ARC Machine Monitoring System

By
Rohan Inampudi
Akhil Kodumuri
Calvin Lee

Final Report for ECE 445, Senior Design, Fall 2022
TA: Zhicong Fan

6 December 2022
Project No. 22

Abstract

There never seems to be enough time to go to the gym. People are busy, and more often than not, the gym is always just as busy. The number one reason that causes long wait times at the gym is the large amounts of people that use the half rack in their workouts. If people were to know about rack availability ahead of time, they would be able to plan their workouts so it takes the least amount of time. Our solution is to have an IOT device that can take in user input at the rack, and send the data to a web server via the lightweight MQTT message to a Raspberry Pi. With this system, users are able to check for the availability of a rack real time, and know immediately whether or not they can do their workout without waiting.

Contents

1. Introduction

1.1 Problem

One question that is always on a college student's mind is: "Is the ARC busy?". There have been many times throughout our college career where we have gone to the ARC expecting a quick workout just to see lines for the machines we want to use. We've always wished that we could see what machines were being used and what were not. To combat this, we would like to create an interface where students can use their phone to visually see which equipment at the ARC are being used and which are not.

1.2 Solution

We have created an interface where students can use their phone to visually see which equipment at the ARC are being used and which are not. This way, students can anticipate whether or not they should go to the ARC. At a high level, there would be a button by the equipment being used. The button, upon being pressed by the user when a machine is being used, would then send a signal to an IoT device which would then send a signal to a server. Our website will then use this server to update a UI which users can utilize to see which machine is being used. It should be noted that this design will be restricted to the Matrix Mega Half Rack which is the most used apparatus at the ARC. This system can be used generally for any machine at the ARC, however, the full capabilities of this system is unlocked for the Matrix Mega Half Rack.

1.3 Visual Aid

The diagram below depicts the full process of how our system works. First, a user will press a button on an ARC machine. Then, the System on Chip (ESP-32) on the PCB will wirelessly communicate the machines used to a Raspberry Pi. This Pi will then send this information to AWS, where our website will get information about whether or not a machine is in use. The UI will then depict to a viewer that a machine is in use.

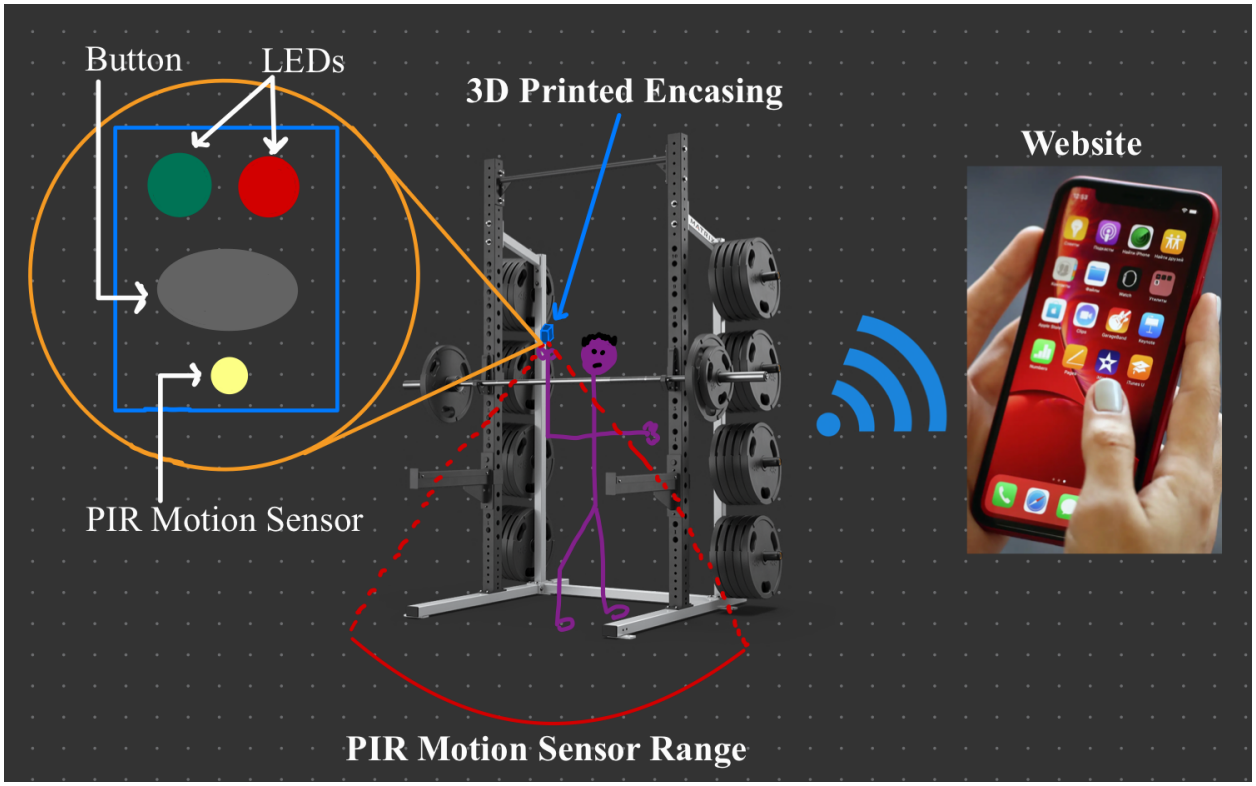


Figure 1: Visual Aid

1.4 High level requirements list

Battery Life: System should be able to last for multiple hours without a direct power source.
Machine Status: Changes in machine availability are correctly and efficiently communicated using multiple peripherals (button and PIR motion sensor).
Website Access: Website is accessible to users and correctly displays the most up to date availability of a machine.

1.5 Subsystem Overview

1.5.1 High Level Block Diagram

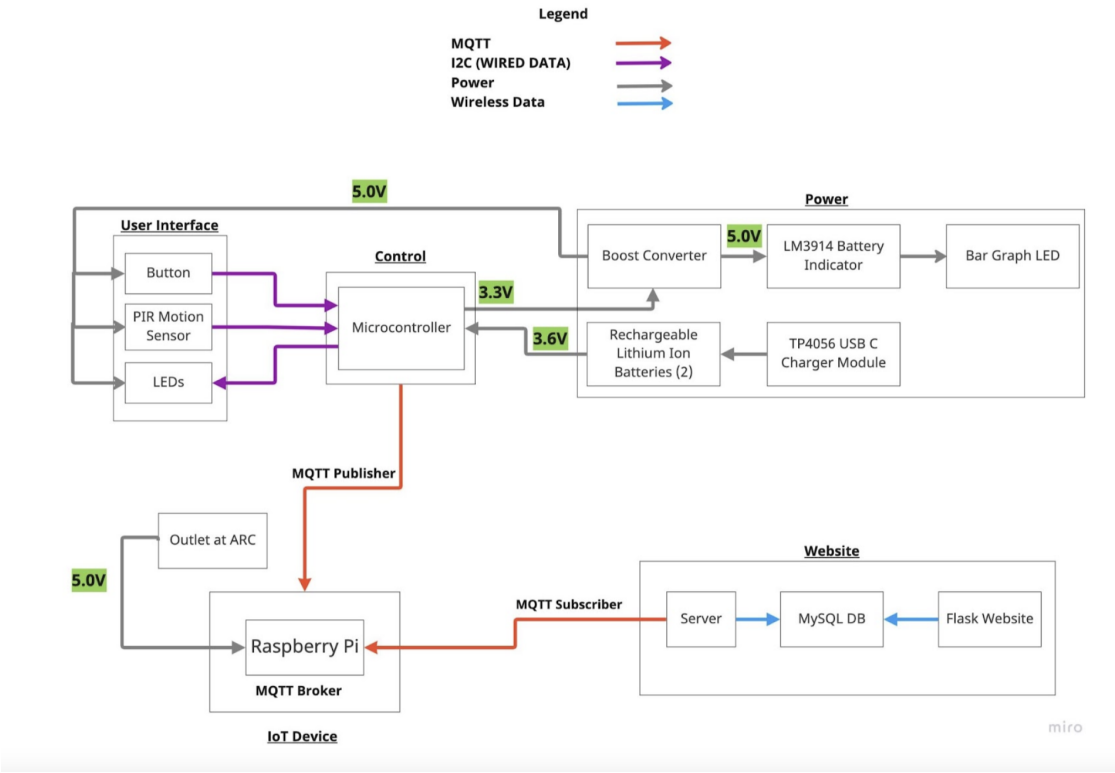


Figure 2: Block Diagram

1.5.2 Power

The goal of the power subsystem is to appropriately supply the correct amount of power to use the various components our PCB contains. These components include but are not limited to: LEDs, buttons, microcontroller, rechargeable lithium ion batteries, motion sensor, and a bar graph LED display. One feature of the system is the ability to sustain operations without the use of an external power source. Thus, rechargeable lithium ion batteries are used to supply our system for long periods of time. The power subsystem contains a module that shows the amount of charge left within the rechargeable lithium ion batteries. FInally, when it is time to charge the lithium ion batteries, we've included a TP4056 safe charging module so the batteries can be safely recharged to full power. One feature of this module is that charging stops when the batteries are completely full.

1.5.3 Control

The control subsystem is composed of our ESP32 microcontroller, button, PIR motion, and LEDs. These components are powered by the Power subsystem in order to operate. The goal of this

subsystem is to send a MQTT network packet communicating whether or not an ARC machine is in use. The button will be pressed by a user at the ARC when they want to use the machine. This action will then turn on the LED and trigger the ESP32 to send a packet to our IoT device to signal the machine is in use. The same steps will occur when the user presses the button to turn off the LED signaling the machine is not in use. The subsystem contains a PIR motion sensor which acts as a fail safe if a user does not press the button before or after use. Note that each machine at the ARC would theoretically have its own PCB with the Control and Power subsystem. Each communicating their respective machine availability information to the IoT device.

1.5.4 IoT Device

The IoT device utilized in this project was Raspberry Pi 3 B. The Pi acts as a "middleman" between the ESP32 within our Control subsystem and the website. This decision was made to reduce the energy consumption on the Control subsystem. Instead of the ESP32 sending MQTT packets to the website, all ESP32 outgoing traffic is sent to the Pi.

1.5.5 Website

We decided to create a Python Flask based website in order to accurately and efficiently communicate the most up to date availability to the user. On the backend, we opted to store data using MySQL, a relational database management system. MySQL's reliability, security, compatibility, and friendly UI allows for seamless integration with our project. Additionally, an email notification service has been created for user's to receive real time updates on whether their favorite machine is available at the ARC.

2. Design

2.1 Control

2.1.1 Button to IoT Device

In order to detect whether or not a machine is in use, we created a PCB with the following components: a button that will be pressed whenever a machine is being used, an ESP32 system on chip, and a LED of different colors to indicate that a machine is being used. We will use the MQTT network protocol to communicate between the ESP32 on our PCB and the Raspberry Pi. Information on machine status will be shared via the MQTT protocol to the Raspberry Pi. The Raspberry Pi will also receive information from our website that will then be communicated to the PCB. Once the button is pressed, the LED on our PCB will also light up to signify the machine is in use. In order to power this subsystem, we will have rechargeable lithium ion batteries, on a battery pack. These batteries can be easily charged. In order for this subsystem to act in accordance with our high level requirements, our IoT device should be able to handle the

communication of multiple machine sensor PCBs at once. Also, this subsystem should be able to last multiple hours with a single 3.7V lithium ion battery.

2.1.2 Motion Sensor to IoT Device

A motion sensor is utilized in order to detect use of the equipment that occurred without pressing the button. This allows our website to be changed if a person forgets to press the button. We include this fail safe in order to provide the most accurate information to students. The motion sensor will be on the PCB, and, thus, will be powered by the rechargeable lithium battery. We will incorporate logic to limit the power consumption of the motion sensor, so it is not being used continuously. An example of this logic is scheduled downtimes for when the ARC is closed. Since we will be using a PIR motion sensor, we will use electronic tape in order to limit the range of view of the motion sensor. This will ensure that we will not take in more information that we don't need and we can control the sensor to take in input from individuals using the machine.

2.1.3 Microcontroller

The ESP-32 microcontroller is responsible for the logic that controls what data is propagated through to the database on the server end. At a high level, when the user presses the button, the availability should be sent through to the MQTT broker (the raspberry pi), and the LED should turn on, visually marking to the user that the machine is in use. The system works similarly when the user leaves the machine, saying that the machine is off. Furthermore, we have a failsafe that is implemented using a PIR motion sensor. We used a queue to implement this feature; we poll the motion sensor every second, and push this value (either a 0 or 1 depending on availability) into the queue. Then, at any point if the number of ones divided by the number of zeros crosses a certain ratio, an availability signal is sent depending on if activity is measured or not.

2.1.4 Control Justification

The control subsystem is essential for determining and relaying the appropriate machine availability information to the other subsystems. We programmed our ESP32 using the Arduino IDE, thus all packets sent from the ESP32 to the Raspberry Pi are able to be correctly verified through the Serial port. Furthermore, availability is updated on the LEDs directly on the PCB. Thus, we were able to utilize the LEDs for testing as well.

2.2 Power

2.2.1 Power Design Description

In order to power all on-board components (button, PIR motion Sensor, LEDs, microcontroller), we use two 3.7V Panasonic NCR18650GA 3450 mAH 10A lithium ion rechargeable batteries. We charge the lithium ion batteries using a TP4056 USB C Li-Ion Charger module. Using this module,

we are able to charge the battery pack using a USB-C cable, thus enabling the entire system to charge without removing the physical batteries from the case. For the convenience of the employees of the ARC, we visually display the battery life of each power subsystem. The battery life indicator we use is the LM3914 bar graph chip. The display will allow ARC workers to know when the rechargeable batteries will need to be plugged in for charging. Some components on our PCB require a 5V input. Due to our batteries being 3.7V, we use a 3.7V to 5V boost converter (MAX1797EUA+) to help achieve this task.

2.2.1 Power Justification

The power consumption for the battery life indicator, LEDs, and PIR Sensor are straightforward to calculate, as that information was directly on each component's datasheet. However, we must closely analyze the consumption of the ESP32 for our specific use case of transmitting and receiving data via WiFi:

When the ESP32 is in Active Mode (WiFi module, bluetooth module, and processing core running at all times), greater than 240 mA is needed for operation. Furthermore, it has been documented that large spikes in power of almost 790 mA occur when WiFi and Bluetooth are in operation together. For our use case, we will only be leveraging the WiFi connectivity feature of the ESP32 for data communication purposes with the Raspberry Pi. Furthermore, we intend to let the ESP32 operate in Deep Sleep mode so that it stays at a low power consumption of 0.011 mA when it is not transmitting or receiving data via WiFi. Therefore, on average, we expect a power consumption of 39 mA - 55 mA for the ESP32.

According to our calculations (displayed in the table below), we estimate a total power consumption (worst case) of around 6553.5 mAh. Thus, utilizing two 18650 batteries is essential as we will be able to output 6900 mAh, lasting a total of 5 full days of operational hours at the ARC. The calculations in the table below can be modeled by the following equation total_power_consumption = current_required_by_component * total_time_component_on.

Component	**Power Consumption**	**Power Consumption (5 days)**
Button	-	-
ESP32	55 mA * 17 hrs * 5 days	4675 mAh
Battery Life Indicator	10 mA * 17 hrs * 5 days	850 mAh
LEDs	12mA * 17 hrs * 5 days	1020 mAh

PIR Sensor	0.1 mA * 17 hrs * 5 days	8.5 mAh
		Total: 6553.5 mAh

Table 1: Power Calculations

2.3 IoT Device

2.3.1 IoT Device to Server

For a high level overview of our project, the button is directly connected to the ESP-32, which will communicate with the Raspberry Pi via Message Queue Telemetry Transport, or MQTT. Once the data is on the Raspberry Pi, we are planning on utilizing a locally hosted server which will receive the MQTT messages from the Pi.

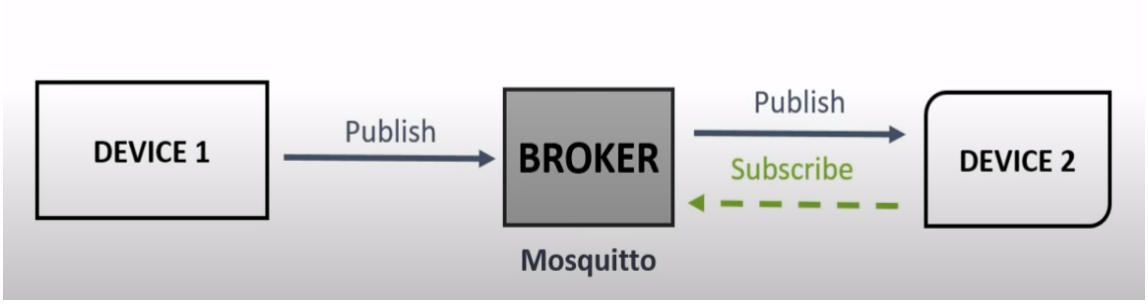


Figure 3:MQTT publish/subscribe model

MQTT is the standard of communication for IoT designs due to its lightweight publish/subscribe system, and is designed for constrained devices with low bandwidth. Using MQTT, we can have multiple clients sending and receiving data to and from a single “broker”, a hub that receives and filters all messages, and publishes the messages to all clients that are subscribed to that topic. In our case, the ESP-32 will be acting as a client, and the Raspberry Pi will be acting as both a client and a broker. Once the button is pressed, the data for a certain topic (for example, “data/arc_availability”) will be published from the ESP-32 to the Pi, and the Pi, which is subscribed to that topic, will receive that published data.

All machine information that is sent to the Pi will be sent to the locally hosted server. Ultimately, the Raspberry Pi will act as a gateway of communication between all ARC machine sensors to the server. The ARC would only need one Pi in order to send ARC machine sensor information to the server. Because of this, in order to charge the Raspberry Pi, we will just plug it into a free outlet at the ARC.

2.3.2 IoT Justification

The IoT subsystem was tested by monitoring the MQTT packets the Raspberry Pi received and checking the packet's payload to check if the correct machine ID and availability were being sent from the Control subsystem's ESP32.

2.4 Website

2.4.1 Website Design

We have created a Python Flask based website that allows users to obtain the most up to date availability for all machines at the ARC serviced by our design. Note that this implementation was done locally to ensure a proof of concept. A local server has been set up to act as MQTT subscriber to the "ESP32/machine_availability" topic on the broker (Raspberry Pi). Upon receiving any new availability information for a given machine, the server will immediately relay this data to a MySQL database and update an availability table accordingly. The Flask website queries the MySQL database for up to date availability and visually displays the status of each machine. Furthermore, users have the choice to sign up for an email notification service. An endpoint has been added to the website where one can enter their email and specify the machines they would like to receive notifications for. Then, when a user's desired machine becomes available, our service will leverage Simple Mail Transfer Protocol (SMTP) to send emails to all individuals registered for that specific machine.

2.4.2 Website Justification

We were able to test the website by displaying the data propagated through the database to our web server. We queried the database using a MySQL python library, and displayed it on a table on our website. We were also able to test the email notification system by verifying that our test email was receiving emails correctly whenever the availability correctly opened up, and if the email was subscribed to the correct arc station.

2.5 Design Alternatives

2.5.1 Step Up Voltage

In order to power specific components on the Control and Power subsystems, 5V input was needed. The rechargeable lithium batteries that we use are 3.7V. To fix this, we used the MAX1797EUA voltage boost converter. This chip boosts any low voltage input to 5V output. However, this decreased the total current circulating within the overall circuit. Because the MAX1797EUA boosts voltage, current is decreased. The issue that arose was our ESP32 did not turn on.

We addressed this issue for our demo by directly using a 5V voltage source to supply the modules that needed it. But, this caused a dangling component to our overall system.

An alternative design that would avoid this issue, is to use batteries in series that are greater than 5V and directly supplying the appropriate modules. Any component that requires less than 5V a voltage regulator should be used. This method ensures that there is no substantive voltage drop allowing for our ESP32 to conduct enough current to power on.

2.5.2 Eliminating Noise

A buffer was used to track whether or not there is activity near the system. However, an issue arose when the buffer was filled with 0s (signifying no activity). When a button was pressed, turning the LED on, showing the machine to be “in use” our LED would instantaneously turn off because the ratio of ones to zeros was below the threshold ratio. This was an edge case within the microcontroller (ESP32) code. This was remedied by adding a condition that dismissed the buffer activity threshold, when significant time had passed since the last time a button was pressed. Not only did this solve our edge case, the added condition completely removed the scenario that “noise” would trigger our system to be in use.

3. Parts and Schedule

3.1 Parts

Description	**Manufacturer**	**Quantity**	**Extended Price**	**Link**
FireBeetle 2 ESP32-E MCU	FireBeetle	2	$8.90	link
Raspberry Pi 3 - Model B	Adafruit	1	$35.00	link
12mm Button	Gikfun	1	$8.78	link
5x Stemedu HC-SR501 PIR Sensor	Stemedu	1	$9.99	link
RGB LEDs	Adafruit	1	$3.95	link
5x AM312 PIR Sensor	Aideepen	1	$9.59	Link updated

Rechargeable 18650 Battery	Panasonic	4	$4.99	link
Raspberry Pi 3 Power Adapter	Canakit	1	$9.95	link
2x18650 Battery Holder	E-outstanding	1	$8.99	link
TP4056 USB C Li-Ion Charger Module	diymore	1	$9.59	link
MAX1797EUA+	Digikey	1	$6.00	link

Table 2: Cost Analysis

3.2 Schedule

Week	Team Deliverables	Rohan	Akhil	Calvin
9/26 - 10/3	Design Doc	Design Doc	Design Doc	Design Doc
10/3 - 10/10	Design Review PCB Board Review	Finalize PCB Design	Finalize PCB Design	Finalize PCB Design
10/10 - 10/17	PCB Order 1 Teamwork Evaluation	Complete Teamwork Evaluation	Complete Teamwork Evaluation	Complete Teamwork Evaluation
10/17 - 10/24	Assemble Full PCB Design	Assemble PCB Start webdev	Assemble PCB Start webdev	Assemble PCB Start webdev
10/24 - 10/31	Ensure PCB Design Works	Test PCB	Test PCB	Test PCB
10/31 - 11/7	PCB Order 2 Individual Progress Reports	Complete Progress Reports	Complete Progress Reports	Complete Progress Reports

11/7 - 11/14	Debug and Refine Software	Debug Full System	Debug Full System	Debug Full System
11/14 - 11/21	Mock Demo Final Testing	Final Testing + Demo Prep	Final Testing + Demo Prep	Final Testing + Demo Prep
11/21 - 11/28	Fall Break	Break	Break	Break
11/28 - 12/5	Final Presentation Final Paper	Work on Final Paper and Presentation	Work on Final Paper and Presentation	Work on Final Paper and Presentation

Table 3: Scheduling

4. Requirements and Verification

All verifications are in the appendix C below. Only requirements and quantitative results are explicitly stated here.

4.1 Control

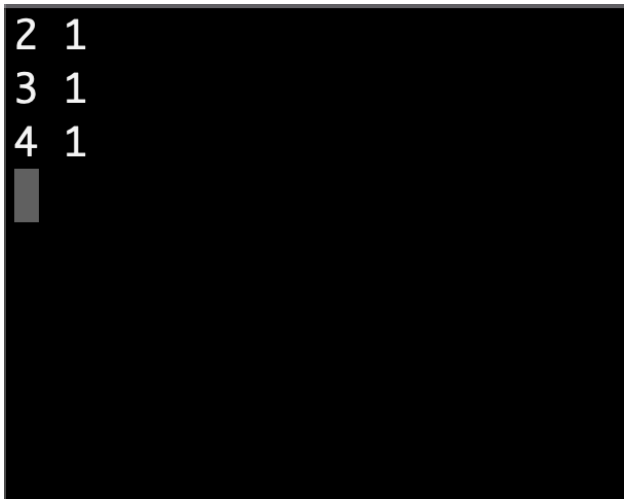
Requirement 1: A button press should be able to send a MQTT packet to our IoT device.
The MQTT broker server on the raspberry pi is able to see every time an MQTT packet is sent from the ESP-32. The first number in each line represents machine id while the second number signifies machine availability (0 not available, 1 available).

Requirement 2: When a user presses the button to use the ARC machine, the LED should light up

to signify the machine is in use.
LED can be seen to light up on our breadboard signifying the output signal and logic from the ESP-32 is working as expected.

Requirement 3: When a user presses the button after using the ARC machine, the LED should be off to signify the machine is not in use.
Similarly to the previous requirement, when the user presses the button the LED turns off, showing the logic and output signal is working as expected.

Requirement 4: Multiple machines must be able to send messages simultaneously
We are able to simulate multiple machines sending messages by manually sending MQTT signals to the same topic (arc_availability) as the one the ESP-32 is sent to.

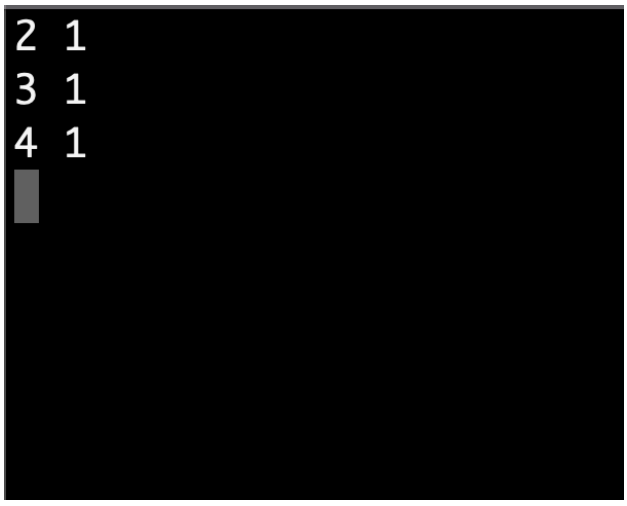


4.2 IoT Device

Requirement 1: Manage multiple MQTT publish messages from a single ESP32 Device
The MQTT broker server on the raspberry pi is able to see every time an MQTT packet is sent from the ESP-32.

Requirement 2: Manage and differentiate multiple MQTT publish messages from multiple ESP32 Devices.

We are able to simulate multiple machines sending messages by manually sending MQTT signals to the same topic (arc_availability) as the one the ESP-32 is sent to.



4.3 Website/MySQL Workbench

Requirement 1: Server must be able to receive MQTT messages from Raspberry Pi

Our script transfers the signal from MQTT and commits a new piece of data to our locally hosted MySQL database. We are able to verify that the MQTT message is successfully delivered by checking to make sure our database is updating properly, which did occur.

Requirement 1: Machine availability is correctly updated on website based on information in

MySQL database

Once the database is updated, our web server is pulling the most updated data directly from the database. We were able to confirm that this was indeed occurring by checking our website right after availability data was sent.

Station ID	Availability
1	Available
2	Available
3	Taken
4	Available

4.4 Power

Requirement 1: Be able to power all on-board components.

We were able to power all of the components that only required 3.3V. This means that the PIR sensor and the LM3914 were unable to be powered. This is due to the ESP-32 not getting enough current for full utilization, as the 3.3V output pin was not outputting the voltage correctly. However, the rest of the circuit was able to be powered by the Lithium Ion batteries correctly.

Requirement 2: Be able to sustain power to all on-board components for 5 days.

Because the entire circuit was not powered by the lithium ion battery alone, we were not able to accurately test this requirement. The battery was able to power the ESP-32 for 5 days during testing.

Requirement 3: Be able to display when a battery needs to be recharged for the convenience of ARC employees.

We were able to confirm this by testing the LM-3914. The integrated circuit works as expected, and is able to properly display the amount of voltage across the lithium ion battery. In this way, the employees will be able to see the amount of battery left in the battery before it is necessary to recharge the device.

5. Conclusion

5.1 Accomplishments

In the end, we were able to accomplish fully working functionality, with all of our subsystems relating to the functionality working together successfully. When a user presses the button, a signal is correctly propagated through WiFi to the Raspberry Pi using MQTT. From there, the

server is able to subscribe to the topic, and commit the availability data to a MySQL database. The data is then successfully pulled and displayed on our Python Flask website. The PIR motion sensor is also correctly working, with the failsafe code working as intended. When the LED is on, but no activity is detected an “Available” signal is sent, but if the LED is off and activity is detected, a “Taken” signal will be sent. The power subsystem mostly worked, the battery level indicator worked and successfully displayed the voltage across the lithium ion battery.

5.2 Uncertainties

Unfortunately, when we connected the circuit together, our ESP-32 was unable to output the correct voltage from the 3.3V output pin. There are a couple things we thought could have been the problem. First of all, our utilization of a boost converter from 3.3V to 5V necessarily lowers the current that is supplied throughout the system. We suspect that the amount of current entering the ESP-32 was not able to exceed the minimum current needed for successful operation of the microcontroller. Another hypothesis we had was that we underestimated the total resistance of the entire circuit. In this way, the total current could also have been lowered, and most likely exacerbated by the boost converter issue discussed before.

5.3 Future Work

In the future, we hope to build upon the current iteration of our project by adding the following features:

1) We would like to expand our system to be fully compatible with different types of machines at the ARC (leg press, shoulder press, lat pulldown). As of now, we are only servicing the Matrix Mega Half Rack.
2) Host website on AWS. As mentioned previously, we hosted our website and database on a local server. For scalability purposes, it would be ideal to integrate with AWS.
3) Create a reservation system. This feature would allow users to schedule a specific time slot where they desire to workout during, thus reducing overcrowding at the ARC.
4) Automatically send the ESP32 into deep sleep mode when minimal motion is detected by the PIR sensor over a large period of time. This would significantly increase the battery life of our system.

5.4 Ethical Considerations

There are a couple ethics and safety issues that our project is facing. We will make sure that nothing in the code of ethics is breached, and that the safety of the user is the utmost priority of the project.

The first ethical issue is the data collection that is performed whenever someone presses the button to modify the machine availability. In order to make sure there are no breaches in privacy, we will not be implementing an account system such that no personal information will

be tracked, nor will patterns be able to be perceived and logged. The only use we currently have in mind for our project is for people to be able to see whether or not the gym is busy regardless of who is utilizing the machines.

The other ethical issue is regarding the email notification function of our website. In order to protect our user's data, we will only be using it for notification purposes, and will make sure the user knows the implications of submitting their email information.

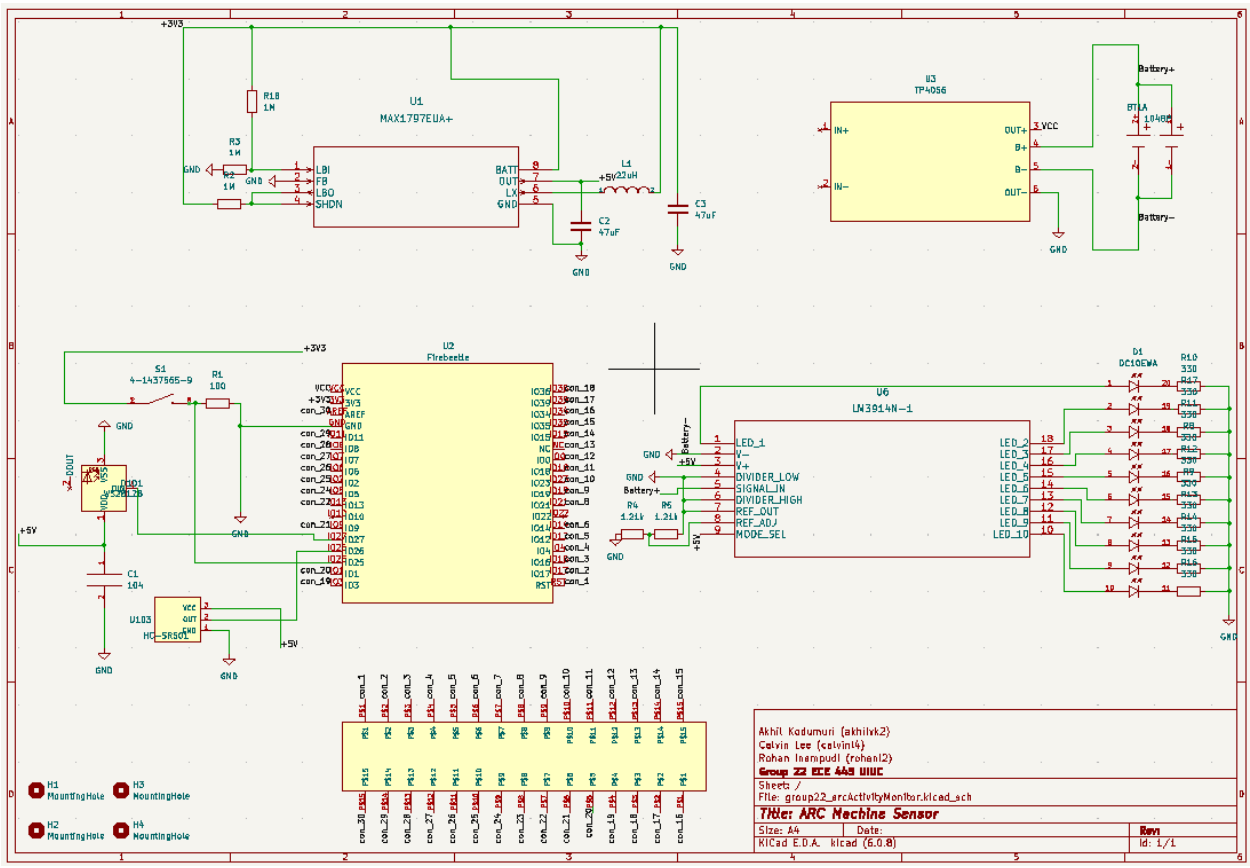
Furthermore, we also have a safety issue due to utilizing a battery. In order to prevent any possible hazards, we will make sure to keep the PCB casing organized, and safely use the batteries within the safe operating range.

5.5 Acknowledgements

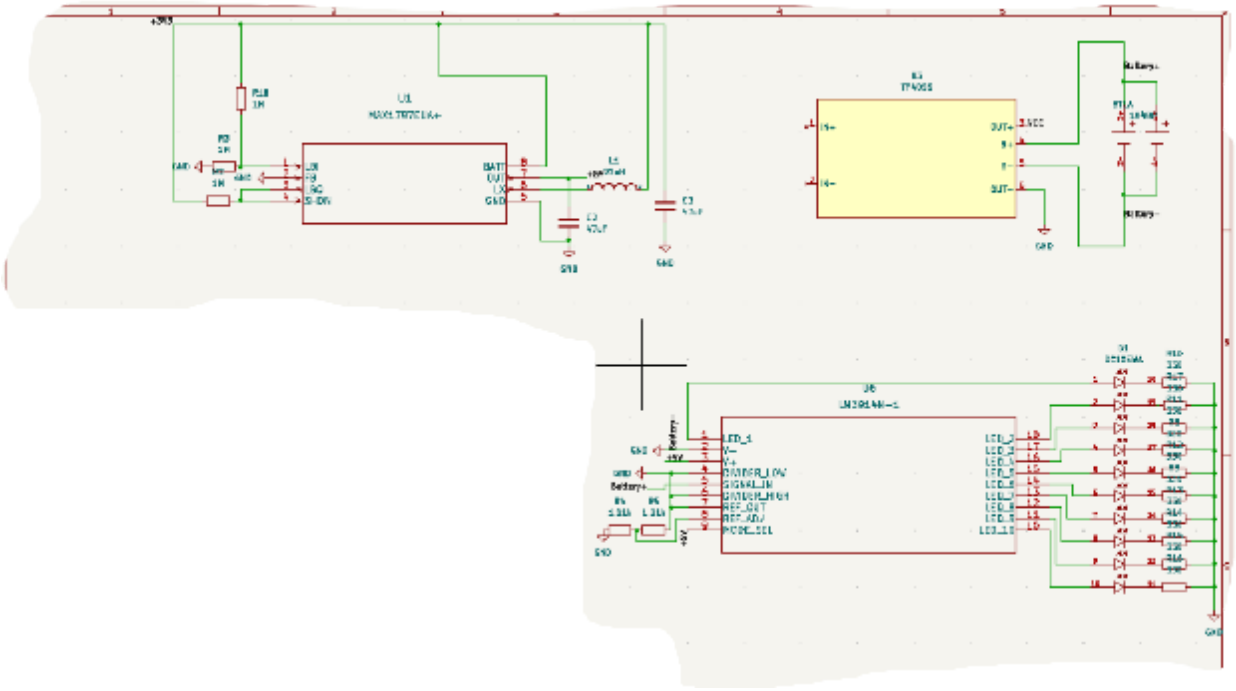
We would like to thank Professor Gruev, first and foremost, who helped oversee our project and act as a mentor throughout the entire process. He gave us priceless advice that not only helped immensely for this project, but also for any projects that we will undertake in the future. Furthermore, we would like to thank Zhicong Fan, the TA for our project, as he helped convey project guidelines, as well as acting as our line of communication with the professors. Finally, we would like to thank all of the other professors and staff that helped set up the class and provide all the resources that we were able to take advantage of.

Appendix A: Schematic

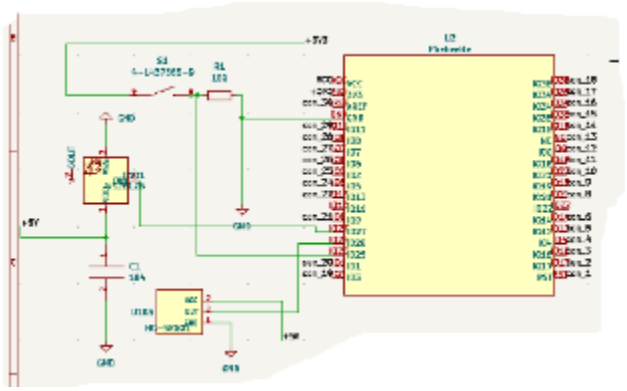
Full schematic:


U1
MAX1797EUA+
U3
TP4056
U2
Firebeetle
U6
LN3914N-1
S1
4-1437565-9
U103
HC-SR501
D1
DC10EWA
Akhil Kodumuri (akhilvk2)
Calvin Lee (calvinl4)
Rohan Inampudi (rohanl2)
Group 22 ECE 445 UIUC
Sheet: /
File: group22_arcActivityMonitor.kicad_sch
Title: ARC Machine Sensor
Size: A4
Date:
KiCad E.D.A. kicad (6.0.8)
Rev:
Id: 1/1

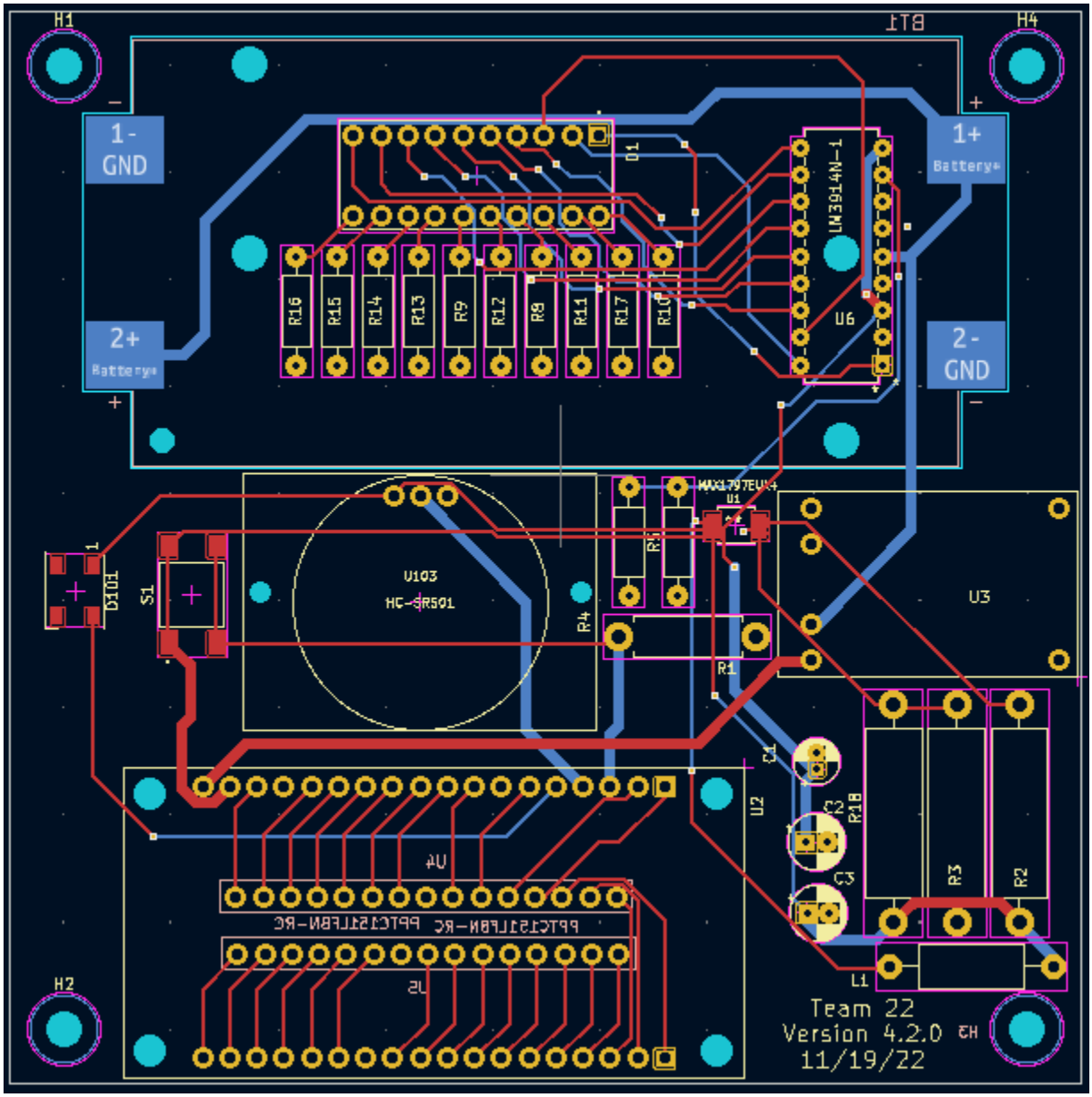
Power Subsystem:



IOT Subsystem:



Appendix B: PCB Design



Appendix C: Requirements & Verification Tables

Subsystem RVs

User Interface

Button Requirements	Verification
Requirement 1: A button press should be able to send a MQTT packet to our IoT device.	Verification 1: On our IoT device, which will act as a MQTT server, we can verify what devices are requesting information on it, by a simple command on the IoT device.
Requirement 2: When a user presses the button to use the ARC machine, the LED should light up to signify the machine is in use.	Verification 2: This can be easily verified by pressing the button on our system and visually checking the activation of the red LED.
Requirement 3: When a user presses the button after using the ARC machine, the LED should be off to signify the machine is not in use.	Verification 3: This can be easily verified by pressing the button on our system and visually checking whether the LED is lit.
Requirement 4: Multiple machines must be able to send messages simultaneously	Verification 4: on the IoT device command line can be used to check what devices are trying to send messages to the device
Motion Sensor Requirements:	Verification:
Requirement 1: Motion sensor should only be able to detect continuous movement from when the machine is in use	Verification 1: We will mimic continuous movement in front of the motion sensor for a defined time interval and check whether or not the website is updated with machine availability.

IoT Device

IoT Device Requirements	Verification
Requirement 1: Manage multiple MQTT publish messages from a single ESP32 Device	Verification 1: Command line within the IoT device interface can be used to verify whether or not each publish message from a particular ESP32 device is being received
Requirement 2: Manage and differentiate multiple MQTT publish messages from multiple ESP32 Devices.	Verification 2: Command line within the IoT device interface can be used to verify if the IoT device is receiving all packets from multiple ESP32 devices

Website

Server Requirements	Verification
Requirement 1: Server must be able to receive MQTT messages from Raspberry Pi	Verification 1: Server has a console to interface with its servers. We can use the server console to ensure the proper packets are being sent from and to the server.
Requirement 2: Server must send packets to our database with information on which machine is in use.	Verification 2: Database has been updated with the correct information.

Flask Application Requirements	Verification
Requirement 1: Machine availability is correctly updated on website based on information in MySQL database	Verification 1: Table displayed on website has the most up to date availability information for all machines.

Power

Power System Requirements:	Verification:
Requirement 1: Be able to power all on-board components.	Verification 1: All subsystems are supplied with the appropriate amount of power to operate.
Requirement 2: Be able to sustain power to all on-board components for 5 days.	Verification 2: All subsystems work as intended for 5 days worth of operation hours at the ARC.
Requirement 3: Be able to display when a battery needs to be recharged for the convenience of ARC employees.	Verification 3: We are using the LM3914 bar graph LED to provide a visual to when the batteries will need to be recharged.

References

[1] *"Amazon.com: Gikfun 12mm waterproof push button momentary on off switch ..." [Online]. https://www.amazon.com/Gikfun-Waterproof-Button-Momentary-Arduino/dp/B07W5TGKQ3. [Accessed: 15-Sep-2022].*

[2] "PS1440P02BT: Digi-key electronics," *Digi*. [Online]. Available: https://www.digikey.com/en/products/detail/tdk-corporation/PS1440P02BT/2236832?utm_adgroup=Alarms%2C%2BBuzzers%2C%2Band%2BSirens&utm_source=google&utm_medium=cpc&utm_campaign=Shopping_Product_Audio%2BProducts_NEW&utm_term=&utm_content=Alarms%2C%2BBuzzers%2C%2Band%2BSirens&gclid=Cj0KCQjw39uYBhCLARIsAD_SzMRocrg56djQZdtSr1banc2WuquRuRWNwZ3Xb1x-w5BqNdJqJw-9-HQaAjB5EALw_wcB. [Accessed: 15-Sep-2022].

[3] "Chipsets: Espressif Systems," *Chipsets | Espressif Systems*. [Online]. Available: https://www.espressif.com/en/products/socs. [Accessed: 15-Sep-2022].

[4] Németi Florian, G. Pauletto, and D. Duay, “IOT: L'émancipation des objets,” *Amazon*, 2017. [Online]. Available: https://docs.aws.amazon.com/iot/latest/developerguide/connecting-to-existing-device.html#gs-device-view-msg. [Accessed: 15-Sep-2022].

[5] Cdaviddav, “Send data from ESP8266 or ESP32 to Raspberry Pi via MQTT,” *DIYI0T*, 07-May-2021. [Online]. Available: https://diyi0t.com/microcontroller-to-raspberry-pi-WiFi-mqtt-communication/. [Accessed: 15-Sep-2022]

[6] “External+Circular+Battery+for+PCB,” *Google Shopping*. [Online]. Available: https://www.google.com/shopping/product/5898283066540390 66?q=external%2Bcircular%2Bbattery%2Bfor%2Bpcb&client=safari&rls=en&sxsrf=ALiCzsZaA5BHehX59EQpajrvZOfjehFTwg%3A1663275125169&biw=1440&bih=735&dpr=2&prds=eto%3A2725741664819496371_0%2Clocal%3A1%2Cpid%3A11856754887431340 41%2Cprmr%3A2%2Crsk%3APC_1026065390291347991 1&sa=X&ved=0ahUKEwiX2P6R15f6AhXpjIkEHVlACSgQ8wIIuBE. [Accessed: 15-Sep-2022].

[7] “2-1775485-1 : Battery holders,” *TE Connectivity*, 03-Jan-2018. [Online]. Available: https://www.te.com/usa-en/product-2-1775485-1.html?te_bu=Cor&te_type=srch&te_campaign=ggl_usa_cor-ggl-usa-srch-smbmktg-fy22-googlefeed_sma_sma-2210_2&elqCampaignId=115724&mkwid=VDybDFVR%7Cpcrid%7C386964346943%7Cpkw%7C%7Cpmt%7C%7Cpdv%7Cc%7Cslid%7C%7Cproductid%7C2-1775485-1%7Cpgrid%7C78782457763%7Cptaid%7Cpla-298884436745%7C&utm_content=VDybDFVR%7Cpcrid%7C386964346943%7Cpkw%7C%7Cpmt%7C%7Cpdv%7Cc%7Cslid%7C%7Cproductid%7C2-1775485-1%7Cpgrid%7C78782457763%7Cptaid%7Cpla-298884436745&gclid=Cj0KCQjwmouZBhDSARIsALYcourXPqfM1fucoo39DQsdyYPBcSBieWtgXxXjzO0ox1Y1Peppnti2pE0aAkw-EALw_wcB. [Accessed: 15-Sep-2022].

[8] Last Minute Engineers, “Insight into ESP32 sleep modes & their power consumption,” *Last Minute Engineers*, 04-Jul-2022. [Online]. Available: https://lastminuteengineers.com/esp32-sleep-modes-power-consumption/#:~:text=The%20chip%20consumes%20around%200.15,coprocessor%20is%20on)%20to%2010%C2%B5A. [Accessed: 28-Sep-2022].

[9] Cdaviddav, “Guide to reduce the ESP32 power consumption by 95%,” *DIYI0T*, 07-May-2021. [Online]. Available: https://diyi0t.com/reduce-the-esp32-power-consumption/.

[Accessed: 28-Sep-2022].

[10] Smt, “18650 battery how to charge - 5 simple builds,” *SM Tech*, 14-Aug-2022. [Online]. Available: https://somanytech.com/18650-battery-how-to-charge-18650-battery-with-charger-and-without-charger/#:~:text=USB%20powered%2018650%20battery%20chargers,you%20can%20say%20slow%20charging. [Accessed: 28-Sep-2022].