Climbing Log and Information Management Bundle (CLIMB) with Linux, Apache, MySQL, and PHP (LAMP)

Paul Kummer

Minnesota State University Moorhead, MSUM

cx3645kg@go.minnstate.edu, pakummer@gmail.com

***Abstract* – With the progression of time, climbers are spending more time on artificial climbing walls and using technology to keep track of their climbing history and progression. This paper discusses a project that allowed users to access their climbing information from a web-based interface. Similar application exists, and they are compared. Also, the design and implementation of the project's webserver, database, and remotely controlled device are discussed. All the systems worked together, allowing climbers to log into a website that could track what climbs they completed and when they completed them. Furthermore, information about current climbing routes could have been accessed to help the climbers select their next climbs. To ensure everything worked properly, testing was done. Some bugs in the software were known about before the tests, but other bugs were discovered after. If the project was to be develop further, future goals of the project are elaborated. Altogether, the project allowed climbers to get information about their climbing history and progression with the added ability to interact with their climbing environment remotely.**

***Index Terms* – bouldering, C++, cascade style sheet (CSS), climbing, CLIMB, computer science and information systems (CSIS), hypertext markup language (HTML), LAMP, PHP hypertext preprocessor (PHP), Raspberry Pi, route management, web application.**

I. INTRODUCTION

The CLIMB project's purpose was to give climbers the ability to manage their climbing sessions from a webpage. Each session involved information about when the session occurred, the route climbed, and the difficulty of the routes. Additionally, after the user logged in, they could search for routes to get its difficulty and any additional information that may be associated with the route. From the same webpage, the user was able to select an angle they wanted the climbing wall to be at, which would then be changed on the actual climbing wall through CLIMB's networking capabilities. Many parts of project had been replicated in other works. However, CLIMB attempted to integrate many facets of climbing sessions into one location.

II. RELATED WORKS

When the project was created, there were commercially available climbing walls that could change climbing angle such as the Moon Board [1] and the Kilter Board [2]. These climbing walls were intended for training and offered their users a database of an ever-expanding amount of climbing routes they could choose to climb. Both products had their own free applications that allowed users to track what climbs they completed and search for their next route to climb based on a variety of criteria like difficulty range, route setter, number of repeats, involves a specific hold, highest rating, etc. More expensive versions of both climbing walls even allowed the user to light up the holds of the climbing routes they wanted to climb. However, to adjust the angle of the climbing wall, the user had to activate a switch that adjusted the angle. Both applications were proprietary, only working for their respective product.

Some applications were more focused on climbing routes and not on a proprietary product. Some applications were free to use and allowed users to search for routes based on their location. Mountain Project [3] was one such application. The application allowed users to search for outdoor climbing routes on natural rock, add new routes outdoors, add helpful information about routes, and track what they have climbed. People primarily used the application as a digital guide, so they would know what to expect when climbing outdoors. Another application to help users with their climbs was Eat Spray Love [4]. This application was intended for noncommercial climbing walls and allowed users to add their climbing walls and its corresponding routes to a database. Then regular people could track their climbing routes like the proprietary climbing applications.

The project took important aspects of existing application designs and attempted to merge them into one location. Since all the applications had their own unique features, some sacrifices had to be made to meet a simpler set of system requirements. Most applications based around climbing routes shared some common designs. They had a database of existing climbing routes and each route had information associated with it like difficulty, who established it, and some helpful information. Also, the user could search the database for routes meeting their criteria. CLIMB achieved storing and accessing route and user information in database like the other applications, but unlike some applications it was not proprietary. One added feature that is missing from other apps was the ability to adjust a climbing wall's angle without physical contact.

III. DESIGN

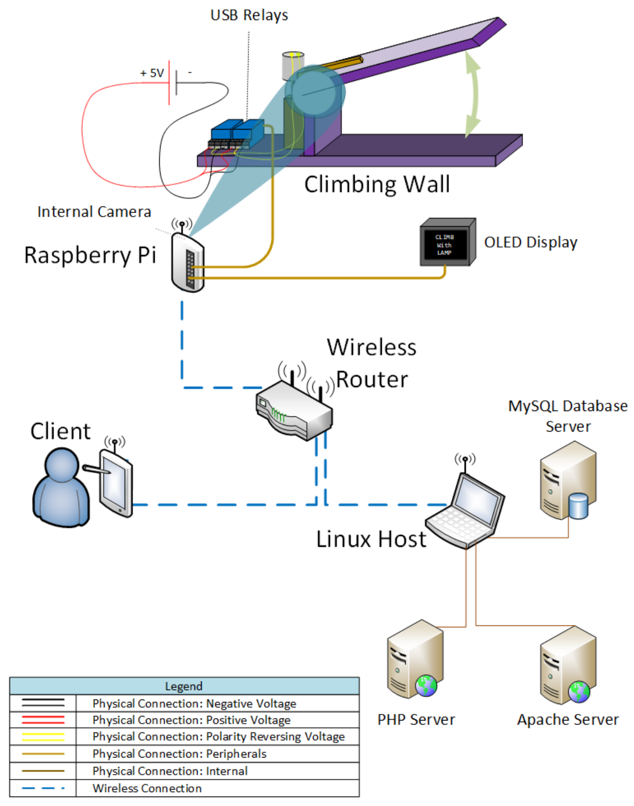
USB Relays
+ 5V
-
Climbing Wall
Internal Camera
Raspberry Pi
OLED Display
Wireless Router
Client
MySQL Database Server
Linux Host
PHP Server
Apache Server
Legend
Physical Connection: Negative Voltage
Physical Connection: Positive Voltage
Physical Connection: Polarity Reversing Voltage
Physical Connection: Peripherals
Physical Connection: Internal
Wireless Connection

Figure 1

The design began by determining the requirements for the system. The project had to do the following:

- Adjust the climbing wall's angle
- Manage users' accounts
- Track users' climbs
- Allow searching for routes
- Manage route information

To meet the requirements, the project was broken into separate components consisting of the Raspberry Pi application, web application, and networking. A representation of the system is depicted in Fig. 1 with the Linux Host delivering what is referred to as the web application to the client over a wireless network.

A. Raspberry Pi Application

The Raspberry Pi component of the project determined the angle of the climbing wall and changed the angle upon a user's request from the webpage interface. Since this part of the project was developed independently of the other components, the user request was initially received as a command line argument, but later converted to a network message. Next, the Raspberry Pi began capturing images with an internal camera and analyzed them to estimate the angle of the object. Then the Raspberry Pi issued commands to its universal serial bus (USB) controlled relays that resulted in the angle controlling mechanism of the climbing wall to signal movement of up, down, or stop. While this process is occurred, an organic light-emitting diode (OLED) display was updated, allowing the user to know what the climbing wall's status was.

B. Web Application

Another component of the project was the users' interface. The interface was created through a webpage that the user got from a LAMP stack. The LAMP stack was three separate servers running on a Linux host computer. Each server worked with another server to attempt to fulfill the user's request. In Fig 1., the servers can be seen as components of the Linux host.

The process began with a client request to the Apache server. Since what the user was requesting was a PHP file, the Apache server communicated with the PHP server to process the file. During the processing of the PHP scripts, the PHP server would communicate with the MySQL server when the script needed information from the database or the Raspberry Pi if the climbing wall's angle was being changed. After the PHP server processed the PHP file, it returned a HTML document to the Apache server that was sent back to the user.

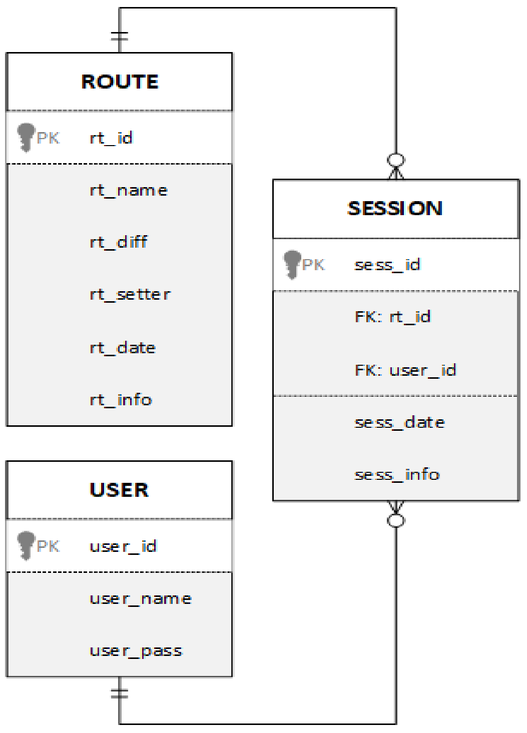

Figure 2

Information that was stored about users and routes was stored on the MySQL database. The database design was simple and consisted of three tables. Two of the primary tables were ROUTE and USER. These tables were connected through a linking table called SESSION. A linking table allowed both a route and a user to be used in zero or many sessions as shown in Fig. 2.

C. Networking

For the entire project and its components to work together, everything had to able to communicate on a wireless network using transmission control protocol (TCP). The Raspberry Pi had to communicate with the PHP server to receive messages containing wall angles and the Apache server had to receive a request and send an HTML file to the requesting user. Since the Apache server, MySQL server, and PHP server were all running on the same host, there were no additional wireless networking steps needed for inter-server communication. They were all done on the local host that required some authentication for the MySQL server. However, some design changes could have resulted in additional networking requirements if those servers were to be located on different hosts. The connections between components are shown in Fig. 1 by using different colors and styles of lines.

IV. IMPLEMENTATION

The project had two components that could work independently of each other. First, a program running on the Raspberry Pi could run as a stand-alone program that was solely responsible for changing the angle of the climbing wall. Second, a web application could work separately except for being able to adjust the climbing wall angle; Its primary purpose was to keep track of routes and the climbing history of climbers. All the project's files could be found in the author's GitHub repository [5].

A. Raspberry Pi Application

1) Setup: Initially, the Raspberry Pi had to be setup with an environment that could compile and run C++ code. This was accomplished by installing the operating system (OS) of Raspbian GNU/Linux 10 (buster) on the Raspberry Pi [6]. Then the angle controlling program could be compiled and ran.

2) Creation: Each API has its own objects and libraries that were needed before the creation of the executable "adjustAngle", the main program. These requisite files had to be made with either the "make" command or a shell script. After they were compiled with their various methods, object files were created and linked. The following folders contained either an API or a C++ file that was compiled separately to get the required objects or libraries needed by the "adjustAngle" executable:

- SSD1306_OLED_RPI-main [7]
- raspicam-master [8]
- makeDispObj
- HIDUSBRelay [9]

Some files had to be moved so the makefile for "adjustAngle" could locate them. The file "usb_relay_device.so" was in the subdirectory "RaspberryPi_App/libs" and "dispFunctions.o" was in the primary directory of "RaspberryPi_App." Also, some shared libraires had to be setup using steps from [10] and communication to the relays was configured to address each relay by following instructions from [11].

Next the executable program that controlled the wall angle called "adjustAngle" was made. Creation of the executable was done by calling "make" in a terminal of the directory that the project resided. After typing the command, the executable "adjustAngle" was created in the working directory.

3) Execution: Additional configuration of the Raspberry Pi's OS was done to automatically run the angle controlling program when the OS booted. The approach used was the creation of a service unit as shown in [12]. When the OS booted, the new service unit executed the angle control program "adjustAngle" because it was wanted by "multi-user.target", which was automatically ran when loading user profiles.

4) Directories and Code: The program "adjustAngle" was written in C++ and was split into multiple files to help keep related code together. Many files were also in separate directories to promote the organization. Directories were used to separate APIs, program functions, program headers, instructional material, libraries, and tests. Some unsorted files resided in the primary directory.

a) Parent directory: In the "RaspberryPi_App" directory all the files and directories needed to create the primary program "adjustAngle" were located. The files were C++, headers, objects, makefiles, or text. The main line of logic for the program was contained in the main.cpp file within this folder. Additionally, some object files needed for compilation were in this directory. Depending on the program's settings, captured images that were analyzed may have been there too.

i) C++ files: The main logic for the "adjustAngle" executable was in the main.cpp file. To begin, the code included the "prototypes.h" header file and the "functions.cpp" file. The "functions.cpp" file was included instead of compiled as an object because there were some unresolved conflicts with the makefile. Then the program initialized variables that were needed in the main function and other functions. These variables were responsible for keeping track of whether the initialization of the program was successful, if the desired angle was reached, the angle of the wall, making a relay object, making an image object, and holding the status of the wall.

After the variables were instantiated, the API [7] controlled peripherals were initialized. First the OLED object was created with its initialization method. Since the display needed to be updated frequently while other functions were executed in the program, the display ran in its own thread that had a looping function, running parallel to the rest of the program. To get updated angle information displayed, pointers were used to pass the angle measurement that could be changed both inside and outside of the thread. Use of threads was chosen over forking a new process for a very important reason. With the use of threads, variables could be shared among all the threads of a process. If the fork function was called, a new identical process would be created with its own variables that could not be shared with the parent process as discussed in [13].

Next the camera was initialized by creating a camera object with the “raspicam” API [8]. The camera object became active when its method named “open” was called. Prior to starting the camera, its parameters like format, width, height, brightness, ISO, and sharpness were set.

The last object initialized was the relays. They used the “HIDUSBRelay” API [9]. Like the other peripherals, it had its own object created with its own initialization method. Once the object was created, the relays changed to their open state, which gave physical evidence that they were working by clicking and their corresponding LEDs being illuminated.

Afterwards, the main function went into a loop if everything was initialized without problems. This loop would continue until the shutdown signal of “-1” was received. In this loop a function was called that listened for a message over a TCP connection containing a wall angle as was demonstrated in a tutorial from [14]. Once this message was received and verified, another loop started that called functions to capture and analyze images and to control the relays if the angle had to be adjusted. The inner loop would only stop when the desired angle was reached, or the shutdown signal was received.

Upon receiving a shutdown command or a failed initialization of a peripheral, the program would begin a shutdown sequence. It began by waiting for the display thread to complete its function and join the main thread as was shown in [12]. The display thread had its own shutdown signal stored in and enumerable type. After the thread joined the main thread, every peripheral switched to an off state and released its resources. Finally, the program would return the integer “0”, signaling a successful execution.

ii) Header files: Within the directory “headers”, there were three header files. These files were “bmp.h”, “prototypes.h”, and “usb_relay_device.h”.

The “bmp.h” file was responsible for the definition of the BMP image class that was described and demonstrated in [15]. Also, a pixel class was created to make working with an image and its pixels easier. The “prototypes.h” file had constants used in the program and forward declarations of functions. With the forward declarations, the functions could be used at any point of the program since the function names and signatures would be known at the time of compilation. Lastly, the file “usb_relay_device.h” was from [9] and holds information needed to control the relays.

iii) Object files: There was one custom built object file called “dispfunctions.o” that utilized the API [7] needed for controlling the OLED display. Despite using the API, this object file had some additional functionality that was added to customize what the API could do, like scroll an indicator upwards and display a scale.

iv) Makefile: When executed, the file “makefile” ran a script that automated compiling or removing compiled code. After successful execution, the “adjustAngle” file was made.

v) Text: There was a readme file that was intended to provide anyone interested in the code an overview of what the project was and what to expect from the files it contains.

b) Functions: One C++ file resided in the “functions” directory. It contained all the programmatic functions not part of the APIs written specifically for this project.

i) Image capturing: One function’s primary task was to capture an image and call other functions to analyze the captured image. When this function was called, it would prepare the camera object to capture an image. During this process, the camera would have a small delay to give it time to setup and take a picture. Once the image was captured, the data was sent to other functions for further processing. The first function it called cleaned the image so that only pertinent information remained. Then with the important information remaining, another function call was made to handle the angle interpretation. Lastly, an optional function call could be made if an output image was desired so that results could be visual inspected to ensure that everything performed as it should.

ii) BMP images: Many functions were used only for managing, manipulating, and creating BMP images. When all these functions were used, an image was outputted that had been transformed to black-and-white and had a colored line representing the measured angle drawn on the image. One function transformed the image to black-and-white and was especially important because it removed erroneous pixel data of points that should not have been measured when calculating the image’s angle. Depending on the lighting conditions and what the image to be captured was, the threshold value for erroneous pixel occasionally was changed. An overview of the process and the structure of a Bitmap image are shown in Fig. 3.

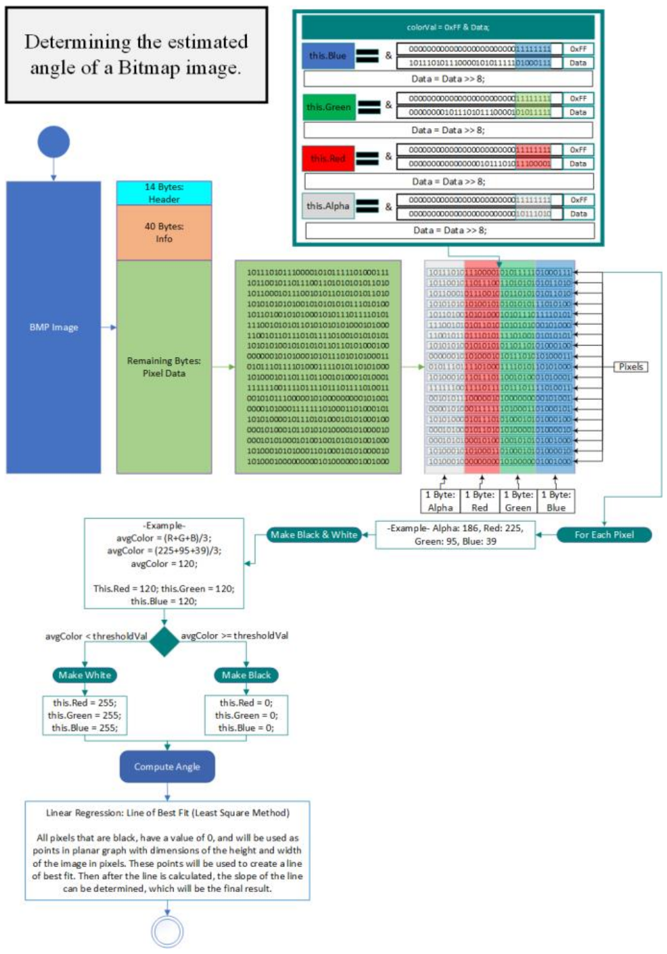
Determining the estimated angle of a Bitmap image.
colorVal = 0xFF & Data;
this.Blue
&
00000000000000000000000011111111
0xFF
10111010111000010101111101000111
Data
Data = Data >> 8;
this.Green
00000000000000000000000011111111
0000000010111010111000010101111
Data = Data >> 8;
this.Red
0000000000000000000000011111111
00000000000000001011101011100001
Data = Data >> 8;
this.Alpha
00000000000000000000000011111111
00000000000000000000000010111010
Data = Data >> 8;
14 Bytes: Header
40 Bytes: Info
BMP Image
Remaining Bytes: Pixel Data
Pixels
1 Byte: Alpha
1 Byte: Red
1 Byte: Green
1 Byte: Blue
For Each Pixel
-Example- Alpha: 186, Red: 225, Green: 95, Blue: 39
Make Black & White
-Example-
avgColor = (R+G+B)/3;
avgColor = (225+95+39)/3;
avgColor = 120;
This.Red = 120; this.Green = 120; this.Blue = 120;
avgColor < thresholdVal
avgColor >= thresholdVal
Make White
this.Red = 255; this.Green = 255; this.Blue = 255;
Make Black
this.Red = 0; this.Green = 0; this.Blue = 0;
Compute Angle
Linear Regression: Line of Best Fit (Least Square Method)
All pixels that are black, have a value of 0, and will be used as points in planar graph with dimensions of the height and width of the image in pixels. These points will be used to create a line of best fit. Then after the line is calculated, the slope of the line can be determined, which will be the final result.

Figure 3

Four other functions worked together and were strictly for making a new Bitmap image. They setup the Bitmap's header and changed the pixel data so it could be saved correctly and interpreted by image reading software when opened. Information from [15] was used to perform correctly construct the Bitmap.

iii) Angle algorithm: To determine the line-of-best-fit and its goodness-of-fit and angle, one function used linear regression with the least squared method, described in [16], and trigonometry. Every pixel from the captured image that had red, green, and blue values of 0 were considered a point on plane. While points were added, the x and y coordinates of each pixel were used to create additional values for formulas needed to find the line-of-best-fit and the goodness-of-fit. With the goodness-of-fit value, the likelihood that the line-of-best-fit represents the actual data could be determined and could indicate if there are any issues capturing a valid image.

iv) Relays: Climbing wall movement was controlled by one function. The function changed the state of two separate relays to either increase, maintain, or decrease the angle of the climbing wall. Each relay had two states that could be in, open or closed, meaning there were four different possibilities of combined states. When the project was made, the relays were setup for polarity reversing the motors power source. However, for actual implementation on a real climbing wall, each relay should be independently responsible for either sending an up or down signal by completing a circuit. Therefore, the "controlSwitch" function should be changed for any configuration that is not the same as the prototype climbing wall used in the project.

The relay controlling function did not loop and had to be repetitively called to update a pointer used by the display's functions and to stop any movement that may have been started. Once the relay state was changed, it would remain unchanged until a new signal was received. When the function was called, the climbing wall would move unless the measured angle was within the tolerance value of two degrees from the desired angle.

v) Networking: Communication with the LAMP stack was performed by one function. The function caused the loop in the main function to pause while it waited to receive a message over a TCP connection on port 8080, demonstrated in a tutorial from [14]. When a connection occurs, a message was received, and the main function's loop continued. The message that was received should have been a valid angle for the climbing wall to be adjusted.

c) Headers: No files existed in the "headers" directory. The intended purpose of the directory was to have all the header files required to create "adjustAngle" located inside.

d) Relay API: The files needed by the API [9] to control the relays resided in the "HIDUSBRelay" directory. There were some test files that could be used after compiling the code that used the API and were useful in determining if the relays and the API were functioning correctly.

e) Instructions: Instructions for how to setup the Raspberry Pi so that the APIs would work could be found in the "instructions" directory.

f) Libraries: The purpose of the "libs" directory was to store any libraries that were needed by the project. One shared object file was inside.

g) Custom display API: Custom code was written to integrate the API [7] into this project. The API didn't include some desired functionality like making a scale and having vertically scrolling shapes. The issues were solved by creating a separate program in the directory "MakeDispObj." By creating a separate program, a new object file was created and used by the main program.

h) Camera API: The API [8] that controls the Raspberry Pi's camera was contained in the "raspicam-master" directory. After the API was compiled, the libraries and objects needed in the main program were created.

i) OLED API: The “SSD1306_OLED_RPI-main*”* directory held the files for the API [7] that controled the OLED display connected to the Raspberry Pi. Like the other APIs, the files in this directory had to be compiled first, and it also contained some tests to ensure the functionality of the API and the display.

j) Tests: In the “tests” directory there were some custom test programs written during the creation of this project to ensure that the APIs and other code were working correctly and could be incorporated into the main program. There were test programs for the Raspberry Pi’s camera, relays, and network connectivity.

B) Web Application

1) Setup: For the web application to work, the LAMP stack had to be setup first. There were numerous tutorials on how to setup the stack. The project used the steps from [17]. The stack installation started by installing the Linux OS on the computer that hosted the Apache, MySQL, and PHP servers. After the OS was installed, the steps to install the servers required only typing in some installation commands as the super-user in a terminal window.

Once the servers had been installed, some configurations needed to be done to ensure the servers could communicate on the network. Ports 80, 443, and 3306 were ensured to be opened and not blocked by any firewall, allowing http and https traffic to the Apache server and queries to the database. Also, the MySQL server needed to be configured with a username and password. Since the MySQL server was on the same host as the other server, the server’s host was localhost addressed as 127.0.0.1 in IPv4, which was a loopback address. The PHP server was installed with the “libapache2-mod-php” and the “php-mysql”, leading to no other configurations being required. However, it may have been beneficial to make the Apache server have PHP files take precedence over HTML. This could have been done by modifying the Apache server’s “dir.conf” file and changing “index.php” to be listed before any other file. Steps to accomplish all the tasks were described in [16].

To make modifications of the MySQL database easier, MySQL Workbench [18] was downloaded and installed. The software provided a graphic user interface (GUI) to the MySQL databases and its configurations.

2) Creation: The creation of the web application was accomplished with two parts. The first part was the webpage that was delivered to the user and the second was the MySQL database that held the information used by the webpage.

a) Webpage: Whenever a user made a request to the host computer, the Apache server attempted to deliver the main webpage. However, since the main webpage was a PHP file the PHP server had to process the file first. The main page used information from many other PHP files. Multiple PHP files were used to group related PHP code so the main webpage, “index.php” was easier to understand.

Making the webpage accessible on the network, PHP and CSS files were placed into the Apache’s webpage directory as was described in [17]. In this project the files were located at “var/www/html”. Once the files were in the directory, the Apache webserver was restarted. Then the webpage was accessible to anyone on the network by them navigating to the IP address of the host on a web browser.

b) Database: For the creation of the database, [18] was used, and a new database was created. In the project, a separate user account was created for the database that had restricted rights to prevent malicious actors from gaining access in clever ways to information they shouldn’t.

Once the database existed, the tables, functions, and stored procedures were added. There were three tables as shown in the entity relationship diagram of Fig. 3. The tables, functions, and stored procedures could all be created by executing the query files of “storedProcedureAndFunctions” and “tablesAndData.” Afterwards, the database was setup and allowed data to be added.

3) Execution: With the host and servers operational, the servers were restarted. Updated settings were in effect after the restart. Then execution of the PHP scripts was done automatically every time a client requested the main webpage from the host. There were times when the PHP file was executing that the MySQL server was accessed automatically by the PHP scripts. The database would be accessed for creating users, authenticating users, anytime route information was added or needed, and in many other circumstances when the client was using the webpage. Most of the time the database was accessed, a stored procedure was being executed to perform pre-written queries.

4) Directories and Code: The files needed by the web application were all located in one directory. Within this directory there was also a subdirectory that contained files for setting up the database. The database files were not required to be where they were, but for convenience of locating web application related files they were put into the webpage’s directory.

a) Working directory: On the host computer, there was a directory called “WWW” that held all the files needed by the Apache and PHP server to generate a HTML webpage. The files were split into related categories to make finding and modifying code easier.

i) Webpage: In the web application’s directory, there were six PHP files used to generate HTML. The first file that the Apache server accessed was “index.php.” This was the main file needed to create the webpage. All other PHP files were imported when the PHP server read the file.

The “index.php” file used sessions so that information between requests was persistent, meaning variable information wasn’t lost every time a new page was created. The type of webpage generated was dependent on the values associated with buttons from the webpage. Whatever the value was, a switch statement determined what HTML to use, which was pre-written in the “forms.php” file. At the end of the “index.php” file was the HTML code for the base layout of the webpage. Code generated from the PHP script was inserted into the HTML layout, which was what the Apache server returned to the client. Review of how to make the sessions and other PHP functions work was reviewed on tutorials from [14], [19].

ii) Functions: Two files were used to hold the functions that created the main webpage. One file was used for general functions and the other was for functions relating to the database.

The general function file, “functions.php”, also held constant values since there were not many functions or constants. There were three functions in the file, and they were used to alternate a Boolean value, send a message to the Raspberry Pi, and to ensure a chosen wall angle is valid.

Separate functions for the database were grouped together in the file “dbFunctions.php.” All the functions called the MySQL server, and were used for generating tables, generating dropdown menus, or executing stored procedures. For the functions to work properly, they imported database information from the “dbInfo.php” file that contained the database’s hostname, name, username, and password. Since the default port and IP address were used, nothing needed to be done to specify those values.

iii) CSS: On the main webpage there was a button near the bottom that allowed the user to toggle between light and dark mode. When the webpage loaded, there were two CSS files that could be read for changing the webpage’s appearance. One file used lighter colors to make the webpage brighter and was set as the default. However, some people preferred darker colors, which could reduce eye strain. That was why the second CSS file exists. It used a dark color profile consisting of greens, grays, and black.

Other than the color profiles, both CSS were the same. They were used to help position and size HTML objects based on what the object was, what class the object was in, and where in the HTML document it was located. The CSS file also attempted to make the webpage look better on smaller screens by changing the layout of containers if the screen size was less than six hundred pixels wide. Information from [19] about CSS was used extensively for learning how to use CSS, like making a page fit on smaller screens.

b) SQL setup: The directory “sqlSetup” existed inside of the main webpage’s directory and only held files consisting of SQL queries needed to setup the MySQL database. These files were used for the following purposes:

- Create tables with data
- Create stored procedures and functions
- Create an exact replica of the existing database

There was also another subdirectory that had files for creating each stored procedure individually. Having the stored procedures separated, stored procedures could be quickly modified and added to the database.

V. TESTING

As the project was developed, portions were tested to ensure functionality before implementation. Each part could be tested independently and be modified without directly interfering with the other part. There were no unit tests ran against the system and more extensive testing could be performed.

A. Raspberry Pi Application

Testing the Raspberry Pi’s software, Each API had its own programs that verified functionality, and the main program was written in progressive steps. Additionally, some customized code was written to test if the APIs could be successfully implemented into the main program. The main program was initially written to be run in a terminal window. When it was executed, it would display real-time statistics about what the program was doing and the measurements it was taking, allowing real-time testing for erroneous data. These real-time tests proved useful for determining values for cut-off thresholds when turning the image to black-and-white and for calculating the image size to be measured. Also, information about the angle algorithm was gained and resulted in some alterations on the formula for handling conditions where the slope was zero or infinite.

The program has not been thoroughly tested with cases of invalid input data. For instance, if the user or malicious actor entered data that exceeded the limits or was of the wrong data type for a variable, the effect on the program was unknown. There needed to be more validation of input.

B. Web Application

Tests of the web application involved various inputs on the webpage and required checks of every function and stored procedure, ensuring the database performed as expected. On the webpage, there were fields that could have custom user information input and sent to the database. These fields had some SQL injection attacks attempted on them, which were shown in an example on [14]. However, since the information was read only as the data type “varchar,” the database didn’t attempt to interpret any of the data as a query.

Limited network security was tested by checking devices’ ability to communicate on the network. If this project was to be implemented on an existing network, it should be tested for network vulnerabilities. User passwords were stored using Secure Hash Algorithm 2 (SHA-2) as described in the MySQL manual [20], protecting some user information, but additional protection could be implemented to prevent database access or manipulation.

The web application had some of the same issues as the Raspberry Pi application with a lack of data validation. Therefore, there could have been potential errors that could have occurred if data too large or of the wrong type was used as input.

VI. KNOWN BUGS

The project did have some known bugs, and potentially more that could have been discovered with more testing. Some bugs were prevented by not allowing the users to get the program into the state where the bug could occur. This was done by either having the programs ignore the data or changing the data into something valid that wasn't necessarily correct but wouldn't cause the program to fail.

A. Raspberry Pi Application

The program on the Raspberry Pi was reliant on TCP communications to adjust the climbing wall angle. However, the program did nothing to verify that the message came from the PHP server, which should have been the only location that the message could have come from. This allowed someone on the network to send a message to the Raspberry Pi that could have contain invalid data. The program attempted to convert the TCP message to a float type, so if the message could not be converted to a float type an error would have occurred. A "try catch" statement could have been used to handle this issue with relative ease, but there may have been better solutions.

Another bug involved a function that controlled the relays. The function didn't do anything to prevent the relays from moving the wall to an unattainable angle. Also, if the project was implemented on a real climbing wall, there should have been safety switches to make it impossible to move the wall past its limits; It would be desirable to have a sensor to ensure that nobody was on the climbing surface while it was moving too. The safety switches would also be beneficial if the Raspberry Pi's camera began capturing bad images. If the camera malfunctioned or took inaccurate images, it was possible that the climbing wall would continuously attempt to move to an angle that was unattainable.

In some conditions, the climbing wall would move back and forth indefinitely while the program attempted to move the wall to a specified angle. This could occur because the climbing wall was moving while the program was capturing and processing image data. After the data was processed, the climbing wall would be in a different location than when the image was taken, potentially causing the wall to perpetually overshoot and undershoot the desired angle. The issue was partially solved by including an angle threshold that allowed the climbing wall to stop moving if it was within a certain range of degrees from the desired angle. However, the angle algorithm may have been introducing some error too. Based on tests of the climbing wall's actual angle compared to the measured angle, there appeared to be some discrepancies.

B. Web Application

There were a few bugs on the web application, but none were known to cause complete system failure. However, the entire project could have had more testing.

There was a bug that occurred when the user pressed the "Clear" button on the lower part of the webpage. After the button was clicked, the session data was cleared, meaning that the PHP server didn't have any button data to calculate what webpage to generate. Therefore, by default, a webpage was created with very little information. The user then had to click the "Main Index" button to go to the main login page.

A bug also existed with user accounts. There was no way for a user to know if they were using a username that already existed when creating a new account. The user would be allowed to use an existing username because all data was tied to the user's unique identifier that was assigned to them when they created an account, not their username. Problems then arose when the user attempted to log into the web application. The stored procedure that authenticated users looked for the first entry of the username and checked if the password matched, which it wouldn't unless the two users had the same password. Therefore, the second person with the shared username thought they are unable to login, but they had to use their unique user identification instead of their username to successfully login.

The route management suffered from a similar issue as the users; routes could have the exact same names because the routes' identification numbers were different, meaning that if two routes had the same name, their identification numbers would have to be used over the route's name. Also, there was no way to archive routes that no longer existed, so users would have been able to see non-existent routes.

For mobile users, the webpage wasn't always displayed in an easy-to-use format. The webpage was primarily designed to be used on a desktop or laptop computer. However, some configurations in the CSS files did make the webpage more viewable, but some features still needed improvement like the dropdown menus.

VII. FUTURE WORK

This project was developed throughout the author's education. Therefore, there were many opportunities for improving the code since some techniques and practices were not known while some code was written. Also, since the development was over a large period, some coding style changed throughout the project and may be evident in the code. When the programs were developed, they were not originally designed to be implemented together. Also, some code was written in a way that may have been logically easier to follow from the author's perspective but could cause more processing overhead. Therefore, improvements could be made to make everything work more harmoniously and efficiently. The project was considered a prototype and its repository was at [5].

A) Raspberry Pi Application

1) Image Processing: One limitation on the Raspberry Pi's program was the speed at which it processed images. For some applications where an object won't move quickly, this may not be an issue. However, the process could be quicker. Features like drawing an image with the line-of-best-fit and showing the statistics could be turned off. Also, the delay between pictures could be reduced until the camera no longer can take quality pictures. Additionally, the process of turning the image black-and-white could be skipped if pixel information is immediately read by the angle algorithm on the first iteration of the image data. It may be possible to remove the pixel array entirely and only use the image data but understanding what is happening in the process could be more difficult.

2) Safety and Failsafe: In an application that isn't a prototype, design changes should be considered to prevent the system from going into a state that could damage itself or injure people. Failure conditions could be detected when angle measurements are beyond the limits, not incrementing or decrementing in accordance with the relays' status, or not being reached within a set amount of time. For instance, if the camera begins capturing incorrect images that feeds the same angle measurement on every capture, the program should detect a failure state and stop any movement and restart, shutdown, or wait for a manual override. There could also be physical limit switches that could force movement signals to stop if any of the limits are reached. For extra safety, there could also be proximity sensors to prevent people from being on the wall or near pinch points as the wall is moving.

3) Data Validation: Messages received by the program should have been checked to ensure that the angle was within the limits and could be converted to a float type value. This validation was done when messages were received as command line arguments but was not added to messages from the TCP connection. Future development should validate more data to prevent program errors.

4) Limit Connectivity: The only device that should have been able to send messages with angles to the Raspberry Pi should have been the PHP server. Any other messages could have been from someone with nefarious intentions. Messages should be restricted to the local host, or they could be authenticated to verify they are from trusted sources.

5) Angle Overrides: On occasion, users did send messages to the program containing an angle they didn't want. The program did not allow the wall to be stopped or changed to a different angle until the requested angle was achieved. Future variations of the project should have some way to interrupt the process once it is motion to allow for the angle to be changed to something else or to stop the movement. This may mean keeping the TCP connection active until the angle is achieved, checking for a client connection between measurements, or creating a separate thread that only manages TCP messages.

6) Makefile: To make the final program, multiple compilation scripts were executed to ensure dependent files existed. Ideally, one makefile could be used to create the final product and all its dependencies. To do this, all the APIs would have to be placed into specific locations with unchanged names. If the APIs all existed in one directory, the task may be easier and increase the level of organization. Development of a monolithic makefile could be difficult with unforeseen problems, but it would be beneficial to end-users.

B) Web Application

1) Improve GUI: User interfaces should be intuitive and graphically appealing to gain acceptance by the users. The interface of the project did work but left room for enhancement.

a) Mobile devices: On some mobile platforms, the webpage was difficult to view and maneuver around. If the CSS files are developed more, the deliverable webpage could be transformed to fit different screen sizes better. Then users could have a pleasant experience no matter what device they use.

b) Graphics: The webpage had a very basic design and made no use of images or special graphics. To make the website more appealing, different color schemes and graphics should be implemented. This will give the webpage a more professional look.

c) Usability: No studies were performed to test how well people could use the website. HTML objects may be better positioned in different spots. There could also be some HTML features that were not utilized that should have been. Some input fields may work better using a slider versus a drop-down menu. Different layouts and features should be experimented with to determine what makes the webpage easiest to use and the most accessible to all people.

2) Additional Functionality

Additional functionality could be added to the webpages to enhance the user's experience. Features could be added to allow access to some content for some users, but not others. Also, more data could be used and analyzed to provide more insightful information.

a) User privileges: There was only one type of user in the system's configuration. One user had the same experience as any other user. Preferably, there would be different privilege levels for different types of users. An administrator would be allowed to access and change anything on the current system whereas a regular user would be limited to only information pertinent to them, following the principle of least privileges. Then there could be different user classes like administrator, wall owner, moderator, and user. These privilege levels would require adding more information to both the database and the PHP scripts.

b) Locations: The database was setup for only one climbing wall. However, in a real application, the system should be able to store and access information from many different climbing walls in different locations. Then users could select what climbing wall they were at and access the climbing walls information. Adding these operations would require new tables and modification of current PHP scripts.

c) Progression tracking and predictions: Being able to track routes, each user should be able to track their progression and climbing ability. The system allowed users to see their past climbs but didn't provide much insight into their climbing history. Information had to be interpreted by the user, which could be biased and misleading. Avoiding this pitfall, graphs and additional information should be provided to the user on a separate "progression" page. The webpage could give the user information like average difficulty climbed, highest difficulty climbed, number of completed climbs, average number of days between sessions, date of last session, most climbed route, and many other statistics. For more serious climbers, predictive analysis could be used to extrapolate data and predict climbing ability based on past progress. Use of artificial intelligence could be utilized to aid in predictions. Predictive abilities could be very difficult to achieve successfully and could be its own separate project.

d) Route suggestions: A feature that may be useful to climbers is route suggestions. Then climbers could spend less time searching for routes they might like. A feature like this could be simple or complex, depending on how much depth is wanted. The suggestion could be as simple as listing five routes at the user's average climbing ability and five routes at their maximum ability. The feature could also be more complex and implement route ratings, style, climb completions, steepness, and difficulty. Then based on the user's preferences, a more refined list of climbs would be suggested.

C) Organization

Directories and files needed to be more organized. The project initially started as separate projects that were never intended to become a larger project and were only ways of practicing what was taught in classes. Therefore, some of the directory structures do not follow a strict policy of placing files in specific locations. Going forward, files of specific purposes like headers, libraries, functions, and tests should be in their own directory. When this is done, files that are dependent on the files that are moved must be given directions to the new file paths.

D) Angle Measurements

Within the Raspberry Pi and Web application there were discrepancies in the way angles were measured. In the initial design of the programs, ninety degrees was considered vertical, but climbers often used the measurement of degrees from vertical, meaning zero degrees was vertical. Therefore, Translation between the two different measurement methods had to occur and caused confusion and errors. In future development, there should be a way to ensure that angle measurements are always in the correct format by always using the geometrically correct way and then only transforming it into the from ninety degrees format right before it is needed.

E) Testing

More tests need to be done on the project to ensure stability and security. The only observed failures of the project that occurred, with proper use, were from slow capture speeds and the climbing wall moving too quickly. This caused the wall to move beyond its limits. No critical software errors were known to occur but could be discovered with more extensive testing.

1) Unit Tests: No unit tests were created for the systems and could prove useful with future development. By creating an extensive unit test, the system can be guaranteed to work in certain circumstances like a malicious actor attempting to exploit input fields, long runtimes, or typical usage. The unit tests should attempt to use both invalid and valid data wherever data can be input to the program. Invalid data can mean the wrong data types, large data that could cause overflows, or data that is outside of an acceptable range. Additionally, the unit test should test the network connection by attempting to access the servers and devices in both unintended and intended ways. To pass the unit tests, the system should not crash and either ignore bad data to convert it to a valid value. Also, any network connection that isn't acceptable should be denied.

2) Data Validation: Data input from external sources should always be validated. Some validation was done, but it was not everywhere in the programs. For instance, if the PHP server sent a message to the Raspberry Pi, it would get converted to a float type. Normally, this was not an issue because the PHP script was prewritten to only send valid strings that could be converted. However, if someone on the network established a TCP connection to the Raspberry Pi and sent a message that could not be converted, the program would have experienced an error and potentially crashed. The problem could be partially solved by limiting the connections, but it is a good practice to verify data before it is used.

VIII. CONCLUSION

The project successfully accomplished its tasks as it was designed. Climbers were given a webpage interface, allowing them to manage their climbing experience. There were many applications that performed similar tasks, but no known application worked with non-commercial products and allowed the climber to adjust a climbing wall remotely.

To fulfill the desired goal of the project, it was designed as two separate applications and later networked together. The Raspberry Pi application moved a climbing wall to the angle of a network message through a C++ program that ran automatically on startup and the web application managed sessions, routes, and users as well as sent out network messages by using a LAMP stack.

Limited testing was done on all parts of the project. Most testing was done to ensure functionality of APIs in the final "adjustAngle" program. However, the web application was tested for some SQL attacks and to ensure that the database could be queried correctly. Throughout the process of testing and implementation of the code, some bugs were found and known about in advance. Typically, the bugs did not cause the system to fail, but they should be addressed if the project is developed further.

There is ample room for improvement in the project and it could be further developed to make it more stable, efficient, secure, and usable. Many additional features could also be added to give the users more insights into their climbing ability and progression. Overall, the project's separate components work as intended and climbers can improve the quality of their climbing sessions while also being able to interact with their climbing environment.

ACKNOWLEDGMENT

I would like to thank my wife A. Kummer and friend E. Odegaard for inspiring me to delve into the world of programming. Without them I may never have enrolled in the Computer Science program at MSUM. My wife supported, encouraged, and endured me with all my programmatic successes and failures. Also, prior to enrolling into the Computer Science program, E. Odegaard assigned informal programming assignments to me, sparking my interest in the field. Furthermore, he helped get the project off the ground by pointing me in the right direction with setting up the Raspberry Pi. He was always willing to talk and provide insights about computer science.

REFERENCES

[1] MoonBoard, 2019. [Online]. Available: https://www.moonboard.com. [Accessed: 17-Apr-2022].

[2] "Kilter Board," Setter Closet, 2022. [Online]. Available: https://settercloset.com/pages/the-kilter-board. [Accessed: 17-Apr-2022].

[3] "Rock climbing guides: Routes, Photos & Forum," Mountain Project, 2022. [Online]. Available: https://www.mountainproject.com. [Accessed: 17-Apr-2022].

[4] G. Mark (May. 29, 2020) "Eat Spray Love." Google Play [Online]. Available: https://play.google.com/store/apps. [Accessed: 16-Apr-2022].

[5] P. Kummer. (Mar. 3, 2022) "CLIMB with LAMP: Senior seminar project." GitHub repository. [Online]. Available: https://github.com/Paul-Kummer/seniorSeminar. [Accessed: Apr. 17, 2022].

[6] "Raspberry pi os," Raspberry Pi. [Online]. Available: https://www.raspberrypi.com/software. [Accessed: 17-Apr-2022].

[7] G. Lyons. (Aug. 25, 2021) "SSD1306_OLED_RPI: C++ Library to support the I2C 128x64 OLED display module driven by the SSD1306 controller for the raspberry pi SBC eco-system." GitHub repository. [Online]. Available: https://github.com/gavinlyonsrepo/SSD1306_OLED_RPI. [Accessed: Apr. 17, 2022].

[8] C. Verstraten. (2020) "RaspiCam: C++ API for using Raspberry camera (with OpenCV)." GitHub repository. [Online]. Available: https://github.com/cedricve/raspicam.

[9] P. A. and C. Gysin and S. Boytsov (Jul. 24, 2017) "usb-relay-hid." GitHub repository. [Online]. Available: https://github.com/pavel-a/usb-relay-hid. [Accessed: Apr. 17, 2022].

[10] A. Allain. "Shared libraries with GCC on Linux." Cprogramming.com. [Online]. Available: https://www.cprogramming.com/tutorial/shared-libraries-linux-gcc.html. [Accessed: Apr. 17, 2022].

[11] G. Trebbin. (Jul. 24, 2015) "Controlling a USB Relay with a Raspberry Pi." Grant-Trebbin. [Online]. Available: https://www.grant-trebbin.com/2015/07/controlling-usb-relay-with-raspberry-pi.html.

[12] "Five ways to run a program on Your Raspberry Pi at startup," Dexter Industries, 2022. [Online]. Available: https://www.dexterindustries.com/howto/run-a-program-on-your-raspberry-pi-at-startup. [Accessed: 17-Apr-2022].

[13] A. S. Tanenbaum and H. Bos, "Processes and Threads." in *Modern Operating Systems*m, 4th ed. Amsterdam, Netherlands: Pearson Education, 2015. ch. 2, pp. 85-173

[14] "A computer science portal for geeks." GeeksforGeeks. [Online]. Available: https://www.geeksforgeeks.org. [Accessed: 17-Apr-2022].

[15] M. M. Hassan, (Jun. 28, 2020) "C++: How to write a bitmap image from scratch." BITMAP. [Online]. Available: https://dev.to/muiz6/c-how-to-write-a-bitmap-image-from-scratch-1k6m. [Accessed: 17-Apr-2022].

[16] C. H. Brase and C. P. Brase, "Correlation and Regression." in *Understandable statistics: Concepts and methods*, 10th ed. Mason, OH, USA: Cengage Learn, 2012. ch. 9, pp. 520-532

[17] M. Drake and E. Heidi, (Jul. 15, 2021) "How to install linux, Apache, mysql, PHP (LAMP) stack on ubuntu 18.04." Digital Ocean. [Online]. Available: https://www.digitalocean.com/community/tutorials/how-to-install-linux-apache-mysql-php-lamp-stack-ubuntu-18-04. [Accessed: Apr. 16, 2022].

[18] "MySQL Workbench, Enhanced Data Migration" Oracle, 2022. [Online]. Available: https://www.mysql.com/products/workbench. [Accessed: Apr. 16, 2022].

[19] "W3Schools free online web tutorials." W3Schools Online Web Tutorials. [Online]. Available: https://www.w3schools.com. [Accessed: Apr. 16, 2022].

[20] "Encryption and Compression Functions." in *MySQL 8.0 Reference Manual*. Oracle, Apr. 7, 2022. [Online]. Available: https://dev.mysql.com/doc/refman/8.0/en. [Accessed: Apr. 16, 20