Interactive Climbing Holds
Final Report

By:
Brian Hu
Daniel Yuan
Jishnu Datta

Final Report for ECE 445, Senior Design, Spring 2018
TA: Jacob Bryan

May 1, 2018
Team Number: 14

Abstract

This document explains design choices and verifications for the Interactive Climbing Holds Senior Design Project. The objective of the project is stated and the solution we have chosen is explained clearly and references the design modules for each part of the solution. All design choices for each module are laid out and explained in detail how they were completed and why they were chosen. The tests performed to verify the project's completion of assigned goals are explained thoroughly and their results recorded. Costs of parts and labor for this project have been calculated and the total cost is explained. The schedule of the three group members' work over the past 16 weeks of working on this project has been documented. Finally, the actual project is evaluated for its completed and uncompleted features and there is a brief discussion of potential future work and ethics of the project.

Table of Contents

1. Introduction

1.1 Purpose and Functionality

The primary goal of this project is to introduce interactive climbing holds to improve end-user experience for climbers and indoor climbing gyms.

In a rock climbing gym, climbers scale a vertical wall using hard plastic rocks called holds, which are bolted onto the wall. To further increase the difficulty of climbing a wall, gyms set “routes” which are a subset of holds on the wall. By limiting the number of holds available to climb a wall, climbers find reaching the top more complex and difficult.

Currently there are three major issues with indoor rock climbing. The first issue is to do with routes. Many climbing gyms use colored tape attached to holds, but with many routes and holds, a wall feels cluttered and confusing. Often, when a climber is on a wall, it can be difficult to tell which holds are on route. Furthermore, some gyms only use colored rocks a means of indicating routes, but this then limits holds to only one route. Our solution to this was to make it so that the rocks could change color, based on the user’s selection. We implemented this by using an LED inside the hold and allowed holds to be changed to any RGB color selected. In this way, we could then set holds to the color of the route currently being climbed, as well as turn off any other holds nearby. This solves both problems at the same time as now holds can be used in multiple routes but when a climber is on the wall, they can only see their current route with any confusion. By using the Interface Module we can display route information to the climber while they are on the wall.

The second issue of rock climbing is the lack of interactivity with the gym and climbers. In most gyms, climbers are encouraged to be independent. They look at a list of routes, climb them and then leave. This pattern is detrimental to both gyms and climbers as gyms have no idea what their climbers enjoy. In some gyms, owners may talk only with regular proficient climbers, and thus may make too many difficult routes as compared to the average user base. Furthermore, climbers have no records of their performance over time. No one records a climber’s data as they climb, and so they have no idea how they are progressing. To solve this problem, we wanted to collect data during a climber’s climbs with a Hall-Effect sensor. By wearing small magnets on a climber’s hands and feet, our project is able to track a climber’s movement up the wall. In this way we can provide detailed climb description to the climber, post-climb. Furthermore, by collecting data about routes, we can now provide gyms analytics on their routes. For example, if you see many people are completing the easy routes in very quick amounts of time, you may want to switch these routes out for something harder. To display all this information and actually interact with the wall we used a number of modules involving the Microcontroller, Wifi Module and Server. We created a webapp in the Software Module that climbers would interact with to see their data and analytics. Furthermore, the server keeps track of the state of all holds and user information.

The Control Unit keeps track of all the states and information flowing through the hold. The WiFi Module allows a way to send and receive information to the server. Finally, the Sensing Module, allows for detailed information about a climber's movements while they scale a route.

Our final problem was the difficult process of setting routes in gyms currently. Gyms periodically take down all the holds/ routes on a certain wall, and redo the wall in order to keep climbs fresh. During this process staff members need to manually unbolt holds on the wall and then bolt in new holds in certain positions for new routes. This is a long process and can take up to a day per wall. Furthermore, only really trained staff are allowed to route set, and so the general gym customers cannot set their own routes. By using our webapp again, we wanted to solve this problem by allowing anyone to set routes simply by selecting the holds they wanted. As everything was already set-up to be able to climb a route, just given a list of holds, we can allow anyone to make routes by choosing holds already existing on the wall. The way that holds can be multiple colors due to our solution in the first problem, makes it so that more routes can be created, with almost no effort. The Battery Module is critical in providing power to the hold so it does not need to be removed very frequently. The goal of the Battery Module is to reduce the number of times a hold must be taken off the wall, so it is critical it has a reliable and long lasting power output.

1.2 Subsystem Overview

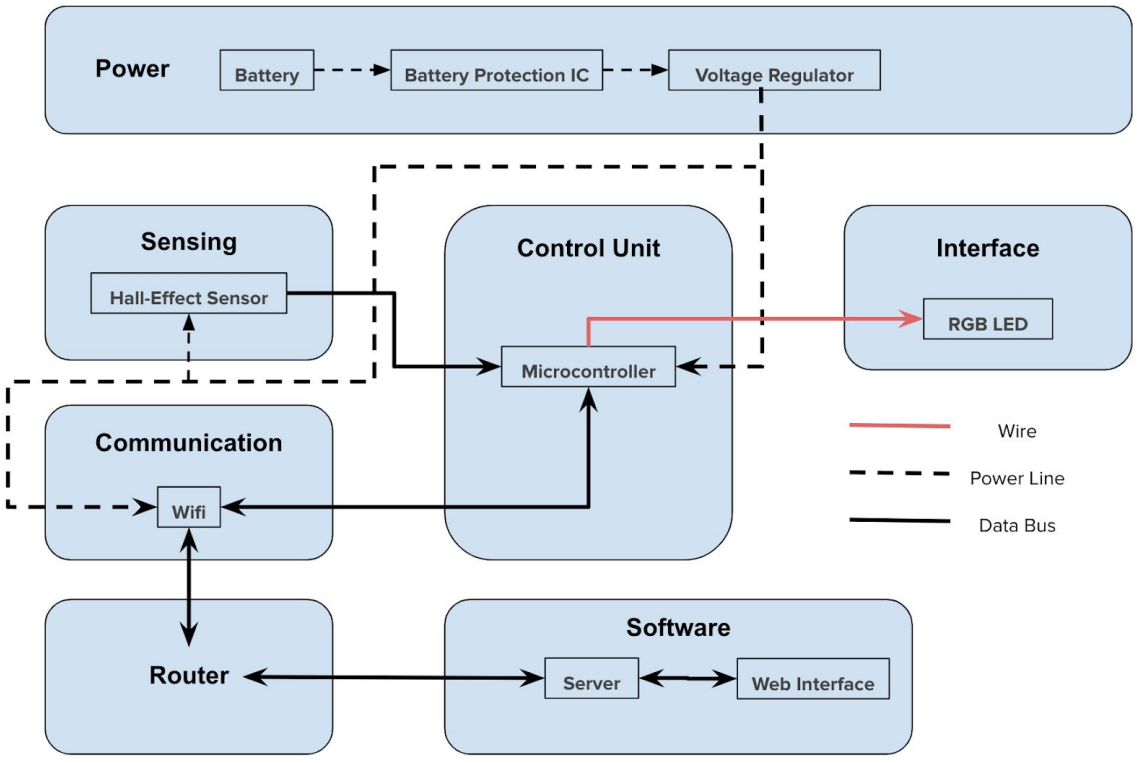

Figure 1.1: Interactive Climbing Hold Block Diagram

Power Module

- Contains a battery and a voltage regulator. The battery protection IC was deemed unnecessary after we changed our battery design choice from a lithium-ion to an alkaline battery.
- Provides power to control unit and wifi module. Supplies 3.3V power.

Control Unit

- Controls, maintains and implements of state of hold.
- Communicates with wifi module to poll state.
- Controls Interface Module logic.
- Receives data from Sensing Module.

Sensing

- Detects climbers wearing magnet when close to hold.
- Sends whether a climber detection data to control unit.

Interface

- Contains an LED to display RGB colors to the climbers outside of the hold.
- Color is controlled by output from control unit.

Communication

- Uses WiFi protocol to act as a bridge between microcontroller and server through router.
- Receives AT commands from the control unit and sends and receives HTTP requests and responses between control unit and router.

Router

- A way of connecting the server to all the hold communication modules through WiFi.

Software

- Contains a web-app with a database to store client data and hold states.
- Client uses web-app to interact with holds.
- Allows holds to view their state through WiFi communication on router.

2. Design

2.1 Power Module

The power module hosts a battery pack and voltage regulator required to power the hold internal hardware. As implied by the title, the setup is key to providing enough power in order to handle a maximum load from the other modules when being run at full capacity.

2.1.1 Battery

When looking for an optimal battery to use in each climbing hold, there were several major considerations which we had to balance. We looked into batteries that could provide a minimum voltage, current, were reasonably priced, and had a long enough lifespan given our average current draw. During our process of considering reasonable battery types for our final product, we considered batteries including NiMH, NiCd, Lithium Ion, and alkaline batteries.

The first major threshold each battery had to pass was in providing enough voltage and current to actually keep the rest of the hardware running. In particular, we wanted a voltage draw of at least 4.5 V and the ability to supply a current of at least 300 mA to the rest of the hardware. The main trade off from meeting this requirement was in battery size, which we had to consider on a strict scale in order to fit inside the battery. After looking at many different types of batteries, the two main battery types that seemed to satisfy both requirements were lithium ion and alkaline batteries. From there, we made the ultimate decision to use alkaline batteries instead of lithium ion batteries. This was due to the large difference in price which made the alkaline batteries much more desirable in a project like ours, since we needed eight battery packs for all of the holds going into our design.

Our final battery setup consisted of three AA alkaline batteries in series providing a total of 4.8 V to the circuit. The testing done on the battery, which will be mentioned in the later sections, show that the battery is more than capable of supplying the desired current draw, and can provide a high enough voltage to maintain the system for over 15 hours. The main downside about the batteries we chose was that they were not rechargeable, but getting alkaline batteries that are would not have been too challenging. Our main goal in choosing non rechargeable batteries was to save costs in the end.

2.1.2 Voltage Regulator

While the battery provides power to the rest of the system at around 4.8 V, none of the actual modules are being powered at that voltage. This is mainly due to the fact that our wifi module can only handle voltages of up to 3.3 V. Due to this factor and the fact that our microcontroller can also operate at 3.3 V, we decided to unify the power line at 3.3 V using a TI LM317 adjustable voltage regulator. A portion of our overall circuit diagram showing on the the relevant voltage regulator circuit portion is shown in Figure A1 in Appendix A.
Since our voltage regulator has an adjustable output voltage, we needed to calculate the optimal resistor setup around the voltage regulator in our circuit design. To do so, we consulted the part's datasheet, and formulated our solution based off of Equation 2.1 shown below[1].

$$V_{OUT} = 1.25V \times \left(\frac{R2}{R1+1}\right)$$

Equation 2.1: Voltage Regulator Output Voltage Equation

In order to achieve a Vout of 3.3 V, we calculate the ratio between R2 and R1 to be R2 = 1.5 R1. Based off of this, we used a R1 value of 100 Ω and an R2 value of 150 Ω. This, along with the fact that the voltage regulator can handle current draws of up to 1 A[1], makes our voltage regulator a solid choice in order to provide power to the main board and sensors.

2.2 Communication Module

Our project required a communication module in order to update individual holds based on a server. We wanted each hold to be modular and self-contained so we opted for WiFi as our communication protocol. We used an ESP8266 Wifi Chip as part of our system in order to allow the microcontroller to update its state. The ESP8266 takes a custom AT command set, which must be sent from the microcontroller using UART at a 9600 baud rate. Firstly, the ESP8266 must be connected to the server router using AT commands. After the connection has been established, the microcontroller is able to create a TCP connection via AT commands with the server. Using this TCP connection, the microcontroller is able to send HTTP requests to the server and receive back data. Once a response is received, a final AT command must be send to the WiFi module to close the TCP connection. Although this is not required, not sending this command relies on the TCP connection to timeout which takes around 500ms of time. This dramatically decreases the performance and speed at which the microcontroller can communicate with the server. The AT commands commonly respond positively by returning an OK message back via the UART. By parsing the response from the ESP8266, it is possible to keep track of AT commands which fail. This increase the robustness of the overall hold software and helps greatly to debug any WiFi issues.

2.3 Sensing Module

The crux of our project focused heavily on the sensing aspect of each hold used to detect when a climber has reached the desired hold. After heavy deliberation over the optimal sensor to use, our final decision was to use TI DRV5032 Hall effect sensors in each relevant hold in conjugation with small wearable magnets on the end user. These sensors detect the presence of a nearby magnetic flux, and toggle as a switch based off of this mechanic. We used Hall effect sensors because they provided a cheap sensing option that had an effective range over that specified by the holds. The distance between the sensor placement on the inner side of a hold wall and the surface of the hold is approximately 1.5 cm. We wanted a sensing range of at least double that distance with a small enough magnet that would not cause any intrusive behavior to the end user. After selecting the Hall effect sensor, we needed to choose a magnet size which balanced overall size with sensing range. Our initial design wanted a 5 cm sensing range, but we eventually settled for a magnet size which had approximately 4 cm as its effective sensing distance. This was done based off of the formula provided in Equation 2.2 below from the datasheet[2].

$$\vec{B} = \frac{B_r}{2}\left(\frac{D+T}{\sqrt{(0.5C)^2+(D+T)^2}} - \frac{D}{\sqrt{(0.5C)^2+D^2}}\right)$$

Equation 2.2: Hall Effect Sensing Distance Formula

After simulating the expected magnetic flux values based on the formula, we came to several choices with their relative magnetic fluxes displayed in Figure 2.1 below, where B_OP is the calculated turn on magnetic flux for the Hall effect under the different magnet sizes.

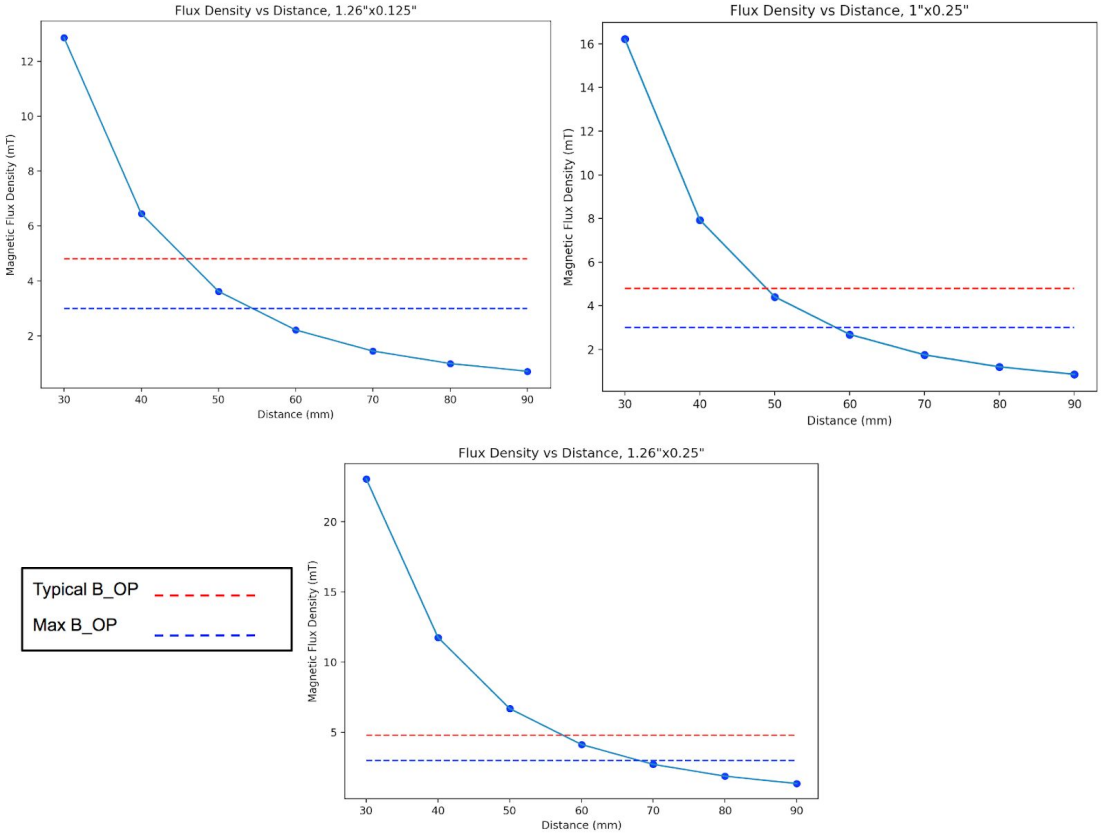

Figure 2.1: Simulated Magnet Size Flux Curves

From the three options presented to us, we decided on a magnet size of 1.26” x 0.125” over the 1” x 0.25” and 1.26” x 0.25” magnets, mainly because it gave us an average of 4 cm tolerance as desired, as well as a readily available size that was cheap enough to buy enough as required by our design. As will be described in a later section, our testing for accuracy and sensing distance with the Hall effect sensor after construction were very sufficient to show that the sensing could be done feasibly for every hold in our design.

2.4 Interface Module

In order for climbers on a wall to see what their selected route was on the wall before and during traversal, inside every hold we have a common anode RGB LED serving as the primary interface for a climber. The LED is sealed within a silicone mold just under the surface of the hold. We decided to choose an RGB LED over single color LEDs because our design requires a wide array of potential colors on a single wall, since every route must be illuminated by a unique color. Since

our microcontroller has the ability to communicate with the RGB LEDs using built in analog PWM ports, there were no complications during software to hardware implementation.

There were two choices of RGB LEDs available to us, a common anode LED and a common cathode LED. The LED wiring setup for the two are shown in Figure 2.2 below.

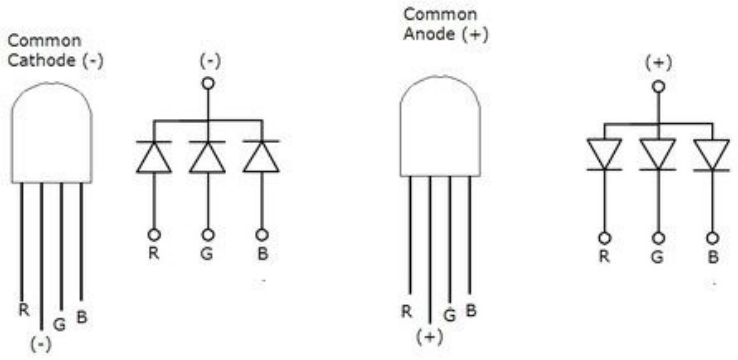

Figure 2.2: Common Cathode vs Anode LED Wiring

From a first glance, it seems more intuitive to use a common cathode LED due to the fact that wiring the LED to the PIC allows for direct connections from each of the RGB pins. However, when we were comparing the two choices, we realized that we could construct a circuit that would allow a common anode LED to function exactly the same as a common cathode LED, while giving us a higher output quality due to the fact that we had free access to expensive common anode LEDs at the time. To accomplish the above task, we created an LED interface circuit shown in Figure A2 in Appendix A.

As shown in the figure, since we wanted our LED on the surface of the hold away from the other electronics, our main board simply has a pinout row for each of the four LED pins. Pin 2 on the LED goes directly to power as depicted previously in Figure 2.2. Each NPN BJT serves as a current drain that allows the signal from the three PWM pins on the microcontroller to act as individual color control pins. The three resistors R3, R4, and R5 were chosen to be 100 Ω, while R6, R7, and R8 were set to 1 kΩ each. The resistor choices were made from testing different combinations to achieve a brightness level that was bearable to look at for a climber, but not so dim that they would not be noticed from a reasonable distance. Ultimately, this circuit setup allowed our LEDs to achieve a large number of color options for different routes.

2.5 Control Unit Module

The control unit module contains the microcontroller, RGB LED control circuit, and the hold software. The primary purpose of the microcontroller is to interface with each component in the climbing hold to handle inputs from the hall-effect sensor, control the RGB LEDs, and communicate with the server. In our implementation, we decided to use the PIC16F18877 microcontroller to communicate with the wifi module through UART communication, read inputs from the hall-effect sensor with generic I/O pins, and have three PWM pins that could control the

color channels of a RGB LED. The hold software was kept minimalistic in order to reduce the power consumption by offloading all data processing into the server.

2.5.1 PIC Microcontroller

The microcontroller we planned on using was the PIC16F18877; however, due to some mistakes in ordering, we also used the PIC16LF18877 microcontroller[3]. Fortunately, the PIC16F18877 and PIC16LF18877 are the essentially the same except for the PIC16LF18877 being the low-power model. Since, our circuit is running at the lower voltage of 3.3V, the different microcontrollers had no effect on our final product. Although ATmega microcontrollers could have been easier to use, we wanted to use a PIC to reduce power consumption. As the holds are on the walls and only powered by battery, it is critical we reduced the power consumption as much as possible. Furthermore, the PIC microcontrollers we chose provided us with enough pins to handle our sensing module, RGB module and WiFi module. The microcontroller was difficult to program and even use at first as various pins needed to be set-up in a highly specific manner. Fortunately, we were able to find an addon for the proprietary PIC programming software, which made setting up the pins effortless [4]. Through the addon we could state that we needed an input pin for the Hall Effect, three output PWMs for the LED and TX/ RX UART pins for the WiFi module, and then integrating the external modules was much easier with the hold software.

2.5.2 Hold Software

The microcontroller was controlled by the client using a finite state machine. The microcontroller could only be in 3 states at any given time: standby, climb, display. The holds were set-up to mirror the server states of the holds. Therefore, clients would perform actions in the front-end of the webapp which would change the server state of holds. Then a hold would update itself after checking against the server. To check, at a set time interval, the hold would use the ESP8266 to establish a TCP connection with the backend. Then it would send a HTTP get request to check its own state. After receiving a message about its state and any given parameters with its state, it would then set itself to that state, and change anything required by the parameters. Standby state is the state used when the holds need to be inactive and consume as little power as possible. During standby state the LED is off and the amount of delay during polling for state is greatly increased. A display state request will come with three parameters, red, green and blue. The LED will then set to this color and hold the color until another state or a display state with a new color is set. Finally, climb state also comes with the red, green and blue parameters. During climb state, the LED will also hold the set colors. Furthermore, the hold will send back any events recorded from the sensor module for the duration of the time the hold is in climb state. An event is either a hold becomes activated (the climber touches the hold and triggers the hall effect sensor from inactive to active), or the hold becomes deactivated in a similar manner. These events are sent through the ESP8266 TCP connection as HTTP post requests, and contain the hold ID and event type. These three states make up the majority of logic in the microcontroller software, however, before the finite state machine begins, the hold does go through a boot sequence. On startup the microcontroller will set-up and make sure all other modules are working. Primarily it checks the wifi module for a stable connection to the climbing wall server.

After start-up is initialized, it defaults to standby state and begins polling to update its state to mirror the server.

2.6 Software Module

The software module involves the server, frontend, and backend for our web application that handles all the hold inputs and determines what states the climbing holds should be in. One of our large design decisions was to offload all processing onto the server to reduce the power consumption on each hold. We also wanted to create a user interface that would be intuitive to use and implement in the short time-frame given for this senior design project. Also, in climbing gyms, most people will only have their phones with them while climbing, so we wanted a user interface that could be used on a mobile device as well as other computers for admin control and other tools.

Since we wanted to have a centralized server to control all the holds, we decided to use a single-board computer called the Raspberry Pi, which runs a lightweight linux kernel based on debian on relatively cheap hardware[5]. The choice to use an actual operating system instead of repurposing another microcontroller is to allow a multitude of server choices later for scaling purposes. For example, if a larger climbing gym were to use hundreds or thousands of climbing holds, a simple microcontroller would not be able to handle that amount of requests while staying responsive; thus, an upgrade to a more powerful machine would be needed. Our software is written in a modular fashion that would make it trivial to port between different machines with varying architectures. This design choice was to make it easy to setup the server for a climbing gym and allow for scalability. Also, in the distant future, the server could also be moved into the cloud, so the climbing holds will be communicating with a centralized server that can handle controlling holds from multiple gyms all over the world. This removes the need for a server to be located within a climbing gym at all, as the climbing holds can communicate over the internet to a data center.

For the web application, we are using a decoupled web stack configuration, which basically means that the frontend server and backend server are separated. This allows for more flexibility for implementation as well as simpler scaling for future-proofing. One of the primary reasons for choosing a decoupled web stack configuration was to completely detach the frontend from the backend to make it simpler to develop with different group members, so we could develop our respective parts at our own paces until the integration phase. Another key consideration was the idea of scalability, as we wanted to make a server that could be moved from a local server within a climbing gym to a centralized server in a data center or on the cloud that could store user information and preferences across multiple climbing gyms, which would allow us to develop a system for integrating users in the scope of nearby climbing gyms instead of just one local one.

One of the main tools that we use in the software module is Docker and its accompanying tool Docker-compose[6][7]. Docker is a tool that performs operating-system level virtualization, which is also known a containerization, which serves to package each software module into an isolated

container that would allow us to deploy software modules across varying platforms and servers. The purpose for using this in our senior design was primarily to solve the issue of handling code between different computer architectures. Since our our personal machines have x86 architecture and the Raspberry Pi server using an ARM architecture, using Docker will make the switch from development to deployment much simpler. Also, docker containers help eliminate the issue of code working on one machine and not another as well as running containers on seperate machine altogether; thus, Docker adds much more flexibility for software development and insurance for differing architectures, which makes it an ideal tool for this project.

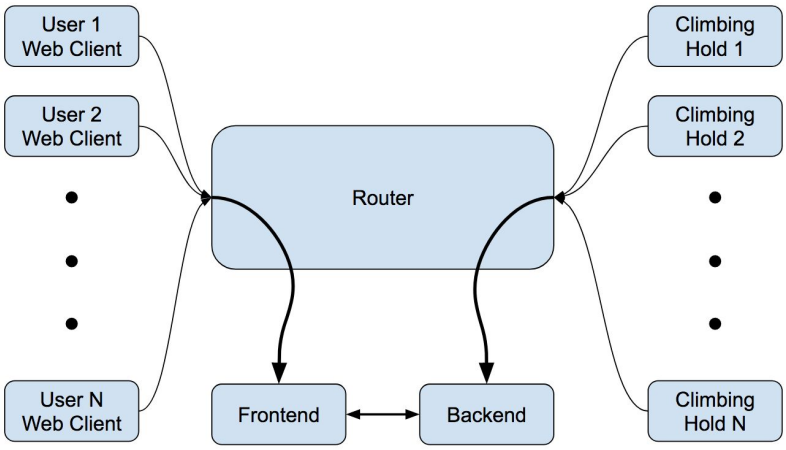

Figure 2.3: Network connection between server, climbing holds, and users

2.6.1 Backend

The backend for our web application serves as an API server, which is essentially a server that handles requests from the climbing holds and the frontend to facilitate actions on the database. In our implementation, we are using an Express backend written in Node.js that interfaces with a PostgreSQL database. The database is responsible for storing all states as well as storing data such as climbing hold and user information. The organization of the API server are routes, which refer to possible requests that a client can make to the server, that have different tiers of authentication that is described in more detail in section 2.6.1.1. All actions that happened to the climbing holds or on the webapp passed through the backend API server that connects the frontend, database, and climbing hold through one clean interface.

2.6.1.1 Authentication and Routes

The API server routes are broken up into three authentication tiers: public routes, private routes, and admin routes. Authentication is enforced using a middleware, which is essentially a function that is called before each request that rejects requests that don't satisfy the authentication tier requirements with an authentication error.

Public routes don't have any authentication and simply just allow those routes to be called by any client. Public routes are mainly used for user registration, user login, hold registration, getting hold state, and hold event registration. To register and login into the system, an user must access

public routes to obtain a json web token, which is a hashed string that stores authentication data. They will be able to use the higher authentication tiers after logging in. Climbing holds automatically register themselves as they are powered on, and update their status in the server by calling the hold registration routes. As stated in the hold software section, the backend server doesn't access a climbing hold directly; instead a climbing hold fetches the state it should be using a public route and mirrors what the backend server tells it to be. Climbing hold event registration refers to the route used by climbing holds to register events, which are the events that notify when a hold is activated or deactivated by a climber.

Private routes have a basic authentication that is based on an json web token that a client must send in order to pass the private authentication middleware. After a user is logged in, the backend server sends a json web token, which is essentially the UUID of the user that is hashed with a secret key and salted with 10 extra characters, that uniquely identifies a user and is functionally impossible to crack as the secret key is local to the server and the salt is unique to the server environment during the generation of the json web token. The client is required to send their token to the server, which will check the authenticity of the token by unhashing the token and checking if the user exists in the database. These routes are used mostly by the frontend for requesting information about walls, climbing holds, routes, and user information. All the data displayed on the frontend is requested from the backend using private routes. However, there are another set of routes called action routes that carry out actions such as starting a climb, ending a climb, and canceling a climb. These action routes are a bit more complicated as their functionality is core of our project, as they handle lighting up holds, switching hold states, and switch states in the database to accurately parse and store events coming from the climbing holds, which is described in more detail in section 2.6.1.2.

Admin routes have the same authentication scheme as private routes as they check the web token from a user for authenticity by unhashing the token and checking if the user exists and has admin permissions in the database. The only difference between admin and private authentication is the extra step of checking if the user has admin permissions. The admin routes are responsible for handling actions such as defining a new wall, defining a new climbing routes, setting a climbing hold into display mode, and setting a climbing hold into standby mode.

2.6.1.2 Action Routes

Action routes are more complicated than normal routes as their logic handles many tasks such as creating a new climb element, updating climbing hold states, and disabling routes with conflicting holds or walls. Essentially, we have created a write lock on climbing routes that is locked using the start climb action route and unlocked using the end climb or cancel climb action routes.

To start a climb, a client makes a start climbing request to the backend, and if the route is not disabled, the backend finds all climbing routes that contain a climbing hold that will be activated by this climbing request as well as any routes that are on the same wall and disables them. Then, the server updates the states of all the climbing holds in the climbing route into climb mode and changes the RGB LED color to the color of the route as well as adding some tracking information

in the holds database so the server knows where to store climbing events from the climbing holds. This effectively restricts any client to only start a climb on routes that will not physically conflict with the activated route and enables the climbing holds to detect user interaction on the holds and allows the registration of events into the newly created climb element.

A client can either cancel a climb or end a climb after a climbing route as been activated, the only difference is that ending a climb keeps the unique climb element created when the start climb request was called, which stores all the climb events, and canceling a climb will delete this climb element. Both of these routes, find all the climbing routes that have been disabled by the start climb request and updates their state into standby mode, which unlocks the routes for future requests.

2.6.1.3 Database

The database that we decided to use is PostgreSQL, a high performance relational database, that we interface through an OEM called Sequelize. The choice for using this is that we wanted to use a relational database, as our data has complex relations with other tables in the database. We defined six different tables to store relevant data for the backend: users, walls, routes, holds, events, climbs. For each table, every element is given a unique identifier using the UUIDv4 scheme.

The user table stores user information including the user's name, unique username, hashed password, email, climbs, permissions, routes created, favorited routes, and skill. Most of these attributes are self explanatory such as the username, password, email, and name. The permissions attribute is an enum that stores if the user is an user or an admin, which is used in the admin routes. The climbs attribute stores the climbs that the users have completed so they can look back into old climbs and the stored events within each climb. We could do a lot more data processing to generate interesting statistics or allow a user to see their previous climb on the wall, but these are all extra features that we didn't get to. The routes created is simply a list of routes that the user has created, so the user has to be an admin for this to have any data. Favorited routes and skill are attributes that we originally wanted to implement, but we simply didn't have time to have these extra features.

The wall table stores a wall, which has a name, status, and list of routes. The wall object is used to help climbing gyms specify physical portions of their climbing wall to make sure that routes that are on the same physical section cannot be activated, which would create a safety hazard as climbers might potentially be climbing over one another. The status attribute simply stores if the wall is currently being used by a climber or a route setter.

The routes table stores information about a route, which includes the name, color, person who created the route, status, difficulty, holds, the start hold, the end hold, an image url, and an active climb id attributes. A route has information like name, color, image url, and difficulty to help users identify and gauge routes by their difficulty as well as visually identify a route when its activated on the wall. The holds, start hold, and end hold attributes store the climbing holds that make up

this route as well as specifying what hold the user should start and end on, which can be tracked with events. The status attribute stores the current state the route is in, such as specifying if this route is disabled, active, or in standby mode. The action routes described in section 2.6.1.2 updates the status of routes when a client issues an action. The active climb id is a tracking attribute that is stored when a route is put into climb mode to help find a climb quickly to store events from the climbing holds.

The holds table stores information about a climbing hold including hold type, state, LED color, image URL, active climb id, active wall id, active route id, and route setting toggle. The hold type and image URL attributes is the help the users identify climbing holds on the wall, by giving them the type of hold it is and what it looks like. The state and LED color attributes are what the climbing holds fetch when they request an update from the server and they then update their local status to match these attributes. The active climb id, active wall id, active route id, and route setting toggle are attributes that store tracking information so the backend can quickly update elements for handling action routes and event registration.

The events table simply stores the time an event was created, the event type, and the hold that the event originates from. The event type is an enum that stores if the climbing hold that created this event was activated or deactivated. These events are typically linked in a climb object that stores the entire timeline of a climb.

The climbs table stores a route id, user id, and an array of events that describe a climb that a user has completed. The user id and route id identify the climb and links it to the user and the route that the user is climbing. The events array stores events in the events table that has occured during this specific climb. This allows us to create an exact timeline about which holds were activated and how they were activated throughout the duration of a climb.

2.6.2 Frontend

This the the outward facing software which the climbers and gym admins actually see and interact with. The frontend acts as a way for climbers to change the backend and therefore holds easily. As the majority of climbers would be accessing the web-app using their phones, it was important to design for mobile.

Initially, we require all users to create an account to access the displays to access most backend routes. By requiring accounts, we are able to track users and store information to their accounts. This helps provide better analytics to both the gyms and users. Once a user logs in they receive a token based on their account, which they can provide to the backend to authorize any routes requiring an account. This token is stored in their local storage on their webpage. This allows users to close their browser and reopen the climbing wall webpage without having to relog.

Once inside the browser there are multiple tabs for different functionalities. The home page greets the user and informs them of how to use the project and various rules of the current gym.

The walls and holds page both respectively list the walls and holds available. Clicking on an item will lead to a more detailed page about a certain hold/wall.

The routes page provides a climber with a list of routes currently available at the gym. Clicking on a route will lead to a more detailed routes page which will contain information such as difficulty, color, wall and a list of holds in the route. Furthermore there is a button to indicate the routes availability for climbing. If the route or holds in the route are currently active, then the button is unable to be clicked. However, if the route is available, button, which is displayed as free to climb, will send a climb request to the backend when pressed. It will then lead to a climbing page, where the user can end the climb or cancel the climb using respective buttons. Once a climb is ended, the frontend will direct the user to a detailed page for that climb. The detailed page for a climb contains information the route, climber, time taken for the route and a list of time-stamped events sent from the holds on the route. With this list, it is possible to chart out a climber's way up the route.

The profile page was intended to give climbers detailed information and stats about their climbing behavior over time. On this page, currently a climber can see their name, a list of all the climbs they have ever done and a list of route they created. This page can be further expanded to include graphs and various charts on the climber's skill.

Finally, the admin page has four buttons to send more controlled commands to the server. Each button launches a form above the page, where a user can input information into the fields and send the form to the server. The create route button allowed for creating routes given a name, color, difficulty, wall and list of hold ids with start and end holds. The create wall button allowed for creating a new wall given a name. The display mode button allowed an admin to set a hold to a certain color in display state given the hold id and color. The standby mode button allowed an admin to set a hold to standby state given a hold id.

3. Design Verification

3.1 Battery

1. **Requirement:** Must be able to supply a 4.5 - 5.5 V source voltage with a minimum 300 mA current draw.
   Verification: Attached 10 Ohm resistor to 4.8 V battery to simulate a 480 mA current draw. The voltage across the resistor was 4.6 V, which is within the acceptable range for the voltage regulator.
2. **Requirement:** Has to be accessible from the back of the hold for convenient removal.
   Verification: Unscrewed back panel of an assembled hold and removed battery. Entire process took slightly less than two minutes to complete.
3. **Requirement:** Battery should last for 24 hours with 6 hours in the climb state and 18 hours in the standby state.

Verification: Our verification for this requirement is based on the graph shown in Figure 3.1 below.

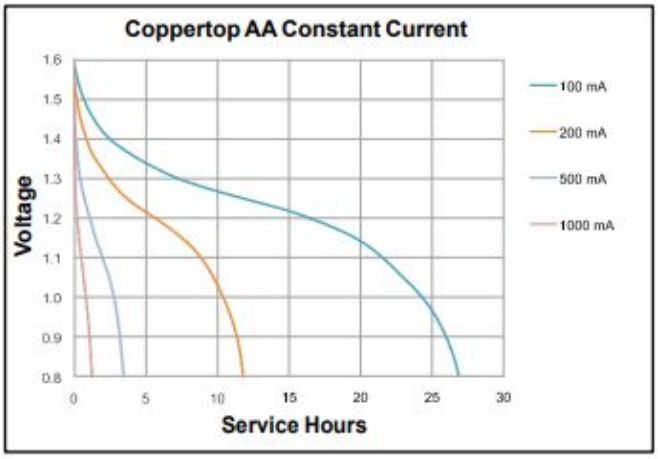

Figure 3.1: Alkaline Battery Lifespan Chart

From our overall calculations of average current draw during the duration of the two phases described above, we came to an average of around 100 mA of constant draw with the entire system running. Since we need a voltage that exceeds 1.1 V individually for each battery, or a total of 3.3 V for the regulator, we can see from the rightmost curve in Figure 3.1 that our estimated battery life is around 22 hours. This is just short of our 24 hour goal, but it is due to the fact that we originally intended on using a lithium ion battery, which has a much higher total power density, but was too expensive to be practical in the scope of our project. However, the results of this verification still show a considerable amount of power from the batteries we ultimately chose.

3.2 Voltage Regulator

1. **Requirement:** Must be able to safely operate our battery at below 60°C.
   Verification: Applied a resistor to the battery at 480 mA and measured the temperature of the battery after two hours with a thermometer. The temperature of the battery was below 60°C (measured at 46°C).

3.3 Wifi Module

1. **Requirement:** Tranceive data with the server at >2 kbps under an IEEE 802.11 b/g/n protocol.
   Verification: Sent a 256 character long string (1 character = 8 bits, total 2048 bits) from the PIC through the wifi module to transmit to the server for ten trials. The server echoed the exact string back every time to the microcontroller almost instantaneously (around 3 ms average response time).
2. **Requirement:** Must be able to communicate with the microcontroller using a UART connection.
   Verification: The results of the experiment conducted in Requirement 1 of this section uses the UART functionality of the microcontroller in order to transfer information. Thus, this requirement is implicitly confirmed by the success of the previous requirement.

3. **Requirement:** Wake up time from Standby mode should be < 20 ms.
   Verification: Timed the wake up protocol on the microcontroller and outputted the time difference for ten trials, which equalled 0 ms every time since the wake up time was so fast that there were not enough bits in C to handle the low floating point value.

3.4 Hall Effect Sensor

1. **Requirement:** Sensor must read accurate input signals between 10 - 30°C.
   Verification: Tested the Hall effect sensor for five trials each at both at 10°C by putting the sensor in a refrigerator and testing immediately after removal, and at 30°C by placing the sensor near a space heater and performing the same test. In both cases, the sensing distance remained around 4 cm, which is the approximate distance calculated from the next section.
2. **Requirement:** Sensor must be able to detect a magnet within 5 cm with a detection rate of 90% within 500 ms.
   Verification: Ran 20 trials measuring the sensing range of our four functioning Hall effect sensors with a ruler using our designated 1.26” x 0.125” neodymium magnet. The total results as well as the average results are shown in Table 3 in Appendix B.
   The average sensing distance of the four Hall effect sensors is 39.3 mm, which falls just around 10 mm short of our projected sensing distance. However, since the width of the hold wall from the hollowed insides to the surface is approximately 8-10 mm, the sensors will have no problem detecting a magnet’s presence on the other side of the hold. This slight mismatch between requirement and verification was due to the extra tolerance which we originally imposed on ourselves. As far as practical use, our current average distance still remains acceptable in our final product.

3.5 Common Anode RGB LED

1. **Requirement:** Seven specific colors (red, green, blue, yellow, aquamarine, magenta, and white) must be visible through the hold in standard room lighting conditions from greater than 10 meter distance from the LED.
   Verification: Conducted an experiment that involved turning the LEDs different colors from a list of colors including: white, red, green, blue, yellow, cyan, and magenta. Each of our group members attempted to identify the color of the LED from 10 meters away in less than five seconds for approximately 20 trials each. For every color we tested, we achieved a 100% accuracy in determining the color of the LED.

3.6 PIC Microcontroller

1. **Requirement:** Should have enough pins to handle WIFI module, hall-effect sensor, and RGB leads.
   Verification: By running the entire module and having everything working simultaneously, we intrinsically verified this requirement.
2. **Requirement:** Should be able to transmit and receive over UART at >2 kbps.
   Verification: Refer to Requirement 1 of section 3.3, wifi module. Since the communication chain includes sending and receiving data from the wifi module through UART, our test in the previous section also directly verifies this requirement without having to run a separate test.

3.7 Web Interface Requirements

*Note that many of these requirements were tasks which we had to fulfill. If the task was not fulfilled, we ultimately deemed it as a feature over a requirement for future implementation in the sense that it may not have been necessary for the final product.

1. **Requirement:** Displays all possible routes available to climb, with detailed information about the routes.
   Verification: By observing the web server we see a list of all possible routes along with their corresponding colors and names.
2. **Requirement:** Easily categorizable and filterable to select best routes on certain walls for the climber.
   Verification: We were not able to implement this feature in time for the demo. Due to time constraints and the fact that this “requirement” does not contribute to the final functionality of the project, we decided to save this as a feature for later implementation.
3. **Requirement:** Displays statistics to climbers about their progress over time.
   Verification: The web interface has each climber stored in their own accounts, which gives this feature from being able to see one’s climbing times in their profile.
4. **Requirement:** Displays statistics about a climb after the climb has ended to a climber.
   Verification: As shown in the functionality of the interface, the climber has the ability to see what time they reached every hold along their route, as well as their overall climb time.
5. **Requirement:** Offers route setting screen for inputting hold ids into new/existing routes.
   Verification: Observing the web server shows that admins have the ability to select holds to turn into routes.
6. **Requirement:** Functional admin panel to set holds to a certain color, to turn holds off, to receive information about battery life of holds.
   Verification: The current state of the admin panel allows all of the features listed above except for the battery life observation. This is because we did not end up using the same battery protection IC that we originally intended to for the final product.
7. **Requirement:** Route setting mode to select certain hold ids on a wall and save them as new routes.
   Verification: Included as a feature in the route setting portion of the server. Seen by observing on the admin and user ends the list of all the holds previously set.

3.8 Web Server

1. **Requirement:** Should be able to handle data from at least five holds.
   Verification: We have verified the data transfer capabilities for up to eight holds by running all eight at the same time under functional conditions successfully.
2. **Requirement:** Should be able to serve at least three clients simultaneously.
   Verification: We tested a configuration of eight holds by referring to each on the software end as an individual client. Once again, this was all done successfully under full functionality conditions.

3.9 Physical Design

1. **Requirement:** All components for a single hold should be able to fit within any hold.
   Verification: After assembling the holds, we verified that no electronic components either cannot fit the hold or are sticking out of the hold, except for the LED, which is intentional.
2. **Requirement:** Hold should be durable enough to hold a minimum of 400 lbs of force for at least 1 minute
   Verification: We verified with the machine shop that the bolt supporting the hold should be able to support much more than 400 pounds due to not having changed any of the original structure of the holds.

4. Costs and Schedule

4.1. Costs

4.1.1 Labor

At 16 weeks, at 10 hours a week, each group member to spent around 160 hours on this project. If we estimate our labor at a $35,000 salary, cost per hour (assuming 40 hours per week for 50 weeks per year) comes out to $17.50. This therefore makes our total student labor cost:

$$\frac{\$17.50}{hour*person} * 160\ hours * 3\ people * 2.5 = \$21,000$$

The machine shop took one worker a day to complete all eight holds. At a rough estimate of $50/hour of work for the machine shop, this makes total machine shop labor:

$$\frac{\$50.00}{hour} * 8\ hours = \$400$$

Therefore, the total labor cost is $21,400.

4.1.2 Parts

See Appendix B, Table 1.

Total cost of parts is **$272.08**

4.1.3 Total Cost

$$Total\ Cost = Labor + Parts = \$21,672.08$$

4.2. Schedule

See Appendix B, Table 2.

5. Conclusion

5.1. Accomplishments

Overall the majority of the project is fully completed. Currently all holds can communicate with the server and are able to be set to certain states and colors via the web application. The web application is easy to use and features a way to create routes, look at detailed information about the wall. Furthermore, when a route is set to climb, the holds in the route send back events, which trigger when a climber grabs and releases a hold. This list of events that occured are stored and are available for viewing after a climb. The server is able to connect to all eight holds while simultaneously serving a number of different clients. Therefore, our project successfully accomplishes all three solutions to the problems set out in the introduction. The LEDs are able to be set through the server, and are easy to see and make climbing a route much easier. The holds are able to keep track of a climber's progress during a route, and sends this information back to the server. All data associated with a climber's account is stored, and ready to be viewed. Gyms are able to see what routes are being climber through the database and thus interactivity between gyms and climbers is solved. Finally, the add route button in the admin panel allows the quick and easy creation of route given existing, registered holds on a wall.

5.2. Uncertainties

There were two major uncertainties in our final design that require further analysis for potential alternative solutions. The first point of uncertainty is our Hall effect sensor, and the second point is with the battery.

As mentioned in previous sections, the Hall effect sensor's final sensing range came to less than the desired 5 cm. The reason why this inconsistency mainly occured is due to the fact that we had to make a tough compromise between magnet size and sensing range, and ultimately chose to favor magnet size slightly. In particular, it may be beneficial in the future to use a slightly more sensitive Hall effect sensor, but our thought process when making our decision was that the sensing range still surpassed the required physical length. Another issue that stemmed after we had everything in front of us physically was that our sensing assumed that the magnet would be directly over the sensor, with a few centimeters of tolerance. When we actually demonstrated some of the holding positions, however, we quickly noticed that the magnet would not always be exactly where we want it, especially since we only have a single sensor in each hold. In order to remedy this issue slightly, we decided to place the Hall effect right next to the LED on the physical design, which would ultimately give the climber a reference point for the location of the sensor should they have trouble getting sensed. By doing so, during our final testing we found it much easier to achieve high trigger rates on the sensor with our magnets, as ultimately intended.

Similar to the sensor uncertainties, we had similar physical issues when it came to an optimal battery setup, particularly from a practical standpoint. We have always had a consistent issue with

how to power our system wirelessly, since that was the ultimate design angle we wanted to pursue. We originally looked into using lithium ion batteries as mentioned before, but could not find batteries within a satisfactory price range that satisfied all of our requirements. Because of this, we had to compromise and use alkaline batteries, which ultimately removed the need to have a battery protection IC circuit as well. This change worked out for us in the end, but we also put heavy consideration into using smaller batteries, and other potential wireless powering options.

5.3. Future Work and Alternatives

The nature of climbing being both a competitive and recreational sport should give enough reason to create a rating system. By assigning climbers with a skill level, people will be more inclined to train harder to increase their rating. Furthermore, by calculating a climber's average skill level, it will make adjusting route difficulties much easier. If a strong climber fails a route which is meant to be easy, the route difficulty can be automatically adjusted.

Climbers often get stuck at route's because they do not know how to complete the next move to go up the route. By storing a completed climb on the route with detailed information, the climber could effectively request a "ghost climber" to demonstrate how to complete the route. By lighting LEDs in a certain way on holds, like red for left hand, green for right hand, etc., a climber could view how exactly to complete a route without asking others for help. Furthermore, a climber could race an old climb on their way to the top.

5.4. Ethical Considerations

Throughout the project, the primary ethical concern of the project has been the safety of climbers while using our project [8]. Many of the safety issues we had have been completely removed after testing and design decisions. The structural integrity of the hold was not compromised after the holds came hollowed out initially. This meant we could store out electronics inside the hold without damaging its internal structure by milling out sections of the hold. We only changed the holds structure by adding tiny wire channels at the cross sectional walls of the hold, not affecting the structural integrity at all. Furthermore, we were concerned with use a lithium-ion battery inside out holds as they can be fairly dangerous in the wrong circumstances. To circumvent this issue, we instead opter for the safer, although less efficient, standard alkaline batteries to power our climbing holds. Our final safety concern was whether or not any chalk or sweat could enter the holds and cause damage to the internal electronics. However, the machine shop created a plate of plastic that is attached to the back of the hold keep the electronics inside secure. Furthermore, the one hole we drilled for the LED which is on the front of the hold is fully sealed with a translucent acrylic paste. One of the tests we completed for the hold integrity demonstrated it could successfully hold around 400 pounds of weight without any damage. With all of these factors considered, it is safe to say our holds are incredibly safe to climb and will cause no harm to any climbers.

References

[1] Ti.com. (2018). LM317 3-Terminal Adjustable Regulator [online] Available: http://www.ti.com/lit/ds/symlink/lm317.pdf [Accessed 1 May 2018].
[2] Ti.com. (2018). DRV5032 datasheet Ultra-Low Power 1.65V to 5.5V Hall Effect Switch Sensor | TI.com. [online] Available: http://www.ti.com/product/DRV5032/datasheet/abstract#x5369 [Accessed 1 May. 2018].
[3] ww1.microchip.com. (2018). PIC16(L)F18857/77 Data Sheet. [online] Available: http://ww1.microchip.com/downloads/en/DeviceDoc/PIC16(L)F18857_77%20Data%20Sheet_%20DS40001825C.pdf [Accessed 1 May. 2018].
[4] Microchip.com. (2018). MPLAB Code Configurator | Microchip Technology. [online] Available: http://www.microchip.com/mplab/mplab-code-configurator [Accessed 1 May 2018].
[5] Raspberry Pi. (2018). Raspberry Pi - Teach, Learn, and Make with Raspberry Pi. [online] Available: https://www.raspberrypi.org/ [Accessed 1 May 2018].
[6] Docker. (2018). Docker. [online] Available: https://www.docker.com [Accessed 1 May 2018].
[7] Docker Documentation. (2018). Docker Compose. [online] Available: https://docs.docker.com/compose [Accessed 1 May 2018].
[8] Ieee.org. (2018). IEEE IEEE Code of Ethics. [online] Available: https://www.ieee.org/about/corporate/governance/p7-8.html [Accessed 1 May. 2018].

Appendix A: Hardware Circuit Schematics

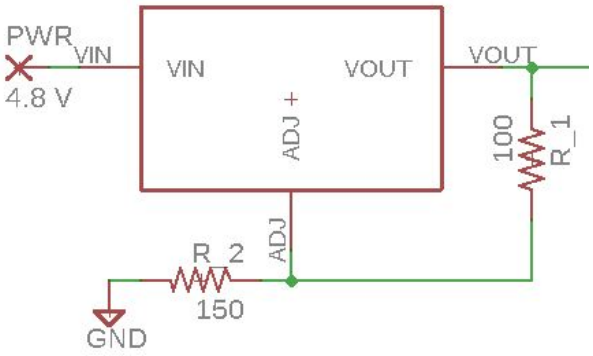

Figure 1: Voltage Regulator Circuit Schematic

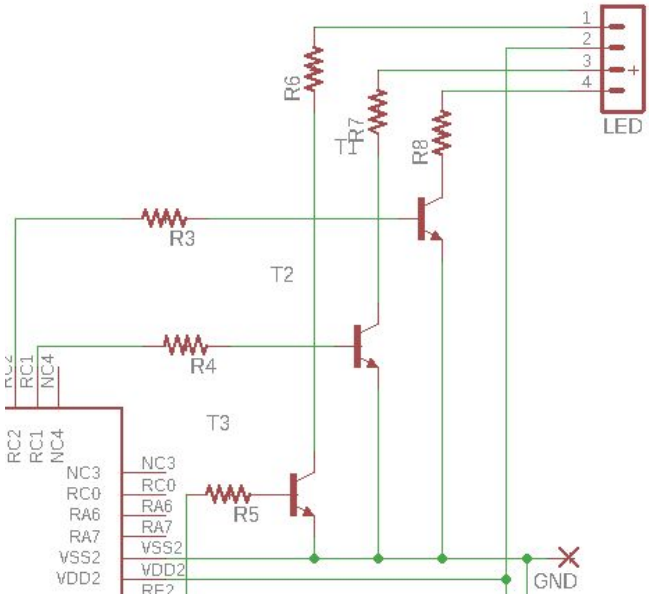

Figure 2: RGB LED Control Circuit Schematic

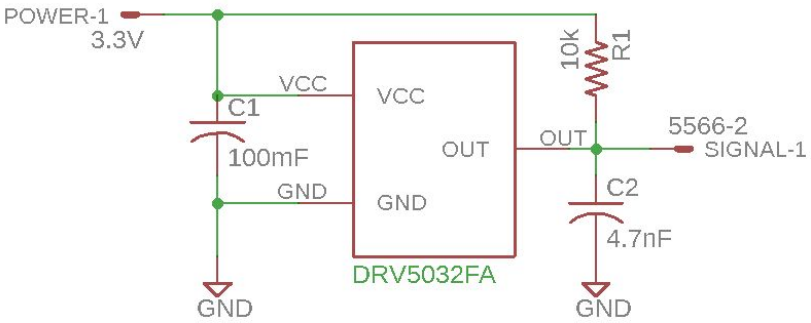

Figure 3: Hall Effect Sensor Circuit Schematic

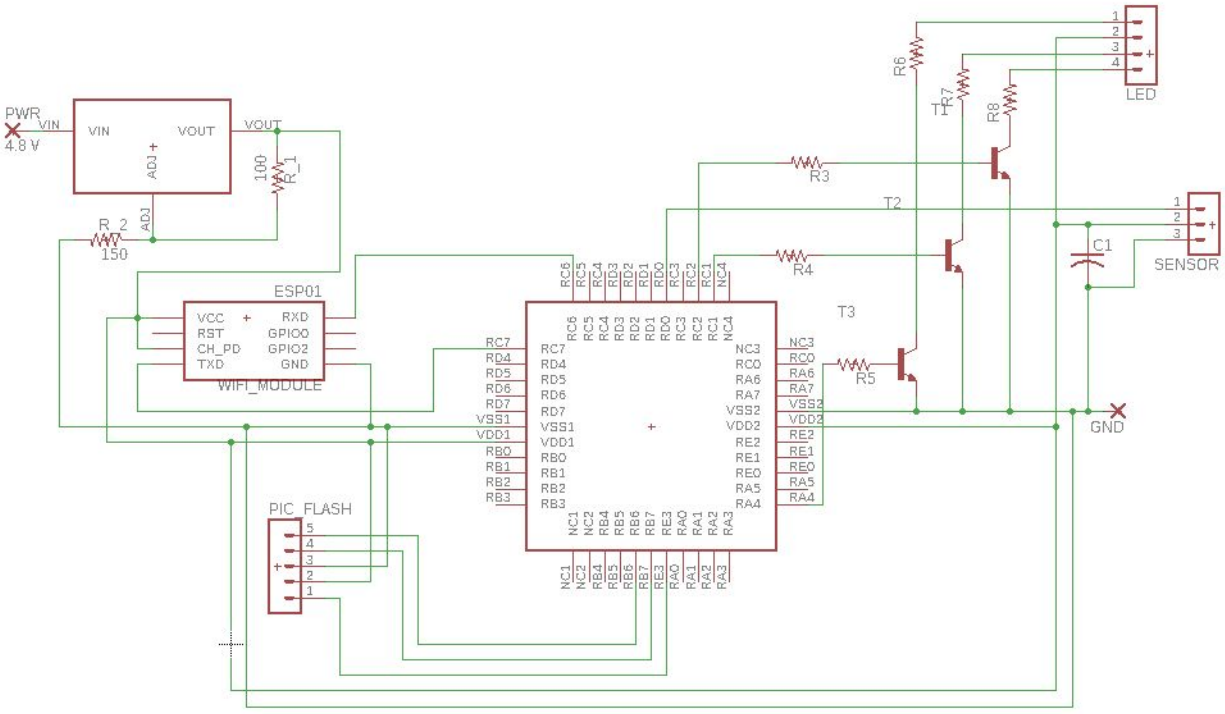

Figure 4: Primary Control Board Schematic

Appendix B: Important Tables

Part(s) Name	Quantity	Total Cost
Hall Effect Sensor (DRV5032FADBZR), Texas Instruments	4	$0.72
Wifi Module (ESP8266), Espressif Systems	8	$25.98
Microcontroller (PIC16LF18877-E/PT), Microchip Technology	8	$14.96
RGB LED (YSL-R596CR3G4B5C-C10), China Young Sun Led Technology Company	8	$15.60
Battery, Alkaline AA	30	$25.89
Climbing Holds, XL Basic Sloper	5	$87.00
Magnets	6	$10.99
Raspberry Pi 3 Model B and PSU, Raspberry Pi Foundation	1	$42.99
Microcontroller Flashing Tool (PICKIT), Microchip Technology	1	$47.95

Table 1: Parts Cost Analysis

Date	Jishnu	Brian	Daniel
2/19/2018	Decided on/order microcontroller and wifi module	Decided on/order hall effect sensor and associated magnets	Decided on/order Climbing Holds, RGB LEDS, misc electronics
2/26/2018	Designed front-end on paper	Finalized order for machine shop	Designed backend on paper
3/5/2018	Began frontend and compete skeleton of frontend	Designed PCB for Microcontroller and parts	Began backend and compete skeleton of backend
3/12/2018	Completed basic functionality of front-end and tested with dummy data	Began working on PIC and Hall Effect Sensor	Completed basic functionality of backend and tested with dummy data
3/19/2018	Complete front-end functionality	Began working on PIC and LED	Began working on PIC and WiFi Module
3/26/2018	Began working on and WiFi module and server	Fixed issue with PIC and WiFi module completed work with PIC and LED/ Hall	Integrated front-end and backend

		Effect	
4/2/2018	Integrated front-end and backend	Fixed PCB bugs and begin testing PCBs	Completed backend and fixed PCB bugs
4/9/2018	Worked on PIC to integrate all modules together	Soldered all hardware together	Soldered all hardware together, worked on PIC
4/16/2018	Finished hold software on PIC	Finalized battery module and integrating hardware	Finished hold software on PIC and helping finish hardware
4/23/2018	Prepared final demo	Prepared final demo	Prepared final demo
4/30/2018	Prepared final presentation	Prepared final presentation	Prepared final presentation

Table 2: Schedule Table by Week

Trial	Sensor	Sensing Distance (mm)	Trial	Sensor	Sensing Distance (mm)
1	1	38	11	3	40
2	1	38	12	3	41
3	1	39	13	3	39
4	1	37	14	3	39
5	1	39	15	3	40
6	2	41	16	4	38
7	2	41	17	4	39
8	2	41	18	4	39
9	2	40	19	4	37
10	2	42	20	4	38

Table 3: Hall Effect Verification Test Results