ECE/COE 1896

Senior Design

Home Position-based Automatic Light System Final Report

Completed by: Buka Agbim

Peter (Jiacong) Liu

Andrew Tran

Jarod Vickers

1. Problem Description

While there are options currently available for residential smart or automatic lighting systems, such as smart device applications and motion sensors, they have a few problems. Applications for smart devices allow users to control any light or group of lights from anywhere and allows for easy setup and customization through a smartphone app. However, the problem with these apps is that they do not allow for hands-free operation and isn't a truly smart system as it still requires the user to manually select the lights they want to turn on and off. Another option is to use motion sensors which does not require any hands-on manual control from the user but has its own problems. For example, the effectiveness of the motion sensors relies on the layout of the room they are in because if the user's motion is blocked by anything in the room, the sensors become useless.

The Position-Based Automatic Lighting System (PALS) attempts to solve these problems by using user position to determine which lights to turn on and off. This system extends the functionality of an existing smart lighting system, specifically, the Philips Hue brand smart home devices. It consists of a portable, internet connected device carried by a user, a server, at least three routers, and a Philips Hue smart bridge. Distances from the portable device to each router are calculated using Wi-Fi received signal strength index (RSSI). The server knows which room's lights to turn on based on which router is closest to the user.

This approach maintains the advantages of the previously mentioned systems as well as eliminating the disadvantages. Using position to determine which lights to control allows for the hands-free operation that motion sensors provide but is not affected by the layout of the room as Wi-Fi signals can penetrate through objects that may be blocking direct line-of-sight to the routers. PALS also takes advantage of an existing smart home system that has its own app, which allows for easy setup and customization. The system should be able to simultaneously calculate the distances from at least three separate routers in three different rooms and be able to determine which room the user is in with a delay of no more than one second.

2. Background

Wireless position tracking has been studied for many years, but has usually focused on commercial uses, such as automation, rather than residential use. Some uses of position tracking make use of RFID. However, due to the short range of RFID, this approach requires installing a large array of RFID tags throughout the building. Position is determined by finding the closest tag to a mobile RFID reader [1]. This approach is currently being used commercially for automation in warehouses but is not feasible for residential use due to the amount of tags that need to be bought/installed. An approach that uses Wi-Fi RSSI is the fingerprinting method that takes three distances, triangulates a position and converts them to coordinates on a preset map. Studies of this method have shown that Wi-Fi based distance calculations can have as low as a 30 percent error [2]. One of the problems with this approach is that setting up the map requires measuring the dimensions of each room which can be difficult for an average user to perform.

PALS will be a cheaper implementation of Wi-Fi based position tracking by using closest distance like the RFID method, but with the range that Wi-Fi signals provide. While this approach will apply distances calculations from the Wi-Fi fingerprinting method, it will not use triangulation to determine an exact location due to the lengthy and complicated setup it requires. Overall, this prototype is a mixture of ideas from already existing methods to provide the best accuracy from commercially available devices, such as Wi-Fi access points and smart bridges, while having a relatively simple setup.

3. System Requirements

3.1. Pi Server Data Transmission

For PALS to function properly, the system must be able to gather data on a Raspberry Pi Zero W, and transfer said data to the backend server that is running on a different machine. The data gathered and transferred via this system are signal strength and distance data. The Pi connects to the 3 routers set up throughout the home, and constantly pings them for signal strength. These signals are individually converted into distances using a distance algorithm and sent via a packet to the sever. Without this core functionality, the system would fail.

3.2. Location Accuracy

Location tracking is the primary goal of PALS. PALS utilizes a passive based Wi-Fi system to track the user's location. The required accuracy of the calculated room location should be 99% or better. Ensuring a high location accuracy will reduce the times when the system may accidently calculate a position not in the room the user is actually in. When this happens, the system can cause lights to flicker on and off as it thinks the user is constantly switching rooms which is obviously not the intended behavior of the PALS system.

This 99% was decided upon by the team as an acceptable benchmark for the system. This 99% accuracy means that when within a room, the device will accurately predict the room 99% of the time spent in said room.

3.3. Time Delay

PALS must also function with a low time delay. The time delay for affecting smart devices should be under 2 seconds.

This time delay is imperative, as users would want their smart devices to react to their presence within a short time. We assumed it typically takes about 2 seconds maximum from when someone enters a room to reach around and find a light switch. If PALS took longer than this to result in a visible change, the product would be obsolete. The team decided then that getting the response time for smart device interaction under 2 seconds was paramount

4. Design Constraints

4.1 Budget vs. Hardware Quality

Our solution was constrained by the quality of the hardware that we could afford on a budget. Specifically, the quality of the router we choose to use can largely affect the accuracy of any distance calculations using RSSI. Cheaper routers will not be able to maintain a stable signal strength between devices compared to a more expensive router. Not only was our group limited by a $400 overall budget for this prototype, this system is meant for residential use, so it is unrealistic to expect an average user to spend close to $100 per router or access point. This means that our solution must assume that the user may always not have a stable connection to the access points.

4.2 Device Size

One of the key components of our design is a carriable device that will continuously send data to the server. This device must consist of a component that can run simple C code, a power supply, and a fan if heat becomes a problem. Ideally, we want this device to be able to run continuously for at least 24 hours which affects the size of the battery used. All of these components need to be able to fit comfortably in a user's pocket which is 6.4 inches by 9.1 inches on average.

4.3 Smart Bridge Open API

Only smart lights that have their own open API can be used in this project to ensure there is minimal delay between the server sending an HTTP request and the lights actually turning on/off. There are third-party services that would allow for PALS to be used with any brand of smart light but sending HTTP requests through these services would cause too much delay. For example, IFTTT, a widely used service, can have delays of as low as 4 seconds which is still too slow for our system [3]. Open APIs will allow the server to send HTTP requests directly to the smart bridge with the shortest amount of delay. Since each brand of smart light has their own API, PALS will be designed to only work with one brand (Philips Hue).

5. Evaluation of Design Concepts

5.1 Position Tracking: Single Distance vs. Triangulation

One of the major design changes that happened during the development of the prototype was the change in the position tracking algorithm. Originally, we had decided to use triangulation to determine the user's position. The calculated position was then used to determine which room the user was in and which lights to turn on/off. While the algorithm itself worked, the results were very inaccurate due to the instability of the Wi-Fi signals between the carried device and the routers. Even if one of the distances was inaccurate due to a fluctuating signal strength, the server could calculate a position that was in a completely different room. This led to lights flickering on and off as the user's position was jumping between multiple rooms even when standing still. Another problem with this approach was

that we had to record room dimensions to allow the server to determine if the user was in a specific room. This meant we had to physically measure the length and width of every room which causes one of the problems we were trying to avoid, that being a difficult, complicated setup.

There were many attempts made to address the fluctuations in signal strength. We tried tweaking the distance formula, increasing the area of each room, and adjusted the rate at which the server received signal strength data from the carried device, just to name a few. However, the results were more or less the same.

The final solution we tried was to just use closest distance to determine which room the user was in. This led to the most consistent results out of the solutions we tried which was the reason we decided to go with this implementation for the final prototype. One of the drawbacks of this implementation is that the location of the access points actually matters. Each room that has smart lights must also have an access point in it. Since this implementation does not require high quality access points, however, we believe that this is still feasible with a small budget.

5.2 Server-Client Vs. Single Device

As mentioned above, the core device we will use to gather data is a Raspberry Pi Zero. Even through the Pi Zero runs an operating system and supports most of the development language and tools, its computational power is still limited for us to implement the entire project on it. Not only do we need to gather the router information, we also need to process that data by running through the pathloss distance algorithms every time a new data point is gathered. Running the entire system on a Pi Zero will result in significant time delays. Therefore, we decided to use a Server-Client approach, where the Pi only focuses on collecting signal strength of routers and sends them to the server, a much more powerful computer that can be process the data within milliseconds. To further reduced delays, we used UDP protocol instead of TCP to establish the server-client communication since we are constantly sending the information and if one packet gets dropped it will immediately get replaced. Using UDP protocol will save us time as we do not need to perform the TCP handshake process or recover from packet loss.

5.3 Programming Language Choice

All code on the server end were written in Python because the large number of libraries it has. We used the network socket libraries for server-client communication, matplot library to generate the data graph (see 7.1.1), and the requests library to send HTTP requests to the Philips bridge. It is the perfect language to combine all requirements of our project together. On the Pi Zero however, all the code was written in C because it runs much

faster, requires less memory, and has native GPIO support. This will allow us to reduce the delays from all components as much as possible.

6. Final Prototype

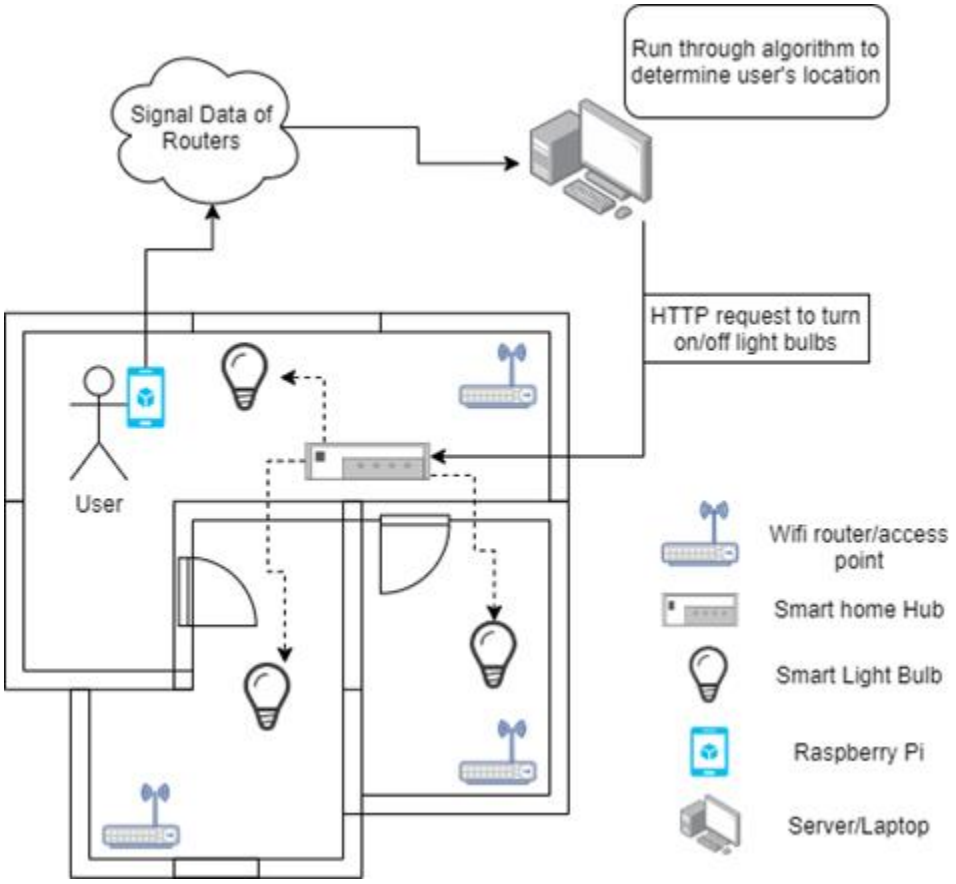

Figure 1: System Interaction Diagram.

The general system interaction diagram is shown in Figure 1. First, the Raspberry Pi Zero carried by the user will detect and collect signal information of all the WIFI routers we set up in the apartment. These data will then be sent to the backend server, where the data will be processed through the algorithm to calculate user's estimated location. The server will record user's current room location, therefore once a change in room is detected, the server will send out HTTP requests to turn on and off the corresponding LEDs in the rooms involved. The HTTP requests will be delivered to the Philips' Smart Home Bridge, which will send out ZigBee signals to targeted LEDs.

6.1 Raspberry Pi Software

6.1.1 Acquiring and Send Router Information to the Server

Since the Raspberry Pi Zero will be running on Linux, we could easily get the connectivity information from each available WIFI router because Linux has countless command line tools. Figure 2 shows one example as how we can acquire such information. The command “nmcli dev wifi” will list all the available WIFI and their signal strength. By redirecting the result, we can put all the information to a .txt file, which can then be sent to the server. There are other commands to show more detailed information including noise level and attenuation, but the general idea is the same.

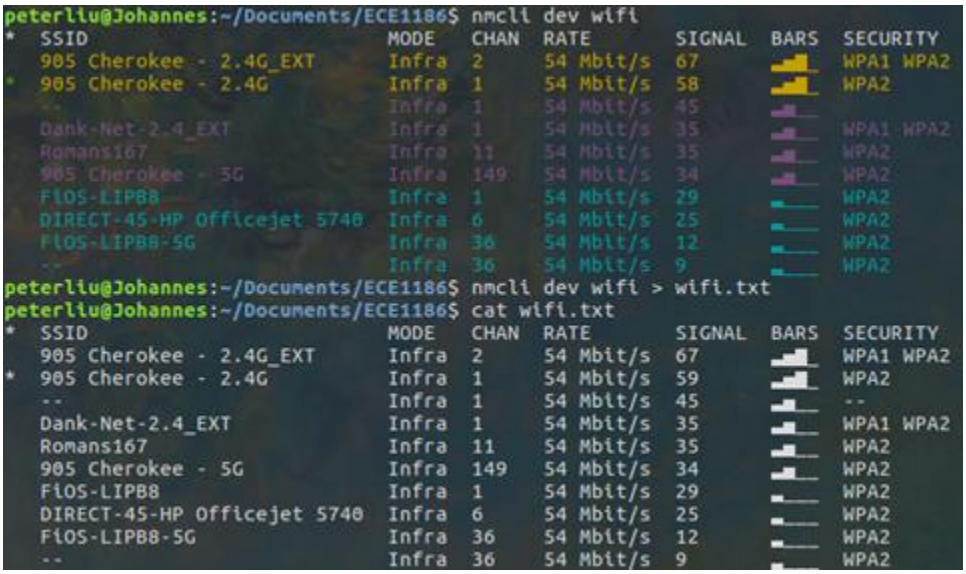

Figure 2: Example Linux Command

6.1.2 Reading User Inputs From GPIO

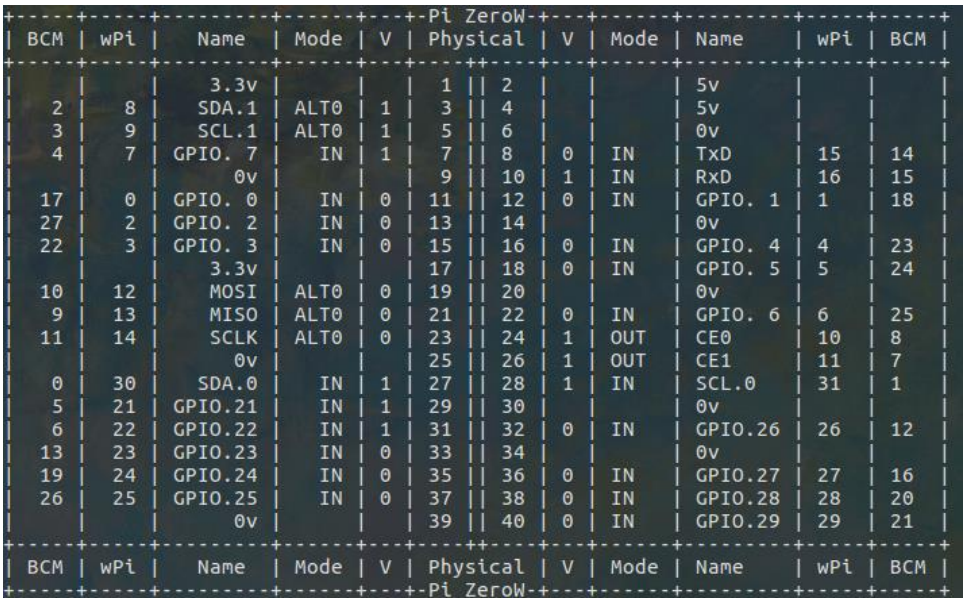

Figure 3: Raspberry Pi Zero GPIO Layout

The Raspberry Pi Zero also has other control functionalities such as adjusting the brightness of the light in current room. This is done by connecting buttons to the Pi's GPIO pins. In our project we set GPIO pin 0, 2, and 3 as input pins. These GPIO pins will read the input voltage to determine whether the button has been pressed and which button has been pressed. Each button represents a brightness level. In our breadboard implementation we had three buttons for 25%, 50%, and 100% brightness, whereas in the Eagle PCB we had an extra button for 75%. The brightness adjustment information will also be sent to the server, where it will be added to the HTTP requests for the LEDs if triggered.

6.2 Hardware Design

6.2.1 Preliminary Research

In the smart technology world, ease of use and the customer experience plays a big role in product development. Many people are comfortable carrying their phones throughout their day in their pockets or purses, which led us to use this as a reference size for the device we are creating. A device such as this, a separate small piece of plastic would be a device carried by a user constantly as they move through their smart homes.

In a similar sense, because users are not actively looking at the device for majority of their time using it, we decided to use a simplistic design. A plastic, phone sized casing with a few large function buttons on it would provide the optimal usability and ease of use.

6.2.2 Eagle Schematic and Design

Creating a parallel circuit between the buttons, the input power could actively be routed through various resistors to then reduce and increase the brightness of the lights. The device only needs a low voltage input to run, so we would set thresholds on different input receivers to alert a change in the brightness of the light using the Phillips Hub API. Here is the Eagle schematic we created:

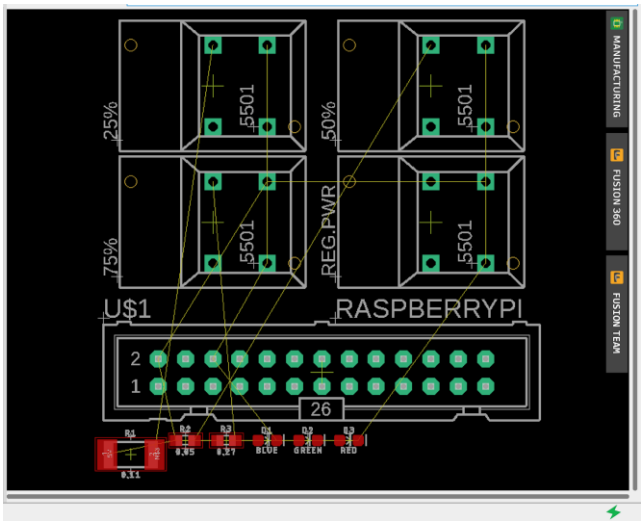

Figure 4: Eagle Device Schematic

Using a custom library to represent the Raspberry Pi, we created a “hat” to cover the circuit board. To implement this without the hat, used a breadboard, a few buttons and resistors, and we were able to create the device. Here is the actual implementation of the schematic, connecting wires from the breadboard to input pins on the Raspberry Pi.

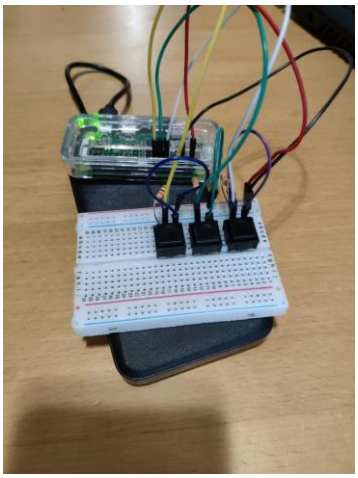

Figure 5: Prototype Device Implementation

6.3 Backend/Server Software

6.3.1 Receiving Information from Raspberry Pi

```
while True:
    data, addr = sock.recvfrom(2048)
    #data = struct.unpack("d",data)[0]
    info = data.split()
```

Figure 6: Python Server Code to Receive Data from Pi

The server-client communication between the Raspberry Pi and the laptop/server was implemented through existing network socket library in Python. Once the router information file is generated on the Raspberry Pi, the server will start receiving the file and extracting signal information of targeted router from the file. As shown is figure 6 the server program will read the file one line at a time and extract useful information from each line. This process will never be stopped unless program is terminated.

6.3.2 Distance Algorithm

$$(p)_{dB} = (p_0)_{dB} - 10n\lg(d/d_0)$$

Figure 7: Pathloss Equation

Above is the pathloss equation we used to determine how far away the Pi Zero is from the signal source. *Po* and *do* are previously hand measured references, *Po* is the original signal strength and *do* is its respective distance from the source. n is a constant that changes based on environment and in our case, we used 2.5 since we are indoor. Once we extract the new signal strength *P* from file, we can use this equation to estimate distance *d* of the location where *P* was recorded.

6.3.3 Data Structure Design

The Data structures in the background must be designed in a way such that we can easily store all the information needed, as well as provide necessary functionality to the system. As such, we need a structure of the overall house to store the information of rooms within the house, as well as locations of objects of interest. These objects of interest include the location of WIFI access points, as well as the location (or containing room) of an IoT device.

6.3.3.1 House Design

The House data structure will be a simple structure that contains the rooms, which are stored in a list, and is able to perform simple calculations given input data. These calculations are get_room which returns the room the given coordinates exist within, and get_closest_room_center, which returns the room whose center the given x, y coordinates are closest to. Both methods could be useful in determining HTTP calls to the Smart Bridge device. The class diagram for the House can be seen in Figure 8.

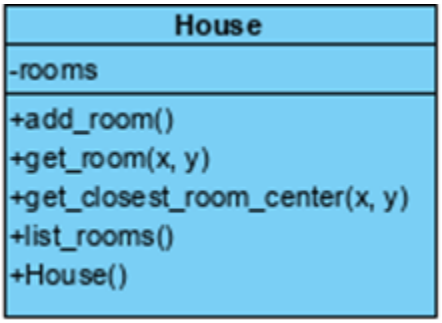

Figure 8: House class diagram

Rooms are simple structures that have a center coordinate in x and y coordinates, as well as a length and width.

6.3.4 Current Location Algorithm

The current location algorithm has deviated from the original x and y coordinate system and has shifted towards a passive approach. Due to complications with signal strength and unreasonable distances, the triangulation of coordinates was scrapped due to extreme variability. The system instead takes a “closest room within reason” approach. The system compares the current distances from each router and chooses the smallest of the 3. From this point, the system then does a check to ensure we are within reasonable range of said room.

Using the room dimensions from the room data structure above, the system checks that we are at least within the bounds of the room in comparison to the router. This algorithm results in PALS accurately being able to choose the room it is in with a passive approach.

```
##SEND HTTP REQUEST HERE
if distanceRoute1<distanceRoute2 and distanceRoute1<distanceRoute3 and not light2:
    if distanceRoute1<max(home.get_room(0,0).length, home.get_room(0,0).width):
        ##TURN OFF LIGHTS
        LC.turnOff(str(3), pals_ip, pals_user)
        LC.turnOff(str(1), pals_ip, pals_user)
        LC.turnOn(str(2), pals_ip, pals_user)
        light2 = True
        light3 = False
        light1 = False
elif distanceRoute2<distanceRoute1 and distanceRoute2<distanceRoute3 and not light3:
    if distanceRoute2<max(home.get_room(6.3,5.3).length, home.get_room(6.3,5.3).width):
        LC.turnOff(str(2), pals_ip, pals_user)
        LC.turnOff(str(1), pals_ip, pals_user)
        LC.turnOn(str(3), pals_ip, pals_user)
        light2 = False
        light3 = True
        light1 = False
elif distanceRoute3<distanceRoute2 and distanceRoute3<distanceRoute1 and not light1:
    if distanceRoute3<max(home.get_room(0,7.6).length, home.get_room(0,7.6).width):
        LC.turnOff(str(2), pals_ip, pals_user)
        LC.turnOff(str(3), pals_ip, pals_user)
        LC.turnOn(str(1), pals_ip, pals_user)
        light2 = False
        light3 = False
        light1 = True
```

Figure 9: Closest router logic

6.4 Philips Hue API Code

6.4.1 Smart Bridge Setup

To gain permission to send requests to the Philips Hue smart bridge, the bridge's internal IP address must be found, and the bridge will also then generate a username that must be sent with every request. On server startup, the setup process will begin, and the bridge's internal IP and generated username will be requested and saved to a JSON file. The setup process is the same as the normal process of setting up any Philips Hue [4]. If a JSON file already exists, the setup is skipped, and the IP address and username are read in from the file.

```
def setup():
    if path.exists("PALS.json"): #already set up
        with open("PALS.json", "r") as inFile:
            palsjson = json.load(inFile)
            ip = palsjson["ip"]
            username= palsjson["username"]
    else:
    # Discover hue bridge and get its id and internal ip
        url = "https://discovery.meethue.com"
        response = requests.get(url)
        response = json.loads(response)
        ip = response["internalipaddress"]

        # set up a username
        url = "http://"+ip+"/api/"
        body = '{"devicetype":"PALS"}'

        while True:
            response = requests.post(url, body)
            response = (response.text).translate({ord(i): None for i in '[]'})
            palsjson = json.loads(response)
            if "error" in palsjson:
                print("Please press the link button your Philips Hue Bridge")
                input("Then Press ENTER to continue:")
            elif "success" in palsjson:
                username = palsjson["success"]["username"]
                print(username)
                break

        palsjson = {
            "ip": ip,
            "username": username
        }
        with open("PALS.json", "w") as outFile:
            json.dumps(palsjson, outFile)

    return palsjson
```

Figure 10: Bridge Setup Code

6.4.2 HTTP Request from Server

HTTP requests can be sent from the server to the bridge by following the format of the Philips Hue Light API standard which requires the ID of the room the lights are in, the IP address of the bridge, the app username, and any attributes such as state (on/off) and brightness [5]. Room IDs are saved on the bridge itself and they can be retrieved from the server on startup using the Groups API. The room IDs are requested on server startup.

```
def getGroups(ip, username):
    url = "http://"+ip+"/api/"+username+"/groups/"
    response = requests.get(url)
    groups = json.loads(response.text)
    print(groups)
    return groups

def turnOn(roomID, ip, username):
    url = "http://"+ip+"/api/"+username+"/groups/"+str(roomID)+"/action"
    body = '{"on":true}'
    requests.put(url, body)

def turnOff(roomID, ip, username):
    url = "http://"+ip+"/api/"+username+"/groups/"+str(roomID)+"/action"
    body = '{"on":false}'
    requests.put(url, body)

def setBrightness(roomID, ip, username, bri):
    if bri > 254:
        bri = 254
    elif bri < 0:
        bri = 0
    url = "http://"+ip+"/api/"+username+"/groups/"+roomID+"/action"
    body = '{"bri":'+str(bri)+'}'
    requests.put(url, body)
```

Figure 11: HTTP Request Code

7. Test and Verification

The primary system requirements that could be properly tested numerically are the location accuracy of PALS, as well as the time delay of PALS. These requirements are paramount to the PALS system, and thus were tested thoroughly as to ensure that PALS functions to its fullest potential. Light uptime and delay were measured both by hand, with time delay also being measured in code to reassure the user.

7.1 Location Accuracy Testing

7.1.1 Coordinate Accuracy

Upon our initial attempt, to create PALS, coordinates were not working properly. The following graph shows the coordinates calculated when walking around our test house:

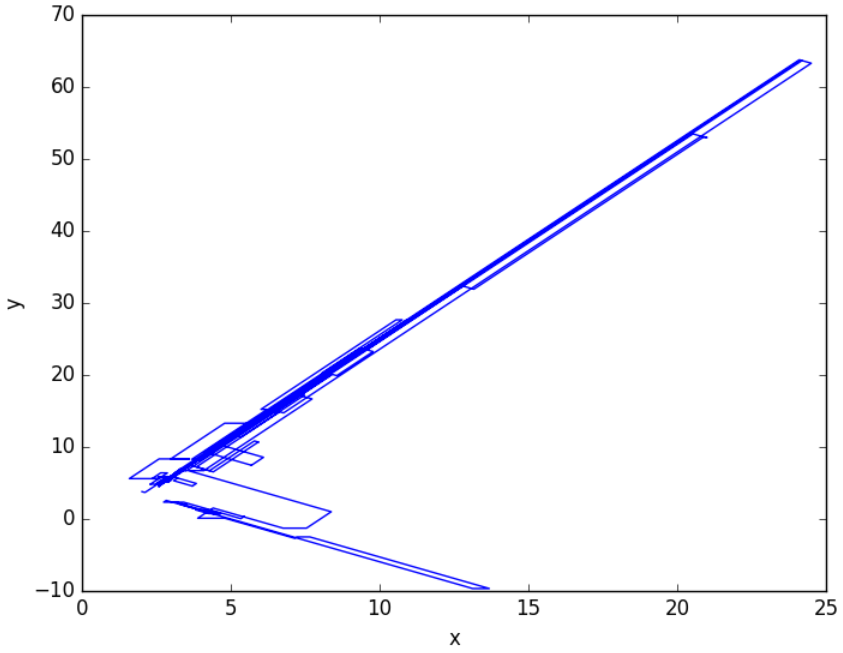

Figure 12: Coordinates calculated from within the test house

The coordinate bounds of the house were roughly [0, 7] for x, and [0,8] for y. Clearly our distance algorithm didn't function with interference, and thus the algorithm was changed.

7.1.2 Current Room Location Accuracy

Following the failure of coordinate generation and subsequent algorithm change, the location accuracy was tested by calculating uptime as a percentage of a light while within the room. The tests were taken at intermediate distances from the routers within the room. Measurements were taken over the course of 30 seconds, and the amount of time spent on was measured. Our testable distance was restricted by the size of the rooms we were testing in. The results of this can be found in Table 1:

Table 1 – Light Uptime

Distance (m)	Light 1 Uptime (s)	Light 1 Accuracy	Light 2 Uptime (s)	Light 2 Accuracy	Light 3 Uptime (s)	Light 3 Accuracy
0	29.94	99.80%	29.92	99.73%	29.84	99.47%
0.5	29.86	99.53%	29.91	99.70%	29.95	99.83%
1	29.89	99.63%	29.77	99.23 %	29.62	98.73 %
2	29.18	97.00%	28.90	96.33%	29.31	97.70%

The above data shows that our PALS system was able to accurately predict the room at least 96.33 percent of the time. It should be noted that the lower accuracies at 2 meters are due to the system being close to the edge of a room at that point, so it is expected that some flickering might occur due to our poor router signals. Excluding these values, the location accuracy was very close to the system requirement of 99 percent.

7.2 Time Delay Testing

Time delay was measured for both a user entering a room to see the light's reaction, as well as the time delay for lights to switch brightness according to user input. These time delays were measured via hand, from entrance/button click to light response, as well as using code to measure the time from when the signal is sent by the Pi to when the HTTP request is created for the lights. Room entrance was considered when a user began entering the door frame for said room. The results of this can be seen in Tables 2 and 3 below:

Table 2 – Room Entrance Delay

Attempt	Room 1 Hand	Room 1 Code	Room 2 Hand	Room 2 Code	Room 3 Hand	Room 3 Code
1	0.85 s	0.81 s	0.75 s	0.73 s	0.90 s	0.85 s
2	0.65 s	0.63 s	0.77 s	0.72 s	0.86 s	0.83 s
3	0.97 s	0.91 s	0.86 s	0.85 s	0.82 s	0.81 s
4	0.81 s	0.74 s	0.82 s	0.79 s	0.73 s	0.69 s
5	0.77 s	0.76 s	0.82 s	0.77 s	0.69 s	0.67 s
Average	0.81 s	0.77 s	0.804 s	0.772 s	0.80 s	0.77 s

Table 3 – User Light Power Input Delay

Attempt	Room 1 Hand	Room 1 Code	Room 2 Hand	Room 2 Code	Room 3 Hand	Room 3 Code
1	0.98 s	0.95 s	0.99 s	0.96 s	0.94 s	0.91 s
2	0.96 s	0.95 s	1.01 s	0.99 s	1.03s	1.01 s
3	0.81 s	0.79 s	1.04 s	1.01 s	0.91 s	0.9 s
4	1.1 s	1.01 s	1.03 s	0.99 s	1.02 s	0.97 s
5	0.95 s	0.76 s	0.88 s	0.86 s	0.97 s	0.94 s
Average	0.96 s	0.892 s	0.99 s	0.962 s	0.974 s	0.946 s

As can be seen from the data collected above, the measured and coded times were precise, leading us to believe these are accurate results. Our system requirement establishes that the delay between room entrance/user input must be below 2 seconds, which our implementation achieved. Also, it should be noted that coded times were slightly shorter than measured times, depicting the time it takes for an HTTP request to be processed by the central control hub. All measured time are measured to 2 points of precision past the decimal point. This helps diminish

the potential inaccuracies of hand timing effects, as humans can only be so accurate with reflexes.

8. Conclusions and Future Work

The final design we created is feasible and could be sold upon further development. Though we were not able to print the final PCB, the components necessary to interact with it and give the expected functionality is no bigger than the average pocket size, so a device like this could be used in the real world. The design of the handheld device also allowed for an easy customer experience. There are only four buttons on the handheld device, each being labeled with its specific function, so training would not be necessary to use it.

Throughout this project, there were many points where we found ourselves "training" to learn a new coding language or get familiar with a new API. Understanding how the Phillips smart hub interacted with the lights and finding ways to manipulate this connection to produce a new functionality is one example of using the knowledge learned throughout the project to progress in it. This is one of the many strong points of our team. We were able to learn and adjust on the fly to allow for the smoothest completion of a task. We also were able to communicate well and divide the work up in ways that highlighted each team member's strengths.

One potential problem that we did not discuss throughout the duration of this project was the system's vulnerability. The system we created utilizes Wi-Fi routers that communicate with a Raspberry Pi to track the user's location. This system is not secure and could easily be vulnerable to threats if the device is taken and reprogrammed. Given that the Raspberry Pi is connected to routers that are possibly connected to the internet, this is a cause for concern. Data could be leaked out of the system if the Raspberry Pi is reprogrammed correctly.

The Wi-Fi routers used for the project were also a limitation that would prevent commercial use. The were lower end routers that did not send a constant, stable signal to the Raspberry Pi, so this led to small issues in our final demonstration. This device would require a certain standard of router to allow for the device to work on a commercial level with a high ease of use.

In the end, the project was very successful given the current circumstances, and if this project were developed further it could easily be marketed as a smart home device. To do so, we would need to focus on the security aspects of the device and of the servers and make the distance calculation more accurate and stable to allow for seamless usage.

9. References

[1] - https://ieeexplore-ieee-org.pitt.idm.oclc.org/stamp/stamp.jsp?tp=&arnumber=6418858&tag=1

[2] - https://ieeexplore-ieee-org.pitt.idm.oclc.org/stamp/stamp.jsp?tp=&arnumber=6802701&tag=1

[3] - https://help.ifttt.com/hc/en-us/articles/115010194247-Can-I-make-Applets-work-faster-

[4] - https://www.support.com/how-to/how-to-set-up-a-philips-hue-light-10110

[5] - https://developers.meethue.com/develop/hue-api/groupds-api/