Dawn Light

Sleep Tracking and Smart Wake Up System

Design Document

UNIVERSITY OF CALIFORNIA
SANTA CRUZ

Jacob McClellan, Zhenyu Zhang, Nicholas Eaton, Steven Appler, Mingyu Ding, Weilin Shu

Executive Summary

The Dawn Lamp is a product that tracks a user's sleep and utilizes lighting to help optimize the wake-up experience for those who sleep at odd hours. The product uses an accelerometer attached to the user's bed. By processing movement data live, we are able to accurately determine when a user is in the 3 basic stages of sleep: very light NREM, other NREM, and REM. Our analysis algorithm is based on current scientific literature and our own testing using both motion and BPM detection. Through an attached lamp, the product produces an artificial sunrise when the user is in the lightest stages of NREM sleep, which has the minimal level of wake-inertia.

As data is collected throughout the night, it is transferred to a Raspberry Pi within the lamp body. The Pi serves as the base station for all processing and control. Data is analysed in conjunction with the user's current settings to decide how and when the user will be awoken. The base station controls a dimmable natural wavelength lightbulb through pulse-width modulation, which is activated upon the user reaching the end sleep stage. Through the Android App, users can change lamp functionality, or control constants used in the algorithm. Users will also be able to see the previous night's sleep data and how it affects the algorithm.

This document outlines the cause, design, and testing of the end product. We will go into detail on the scientific background for why night shifts are detrimental for sleep, and why our product could improve sleeping for impacted workers. The design will be broken down into each module of the system for full understanding of the intercommunications and parallel operations of the system. Finally, a full runthrough of all functionality of the product will be described for each module, proving that it is ready for production

Table of Contents

1 Introduction

1.1 Need Statement

People who sleep at unconventional hours experience detrimental sleep quality due to daytime lighting in their sleep environment.

1.2 Goal Statement

Enhance the sleep quality for people sleeping at unconventional hours by manipulating their bedroom lighting.

1.3 Design Objective

Design a low-cost, low-maintenance system that simulates the rising sun, activating dynamically based on the user's sleep stages.

1.4 Personas

As a warehouse employee, John frequently works night shifts. John often finds himself going to bed and waking up while the sun is still in the sky. As John continues to work odd and late hours, he begins to feel tired all the time. Although he tries to sleep as much as possible every day, it never seems to be enough. John's boss notices his dour mood at work, and his friends don't see the same energy in him that he used to have.

1.5 Scientific Background

Night shift workers, those characterized as working outside the window of 7AM-6PM, are at a high risk of Shift Work Sleep Disorder (SWSD). This is a condition caused by a misalignment of an individual's circadian rhythm and their sleep preferences brought on by the hours in which they are required to be awake and work[1]. This results in worse quality and quantity of sleep in these individuals which in turn affects their health and well being outcomes. To name a few, poor sleep quality worsens cognitive ability [2], amplifies depressive symptoms [3], and accelerates age related diseases such as Heart Disease [4] and Alzheimer's Disease [5]. It is no wonder why those who work Shift hours are associated with worse health outcomes [6].

Our product hopes to address one aspect of this problem by addressing one of the central issues with the circadian - sleep misalignment, lighting. Light is an important regulator of our circadian rhythms. Natural wavelength light exposure in the morning influences the circadian rhythm by boosting cortisol and suppressing melatonin. Proper exposure to light at wake up shifts circadian clocks forward, helping individuals wake up faster and help fall asleep earlier [7].

The problem facing shift workers, and all those who sleep very late hours, is that they have too much light exposure late in the day and not enough when they wake up, which would make sense for those going to bed just as the sun begins to rise. The current prevailing dichotomy is to either not do anything about the excessive morning lighting during one's sleep, which reduces sleep time and quality, or block out all light from the room during that duration to allow for a full night of sleep.

The latter option creates the issue of no natural morning wake-up signals which will leave individuals groggy and slow to wake up in their dark environment. This is despite having received the benefits of a full night's rest. By the time they are out of bed and begin to introduce themselves to light, enough time has passed to shift the circadian backwards to make prospects of sleep increasingly difficult the following night. Not to mention the unaddressed issue of late-night exposure to light common among shift workers, further amplifying the issue.

1.6 Existing Designs

Format:"Product name: Company"

Sleep Cycle: Sleep Cycle: This product is an alarm that utilizes your phone as a sleep tracker. It uses the microphone and accelerometer on the phone to track sleep cycles. It uses this tracking to determine an ideal time to wake the user up. The upside to this device is no additional hardware is needed besides the phone app. This app does not utilize any additional methods of 'gentle wake up' besides the conventional alarm. Users still need to deal with poor lighting conditions inherent to the type of user who would use this product.

Hatch Restore: Hatch: This product is a sunrise alarm that simulates the sunrise routine of places around the world. It would start the sunrise wake up sequence at a fixed time before the input wake up time, and the setting can also be selected at the phone app. The advantage of this product is that it can provide various simulations of sunrise routines, and the access is easy since it can be easily modified on the phone. The disadvantage of this product is that it is a fixed wake sequence, and it can't track the user's sleep cycle to provide a better wake up window. The user also can't access their sleep data through the night.

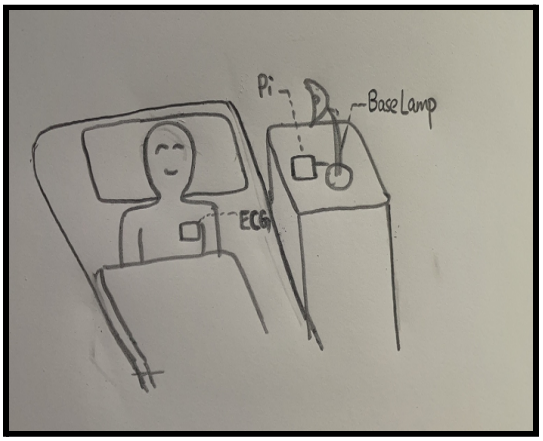
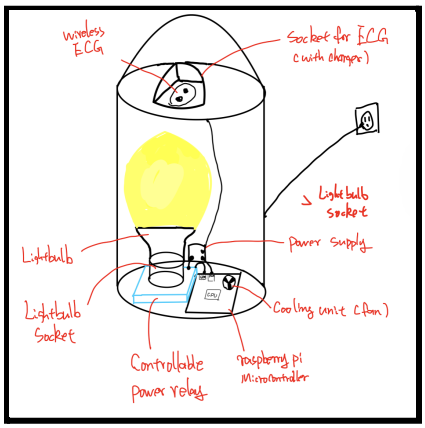
Apple Watch: Apple: This product tracks the user's sleeping condition using a heart rate monitor and accelerometer. It provides the user with an evaluation of their sleep conditions. It's a great product if the data collection and analysis are the major concerns. However, it does not provide a solution to improve sleep quality.

1.7 Sustainability Statement

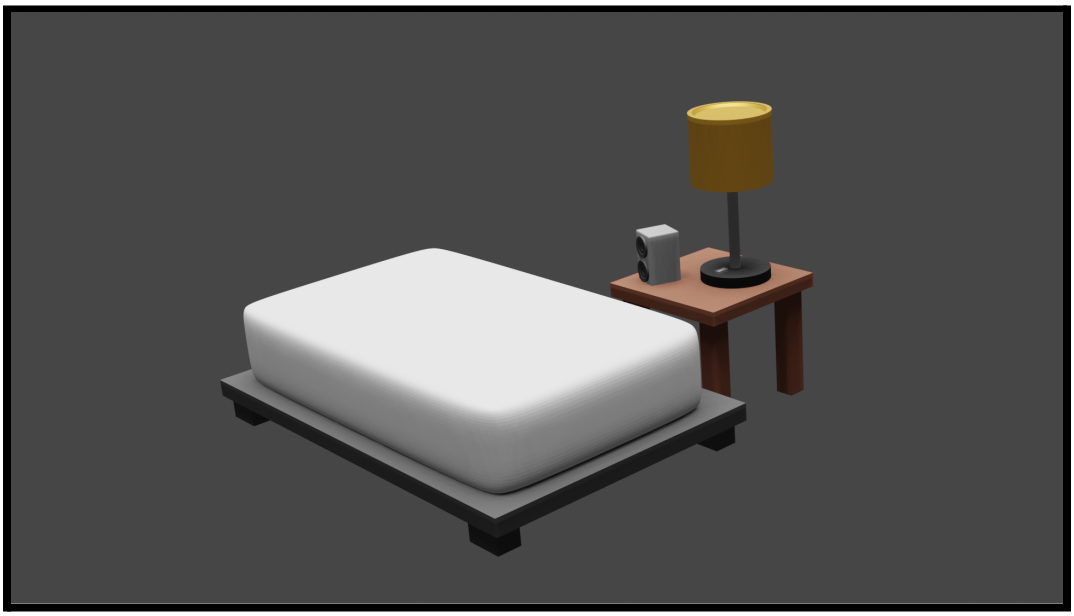
1. We will use recyclable materials/electronics when available and be responsible for keeping our user's data confidential.
2. No persons or animals will be harmed in testing
3. We will not require an internet connection or terms of service agreement for app users, as we consider data collection to be a privacy violation.
4. Ease of use on the consumer side will be a priority, with a simple app interface for quick setup and deployment. The machine should be completely autonomous after setup, and use as little energy (electricity, phone charge) on the consumer's side as possible.

2 Design

2.1 Aesthetic Prototype

Global View	Prototype Light Assembly
	

Drawings: Pre-Development Idealized Prototype



Render: Ideal Prototype

The aesthetic prototype was modeled and subsequently rendered using Blender 2.83 and represents the idealized final design of our system. The bulk of the system is contained within the base of the lamp body, namely the microcontroller and circuit wiring. The reset switch is placed atop the lamp base, and a speaker for the alarm is connected to the main circuit within the lamp but placed outside the housing.

2.2 Design for Manufacture & Assembly

Manufacture

In the interest of ease of prototyping, we have chosen mainly commercial grade products, such as the Raspberry Pi 4b microcontroller rather than printing custom PCBs, as well as using a low-wattage dimmable-bulb in place of creating our own from scratch. If we were to fully flesh out and bring our product to market, we would be creating custom boards and parts. In either case, the componentry is neatly organized into sub-assemblies; i.e. the lightbulb + microcontroller, lamp body and app deliverable.

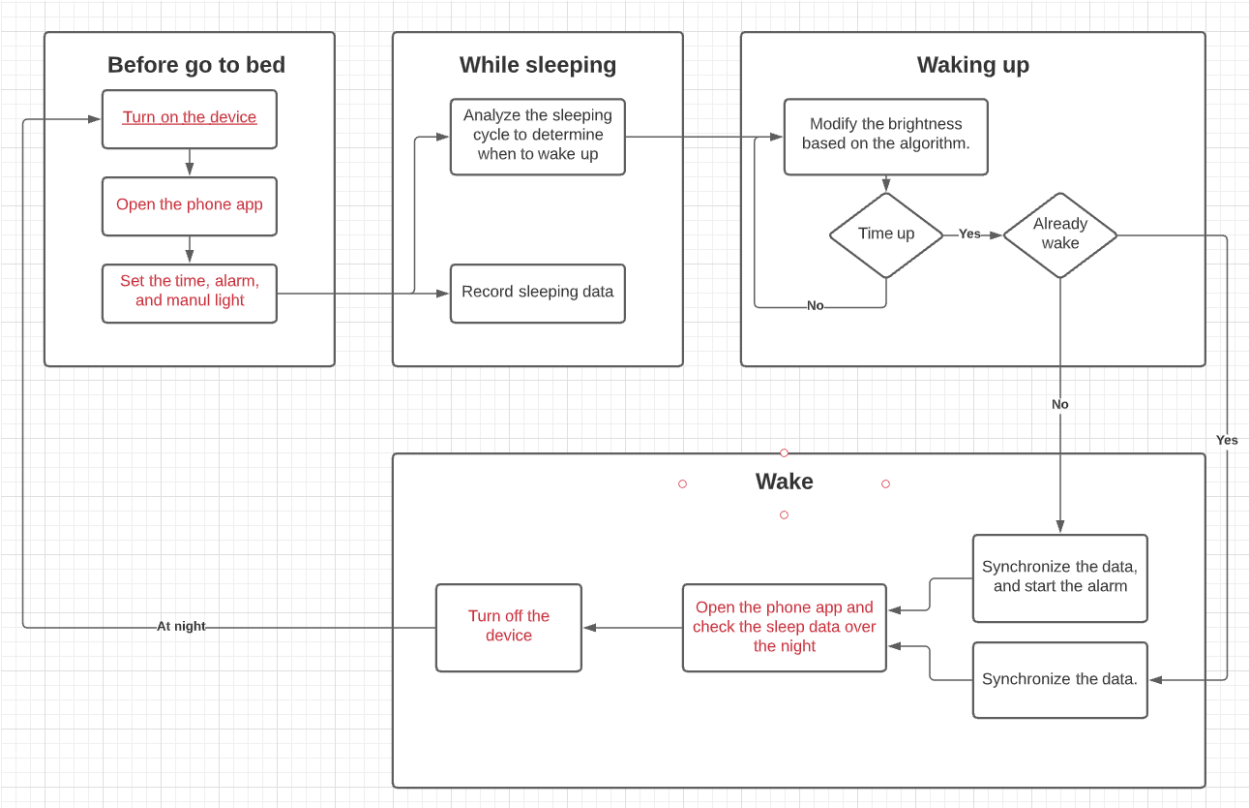
Assembly

Assembly of the device would involve putting together the circuitry by hand, and flashing the Pi with an image of the most-updated code. Once the assembly is confirmed to work, it can be secured into a lamp base as the final product. As of now, there is no realistic way to automate assembly of the prototype. If we moved to custom PCBs, etc.. We could easily automate board assembly and programming, but putting the entire lamp together afterwards would likely either require manual work or an expensive assembly machine.

Maintenance

Unless something goes technologically wrong, the device will not need to be maintained as it is autonomous after the first setup. As electronic components are not perfect, we have ensured that all external parts are cheap and easy to repurchase or replace if damaged. However, as it is unwise to let customers modify electronics, we would require any malfunctioning products to be replaced or maintained by us. The light bulb is an exception, as it can be easily unscrewed and replaced by the customer.

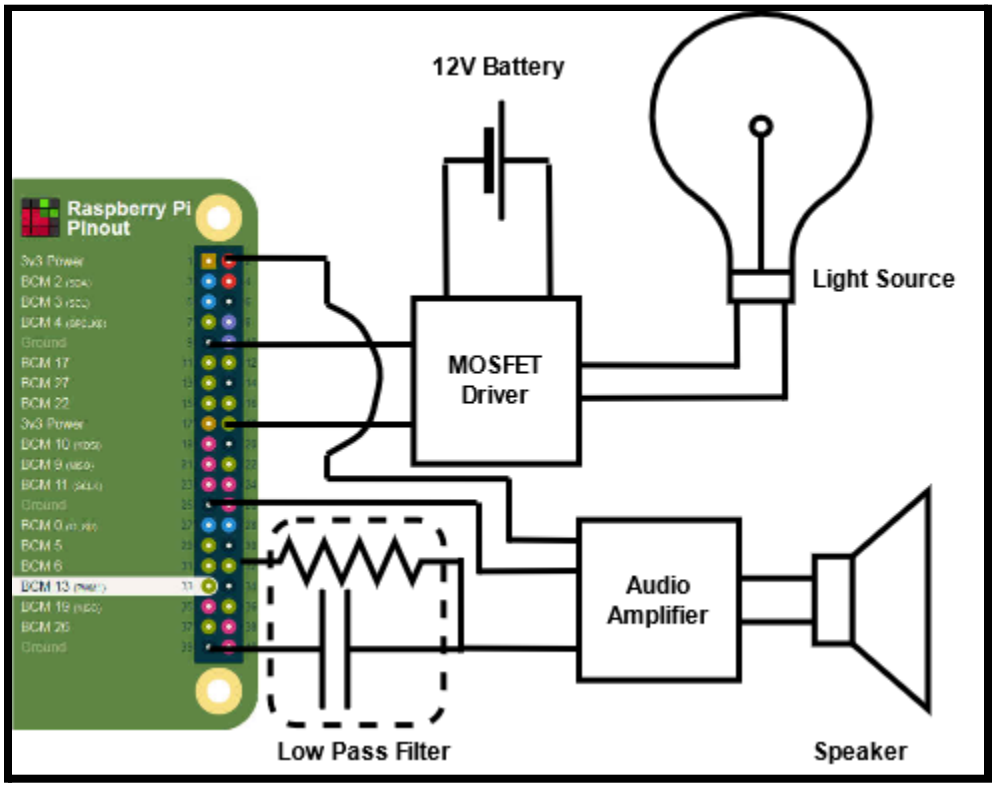
2.3 Block Diagrams



This shows the overall flow of the system for first-time usage. The red font messages indicate the steps needed by the user. The black font messages show the internal logic of the product. This is achieved by running multiple scripts in parallel, such as the data interpretation, light modulation, and phone app server .

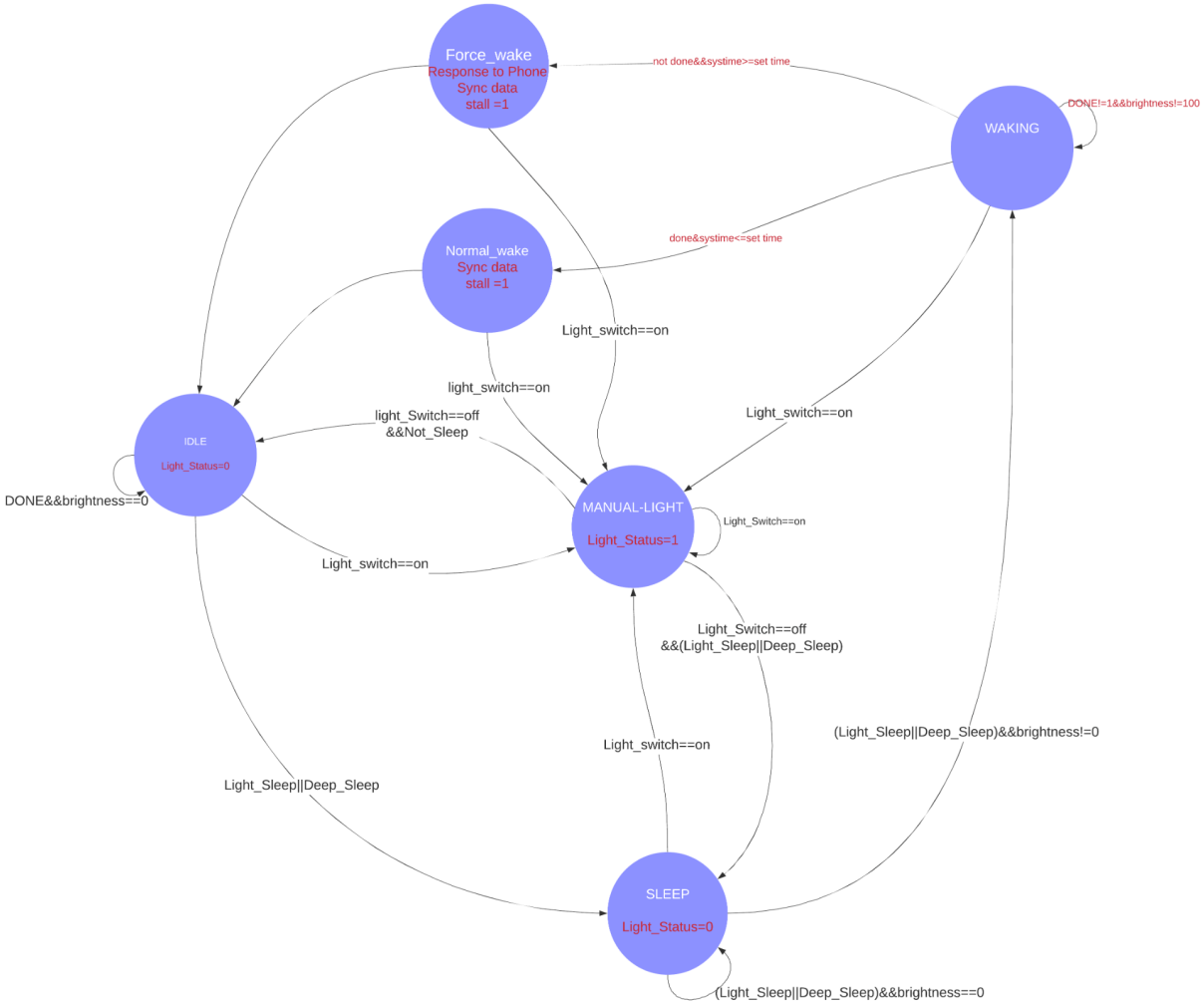
Note: Changing settings through the phone is only needed for first-time setup.

2.4 Wiring Diagram



The wiring diagram shows how the signal from the Raspberry Pi has been processed. The MOSFET switch is used to level up the control signal coming from the Raspberry Ri to power the light bulb with 12V. The low pass filter is used to set a threshold frequency to eliminate high frequency noise for the audio output. The audio amplifier amplifies the output frequency signal to the speaker.

2.5 State Diagram



The state diagram represents how the state machine works on the Raspberry Pi. The basic flow of the state machine would be: IDLE_go to sleep→SLEEP_time to start waking up→WAKING→(awake/sleep at set time)→ NORMAL_WAKE/FORCE_WAKE→IDLE

The MANUAL_LIGHT state can be entered at any time when the user opens that option in the phone app. If the user is still sleeping after turning off the manual light, the state will transition back to sleep to keep monitoring the sleep status of the user.

2.6 Technology

Hardware
For our base station, we chose the Raspberry Pi 4B for its power, ease of programmability, and wireless capabilities. uECG by Ultimaterobotics and MPU6050 are our data collection sources. The uECh is connected to the Pi4 using BLE, and the MPU6050 is wired, and uses the I2C communications protocol. A 12V LED Light Bulb and MOSFET driver are used for light modulation. A resistor, capacitor in a low pass setup, as well as an audio amplifier and 8-ohm speaker are used for tone generation. (Elaborated on in 3.1 Functional Prototype).

App & Pi Controls
Our app was designed in Android Studio using Java. The server on the pi used for interfacing with the app was written in Python with the help of PyBluez [26], an extension of Bluez, the de facto Linux bluetooth library.

Sleep Data Parsing
The data parse algorithm is a perpetual script written using Python 3 utilizing the matplotlib, numpy and sklearn libraries. This program has the role of taking as input a constant stream of data from the accelerometer and outputting a string tuple containing the sleep state (either Deep or Light) and the current intensity of the light on the range [0, 100]. In previous iterations this algorithm was linked with the uECG but due to issues with reliability in data transfer we swapped from the wireless uECG to a wired accelerometer with near 100% reliability.

Backend
State Machine:
The sleep data has been transmitted to the state machine, and the state machine takes control of the actual working state on the Raspberry Pi. All State Machine programs are written using C language. Most of the settings are done using the file input/output feature. Other parts like the bluetooth communication program and the sleep data parsing program will update files with specific names, then the state machine will read or write to these files based on needs.

Startup Script
In order to facilitate ease of use a startup script was written which will automatically run upon turning on the Raspberry Pi and proceeds to start all necessary programs on the Pi for the system to run. First, a receive server is started which will accept feedback from either the ECG or accelerometer and dump this data to a file which the data parser can interpret. Then a bluetooth server is established to allow the user to change preferences using the app and set their wakeup time should they desire. Next, the state machine is both compiled and run; it now waits to accept a connection from the data parse program. Finally, the data parser is run which reads settings from several configuration files and links with the state machine on a local socket to facilitate

their communication. With this the system is prepared for the user to go to sleep and will appropriately awaken them as per their specification in the app. When the user wakes up in the morning they will turn off the lamp/alarm using the reset button, which will finish dumping the sleep data to Pi which can be synced with the Android application to allow the user to view their biometric sleep data with our own analysis. The Pi is finally restarted, preparing the system for another sleep cycle.

External hardware manipulation:
MOSFET switch & Light bulb:
Level up the voltage of the control signal to provide valid voltage to manipulate the brightness of the light bulb.

Amplifier & Low pass filter & speaker:
The low pass filter eliminates frequency that is not needed for our user from the signal (noise). The amplifier also levels up the audio signal to operate the speaker.

2.7 Simulation

Simulation of the system was broken into several distinct tasks. In order to simulate real sleep data for more validation testing, we created the algorithms “fake-data” and “fake-data-v2” to output CSV files of data spanning 8 hours. This data was either in the form of heart rate variability vs time or acceleration vs time, although as we iterated through the design process we ultimately decided that the acceleration data was more reliable and useful and thus limited the HRV data in favor of the superior accelerometer data.

We modified our data interpretation program to include a “simulation mode” in order to process these datasets with custom timing. These two developments allowed us to test our software in isolation for variable intervals of time, identifying common trends in the data to improve our “predictive wait” algorithm. This allowed us to discover and rectify several key bugs with the data parser, namely an issue that would detect falling asleep as awakening and thus almost immediately triggering the light. We were able to implement restrictions on the wakeup window which set a boundary for the wakeup interval to disallow this behavior.

2.8 Modeling

Although the overall design that was agreed on in our decision tables had not been changed, there were a few replacements in technology as the project progressed.

1: Method of light control
Problem:
Our original plan to simulate sunrise was to buy a dimmable lightbulb and connect it to a commercial relay, which would be controlled through I/O pins on the Pi.
It was discovered that, while we could successfully control the relay from the Pi, the relay itself only functioned as an On/Off switch, contrary to its advertised specs.

Solution:
After fruitless searching for a commercial relay or lightbulb that would satisfy our needs, we decided to implement a dimming mechanism from scratch. This circuit, as seen in our wiring diagram, uses a MOSFET driver which is controlled through a PWM signal by the Pi. The resulting system was understandably not as bright, but it was safer and easier than manually modulating a 100W Bulb.

2: Method of data collection

Problem:
uECG, the product we ordered for its open source, wireless data collection, began to shut off or stop transmitting data during the night.

Solution:
We replaced the entire uECG system with a simple accelerometer. After testing and analysis, we determined that the accelerometer was not only much more accurate and easier to implement, it was about $90 cheaper.

3: Method of manipulating the brightness:

Problem:
In the initial design, the modification of the brightness is controlled by the states and input to the state machine in the state machine program. However, brightness can be controlled more ideally in the data interpretation algorithm.

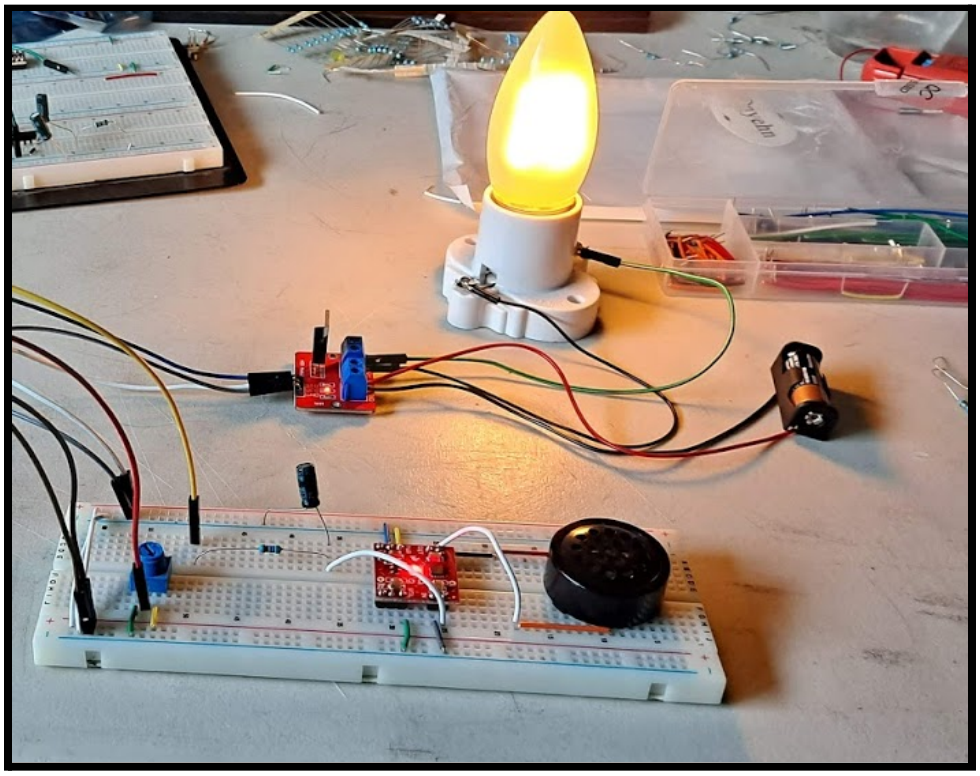
Solution:
We modify the previous complicated state machine to a simple state machine that can reliably control all parts and communicate with the system. The brightness decision moves to the data interpretation program, and the state machine controls brightness based on the output from the data interpretation program.

3 Evaluation

3.1 Functional Prototype

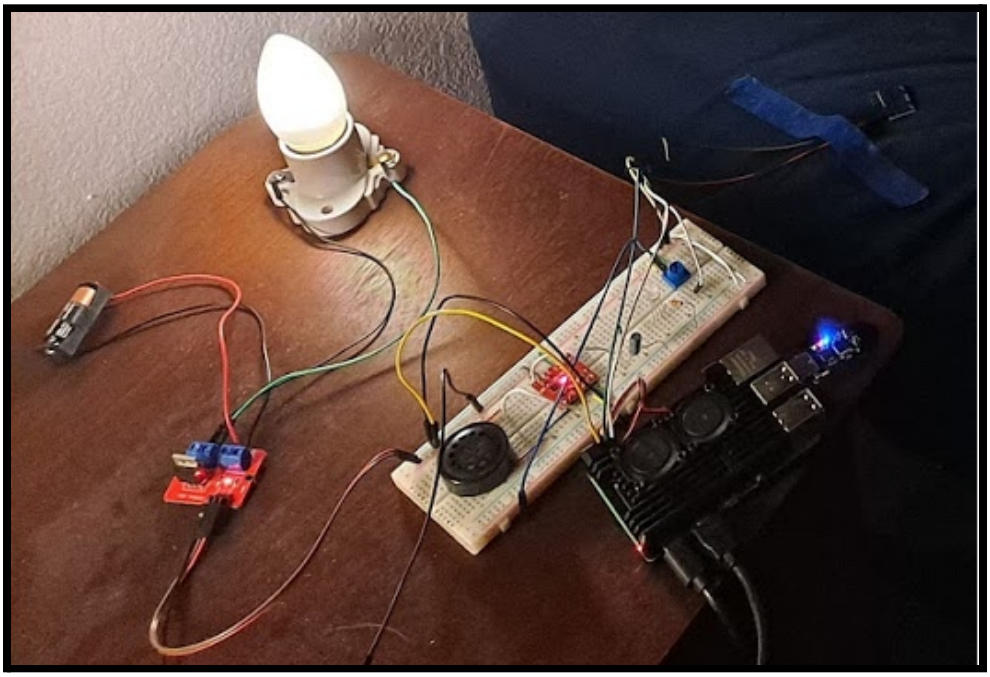
The functional prototype of our project includes three main components: a lamp/speaker module that perform the actual waking signal to users, a raspberry Pi4 which perform all the computations and act as a communication platform for connecting up all three components, and the third component is our dedicated phone App/pi server interface, where users can setup the alarm, view sleep history, and manually turn the light ON/OFF.

Lighting: Pre-Aesthetic prototype



In picture: Alarm speaker (black)

Final Pre-Aesthetic Prototype: All Components



Light Module

The light module, shown in the upper image, consists of a 12V LED Bulb, a 12V battery and holder, and a MOSFET Driver controlled by the Pi4. The MOSFET connects the 12V battery and the LED. It operates as a switch, controlled by a 3.3V PWM signal from the Pi. The brightness of the duty cycle is relative to the PWM duty cycle.

Speaker

The prototype speaker includes an 8-ohm speaker, audio amplifier, and low pass filter. A PWM signal is sent from the Pi4 through the low pass filter and into the audio amplifier. The amplifier boosts the signal and feeds it into the 8-ohm speaker. The low pass filter eliminates any noise from the system larger than the cutoff frequency of 700 Hz. The cutoff frequency is set by a resistor / capacitor combo. This cutoff frequency was chosen because it is high enough to where any usable tone is still available, but low enough to not be overly inclusive of high frequencies.

Raspberry Pi 4

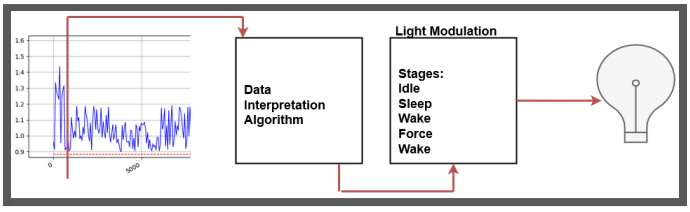
The Pi4 controls the speaker and light signals, as well as runs the state machine and data collection and interpretation. Because there is a large amount of processing occurring while the system is running, and because the system will run continuously for hours on end, we also attached a heatsink / fan combo to help keep the Pi4 cool throughout the night.

Accelerometer

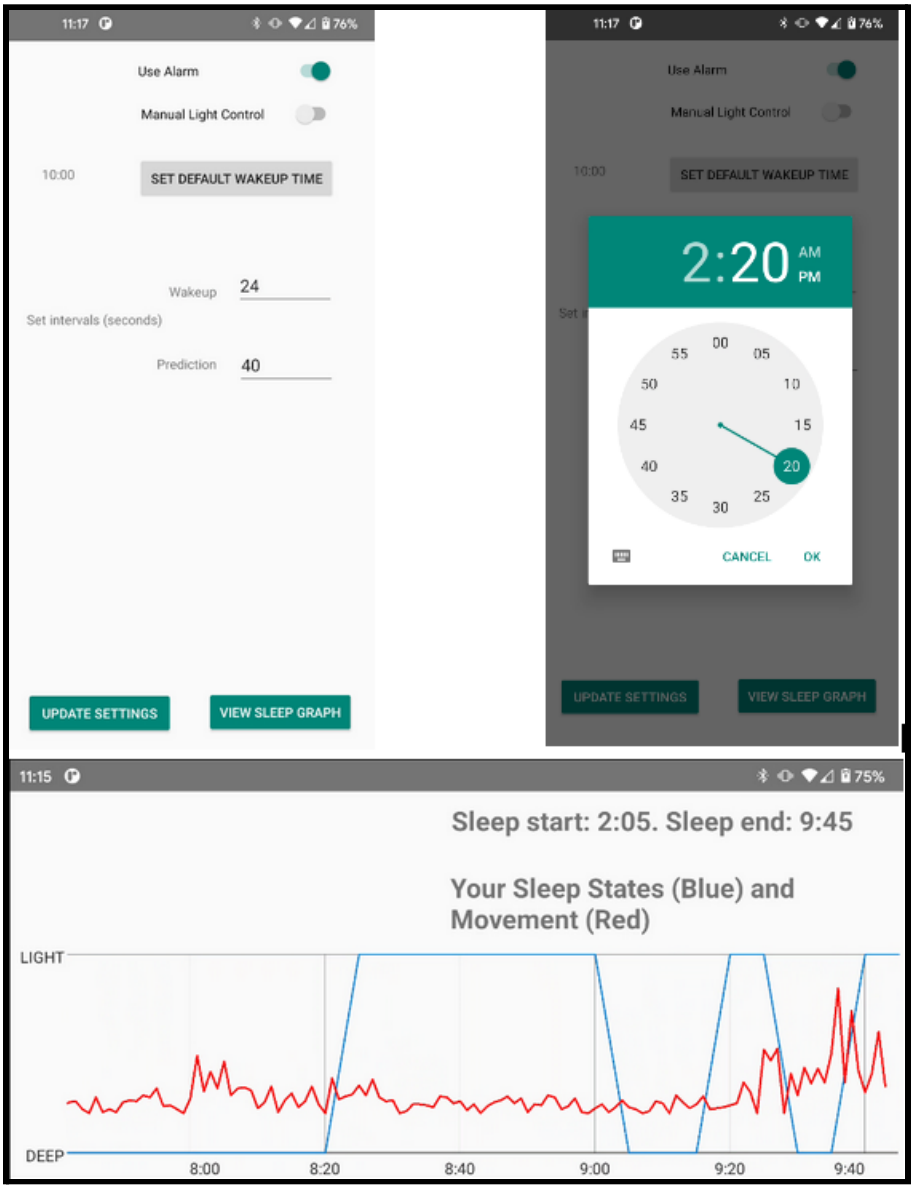
Our functional prototype relies on a bed attachment wired accelerometer. Originally we did heart rate and accelerometer data collection using a bluetooth uECG module, but due to its inconsistency, moved on to the very reliable wired accelerometer for the prototype. The accelerometer is attached to the pi via four cables. Two for ground and power, and two for I2C communications. The accelerometer's data is collected using a weighted moving average. Every 15 seconds, the average of the data is collected and weighted using a logarithmic function, and passed onto the data interpretation algorithm.

Data Interpretation and State Machine

The data interpretation algorithm processes the incoming data in real time and parses out deep and light stages of sleep based on increased and decreased movement activity. It sends control signals to the state machine, which in turn controls the lighting and speakers. The interpretation algorithm was written in Python 3 utilizing the matplotlib, numpy and sklearn libraries. This algorithm uses a bucket system with running averages to determine the sleep state of the user and to find where they swap between NREM, Light and Deep sleep. Data is input to the system via reading from a dump file every 15 seconds, adding all new points to the current dataset and then clearing the file for subsequent reads and writes of new data. As data points are parsed, new averages are computed and compared with those previously calculated, if there is a strong enough difference between the previous and current average a change in the sleep state is triggered. Whether or not the sleep stage has changed, once every 10 minutes while asleep the algorithm writes the current sleep state to a log file and sends it to the state machine. Once enough time has elapsed the program enters the prediction window where it will look for volatility within a light phase of sleep which indicates a transition to non-REM sleep; the ideal time to start the light system and begin to awaken the user. If such a point is found, the light will begin to turn on following a linear pattern increasing the intensity until the manual wake-up time. If the prediction window has elapsed without these conditions being met, the signal is manually triggered in the code and the light will begin to be modulated, reaching max brightness when the wakeup time is reached and the alarm will play through the attached speaker. While the light is being modulated, the state and intensity of the light are updated once per minute, logged, and sent to the state machine to control the PWM of the light. Finally, the system is reset when the user toggles the lamp and is prepared for another night of sleep.



Android App (for server, see Testing 3.2)



Top: Home screen after connection. Displays settings for changing use of alarm, manual light control, default wakeup time, as well as intervals for letting the algorithm predict the change of sleep states.
Bottom: Screen is displayed after pressing “View Sleep Graph”. Both sleep logs are then transferred to the user's phone from the Pi. The graph is able to be stretched for increased detail.

3.2 Testing

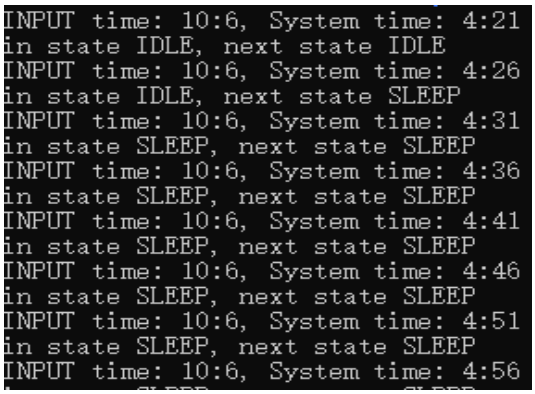
Data Scrubbing

The data collection from our collection apparatuses were gathered with moving averages every 15 seconds. The accelerometer data, whether from the uECG or wired accelerometer, needs to be weighted to be usable. Because of null offsets and gravity,even at rest the accelerometer never reads 0 or 1G. What we care about in our data collection is movement, so the program is actually recording the jerk. If the device is still, it should measure 0, and if the device begins moving, it will record a non-zero value. In addition, the jerk measurement is weighed using a logarithmic function. This provides extra emphasis on the smaller movements, which was shown in testing to be very important in identifying deep sleep stages. The exact weights within the log function were adjusted to where all expected values sent to the interpretation algorithm would fall between 1-10. Because it is a natural logarithm, a 5 on the scale would be a factor of e (~2.718) greater than a reading of 4.

Light Modulation State Machine

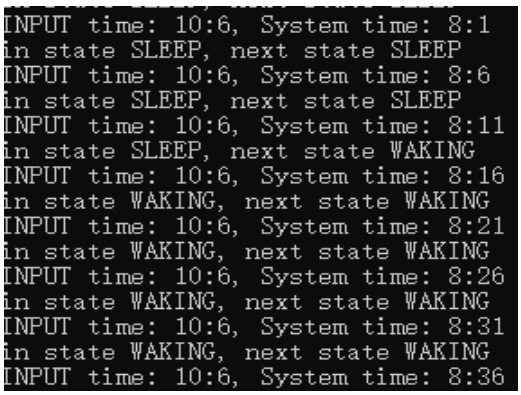
The state machine is tested by observing the response of the input data files and configuration files. The response is given every 5 mins with an indication of the state transition, the time, and the desired wake up time.

Falling asleep:



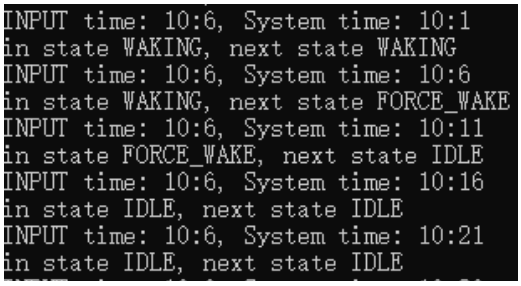
If the user is still awake, the state will stay in the IDLE state until the user falls asleep. When the user falls asleep, the state will transit from IDLE to SLEEP, and the sleep status will be monitored at the SLEEP state.

Start waking sequence:



While the user is sleeping, the data interpretation algorithm will keep analyzing the sleep stage of the user, and based on the current time, the algorithm will give the state machine an input to indicate a proper time to start the wake up sequence. The state will be transiting to the WAKING state, and in that state, the brightness of the light will be modified.

Force wake up the user:

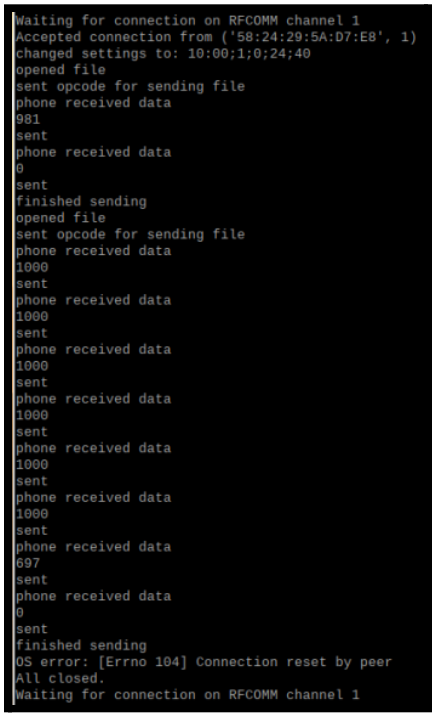


When the time is running out, the state will transit to FORCE_WAKE if the user is still asleep, and the state will transit to NORMAL_WAKE if the user is awake already. In the left Figure, the state transit from WAKING to FORCE_WAKE to start the alarm. The state will transit back to IDLE after that.

Result:

The detailed test schedule is in the Appendix A-3.2. The result is promising. The state changed correctly with the generated file from the data interpretation algorithm. With the hardware setup, the brightness of the light was modified correctly. The alarm could be started with the corresponding case. This test information comes from simulation. The clock should be system time, but since the real test is very time consuming, I wrote an accelerated time that one second in the real world represents 5 mins in the simulation.

Pi Server



Console log in order of server events:

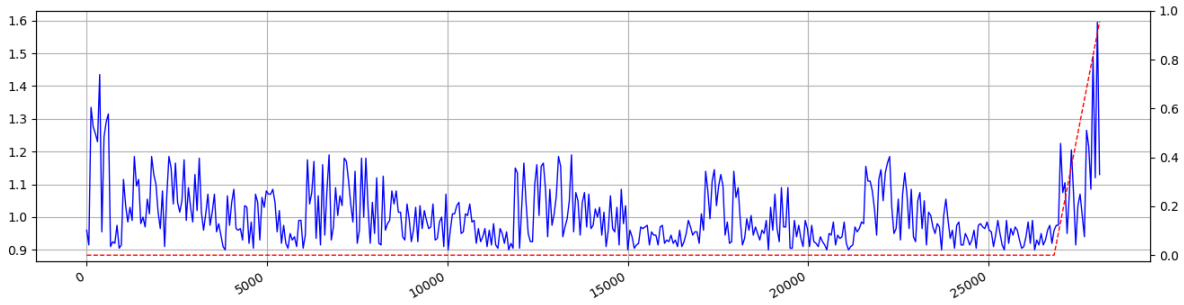
1. Connection opened
2. settings changed
3. latest sleep data files transferred
4. connection loss
5. Reset socket and wait

This is a continuously running bluetooth server that acts as an interface between the user's app and the pi.

Upon receiving a connection from an app client, the server waits for commands:

- **Settings command**. The pi writes the entire received line to the config.txt file, which is parsed by our data interp algorithm on startup. A confirmation code is returned.
- **File transfer command**. The server returns a confirmation code, then proceeds to send both the sleep state log and acceleration log files in chunks of 1000 bytes with opcodes indicating progress. Sending is staggered, as Android's Bluetooth module cannot handle large continuous streams of data.

Simulation with Data Parser



This is a graph output by the data parse algorithm after a successful simulation run. The blue line represents the input data points from the fake acceleration data with the left y-axis showing the corresponding values. The x-axis represents time in seconds that elapse as the simulation progresses. The dotted red line represents the intensity of the light at any given point in time, and the value on the range [0, 1] representing completely off to max brightness is shown on the right y-axis.

3.3 Cost

From the beginning, overall cost for the prototype was not a major concern. After switching from a uECG to a simple accelerometer, our total cost fell from $181 to $81 for the entire prototype. Similarly, moving from a commercial-grade lightbulb to a simple, lower-voltage bulb made the design cheaper and more feasible.

Parts	Price	Quantity	Cost
Lamp Base	$30	1	$30
Raspberry Pi 4B (2GB)	$35	1	$35
uECG [Optional]	$100	[1]	[$100]
Lightbulb	$5	2	$10
Accelerometer	$3	1	$3
MOSFET	$0	1	$3
Grand Total			$81 / [$181]

Appendix

A-1 Problem Formulation

A-1.1 Conceptualizations:

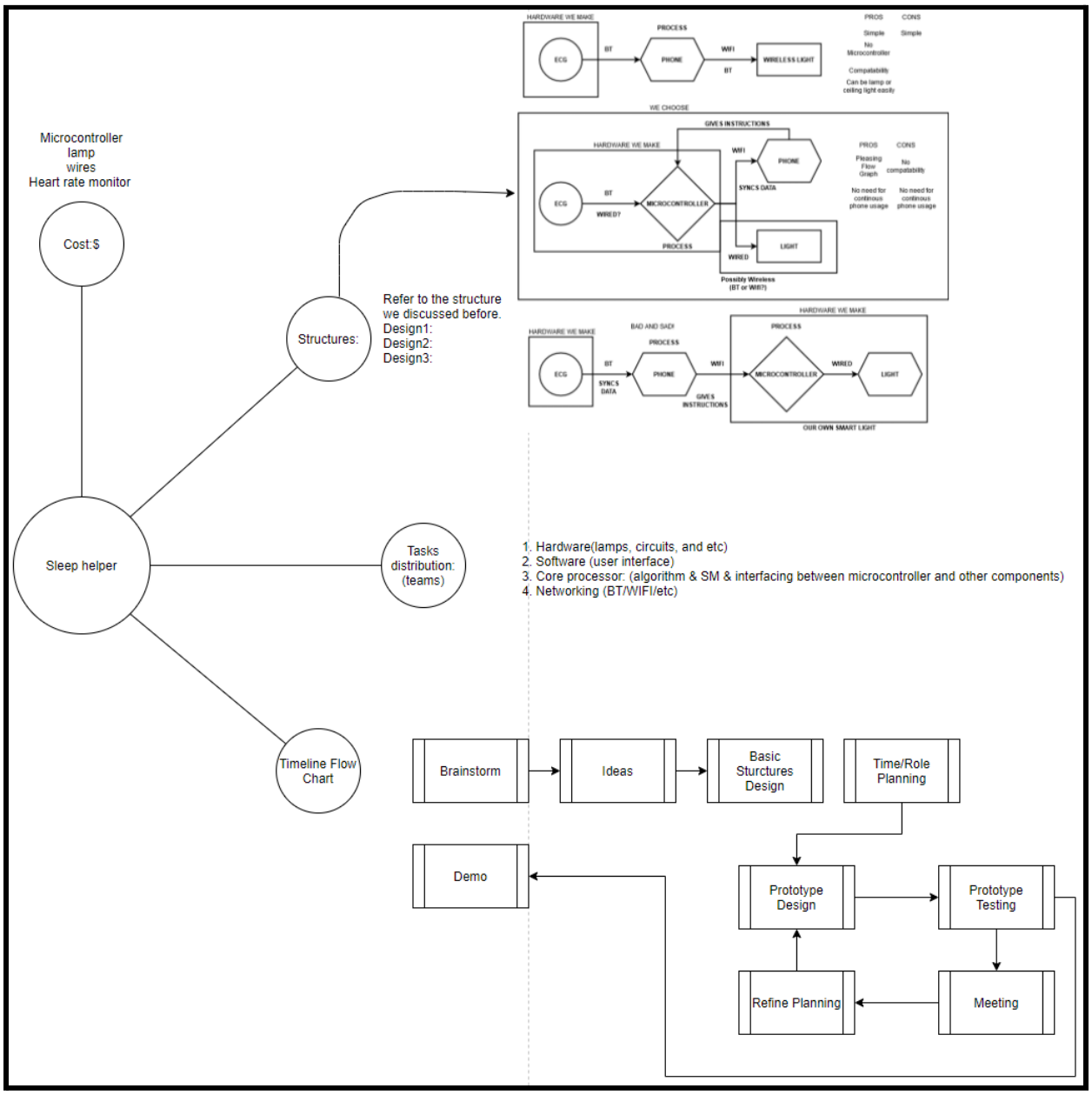
Dawn Light:

- End devices & Interface between the end device and the microcontroller (between user and device):
    - Via computer
        - Computer Apps
        - Command line
    - Via Phone
        - Android
        - IOS
    - Connection to the controller:
        - Bluetooth
        - WIFI
        - Wire connection
- Data processing - Location of data processing:
    - On end device:
        - Programming language: Java, C, Python, etc
    - On microcontroller:
        - Programming language: C, Java
- Hardware for collecting data for examine the sleeping quality:
    - Options:
        - ECG:(preferred)
            - Wired: Pulse Sensor - openBCI ~24.99 [20]
            - Bluetooth/WIFI: uECG (tindie) ~$7 [21]
        - Accelerometer:
            - Wired
            - Bluetooth
            - WIFI
        - Microphone:
            - Wired
    - Location to receive the collecting data:
        - ECG/Motion detector:
            - End device:
                - Data interpretation on the end device. (major processing duty)

                - Microcontroller:
                    - Data interpretation on the microcontroller. (major processing duty)
            - Microphone:
                - Microcontroller.
- Microcontroller:
    - Tasks:
        - If major processing duty is on microcontroller:
            - Process the data from (3) for analyzing the sleep cycles.
            - Running the state machine and algorithm.
            - Control the lamp.
        - If major processing duty is on end device:
            - Process data from (3) for analyzing sleep cycles on the end device.
            - Run algorithms on the phone.
            - Send instructions to the microcontroller\ lamp to control the lamp.
    - Microcontroller options:
        - PSoC:
            - Available for formal CSE 121 students.
            - Bare metal programming.
            - Bluetooth available
        - Raspberry Pi:
            - Available for formal CSE 121 students.
            - More functionalities can be achieved.
            - WIFI/ Bluetooth available
        - Arduino Uno:
            - WIFI function available
- Lamp:
    - Tasks:
        - Mimic the change of brightness based on the received instruction.
        - Controllable power relay for controlling the voltage for the lamp.
    - Hardware options:
        - Wired lamp designed by hardware team:
            - Need to have wired connection to the microcontroller.
            - No need for internet or bluetooth connection.
        - Bluetooth/WIFI based lamp: (more work for software team)
            - Controlled by either the phone or microcontroller.
            - May need internet/ bluetooth connections.
- WIFI/ Bluetooth networking:
    - Task:

- Define a protocol for transmitting data between the ecg and phone, and between the phone and microcontroller.
    - Reliability
    - Data Integrity
- Maybe sync with the Internet as well? Could be a stretch goal
    - Establish reliable WIFI/ Bluetooth channel for transmitting data among end devices, microcontroller, and data collecting devices.

A-1.2 Brainstorming



A-1.3 Decision Tables

For more detailed item descriptions, see decision table inputs: 8.2

Macro-design Table

Criteria	Weight	Design 1		**Design 2**		Design 3	
		raw	weighted	**raw**	**weighted**	raw	weighted
Autonomy	40	3	120	**9**	**360**	3	120
Cost	30	10	300	**5**	**150**	5	150
Ease of use	10	6	60	**7**	**70**	6	60
Versatility of implementation	20	3	60	**9**	**180**	5	100
Outcomes			540		**750**		430

Microcontroller Table

Criteria	Weight	PSoC 6		RPi		Arduino	
		raw	weighted	raw	weighted	raw	weighted
Power Consumption	5	8	40	3	15	4	20
Cost	20	10	200	4	80	8	160
Familiarity	10	8	80	7	70	6	60
Memory	5	3	60	9	180	5	100
Versatility	40	2	80	8	320	6	240
Ease of Use	20	9	180	7	140	7	140
Outcomes			640		805		720

Wi-Fi vs Bluetooth for ECG comms (assuming both are possible)

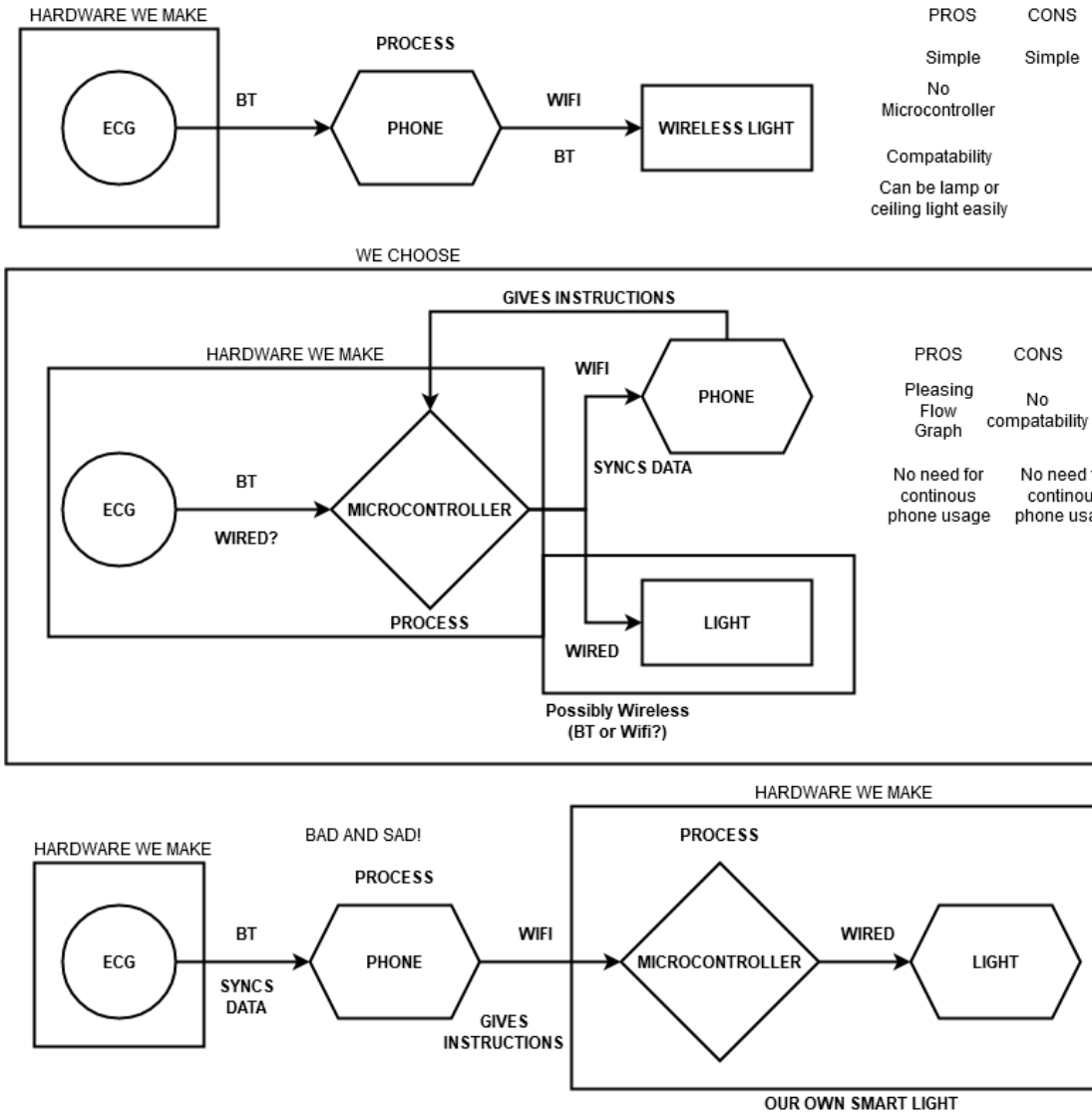
Criteria	Weight	**BT**		Wi-Fi	
		raw	**weighted**	raw	weighted
Ease of Implementation	20	**7**	**140**	8	160
Energy Cost	40	**8**	**320**	5	200
External Dependencies	10	**10**	**100**	7	70
Comms Reliability	30	**6**	**180**	7	210
Outcomes			**740**		640

Smart light (wifi/bluetooth) vs diy smart light (hardware hookup)

Criteria	Weight	Smart light		**DIY**	
		raw	weighted	**raw**	**weighted**
Ease of Implementation	20	10	200	**8**	**160**
Customizability	40	5	200	**8**	**320**
Wifi independent	40	3	120	**10**	**400**
Outcomes			520		**880**

A-1.3 Decision Table Inputs/Elaboration

Macro Designs:



- Microcontrollers**:**
  - PSoC 6: Chosen for convenience [23].
  - Arduino: Chosen for ease of programmability & documentation [24].
  - Rpi: Chosen for convenience, also linux capabilities & state-saving [25].

- Lights**:**
  - Smart-light: Commercial grade smart-light with dimming and API capabilities. Bulb optional in final product.
  - DIY relay: Will use wired hookup to pi, with commercial-grade relay for prototyping. Light Bulb and base-station included in the final product.

A.1.4 Morphological Chart:

Components/functions	1	2	3	4
Wake Up Ways	Alarm	Changing light +reminder	Changing light	vibration
Collecting data for examine the sleep cycle	Wired heart rate monitor	Wireless ECG - node.js on pi	Microphone - wired	Motion detector wired
Controlling brightness of lamp	Software (wireless lamp)	PWM (Wired lamp)	Analog control (Wired lamp)	
Interface with user via end devices	Computer	Android App	IOS App	web service
Connection between End device and Microcontroller	Wired	Bluetooth	WIFI	
Controlling the lamp/other ways of wake up	Wired	Bluetooth (smart device)	WIFI (smart device)	
Location of major processing	End device	microcontroller		
Microcontroller options	Raspberry Pi	PSoC	Arduino	
Network Protocol	TCP	UDP	Modified TCP/UDP	

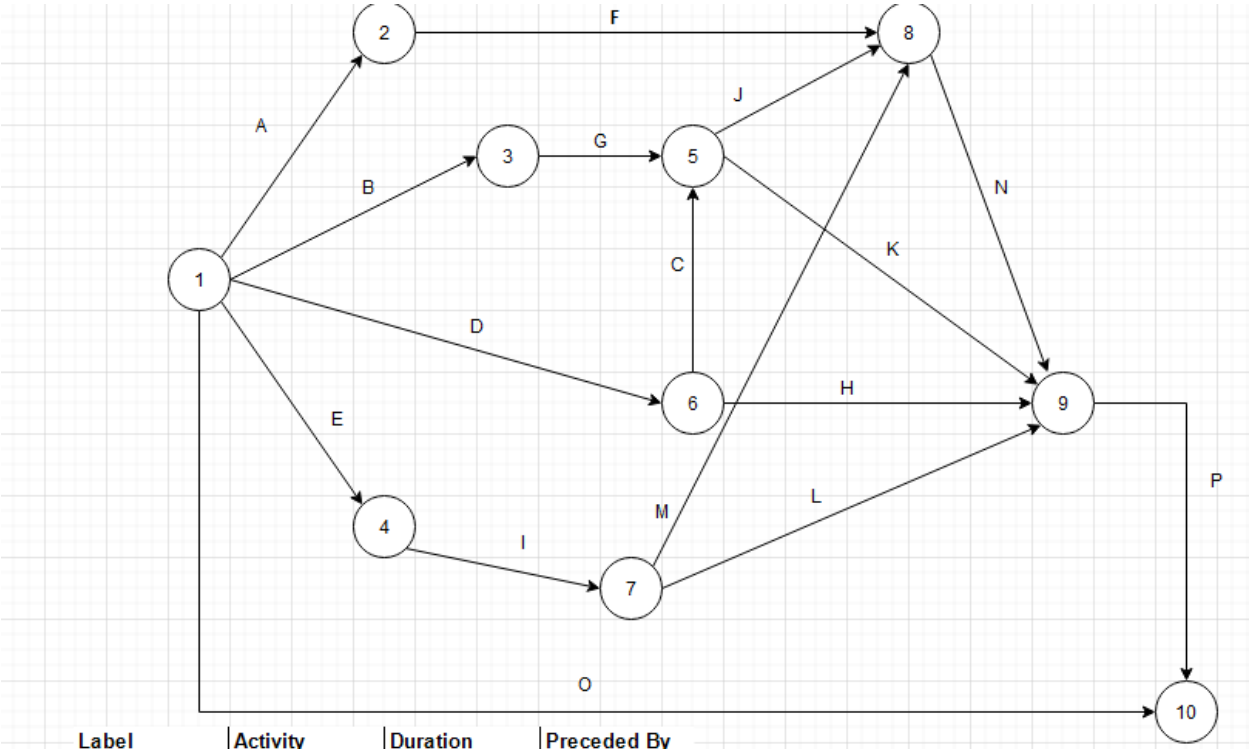
A-2 Planning

A-2.1 Plan

WBS NUMBER	TASK TITLE	TASK OWNER	START DATE	DUE DATE	DURATION (days)	PCT OF TASK COMPLETE
1	**Planning and Design**					
1.1	Timeline Creation	group	2/3/21	2/5/21	2	100%
1.2	Part Selection	group	2/3/21	2/12/21	9	100%
1.3	Design Formalization	group	2/3/21	2/13/21	10	100%
1.6	Materials Acquisition	group	2/3/21	2/27/21	24	100%
2	**Early Prototyping and Design**					
2.1	Basic Data Collection	Group	2/21/21	3/6/21	15	100%
2.2	Data Generation Algorithm	Jacob	2/21/21	4/17/21	56	100%
2.3	Microcontroller State Machine	Andy and Mingyu	2/21/21	4/10/21	49	100%
2.4	Basic Data Interpretation	Nick	2/21/21	4/17/21	56	100%
2.5	Basic Network Interface	Steven	2/21/21	3/6/21	15	100%
2.6	Learning to make Android Apps	Weilin and Steven	2/21/21	3/20/21	29	100%
2.7	Light Modulation	Andy and Mingyu	2/21/21	3/20/21	29	100%
2.8	Update Plan / Timeline	Group	3/19/21	3/20/21	1	100%
3	**Integration and Betterifying**					
3.1	Live Data Transfer	Steven	3/28/21	4/24/21	26	100%
3.2	Taco Night	Group	3/29/21	5/1/21	32	0%
3.3	Dynamic Data Implementation	Nick/ Jacob	3/30/21	5/15/21	45	100%
3.4	Advanced Light Modulation	Andy/Ming	3/31/21	5/8/21	38	100%
3.5	Android App	Steven/Weilin	3/28/21	5/15/21	47	100%
3.6	Aestetic Protoyping	Nick	5/17/21	5/28/21	11	100%
3.7	Update Plan / Timeline	Group	5/24/21	5/24/21	0	100%
4	**Optimization and Delivery**					
4.1	Test & Optimize Design	Group	5/8/21	5/24/21	16	100%
4.2	Documentation Finalization	Group	5/17/21	6/4/21	17	30%
4.3	Presentation Practice	Group	5/24/21	6/4/21	10	0%

We formulated a basic outline of our progress using a Gantt chart. As the project moved forward and we were more capable of finalizing our chart, we assigned more concrete timelines and members to each task. Our overall progress was not fulfilled in top-down order, as a few tasks: data interpretation, backend, and the app ended up all being developed in tandem from beginning to end.

A-2.2 Critical Path Method



Label	Activity	Duration	Preceded By
A	Create State M	1 week	-
B	Mock Data	1 week	-
C	ECG Collect	2 weeks	D
D	Network Design	2 weeks	-
E	Learn Andoid	3 weeks	-
F	Control LEDs	<week	A
G	Basic Analysis	1 week	B
H	App Comms	4 weeks	D
I	Design Function	2 week	E
J	Fnct Anlys	1 week	BGCD
K	Alg Refine	3 weeks	BGDC
L	Create App	1 week	EI
M	Impl Sett	2 weeks	EI
N	LB Relay	4 weeks	BGDCJAFEIM
O	Aes. Proto	2 weeks	-

Notable Events

8 Software Prototype

9 Functional Protoype (Class Goal)

10 Final Prototype

This was our initial Critical Path estimate. Although accurate for the most part, there were a few steps that were completed out of order:

- N-LB Relay: This was able to be developed independently of the other steps, only needing an independent script to test functionality
- H/L/M: These were all developed in tandem as part of the total app design. Settings were easy to change, since we only needed to modify some buttons and change the format of the text being sent back and forth.

A-2.3 Division of Labor - Prototyping

Task:	Name:
ECG & Accelerometer data collection	Jacob & Nicholas
Bluetooth App & Microcontroller comms	Steven
Android app	Steven & Weilin
State machine & Hardware	Zhenyu & Mingyu & Jacob

A-2.4 Collaboration

A Github Repository was used for all code, with a main branch for official progress and side branches for experimentation. Roles were initially assigned based on each member's individual specialities. As it became apparent that certain tasks required different levels of effort, the roles became less defined and each member would fill in work based on urgency and skill-set.

A-3 Test Plan & Results

A-3.1 Test Plan - Critical function

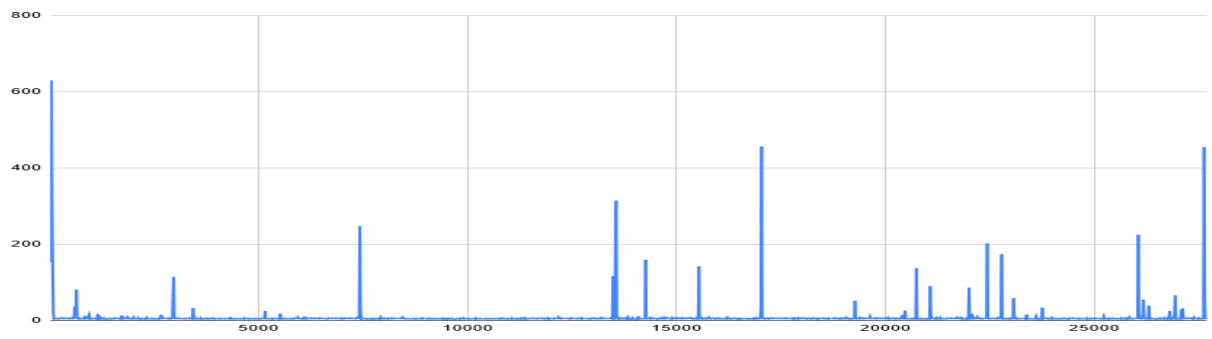
1. BPM Accuracy: Test if ECG has the correct data for BPM, using a traditional blood pressure monitor to check with ECG input.
2. Movement Accuracy: To test if the project is able to detect the movement, and use it for determine the stage of a sleep cycle
3. Equipment Reliability: To test if the ECG is able to correctly mount on the user's chest during the night, no shift in place.
4. Magnetic: To test if the project is able to work correctly under magnetic conditions, such as close to a wireless charger.
5. High Temperature: To test if the project is able to work correctly under different normal temperatures from 32F to 100F
6. Durability: To test if the lamp is able to handle moderate usability without breaking. If the connection between the microcontroller and LED lamp is well connected. The ECG should be able to withstand stomach-sleeping
7. Continuity: To check if the product functions properly in the case of temporary ecg removal during the sleep period.
8. Cost: If the product can be made at a certain cost for profit.

A-3.2 Test Results - Critical

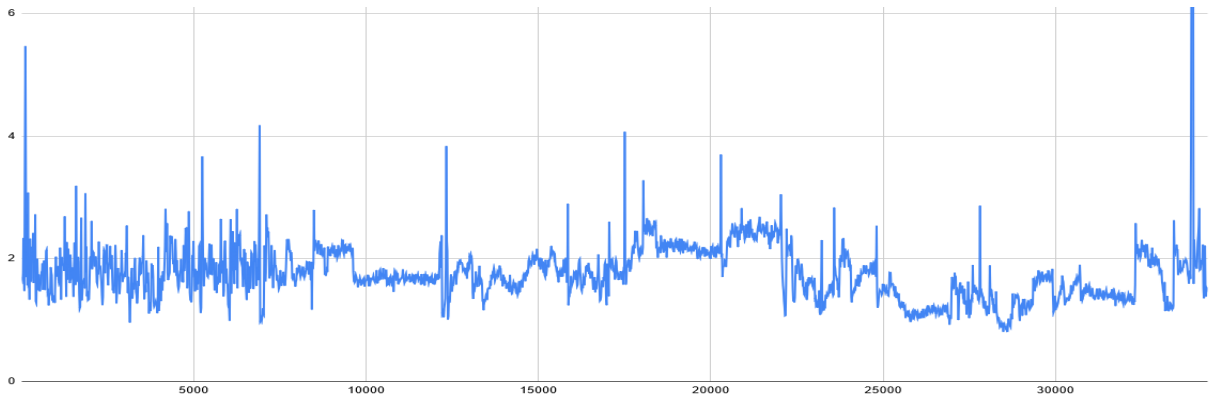
Objective	Result	Discussion
BPM (less or equal to 5B/Min)	Yes (conditional)	If not accurate enough, will use BPM in parallel with movement.
Movement(Able to detect user movement)	Yes	Both uECG and Accelerometer met this requirement.
Equipment(Able to attach to user all time)	Yes	Both uECG and accelerometer met this requirement, with the accelerometer placed next to a pillow.
Magnetic (Able to sustain moderate magnetic condition)	Unsure	Test results inconclusive
Temperature (32-100 F)	Yes	A good product should not interfere with outside environment
Durability (Connection secure between PI and LED)	Yes	If contained in a lamp-base with proper cushioning, no issues will arise.
Continuity (Error-Proof operation without the ECG)	Yes	The timescale will be determined by the microcontroller, so we will need to decide what should happen if removal occurs.
Cost (Profit)	Hardware Prototype: ~60 (no	Likely impossible in the scope of the project.

A-3.2 Test Results - Misc

Data Collection		
Time:	Testing target:	Date, Comments
uECG: short, 2 minute, detect minor but relevant movements	pass	4/13
uECG: short 2 minute, small to no signal for very insignificant moments	pass	4/13
uECG: long 5 hours: maintain connectivity, maintain 50% of battery life	possible fail battery life, connectivity fail (lost connection after about 3 and a half hours)	4/14
uECG: long 8 hours: maintain connection w/ full battery (below)	Fail: uECG is unreliable	4/18
		
Accel: sample data collections, exponential weighting (below)	partial success	5/8 Output is decent, most likely need to introduce another logarithmic weight, since small values are too dampened and large values too highly weighted. would also like to try moving the placement to somewhere more optimal



Accel: 8 Hour Log Weight (below)	pass	output is visually good and usable



App/Server		
Function	Result	Date
APP: File transfer with 10000+ characters	pass	4/02
APP/server: app/server works on all machines	pass	5/24
Settings change message is correct	pass	5/5
States manipulating and switching	pass	4/07

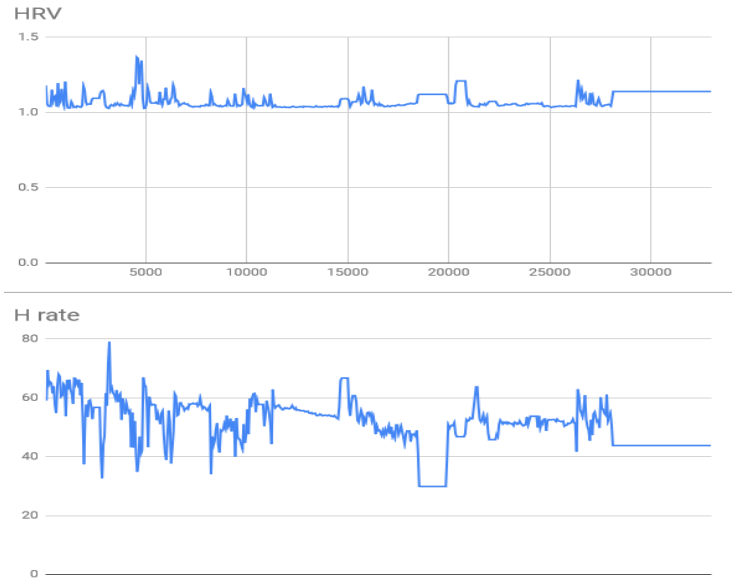
App can be programmed to multiple different devices	pass	4/13
Simulation: Live Dataset test (1 hour)	pass	The signal was sent as expected (5/9)
APP: Settings can be changed	pass	5/5
Pi server: Server remains open and online during all hours	pass	5/5
APP: All forward/back buttons work w/out crashes	pass	5/5

State Machine		
Testing target:	Result:	Time:
Logic of state diagram	Fix flaw state diagram	2/21
Test the state machine code in C	Fix transition error in the code	3/18
Modify the state machine based on the modification of the overall design	Move the implementation of the wake up sequence to the data analysis algorithm.	4/14
Test the newly implemented state machine design with simulation of state transition	Pass test cases and generate output for viewing	4/23
Implement Input parsing and Output writing portion for the state machine.	Finish the input without all testing cases.	4/27
Testing all cases	Pass all cases with minor debugging	5/1
Simulation: Simple Dataset test	Validated that control signal sent at appropriate interval in parsing (5/3)	5/1
Simulation: Live Dataset test (5 minutes)	The signal was sent as expected (5/9)	5/1

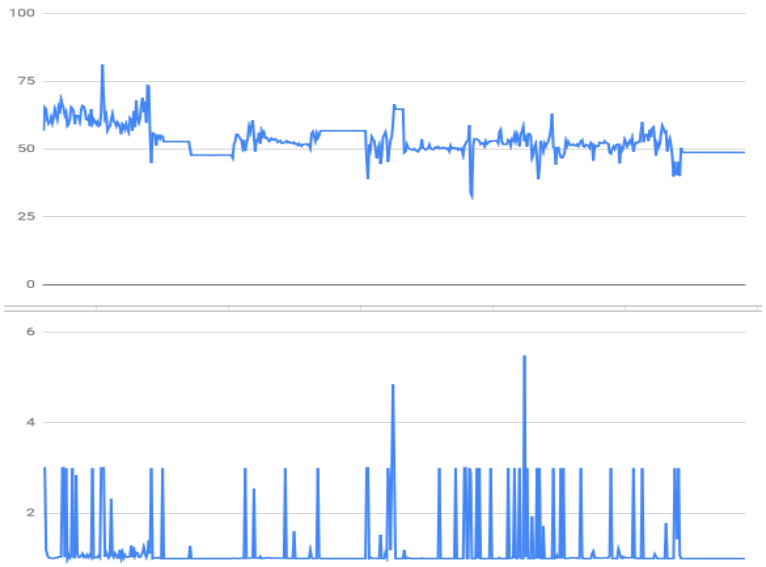
Linear wakeup light intensity gradient based off user config	pass	5/1
Simple timed wakeup control signal for SM	pass	5/1
Prepare the simulations of the state machine for demo.	Modify a few parts for operating the simulations for demo.	5/5
Testing Socket communication	Successfully create server and client in a separate C file to communicate in real time.	5/11

A-3.3 Full Night Run Data Sets Samples

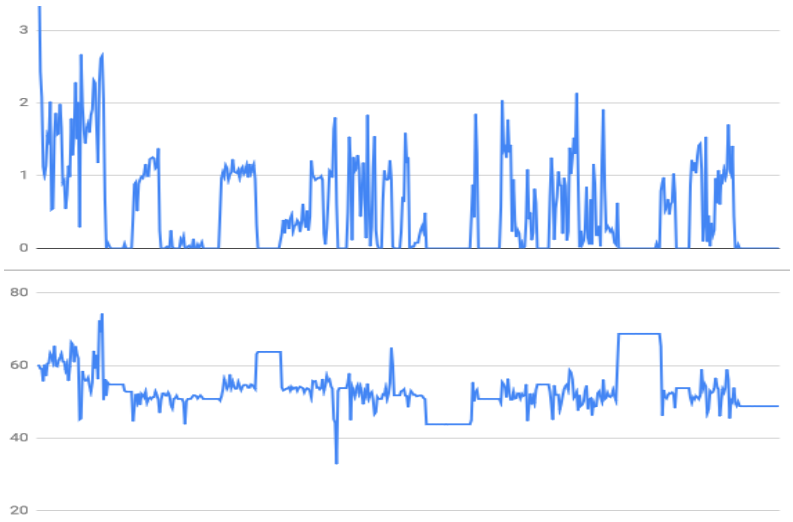
uECG - Accel and HR
3/17/21



4/23/21 - Exponential Accel Scaling



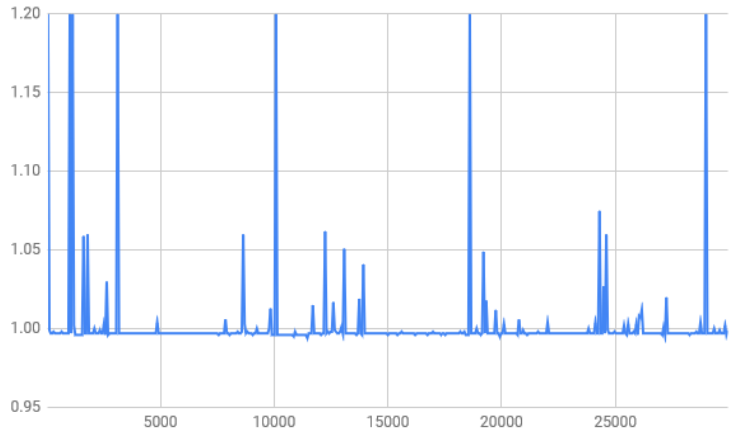
4/26/21



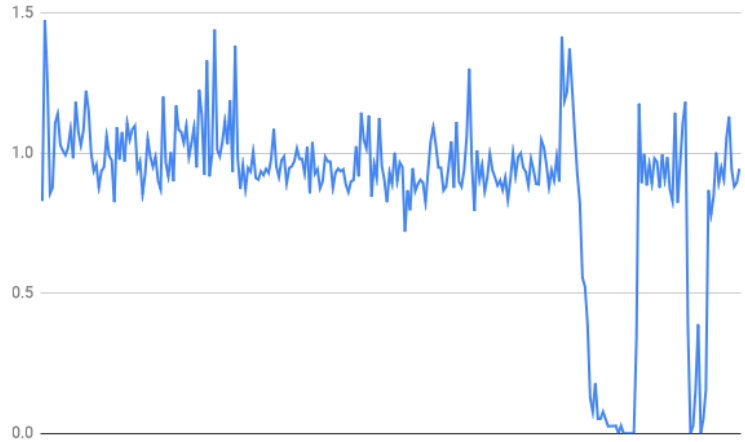
---

uECG - Accel Only

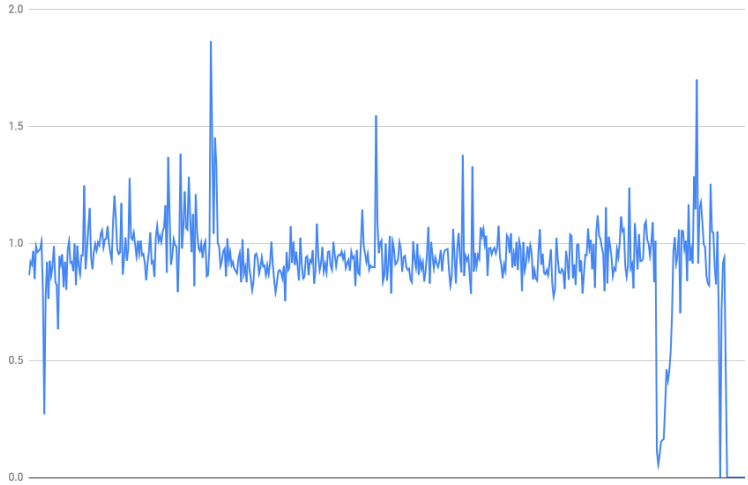
4/20/21 - Exponential



4/28/21 - Logarithmic

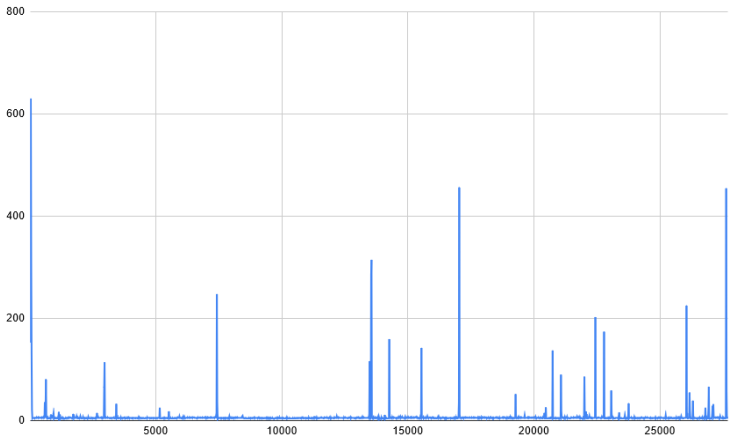


4/29/21 - Logarithmic

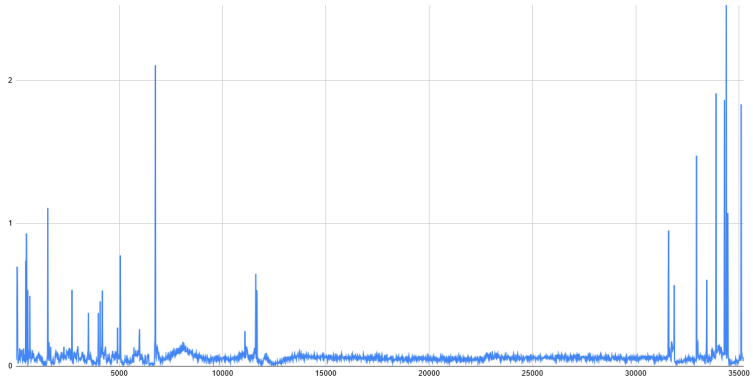


Wired Accelerometer

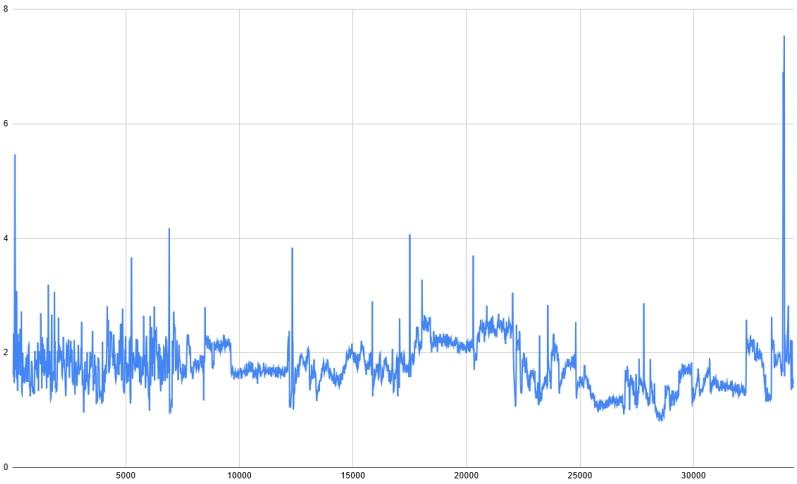
5/8/21 - Exponential



5/12/21 - Logarithmic



5/15/21 - Logarithmic / Adjusted Weight



A-4 Review

Jacob McClellan

I worked primarily on the data collection side of the project. Of course, a lot of time was spent on trying to use the uECG device. If it worked fully as intended, it would have been an excellent component to our project, and I believe the heart rate data may have proved to be an incredibly important part tool for the data interpretation algorithm. However, it ended up being unreliable, and a lot of time was sunk into the uECG that could have been spent elsewhere. The simulated ECG data, which was supposed to work around the unreliability issue, was also limited by the lack of abundant, diverse and continuous data from the uECG. When we transitioned over to the wired accelerometer as our primary apparatus for data collection, it was more effective than I thought it would be. Throughout my time using and testing it, it had 100% reliability. Consequently, once I was able to get proper weights onto the logarithm function, that data also became very readable, clear and easy enough to base the advanced data interpretation program around. If I had more of that lost time back, I could have used it to put more effort into collecting a large assortment of full night data sets, which could have been used to fuel our mostly defunct machine learning algorithm. A lot of my time, if I were to be able to do this project again, could be spent with Nick implementing machine learning.

Nicholas Eaton

Overall I am quite satisfied with the functionality of the data parse algorithm for our prototyped system. If I were to improve the program my priority would be to implement machine learning to provide a stronger sleep detection and more nuanced data parsing algorithm. Our current system uses several running averages to provide a fair estimation of the current sleep stage, but it doesn't do a fantastic job with particularly noisy data. If we were to use machine learning with a large number of sleep datasets we could expect a significantly more accurate sleep stage detection and could do a better job predicting the perfect time to awaken the user. Furthermore, this could dovetail into creating better analytics and information for the user to review their sleep patterns and make changes accordingly.

Steven Appler

For functionality, I am very happy with how the app came out. The data transfer protocol I designed runs reasonably quickly, and the pi server runs seamlessly with good customizability. The sleep graph is good, but could use some additional information in the future. I would personally implement a backlog for users to see more than one day of sleep data in the app, as well as some sort of qualitative data for how well each night's sleep went for users to see their progress. If we were to bring the product to market, I would want to add the ability to integrate with smart lights since most clients wouldn't want to have to add a bulky lamp to their room. We

avoided this for the prototype, since we wanted to have as little commercial equipment as possible.

Zhenyu Zhang

If I could further improve the design, I would integrate all parts to a more complete and good looking product. For the software part, I would implement the real time communication to it. Since our simulations utilize files to provide datas, I really want to integrate all parts together to make it a self-sustaining product. I believe till now, our product is still in the development phase. It still needs some work to make it deliverable to the market. Right now, the communication between the state machine and the phone app is implemented by writing and reading to the same file from both ends. I want to have the send/receive of the bluetooth embedded in the state machine, so that it could communicate in a more efficient way in real time.

Mingyu Ding

I am satisfied with how our project came out at the end, especially how we shape it into the final functional prototype based on our brainstormed ideas. If I have more time to refine our project, I would add the support to smart light bulbs that are currently available on the consumer market. The reason for implementing this functionality is to minimize the cost on the user's end. Since increasingly more people are installing smart light bulbs in their homes, the compatibility from our dawn light system could save them a relatively huge amount of money compared with purchasing the entire system from us as a complete system. As our classmates posted in the discussion form, the addition of smart bulb compatibility will also help in attracting more potential customers to try out our product. So it would be a win-win situation on the journey of developing and promoting the dawn light product line. The challenging part to achieve this idea is to implement the ability to communicate with multiple manufacturers' protocols, such as Phillips, Lifx, Yeelight, etc. But I think we would have the ability to implement this. To achieve great things, two things are needed: a good enough plan, and not quite enough plan.

Weilin Shu

I would add more user friendly features to the phone part. For the alarm clock feature in the app, it successfully sets the correct time. There are still things to improve, For example, I would add weekdays/weekend only wake up time for convenience. I could potentially add a sleep history tracker so that the user could see wake time history, and the user can then adjust sleep time for better sleep habits. I would also try to think about a better way to showcase the app such as trying to make a phone widget. During the graphic part, I would experience more possibilities such as adding different types of graphs, a useful type would be pie graph, so that users can see what percent of their sleep is in light sleep/deep sleep etc.

References

Project GitHub:

https://github.com/sappler/dreamteam9

Accelerometer Datasheet:

https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf

uECG github:
https://github.com/ultimaterobotics/uECG

[1]
Akerstedt, T., \& Wright, K. P., Jr (2009). Sleep Loss and Fatigue in Shift Work and Shift Work Disorder. Sleep medicine clinics, 4(2), 257–271. https://doi.org/10.1016/j.jsmc.2009.03.001

[2]
Alhola, Paula, and Päivi Polo-Kantola. “Sleep deprivation: Impact on cognitive performance.” Neuropsychiatric disease and treatment vol. 3,5 (2007): 553-67.

[3]
Watson, Nathaniel F et al. “Sleep duration and depressive symptoms: a gene-environment interaction.” Sleep vol. 37,2 351-8. 1 Feb. 2014, doi:10.5665/sleep.3412

[4]
Nagai, Michiaki et al. “Sleep duration as a risk factor for cardiovascular disease- a review of the recent literature.” Current cardiology reviews vol. 6,1 (2010): 54-61. doi:10.2174/157340310790231635

[5]
Macedo, Arthur Cassa et al. “Is Sleep Disruption a Risk Factor for Alzheimer's Disease?.” Journal of Alzheimer's disease : JAD vol. 58,4 (2017): 993-1002. doi:10.3233/JAD-161287

[6]
Costa, Giovanni. “Shift work and health: current problems and preventive actions.” Safety and health at work vol. 1,2 (2010): 112-23. doi:10.5491/SHAW.2010.1.2.112

[7]
Blume, C., \& Corrado G. “Effects of light on human circadian rhythms, sleep and mood.” doi:10.1007/s11818-019-00215-x

[20]
“Pulse Sensor (Heart-Rate Monitor).” OpenBCI Online Store, shop.openbci.com/products/pulse-sensor?variant=22543672899¤cy=USD&utm_medium=cpc&utm_source=google&utm_content=&utm_campaign=9567225767&gclid=Cj0KCQiAmL-ABhDFARIsAKywVafv8vkPXpGLHrpDqg4N_TLxODqZmjTAPpF1wuxXAAcJ25yWLLXEnvMaArHOEALw_wcB.

[21]
“UECG - Smallest Open Low Power ECG by Ultimate Robotics on Tindie.” Tindie, www.tindie.com/products/ultimaterobotics/uecg-smallest-open-low-power-ecg/.

[22]
Raspberry Pi Energy Consumption
https://www.pidramble.com/wiki/benchmarks/power-consumption

[23]
PSoC 6
https://www.mouser.com/new/cypress-semiconductor/cypress-psoc-6-soc/#:~:text=Cypress%20Semiconductor%20PSoC%C2%AE%206%20Microcontroller%20(MCU)%20is%20an%20ultralow,%C2%AE%20Cortex%C2%AE%2DM%20architecture.

[24]
Arduino-wifi
https://store.arduino.cc/usa/arduino-uno-wifi-rev2

[25]
Pi 4-base stats.
https://www.raspberrypi.org/products/raspberry-pi-4-model-b/
[26]
PyBluez:
https://github.com/pybluez/pybluez