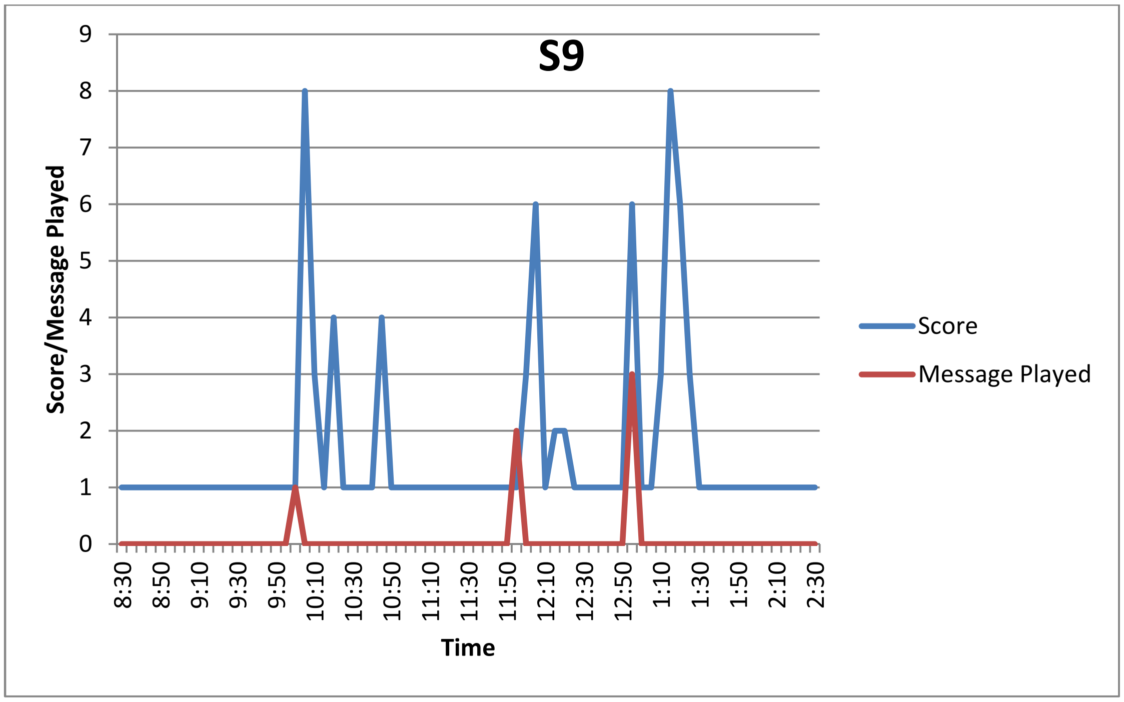


Figure 15. Score data graph for Subject 9.

Subject #	SBL	SMP	Δ*S*.
10	1.000	2.400	+1.400

Table 13. Results for Subject 10.

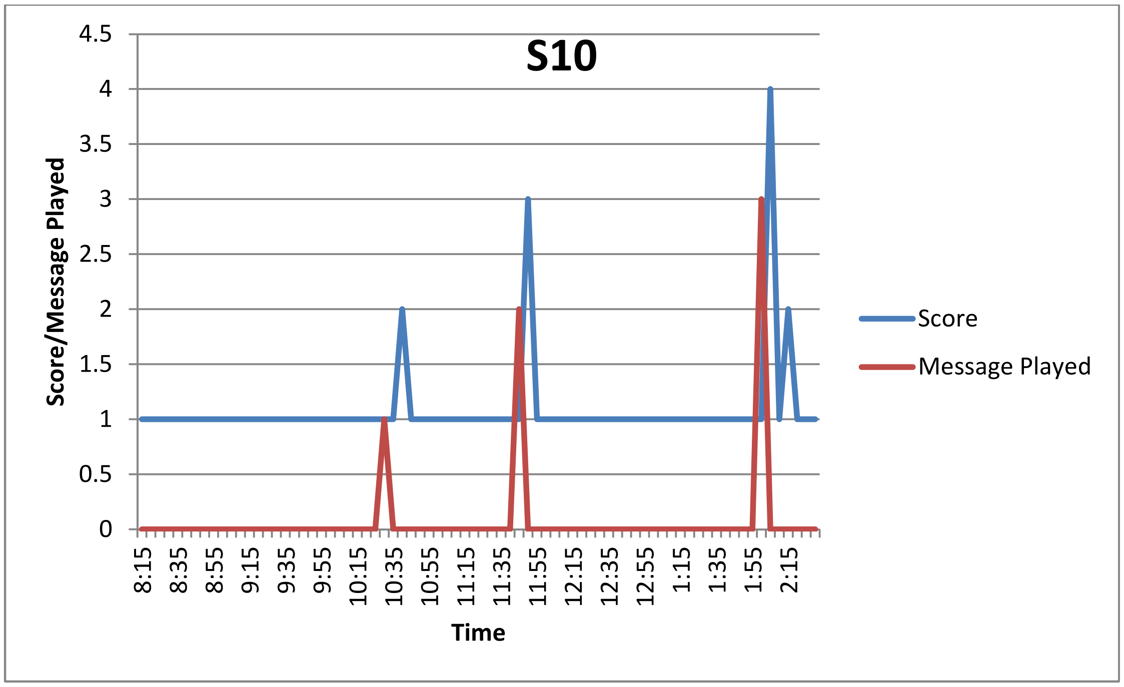


Figure 16. Score data graph for Subject 10.

Based on the data collected for the ten subjects, the exercise scores of the subjects increased post message-play for all of the subjects. The table below lists more statistics on the overall study.

SBL average	SBL maximum	SBL minimum	SMP average	SMP maximum	SMP minimum	ΔS *average*	ΔS *maximum*	ΔS *minimum*
1.041	1.233	1	4.626	6.500	2.400	+3.585	+5.457	+1.400

Table 14. Combined Statistics for All Subjects.

The baseline score for all of the subjects are within a relatively tight range, with the maximum baseline equal to 1.233 and the baseline minimum equal to 1. A score of 1 corresponds to very little movement of the AAGILE since the score is on a scale from 1 to 10. However, when the message was played the score increased by a wide range of scores. This is best explained because each subject exercised with different levels of intensity and duration, yielding an increase in score ranging from 2.4 to 6.5. For subject #10, the average score after message only caused for an increase of 1.4, meaning that the subject only exerted slightly more movement for the AAGILE. On the other hand, subjects #7 and #8 each had an increase in score of 5.457 after a message is played, which corresponds to a much larger acceleration applied to the AAGILE. Subject #9 has an increase of exercise score after a message had played, but due to a longer duration of exercise after each message. Due to the nature of the exercise type and that it was not a control for this study, the range for exercise score increases is rather high, with a range of 4.1 for score after a message was played.

Finally, the results of the post-study survey are summarized. All ten subjects had said that the AAGILE was easy to use. Usability of a device is very important, since it provides benefits to the user with minimal effort. This is especially true for older adults, since they may be less prone to learning how to use new technologies. Eight out of the ten subjects said that the messages were easy to hear. This is also important for the AAGILE's effectiveness, since if the message is not easy to hear, then the user may not hear the motivational message to cause an increase in exercise.

Nine out of ten subjects agreed that the message was not easy to understand. This may be due to insufficient design in the audio amplifier circuitry. The lack of comprehension of a message may significantly hinder the AAGILE's effectiveness to encourage exercise, since the content of the messages are important when the messages are personalized for the user.

Four out of the ten subjects explained that the AAGILE was bulky or uncomfortable to wear. The comfortableness of the AAGILE is crucial in that if the device is not comfortable for the user to wear, he or she may be less likely to use the AAGILE. This indicates that there may be a potential mechanical design flaw in the AAGILE design. In order to promote comfort and usability, wearable devices need to be sleek and have a natural feel to the wearer. In fact, a successful wearable device should have the effect as if the device was not there at all. Some concerns with the AAGILE's mechanical design include that the AAGILE was in the way of seatbelts, the AAGILE would constantly bump into objects when walking about, and the AAGILE seemed fragile and the wearer was nervous about damaging the device. These concerns should be addressed in order to improve upon the AAGILE device in the future.

The final survey question inquires the subject to what extent did the message motivate that person to be more active. The results average to 4 out of 5, indicating that the message did encourage an increase in physical activity by a significant amount, which correlates to the calculated ΔS average of +3.585. Since this function is the key feature of the AAGILE device, the results are promising in that the personalized messages do successfully motivate the user to exercise. The tally for this question is three subjects gave a 5, four subjects gave a 4, and three subjects gave a 3.

All of the data presented is compiled and analyzed and summarized in the next chapter.

CHAPTER 7

CONCLUSION

After analyzing the results of the study, the data can be summarized with two statements: the AAGILE can successfully motivate someone to increase exercise activity, and the overall design of the AAGILE has plenty room for improvement. The AAGILE monitors activity in a well-designed matter, and is capable of recording and reporting an averaged exercise score to the user. The AAGILE can correctly keep track of the time of day, and can determine when to play a message based on that clock. The AAGILE saves score information in non-volatile memory via the EEPROM, and the AAGILE can utilize its USB port to communicate with the Windows® PC. Lastly, the PC application successfully receives data from the AAGILE hardware and correctly displays the data in a clear manner, and also conveniently exports the data to an Excel file with a simple and efficient graphic interface.

The AAGILE's hardware implementation satisfies the self-imposed electronics requirements laid out in chapter 2. The AAGILE's firmware controls the hardware to satisfy the functional requirements of the system as a whole. Finally, the AAGILE's Window application software satisfies the graphical user interface requirements. Therefore, the AAGILE's design implementation is sufficient per its system level requirements, and the execution of the study further indicates that the design works as designed.

Besides design verification, the study also sheds some light on the effectiveness of the intended design based on the device’s essential performance. As indicated in the study results, the message played from the Activity Analyzer did cause for an increase of exercise activity in all ten subject cases. This further demonstrates that the AAGILE is capable of encouraging physical activity from an individual by using its personalized message playing capability. To further support this hypothesis, repeating the test with different types of messages, i.e. positive or negative feedback, can be done. Another example of an alternative study is to conduct the study on different age groups of people. The AAGILE design has been verified in this study, and should be able to be a part in the study examples listed above with no additional design changes.

While the study showed positive results, there were some limitations of the study. One example is that there was a relatively small sample size, both in the amount of subjects and the amount of data extracted from each subject. In other words, six hours of data might not reflect the true effectiveness of the AAGILE throughout all times of the day. This limitation is primarily attributed to the lack in EEPROM storage space on the processor. Another possible limitation is that some of the subjects were aware of the study’s goals. Although the study is meant to be objective, knowing the goal of the study could result in biasing of the data by the subjects. Finally, since there was no monitoring of the subjects while the study was conducted, there is no guarantee that the subject exercised to increase the activity score, or if the subject wore the device throughout the entire 6 hour period. As previously mentioned, there are many different variations of the study that can be conducted to better support the hypothesis.

While the design has been shown to pass all of the system level requirements of the device, as well as display that it can also execute its essential performance, the design and construction of the device needs to be refined. For a device to be developed successfully, it must not only have appropriate functionality, but should also have a good sense of usability. Based off of the survey results, there is much room for improvement on the usability of the device. This is evident in the survey responses related to inability to comprehend messages, discomfort while device is worn, and overall lack of esthetics of the device. Design improvements are strongly recommended for the AAGILE to ensure its marketability with competing devices with similar functions. As existing activity analyzers continue to improve in design, functionality, and usability, the AAGILE must too improve to stay competitive. Some design improvements that are strongly recommended include significantly shrinking the package of the device, improving sound quality of the audio circuitry, and simplifying the Windows® application user interface to make it easier to use for people with a wide variety of computer skills.

Despite the design flaws, the firmware bugs, the mechanical construction, etc., the AAGILE as a whole shows promise. Using a personalized message to encourage exercise is a unique quality of the AAGILE that no other activity analyzer has. From this study, the personalized message shows significant capability to motivate the user to exercise. The current AAGILE design provides a strong foundation that can support future design improvements and maintain a basic, but promising function in motivating people to exercise, and thus, encouraging a healthier lifestyle. The

AAGILE strives to reach the same single goal for its impact on society as other devices, which is to better improve quality of life.

APPENDICES

Some code bits for AAGILE firmware:

```
// __________________________ if (main_menu) sequence __________________________ /
                  if (main_menu) {
                           if (update_display) {
               Clear_screen();
               Set_position(0);
               Print_line("Main Menu", 9);
               Set_position(64);
               switch (main_menu_mode) {
               case 0:
                        Print_line("Exit", 4);
                        break;
               case 1:
                        Print_line("Set Clock", 9);
                        break;
               case 2:
                        Print_line("Edit Play Times", 14);
                        break;
               case 3:
                        Print_line("Edit Messages", 13);
                        break;
               case 4:
                        Print_line("USB Data Upload", 15);
                        break;
               case 5:
                        Print_line("Erase Memory", 12);
                        break;
               case 6:
                        Print_line("Sensitivity", 11);
                        break;}
                           update_display = 0; }
                           if (scroll) {
```

```
                                        scroll = 0;
                                        main_menu_mode++;
                                        if (main_menu_mode == 7) main_menu_mode = 0;
                                        update_display = 1; }
                            if (select) {
                select = 0;
                main_menu = 0;
          switch (main_menu_mode) {
          case 0:
          clock = 1;
          break;
                    case 1:
          set_clock = 1;
          break;
            case 2:
          set_play_times = 1;
          break;
          case 3:
          edit_messages = 1;
          break;
          case 4:
                                                  //Clear_screen();
                            //Set_position(0);
                                                  //Print_line("Sending Data...", 15);
                                                  //Delay_ms(10);

                                                  SpecifyEEPROM(0xFF);
                                                  unsigned char curLocation = readEEPROM();

                                                  SpecifyEEPROM(0x00);
                                                  Delay_ms(10);
                                                  PORTD.2 = 0;        //deselect LCD
                                                  Delay_ms(10);
                                                  PORTD.3 = 1;        //select USB
                                                  Delay_ms(10);
                    while(EEADR != 0xFF)
                    {
//                          PORTD.3 = 1;        //select USB
//                          PORTD.2 = 0;        //deselect LCD


                            ARHG = readEEPROM();
                            Transmit(ARHG);
                            INCEEPROM();
                    }

                    ARHG = readEEPROM();
                    Transmit(ARHG);

                    Delay_ms(10);
                    SpecifyEEPROM(curLocation);
                    usb = 0;
        main_menu = 1;
        PORTD.3 = 0;        //deselect USB
                    Delay_ms(10);
                    PORTD.2 = 1;        //select LCD
                    Delay_ms(10);
        break;
                case 5:
                    Clear_screen();
            Set_position(0);
                    Print_line("Erasing...", 10);
                    SpecifyEEPROM(0x00);
                    while(EEADR != 0xFF){
                            writeEEPROM(0xFF);
                            INCEEPROM();}
                    SpecifyEEPROM(0xFF);
                    writeEEPROM(0x00);
                    SpecifyEEPROM(0x00);
```

```
                                        messageaddr = 0x00;
                                        stop_save = 0;
                        main_menu = 1;
                                    break;
                                  case 6:
                                        set_sensitivity = 1;
                                    break;}
                         main_menu_mode = 0;
                         update_display = 1;}}


// ____________________________ Setup for Scoring _____________________________ /
//Motion detection algorithm: Every few seconds, a 16 point array will be populated
//with 16 consecutive data points from the accelerometer. These points are then
//averaged into the value 'average'. We then compare all incoming data to this
//average and compare it to a threshold. If data is 3 points higher than threshold,
//then we increase motion counter. This algorithm takes drift of accelerometer into
//account as average is constantly recalculated around the current acc. data
                         if(counter == 0){
                              total = 0;
                              for(i=0; i<16; i++) array[i] = ad_input;
                              for(i=0; i<16; i++) total += array[i];
                              average = total/16;
                              counter = 250; }
                         data0 = ad_input;
                         threshold = average + 3;
                         if(data0 < threshold)baseline = 0;
                         if((data0 > threshold) && baseline == 0){
                              baseline = 1;
                              motion++;
                              Delay_ms(70); }

          if((minute==1||minute==6||minute==11||minute==16||minute==21||minute==26||minute==31||minute==36||minute==41
||
                              minute==46||minute==51||minute==56)&&TimeF==1)TimeF = exercise2f = 0;
```

```
// ____________________________ Score/Save Motion _____________________________ /
//Save to memory every minutes: we had to choose five minute resolution for mem.
//space reasons. Every time we save data to memory, we need to save the score,
//hour, and minute for data analysis at the end of the day. The EEPROM of the
//PIC has 256 bytes of memory. Using a full byte of memory for score, hour, and
//minute would only allow for 7 hours of storage. To get around this, we created
//an algorithm that only uses half of a byte for each, or one and a half bytes
//per storage which doubles the memory storage capability.


        if((minute==0||minute==5||minute==10||minute==15||minute==20||minute==25||minute==30||minute==35||minute==40
||
                        minute==45||minute==50||minute==55)&& TimeF==0 && stop_save == 0){
                        TimeF = 1;                  //Flag used to make sure we only save data once
                        counter = 0;
                        Delay_ms(70);
//The following are different sensitivity settings for people with more motion
                        if (high == 0 && medium == 0 && low == 0) high = 1;
                        if (high == 1){
                                if(motion >= 0 && motion <= 30) motion = 1;
                                if(motion > 30 && motion <= 60) motion = 2;
                                if(motion > 60 && motion <= 90) motion = 3;
                                if(motion > 90 && motion <= 120) motion = 4;
                                if(motion > 120 && motion <= 150) motion = 5;
                                if(motion > 150 && motion <= 180) motion = 6;
                                if(motion > 180 && motion <= 210) motion = 7;
                                if(motion > 210 && motion <= 240) motion = 8;
                                if(motion > 240 && motion <= 270) motion = 9;
                                if(motion > 270) motion = 10;
                                score = motion;}
                        if (medium == 1){
                                if(motion >= 0 && motion <= 45) motion = 1;
                                if(motion > 45 && motion <= 90) motion = 2;
                                if(motion > 90 && motion <= 135) motion = 3;
                                if(motion > 135 && motion <= 180) motion = 4;
                                if(motion > 180 && motion <= 225) motion = 5;
```

```
                                        if(motion > 225 && motion <= 270) motion = 6;
                                        if(motion > 270 && motion <= 315) motion = 7;
                                        if(motion > 315 && motion <= 360) motion = 8;
                                        if(motion > 360 && motion <= 405) motion = 9;
                                        if(motion > 405) motion = 10;
                                        score = motion;}
                                if (low == 1){
                                        if(motion >= 0 && motion <= 60) motion = 1;
                                        if(motion > 60 && motion <= 120) motion = 2;
                                        if(motion > 120 && motion <= 180) motion = 3;
                                        if(motion > 180 && motion <= 240) motion = 4;
                                        if(motion > 240 && motion <= 300) motion = 5;
                                        if(motion > 300 && motion <= 360) motion = 6;
                                        if(motion > 360 && motion <= 420) motion = 7;
                                        if(motion > 420 && motion <= 480) motion = 8;
                                        if(motion > 480 && motion <= 540) motion = 9;
                                        if(motion > 540) motion = 10;
                                        score = motion;}
//The following code is used to determine what should be save to memory and
//the actual saving of that data. Explaining how it works can be complicated
//so the variables are named to to represent their function. Going through
//the code and writing a flow chart is the best way to understand what is
//being done. There is a lot of manipulation of memory and data to break the
//bytes in memory in half. Also, it should be noted that the scores and times
//are stored in decimal form. This allowed us to debug more efficiently as we
//did not have to convert everything from hex to decimal. Last note: The mem.
//is stored as follows: score, hour, minute. to find the minute, multiply
//whatever is stored in memory by five.
//Example from memory: 1C 34 C4. This would mean that their was a score of 1
//at 12 (C) fifteen (3*5) and a score of 4 at 12 (C) 20 (4*5)
        readEEPROM();
        if(EEDATA == 255){
                                        mincheck = 0;
                                        hour1 = hour; minute1 = minute;
                                        if(score == 1) score = 16;
```

```
		if(score == 2) score = 32;
		if(score == 3) score = 48;
		if(score == 4) score = 64;
		if(score == 5) score = 80;
		if(score == 6) score = 96;
		if(score == 7) score = 112;
		if(score == 8) score = 128;
		if(score == 9) score = 144;
		if(score == 10) score = 160;
		data = score + hour1;
		if(score == 16) exercise++; }
	if(EEDATA != 255){
		mincheck = 1;
		hour1 = hour; minute1 = minute;
		readEEPROM();
		data = (EEDATA - 15) + score;
		if(score == 1) exercise++; }
	writeEEPROM(data);
	Delay_ms(100);
	INCEEPROM();
	Delay_ms(100);
	readEEPROM();
	if(EEDATA == 255 && mincheck == 0){
		hour1 = hour; minute1 = minute;
		if(minute1 == 0) minute1 = 15;
		if(minute1 == 5) minute1 = 31;
		if(minute1 == 10) minute1 = 47;
		if(minute1 == 15) minute1 = 63;
		if(minute1 == 20) minute1 = 79;
		if(minute1 == 25) minute1 = 95;
		if(minute1 == 30) minute1 = 111;
		if(minute1 == 35) minute1 = 127;
		if(minute1 == 40) minute1 = 143;
		if(minute1 == 45) minute1 = 159;
		if(minute1 == 50) minute1 = 175;
```

```
        if(minute1 == 55) minute1 = 191;
        data = minute1; }
if(EEDATA == 255 && mincheck == 1){
        hour1 = hour; minute1 = minute;
        if(hour1 == 1) hour1 = 16;
        if(hour1 == 2) hour1 = 32;
        if(hour1 == 3) hour1 = 48;
        if(hour1 == 4) hour1 = 64;
        if(hour1 == 5) hour1 = 80;
        if(hour1 == 6) hour1 = 96;
        if(hour1 == 7) hour1 = 112;
        if(hour1 == 8) hour1 = 128;
        if(hour1 == 9) hour1 = 144;
        if(hour1 == 10) hour1 = 160;
        if(hour1 == 11) hour1 = 176;
        if(hour1 == 12) hour1 = 192;
        if(minute1 == 0) minute1 = 0;
        if(minute1 == 5) minute1 = 1;
        if(minute1 == 10) minute1 = 2;
        if(minute1 == 15) minute1 = 3;
        if(minute1 == 20) minute1 = 4;
        if(minute1 == 25) minute1 = 5;
        if(minute1 == 30) minute1 = 6;
        if(minute1 == 35) minute1 = 7;
        if(minute1 == 40) minute1 = 8;
        if(minute1 == 45) minute1 = 9;
        if(minute1 == 50) minute1 = 10;
        if(minute1 == 55) minute1 = 11;
        data1 = hour1 + minute1; }
if(mincheck == 0)writeEEPROM(data);
if(mincheck == 1){
        writeEEPROM(data1);
        Delay_ms(100);
        INCEEPROM(); }
messageaddr = EEADR;
```

```
                        SpecifyEEPROM(0xFF);
                        Delay_ms(100);
                        writeEEPROM(messageaddr);
                        Delay_ms(100);
                        SpecifyEEPROM(messageaddr);
                        Delay_ms(100);
                        motion = 0; }
// _____________ Define High Priority Interrupt Service Routine _________________ /
#pragma origin 0x8
interrupt highPriorityInterrupt (void) {
        #pragma fastMode
        _highPriorityInt (); }
void _highPriorityInt (void) {
checkflags:
        if (TMR0IF == 1) {
                TMR0IE = 0;
                TMR0IF = 0;
                TMR0H = 0xFC;
                TMR0L = 0x36;
                ad_input = Read_adc();
                if(counter != 0) counter --;
                if (button_delay != 0) button_delay--;
                if (millisec < 999) millisec++;
                else if (second < 59){
                        update_display = 1;
                        millisec = 0;
                        second++;
                        relay_time++;
                        relay_time1++;
                if (message_timer) {
                        message_timer_count_short++;
                        message_timer_count_long++; }
                else message_timer_count_short = message_timer_count_long = 0;}
                else if (minute < 59) {
                        update_display = 1;
                        millisec = 0;
                        second = 0;
                        minute++;
                        no_play = 1;}
                else if (hour < 12) {
                        update_display = 1;
                        millisec = 0;
                        second = 0;
                        minute = 0;
                        hour++;
                        if (hour == 12) pm = !pm; }
                else {
                        update_display = 1;
                        millisec = 0;
                        second = 0;
                        minute = 0;
                        hour = 1; }
                TMR0IE = 1; }
        if(INT0IF == 1){
                INT0IF = 0;
                if(button_delay == 0){
                        usb = !usb;
                        scroll = !scroll;
                        button_delay = 200;
                        goto checkflags;}}
        if(INT1IF == 1){
                INT1IF = 0;
                if(button_delay == 0){
                        select = !select;
                        button_delay = 200;
                        goto checkflags;}}
        if(INT2IF == 1){
                INT2IF = 0;
                if(button_delay == 0){
                        replay = !replay;
```

```
button_delay = 200;

goto checkflags;}}}
```

Some code bits for AAGILE software:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO.Ports;
using Excel = Microsoft.Office.Interop.Excel;

namespace AAVIDinterface
{
    public partial class Form1 : Form
    {


        public Form1()
        {
            InitializeComponent();
        }



        public static SerialPort sp = new SerialPort();
        public static DataTable dt;
        public static DataColumn dc;
        public static int strindex = 0;
        public static string shm;

        byte[] input = new byte[256]; // should be 255
        int recv;

        public static Excel.Application xl;
        public static Excel.Workbook xlWorkBook;
        public static Excel.Worksheet xlWorkSheet;
        public static int rowindex;

        public static int ihour, iminute, iscore;


        private void readyToConnect_Click(object sender, EventArgs e)
        {
            if (SIM == 0)
            {
                richTextBox1.Text = "BaudRate = " + sp.BaudRate.ToString();
                richTextBox1.Text += "\nStop Bit = " + sp.StopBits.ToString();
                richTextBox1.Text += "\nDataBits = " + sp.DataBits.ToString();
                richTextBox1.Text += "\nParity = " + sp.Parity.ToString();
                richTextBox1.Text += "\nReadTimeout = " +
sp.ReadTimeout.ToString();

                string comportnum = numericUpDown1.Value.ToString();
                sp.PortName = "COM" + comportnum; // set COM port here
                richTextBox1.Text += "\nCOM Port = " + sp.PortName.ToString();

                try
                {
                    //open serial port
                    sp.Open();
                    sp.ReadTimeout = 3000;
                    richTextBox1.Text += "\nStatus...Connected\n";
                }
                catch (System.Exception ex)
                {
                    richTextBox1.Text += "\nStatus...Failed to Connect ---> " +
ex.Message + "\n";
                }
            }
            else
            {
                richTextBox1.Text += "\nStatus...Connected\n";
            }
        }

        private void import_Click(object sender, EventArgs e)
        {


                try
                {
                    if (SIM == 0)
                    {


                        //int dec = sp.ReadByte();

                        recv = sp.Read(input, 0, 256); //should be 0,256

                    }
                    else
                    {
                        input = sim_input;
                    }
                    xl = new Excel.Application();
                    xlWorkBook = xl.Workbooks.Add(Type.Missing);
                    xlWorkSheet =
(Excel.Worksheet)xlWorkBook.Worksheets.get_Item(1);
                    rowindex = 2;
                    dt = new DataTable();
                    dc = new DataColumn();
                    dc.ColumnName = "Time"; // x-axis title;
                    dt.Columns.Add(dc);
                    dc = new DataColumn();
                    dc.ColumnName = "Score"; // y-axis title
                    dt.Columns.Add(dc);
                    dc = new DataColumn();
                    dc.ColumnName = "Message Played"; // message title
                    dt.Columns.Add(dc);

                    xl.Cells[1, 1] = "Time";
                    xl.Cells[1, 2] = "Score";

                        //richTextBox1.Text =
System.Text.Encoding.UTF8.GetString(input, 0, 240);

                        string hex = BitConverter.ToString(input);
                        hex = hex.Replace("-", "") + "\n";
                        richTextBox1.Text += hex + "\n";

                        //now make substrings with 3 chunks at a time
                        for (int w = 0; w < 510; w += 3)
                        {
                            shm = hex.Substring(w, 3);
                            richTextBox1.Text += shm + "\n";




                        //    //string test = "A42";
                            string score = shm.Substring(0, 1);
                            string hour = shm.Substring(1, 1);
                            string minute = shm.Substring(2, 1);

                            richTextBox1.Text += "score: " + score + " hour: "
+ hour + " minute: " + minute + "\n";

                            if (shm != "FFF")
                            {
                                if (score != "F")
                                {
                                    //***Converts string hex values to
decimal***

                                    if (score == "A")
                                        score = "10";

                                    if (hour == "A")
                                        hour = "10";
                                    else if (hour == "B")
                                        hour = "11";
                                    else if (hour == "C")
                                        hour = "12";

                                    if (minute == "A")
                                        minute = "10";
                                    else if (minute == "B")
                                        minute = "11";

                                    richTextBox1.Text += "score: " + score + "
hour: " + hour + " minute: " + minute + "\n";


                                    iscore = Convert.ToInt32(score);
//***Converts string to int***
                                    ihour = Convert.ToInt32(hour);
                                    iminute = Convert.ToInt32(minute) * 5;
                                    string doublezero = "00";

                                    DataRow dr;//add rows
                                    dr = dt.NewRow();
                                    if (iminute == 0)
                                    {
                                        dr["Time"] = ihour.ToString() + ":" +
doublezero;//Concatonate hour and minute to make a full time
                                        xl.Cells[rowindex, 1] =
ihour.ToString() + ":" + doublezero;
                                    }
                                    else if (iminute == 5)
                                    {
                                        doublezero = "05";
                                        dr["Time"] = ihour.ToString() + ":" +
doublezero;
                                        xl.Cells[rowindex, 1] =
ihour.ToString() + ":" + doublezero;
                                    }
                                    else
                                    {
                                        dr["Time"] = ihour.ToString() + ":" +
iminute.ToString();
                                        xl.Cells[rowindex, 1] =
ihour.ToString() + ":" + iminute.ToString();
                                    }

                                    dr["Score"] = iscore;
                                    dr["Message Played"] = 0;

                                    dt.Rows.Add(dr);
                                    xl.Cells[rowindex, 2] = iscore;

                                    rowindex++;
                                }
                                else  //score = F, message played here instead
of a score. Message played will be minute, score will remain the same
                                {
                                    string doublezero = "00";

                                    DataRow dr;//add rows
                                    dr = dt.NewRow();
                                    if (iminute == 0)
                                    {
                                        dr["Time"] = ihour.ToString() + ":" +
doublezero;//Concatonate hour and minute to make a full time
                                        //xl.Cells[rowindex, 1] =
ihour.ToString() + ":" + doublezero;
                                    }
                                    else if (iminute == 5)
                                    {
                                        doublezero = "05";
                                        dr["Time"] = ihour.ToString() + ":" +
doublezero;
                                        //xl.Cells[rowindex, 1] =
ihour.ToString() + ":" + doublezero;
                                    }
                                    else
                                    {
                                        dr["Time"] = ihour.ToString() + ":" + 
iminute.ToString();
                                        //xl.Cells[rowindex, 1] = 
ihour.ToString() + ":" + iminute.ToString();
                                    }

                                    dr["Score"] = iscore;
                                    dr["Message Played"] = 
Convert.ToInt32(minute);

                                    dt.Rows.Add(dr);
                                }

                                dataGridView1.DataSource = dt;
                                chart1.DataSource = dt;
                                chart1.Series["Series1"].XValueMember = "Time";
                                chart1.Series["Series1"].YValueMembers = 
"Score";
                                chart1.Series["Series2"].XValueMember = "Time";
                                chart1.Series["Series2"].YValueMembers = 
"Message Played";
                                chart1.DataBind();

                            }
                        }

                }
                catch (System.Exception ex)
                {
                    richTextBox1.Text += "Status...Read Failed ---> " + 
ex.Message;
                    sp.Close();
                }

        }

        private void helpToolStripMenuItem_Click(object sender, EventArgs e)
        {
            MessageBox.Show("Welcome to the Activity Analyzer with 
Individualized Voice Direction (AAVID) computer program. This program enables 
the user to evaluate the amount of activity that was completed throughout the 
day. The program has features that include saving the activity graph, exporting 
data to Excel, etc. Click 'Ready' to connect to the docking station. Then press 
the button on the docking station to send the data. Last, click on the 'Import' 
button to get the data and display on the graph.","Help");

        }

        private void saveasToolStripMenuItem_Click(object sender, EventArgs e)
        {
            saveFileDialog1.Filter = "Image Files (*.jpg)|*.jpg";

            if (saveFileDialog1.ShowDialog() == 
System.Windows.Forms.DialogResult.OK
                && saveFileDialog1.FileName.Length > 0)
            {
```

```
chart1.SaveImage(saveFileDialog1.FileName,System.Drawing.Imaging.ImageFormat.Jp
eg);
            }
        }

        private void exportToolStripMenuItem_Click(object sender, EventArgs e)
        {


            saveFileDialog1.Filter = "Excel Files (*.xls)|*.xls";

            if (saveFileDialog1.ShowDialog() ==
System.Windows.Forms.DialogResult.OK
                && saveFileDialog1.FileName.Length > 0)
            {
                 xl.ActiveWorkbook.SaveCopyAs(saveFileDialog1.FileName);
                 xl.ActiveWorkbook.Saved = true;

                 xl.Quit();
            }
        }

        private void button1_Click(object sender, EventArgs e) //close
        {
            if (sp.IsOpen)
            {
                sp.Close();
                richTextBox1.Text += "Serial Port Closed\n";
            }
        }

        private void Form1_FormClosing(object sender, FormClosingEventArgs e)
        {
            if (sp.IsOpen) sp.Close();
        }



    }
}
```

BIBLIOGRAPHY

Anderson, Virginia, and Brunilda Nazario. "Exercise Motivation: How to Get It, How to Keep It." *WebMD*. WebMD, 8 July 2013. Web. 2 Nov. 2014. <http://www.webmd.com/fitness-exercise/features/exercise-motivation>.

Booth, Frank, Christian Roberts, and Matthew Laye. "Lack of Exercise Is a Major Cause of Chronic Diseases." *Comprehensive Physiology* 2 (2012): 1143-211. Print.

Bouchard, C., Depres, J.-P. and Tremblay, A. (1993), *Exercise and Obesity*. *Obesity Research*, 1: 133–147. doi: 10.1002/j.1550-8528.1993.tb00603.x

Burbank P, Riebe D. “Promoting Exercise and Behavior Change in Older Adults: Interventions and Transtheoretical Model.” NY: Springer, 2002.

Burbank P, Sun Y. “Activity monitor and analyzer with voice direction for exercise.” U.S. Patent and PCT pending, US Application No.: 13/475532, May 18, 2010.

Centers for Disease Control and Prevention. “State Indicator Report on Physical Activity, 2014.” Atlanta, GA: *U.S. Department of Health and Human Services*, 2014.

Greene H, Dulude C, Neves A, Sun Y, Burbank P. “Performance evaluation of the activity analyzer.” Proceedings 38th Northeast Bioengineering Conference, Philadelphia, PA, pp. 31-32, March 16-18, 2012.

Hulst, Monique Van Der. "Long Workhours and Health." *Scandinavian Journal of Work, Environment & Health* 29.3 (2003): 171-88. Print.

Laforge, Robert, Joeseph Rossi, and James Pochaska. "Stage Of Regular Exercise And Health-Related Quality Of Life," *Preventive Medicine* 28.4 (1999): 349-60. Print.

McInnis, Kyle. "Exercise and Obesity." *Coronary Artery Disease* 11.2 (2000): 111-16. Print.

Newson Rachel S, Kemps Eva B. *Factors That Premote and Prevent Exercise Engagement in Older Adults. J Aging Health June 2007.* vol 19 no 3. 470-481. doi10.1177/0898264307300169.

Rafferty K, T. Alberg, H. Green, G. Ausfresser, Y.Sun, and P.M. Burbank. “Development of an Activity Analyzer with Voice Directions for Exercises.” 38th Annual Northeast Bioengineering Conference, Temple University, Philadelphia, PA, March 16-18, 2012

Raglin, John. "Exercise and Mental Health." *Sports Medicine* 9.6 (1990): 323-29. Print.

Regan, C., Katona, C., Walker, Z. and Livingston, G. *Relationship of exercise and other risk factors to depression of Alzheimer's disease: the LASER-AD study. Int. J. Geriat. Psychiatry*, 20: 261–268. doi: 10.1002/gps.1278. February 16 2005.

Roth, D L, and D S Holmes. "Influence of Aerobic Exercise Training and Relaxation Training on Physical and Psychologic Health following Stressful Life Events." *Psychosomatic Medicine* 49.4 (1987). Print.

Sparks, Kate, Cary Cooper, Yitzhak Fried, and Arie Shirom. "The Effects of Hours of Work on Health: A Meta-analytic Review." *Journal of Occupational and Organizational Psychology* 70.4 (1997): 391-408. Print.

Steinberg, Scott. *The Modern Parent's Guide to Kids and Video Games*. Lilburn, GA: P3: Power Play, 2011. Print.

St. Marie, Geoffrey. "Reasons for a Lack of Exercise." *LIVESTRONG.COM*. LIVESTRONG.COM, 21 Oct. 2013. Web. 2 Nov. 2014. <http://www.livestrong.com/article/480858-reasons-for-a-lack-of-exercise/>.

“Technical Datasheet for the EDR Series: Reed Relay.” Mobicon Electronics Components.

“Technical Datasheet for the FT232R USB UART IC: Incorporating Clock Generator Output and FTDIChip-ID Security Dongle. Revision 1.04.” Future Technology Devices International Ltd. January 30, 2006.

“Techincal Datasheet for the ISD1700 Series: Multi-Message Single-Chip Voice Record and Playback Devices. Revision 1.31.” Nuvoton Technology. October 31, 2008.

“Technical Datasheet for the L78L: Positive Voltage Regulators. Document 2145, Revision 22.” STMicroelectronics. April 14, 2014.

“Technical Datasheet for the LIS302SG: MEMS motion sensor 3-axis - ±2g analog output “piccolo” accelerometer. Revision 1.” STMicroelectronics. March 2008.

“Technical Datasheet for the LM386: Low Voltage Audio Power Amplifier. Document SNAS545A.” Texas Instruments. August 2000.

“Technical Datasheet for the MC14016B: Quad Analog Switch/Quad Multiplexer. Document MC14016B/D, Revision 10.” On Semiconductor. April 2013.

“Technical Datasheet for the MCP1702: 250 mA Low Quiescent Current LDO Regulator. Document DS22008E, Revision E.” Microchip Technology. November 2010.

“Technical Datasheet for the PIC18FXX2: High-Performance, Enhanced Flash Microcontrollers with 10-Bit A/D. Document DS39564C, Revision C.” Microchip Technology. October 2006.

“Technical Datasheet for the SerLCD v2.5: Serieal Enabled LCD.” Spark Fun Electronics. July 5, 2006.

T. Wang, J. Harvey, E. Chabot, PhD, Y. Sun, PhD, P.M. Burbank, DNSc, RN, “Activity Analyzer for Guided Independent Living Environments (AAGILE): A preliminary study on healthy young adults.” 39th Annual Bioengineering Conference, Syracuse University, Syracuse NY, April 5-7, 2013.

"Video Game Industry Statistics | Entertainment Software Rating Board." *Video Game Industry Statistics | Entertainment Software Rating Board*. Web. 2 Nov. 2014. <http://www.esrb.org/about/video-game-industry-statistics.jsp>.