Akhil Kodumuri Journal Entries

9 - 15 - 2022

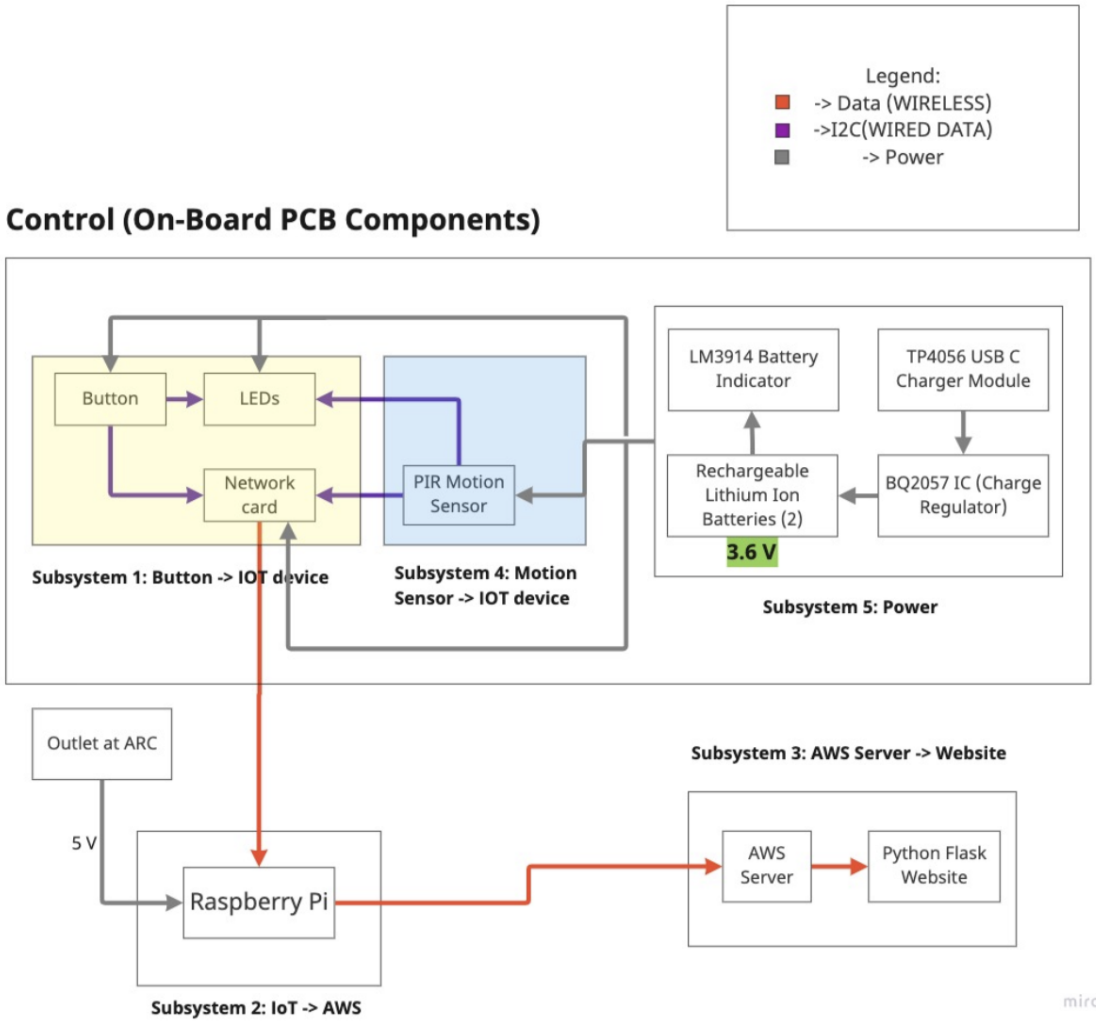
Today the group finished writing the proposal for our (https://github.com/akod0883/ArcMachineMonitor/blob/main/proposal.pdf). We discussed what we were all passionate about and how to incorporate it all into the project.
In short, the main parts of our project proposal includes what our project is about, and the subsystems associated with them
We are aiming to build an "ARC Machine Monitor" which will detect when a student at the arc is using a machine. There will be a subsystem including a Button and motion sensor. If the button is pressed or the motion sensor detects movement, it will communicate data to an ESP32, which will communicate this data to a raspberry pi, which will then send data to a server where our website will be hosted. Our website will have an interface where people can see what machines are in use.

9 - 16 - 2022

Today the group met with the ece machine shop and gained insight on website and resources to research parts to use for our project. We will now do a deep dive on Sparkfun and Adafruit. The employees at the shop also gave us the names of the following sensors: Flex sensor, Holofect magnetic sensor, and Nfc tag which can be used to create the load bearing sensor we will use.

9 - 25 - 2022

Today I worked on looking for specific parts to order from the ECE machine shop. I also did some work on the design document. With this, we gained a better understanding of how our circuit will work and what each subsystem will look like. We got a preliminary look at what the overall block diagram will look like. There were concerns about the quality of our block diagram, so we will attempt to further improve the block diagram. The concerns: blocks were not specific to the parts we will actually use in our project. Subsystems did not explain itself, we did not properly include the connections of data.



9 - 29 - 2022

Today I worked on updating the design document to include RV tables and tolerance analysis. We also spent time calculating the total amount of energy our circuit and project uses. This is useful because one of the goals of our project is to sustain for at least 4 days of the ARC operation. This is because we don't want the workers of the arc to recharge our batteries every day. Below in the table, the calculations for energy consumption can be seen.

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

In order to determine the calculations within this table, we used the following equation to generate the total power consumption:
$PowerConsumption = OperationCurrent * TimeOfOperation$

$OperationCurrent$ can be found by looking the datasheets for each of the individual components that we will use within our project and the value of $TimeOfOperation$ will be substituted with our High Level requirement of 5 days. Thus, we are able to generate a value that signifies the total power operation of our circuit. With this information, we can now figure out what battery to choose. After much research, we decided to go with the following rechargeable lithium battery https://www.18650batterystore.com/products/panasonic-ncr18650ga-ga6?variant=40563598000279.

10 - 4 - 2022

We had our design review where we got more information about how to change our project. We need to make our block diagram more specific to accurately represent our project. Our individual subsystems do not speak for themselves. With this information, we decided to reorganize how we describe the various subsystems within our project. Previously, we communicated our subsystems according to how the various components of our project interacted with each other. For example, the process of ESP32 communicating its information to our IoT device (Raspberry Pi) would be a subsystem. However, we now recognize that this is a very confusing way to depict the working components of our website. Instead, we will divide the subsystems into abstractions that represent each component of our project. Note, that these abstractions contain the processes of our old subsystem breakdown. With the previous example that was used, the process of ESP32 sending its data to the Raspberry Pi would fall under the Control subsystem because the data being communicated is an outgoing packet from

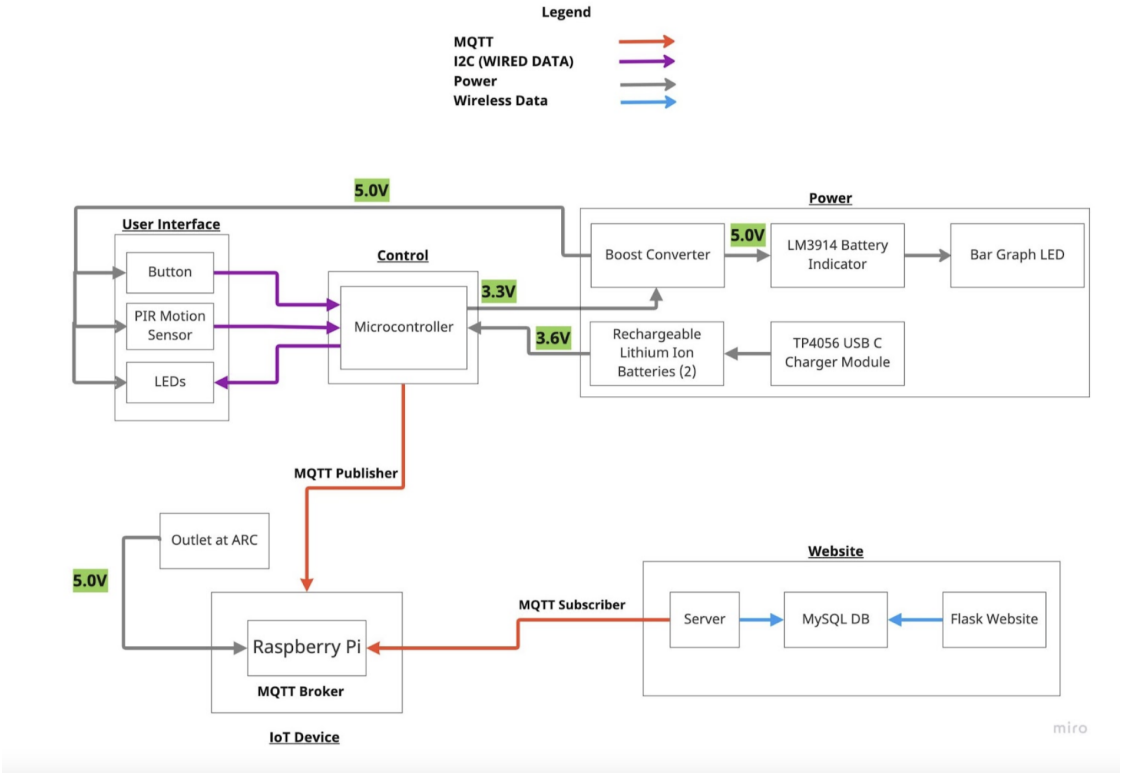
the ESP32. Thus, it would fall under the Control subsystem. A high level break down of each subsystem is as following:

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.

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.

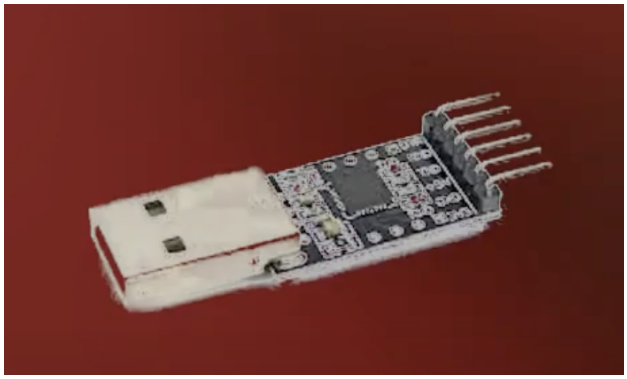
The IoT device utilized in this project was Raspberry Pi 3 B. The Pi acts as a "middleman" between the ESP32 within our Control subsystem and the website. This decision was made to reduce the energy consumption on the Control subsystem. Instead of the ESP32 sending MQTT packets to the website, all ESP32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

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.


Legend
MQTT
I2C (WIRED DATA)
Power
Wireless Data
5.0V
Power
User Interface
Button
PIR Motion Sensor
LEDs
Control
Microcontroller
3.3V
3.6V
Boost Converter
5.0V
LM3914 Battery Indicator
Bar Graph LED
Rechargeable Lithium Ion Batteries (2)
TP4056 USB C Charger Module
MQTT Publisher
Outlet at ARC
5.0V
Website
MQTT Subscriber
Server
MySQL DB
Flask Website
Raspberry Pi
MQTT Broker
IoT Device
miro

10 - 8 - 2022

We conducted research on what development board to use. We decided to go with the Firebeetle. There were many reasons for why we went with the Firebeetle (https://www.dfrobot.com/product-1590.html). It should be noted that the Firebeetle is a ESP32 development board. The development board allows an easy way to program the ESP32. Also, one benefit of using the Firebeetle is that it will manage the amount of power the ESP32 will draw. The designers of the development board had efficiency in mind and, thus, the ESP32 active mode and passive mode conduct less power. The alternative for this is to use a USB, UART TTL, and ESP32. The UART allows for programming the ESP32. However, there is a lot of risk with using this method. Because of the GPIO pins of the ESP32 and the way the pins interface with the flash memory, we decided the risk out weighed the rewards. But, because the overall footprint of this model would be significantly smaller than using the Firebeetle dev board. So, because of this, we will order these parts and try our best to work with it. In order to understand this process we will refer to this video from youtube: https://www.youtube.com/watch?v=lH7hVTUL7W8



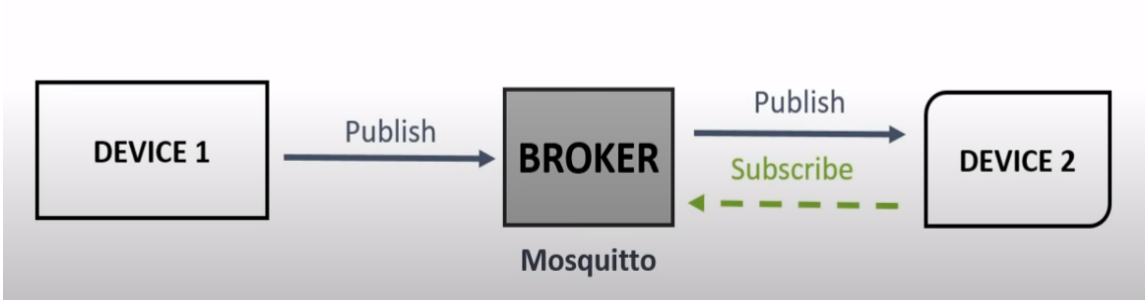
The image above shows a USB UART TTL which allows for usb communication to the ESP32.

10 - 9 - 2022

I conducted research on my own on how the MQTT network protocol will be used to connect the ESP32 with the Raspberry Pi 3 B+. The following is the research that I found about the MQTT.

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

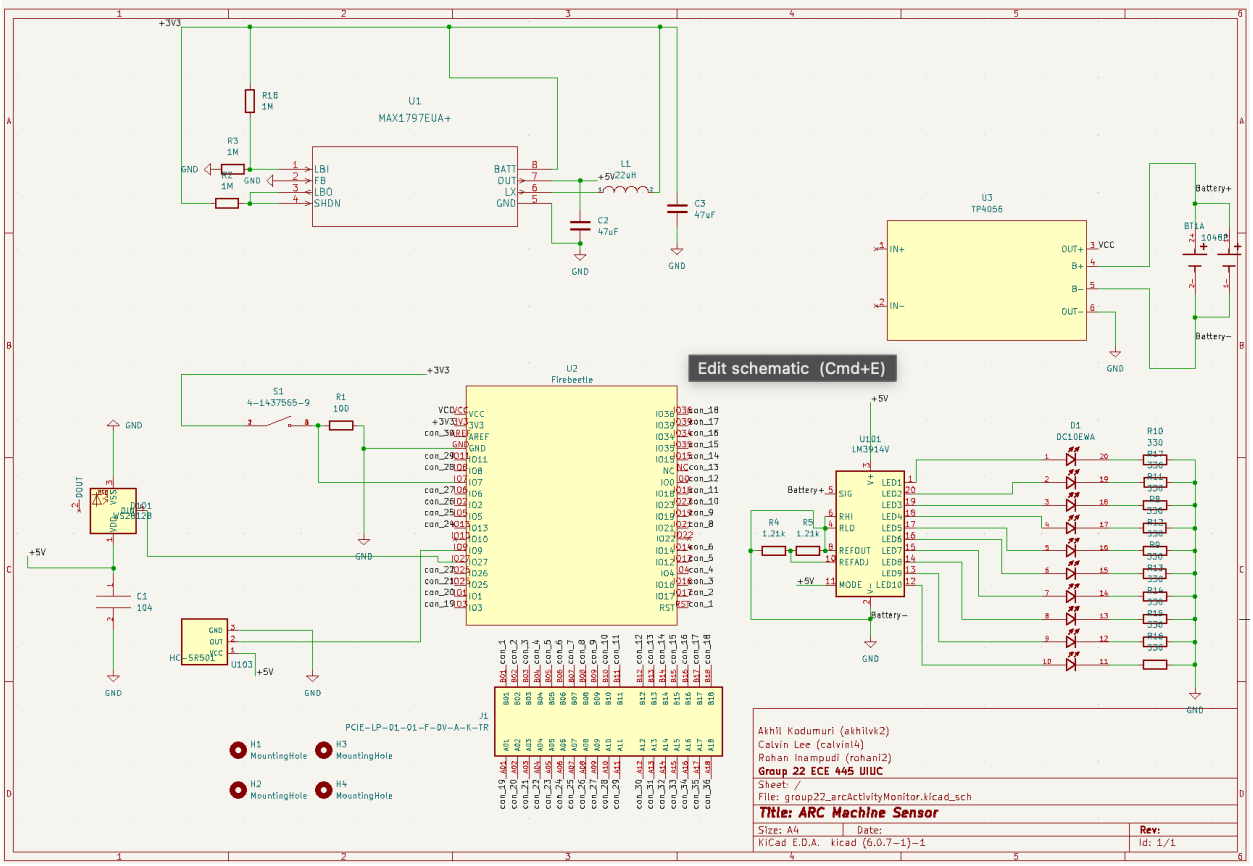
and filters all messages, and publishes the messages to all clients that are subscribed to that topic. In our case, the ESP-32 will be acting as a client, and the Raspberry Pi will be acting as both a client and a broker. Once the button is pressed, the data for a certain topic (for example, “data/arc_availability”) will be published from the ESP-32 to the Pi, and the Pi, which is subscribed to that topic, will receive that published data.

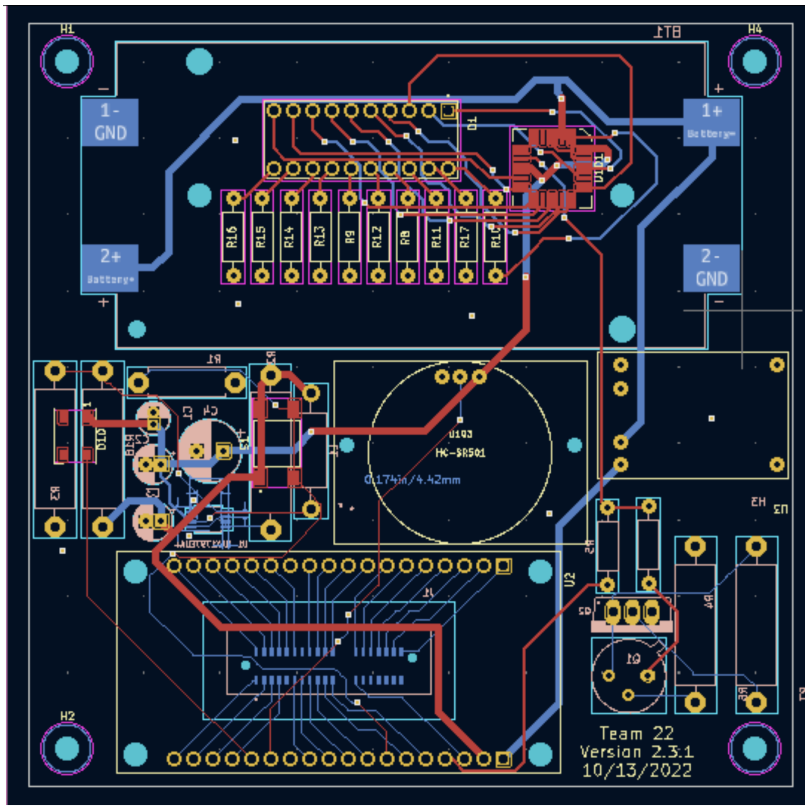


Now with this information, I am now able to understand more details about how the Control, IoT, and Website subsystem will communicate machine availability data. As shown in the image above, device 1 would be signified with the EPS32, the broker would be the Raspberry Pi, and device 2 will be the server hosting our website. The that each block in the model would subscribe/publish to would be called eps32/machine_availablilty. The packets being sent would then contain the following two pieces of information: machine id (unique number to signify a machine at the ARC) and machine availability (0 machine is in use, 1 machine is in use).

10 - 10 - 2022

Since the first round PCBWay order is due the 11th, we spent extensive time working on making our circuits and designing our PCB. We designed the PCB in a way that the button would be in a spot which will benefit a user at the ARC. The button should be in a visible spot so a person at the ARC can easily press it to signal machine availability. Also, the motion sensor should be in a spot to maximize range and versatility. We also designed the battery pack to be on the back of the PCB, so it is out of the way of the components that a user would be interacting with. For this reason the batteries are on the bottom of the PCB. The TP4056 charging module that we use will be in a spot where the employees at the ARC will have an easy time plugging in a USB C cable. Also, we chose USB C as the cable type because electronics in our current world are shifting towards this standard. Below is the schematic and pcb that we will submit to PCBWay.


Edit schematic (Cmd+E)
U1
MAX1797EUA+
U2
Firebeetle
U3
TP4056
U101
LM3914V
U103
HC-SR501
D1
DC10EWA
Akhil Kodumuri (akhilvk2)
Calvin Lee (calvinl4)
Rohan Inampudi (rohani2)
Group 22 ECE 445 UIUC
Sheet: /
File: group22_arcActivityMonitor.kicad_sch
Title: ARC Machine Sensor
Size: A4
Date:
Rev:
KiCad E.D.A. kicad (6.0.7-1)-1
Id: 1/1


1-
GND
1+
Battery+
2+
Battery+
2-
GND
U103
HC-SR501
0.174in/4.42mm
Team 22
Version 2.3.1
10/13/2022

During the process of creating the schematic, we realized that we needed to include extra components in order to correctly activate and use some components on our PCB. For example, the LM3914 battery graph module requires 5V to activate. In our original thought process, this was not a consideration that we made. We stumbled across this information when we were reading datasheets. So, we included the MAX1797EU which is a step up voltage converter. Given any low input voltage, the output of the MAX chip is 5V (which we programmed using the resistor and capacitor values shown in the image above). Using this chip we are able to power the following modules: HC-SR501 (PIR motion sensor), LM3914, and LED. We also noticed that the Firebeetle contains an output pin that consistently outputs 3.3V. Furthermore, 3.3V is enough to use the rest of the components in our circuit. We are able to now think of the Firebeetle as a power source. This simplifies the circuit, so it is easy to create a PCB around the Firebeetle.

Another way this can be implemented is instead of using a step up voltage converter, we could use a step down voltage converter and use a main battery power source of 7V. This way we would only step down to 5V and then feed current into the Firebeetle and utilize the 3V3 output pin to all the components that need 3.3V.

We also decided to include a connector module. This will allow for freedom when wiring or debugging the Firebeetle module.

10 - 11 - 2022

Today, we sent in our PCB order on PCBWay. Now, we started researching where to buy all the parts necessary to build our PCB. Below, in the table, is a breakdown of all the parts that perform significant functions within our PCB. The rest of the components are resistors and capacitors that can be found in the lab. This breakdown is important to have because we can figure out how expensive it would be to scale our project to handle multiple machines. If we were to present our project or think about going commercial, investors would ask how our project could be horizontally scaled. Theoretically, one PCB would be on every machine at the arc. One raspberry pi would be enough because it will act as a single middle man between our AWS server and ESP32s. In order to obtain these parts we are ordering them through the ECE shop and any part that they cannot fulfill we will order ourselves through Amazon or any other distributor.

Description	**Manufacturer**	**Quantity**	**Extended**	**Link**

			Price	
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

Complete parts breakdown can be found in the following link: https://docs.google.com/spreadsheets/d/1l01G8KRwHKcs5DLiIkFAdf3STNrrx1jut49VXsr8v5M/edit?usp=sharing

10 - 13 - 2022

We received an email from Professor Gruev explaining that one of the modules that we plan to use should not be used. Because of this, we met him on a Zoom call to discuss why and what should be used instead. Below are the key points from the meeting.

1. TP4056 Charging module shouldn't be used since its a circuit bought from Amazon
2. We should design our own charging circuit if we get a working pcb with the module bought from Amazon

3. We can refer to Professor Gruev if we need any help and for confirmation on our circuit

After the meeting the group and I discussed whether or not to order a new PCB based on these criteria. We came to the following conclusion: we will first focus on getting our initial PCB design soldered working, then we will solder the new design if there is time. But, since we still have not received our pcb order, we will create a new schematic for the design that Professor Gruev recommended.

10 - 14 - 2022

Today we met to make the necessary changes to our circuit to adhere to what Professor Gruev mentioned. We removed the TP4056 charging module from Amazon from our circuit and replaced it with our own implementation of a charging circuit. We have a USB-C module for changing, the tp4056 for charge control, and another chip in order to make sure that there is no overflow charge going into the rest of the circuit. Now we will schedule a meeting with professor Gruev in order to see if our circuit logically makes sense. We have some questions about the shields we will use for our USB-C and whether or for the overflow charge chip is necessary. We were confused on what to connect to the shield pins of the USB chip.

10 - 16 - 2022

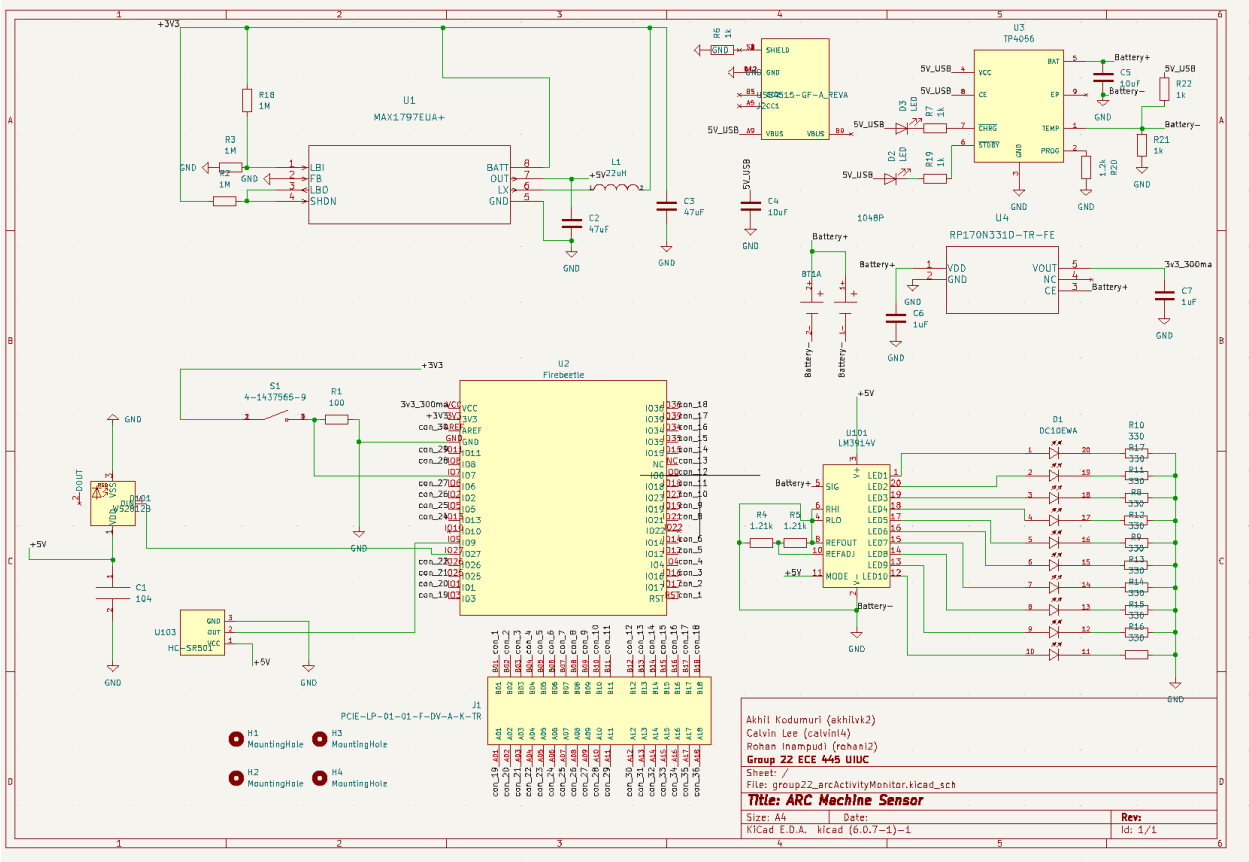
We had our meeting with Professor Gruev, and he gave us the following recommendations.

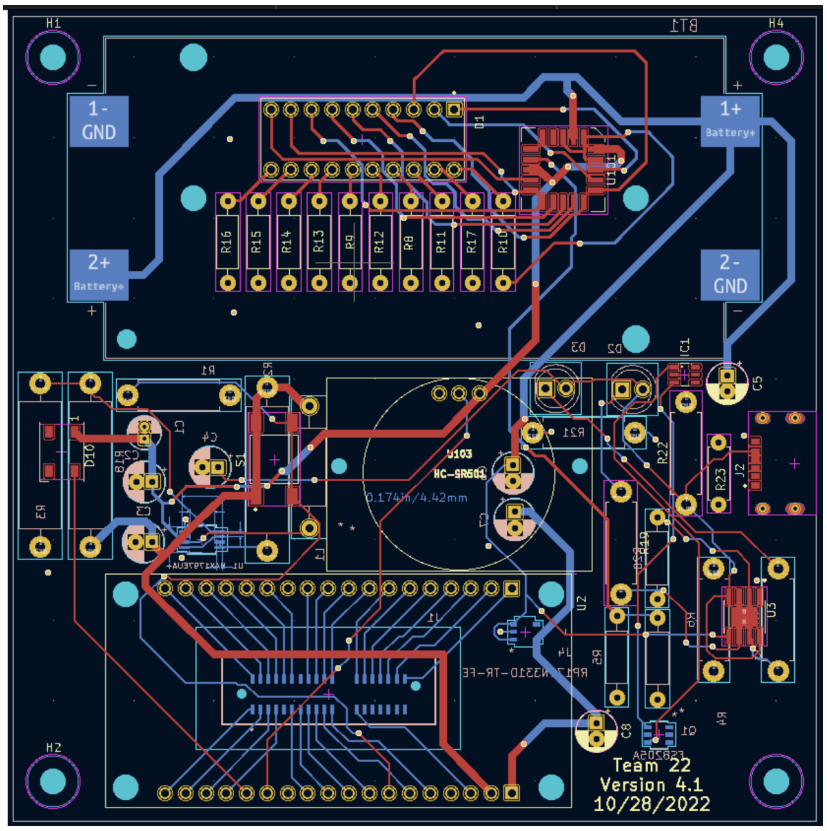
1. Connect the shield pins to pull down resistors and GND
2. Remove the overflow charge chip because the TP4056 should do the same function
3. Add a current regulator to ensure that the Firebeetle is not getting too much current

With this information we will change our schematic accordingly.

10 - 18 - 2022

We met as a group to change our schematic and order the PCB that will only be soldered and made if our first design PCB works with 100% functionality.


Akhil Kodumuri (akhilvk2)
Calvin Lee (calvinl4)
Rohan Inampudi (rohani2)
Group 22 ECE 445 UIUC
Sheet: /
File: group22_arcActivityMonitor.kicad_sch
Title: ARC Machine Sensor
Size: A4
Date:
Rev:
KiCad E.D.A. kicad (6.0.7-1)-1
Id: 1/1


1-
GND
1+
Battery+
2+
Battery+
2-
GND
Team 22
Version 4.1
10/28/2022

Note that all the recommendations that Professor Gruev made were accurately reflected in the schematic above.

1. All the Shield pins are grounded with pull down resistors included
2. We do not included any overflow charge chip
3. We included the RP170N331D chip which is a current regulator that ensures that only 330 mA of current go into the Firebeetle. Note that this is the current necessary for Active mode of the ESP32 to operate (active mode allows for WiFi connection).

10 - 24 - 2022

We are currently waiting for all of our parts to arrive. Until we get all our parts we cannot begin to breadboard and understand if everything works correctly. So, we met to take this time to plan out how we will organize our lives, schedules, and work to make sure we make substantial progress when all of the parts arrive. We all decided to carve out the same time from our schedules in order to make sure we all get work done. Also, if we have any questions for each other we can easily ask each other. Once all the parts arrive we will benign to put more consistent work into the PCB, but we currently cannot due to the nature of the parts not being here.

In the meantime, since I have my own Raspberry Pi 3b+, I have started to familiarize myself with the Pi because I have never used one before. I conducted some research on how to get started with the Raspberry pi. I used the following links in order to learn how to format, boot, and ssh into the Raspberry Pi: https://libguides.nyit.edu/c.php?g=469894&p=3365470, https://www.imore.com/how-get-started-using-raspberry-pi. In order to reproduce these steps, the next part of today's post will contain steps on what I did to set up the Raspberry Pi 3B+.

1. Download the Raspberry Pi Imager application on laptop (https://www.raspberrypi.com/software/)
2. Insert MicroSD card that will be used for the Pi into laptop MicroSD card reader
3. Choose the 32 bit Raspbian image with Desktop mode included
4. Once the image is installed on the MicroSD card, insert the card into the Raspberry Pi.
5. Connect on end of the ethernet cable into the Pi and the other into the laptop
6. Run the following command in order to ssh into the Raspberry Pi `ssh pi@raspberrypi.local`

With these download steps, a reader of this journal can also reproduce the Raspberry Pi setup into their own Pi. The next step for the Raspberry Pi is to figure out how to convert the Raspberry Pi into a MQTT broker. I will continue to do research on my own on how to do this in

a way that will benefit the project. I was able to connect the Pi to the same network on the Raspberry Pi. I was also able to ping the Raspberry Pi and my personal laptop. This is a good sign because at least they are able to recognize each other on the same network.

10 - 28 - 2022

We have now received enough parts to start soldering them onto the PCB. However, when we started soldering, we noticed that we never received our LM3914 and the order was never received at the machine shop. We are unsure if there were any issues on our end, the machine shop, or the chip digistributer we told the machine shop to order from. I will send an email to find our if there are any issues, if not, we will order the chip ourselves.

Since we have all of our parts, we decided we would breadboard all of the parts in order to figure out if we understand how all of the parts work. During this process we wanted to see if we could get the esp32 to recognise a button press or input from our PIR motion sensor. In order to program the ESP32 we used the Arduino IDE. It run with C++ like code so we were able to quickly pick up how to program the ESP32. The Firebeetle documentation provides starter code and projects. So we took the following snippet to get the ESP32 to just blink (https://wiki.dfrobot.com/FireBeetle_Board_ESP32_E_SKU_DFR0654#target_6)

```
// the setup function runs once when you press reset or power the board
   void setup() {
     // initialize digital pin LED_BUILTIN as an output.
     pinMode(LED_BUILTIN, OUTPUT);
   }

   // the loop function runs over and over again forever
   void loop() {
     digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the voltage level)
     delay(1000);                       // wait for a second
     digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage LOW
     delay(1000);                       // wait for a second
   }
```

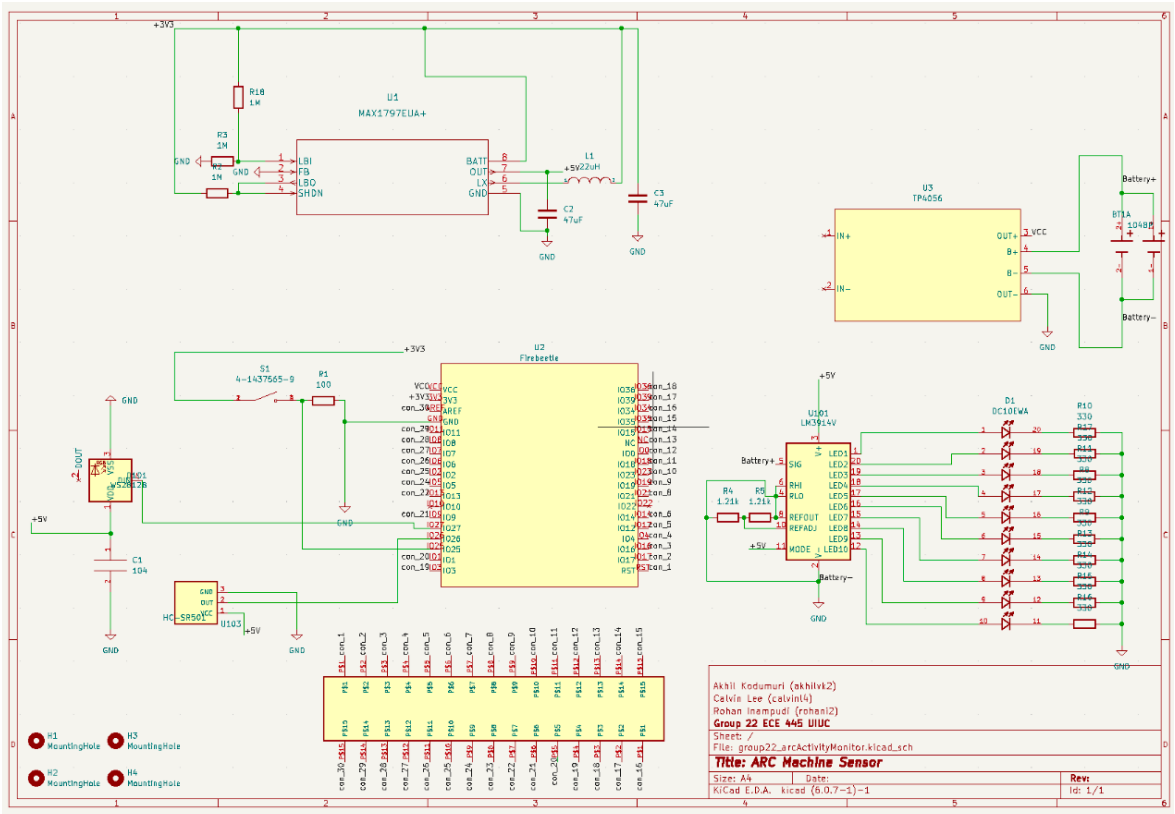
When soldering, one of our group members realized that our original schematic was incorrect. We accidentally used pins 6-11 on the Firebeetle which are unusable pins for the ESP32 because they lead directly into the Firebeetle's flash memory. Thus, we can't program the ESP32 to recognize the input from our button and pir motion sensor. Another mistake that we made was ordering the wrong part for our wire connectors. We accidentally ordered a PCI connector module for PCs. So we took some time to order the correct wire connectors.

Debug Steps:

1. Check whether or not esp32 blink code was working
2. Checking whether or not we can pass current to ESP32 pin, if current detected output current from another pin to led
3. LED did not light up but current was detected
4. Research the pins we are using
5. Realized that pind 6-11 lead directly into flash memory so they cannot be reliably used as I/O

10 - 30 - 2022

As mentioned in yesterday's post, we realized that we made a mistake on our schematic leading to an incorrect pcb design. Today we met and made changes to our schematic which should fix the issues we are encountering. We changed the footprint of the wire connectors and changed the pins that we will be using for the ESP32. We updated our PCB design and schematic. Below is our updated schematic:



Notice in the schematic above there is a different connector and I/O on the Fire Beetle.

10 - 31 - 2022

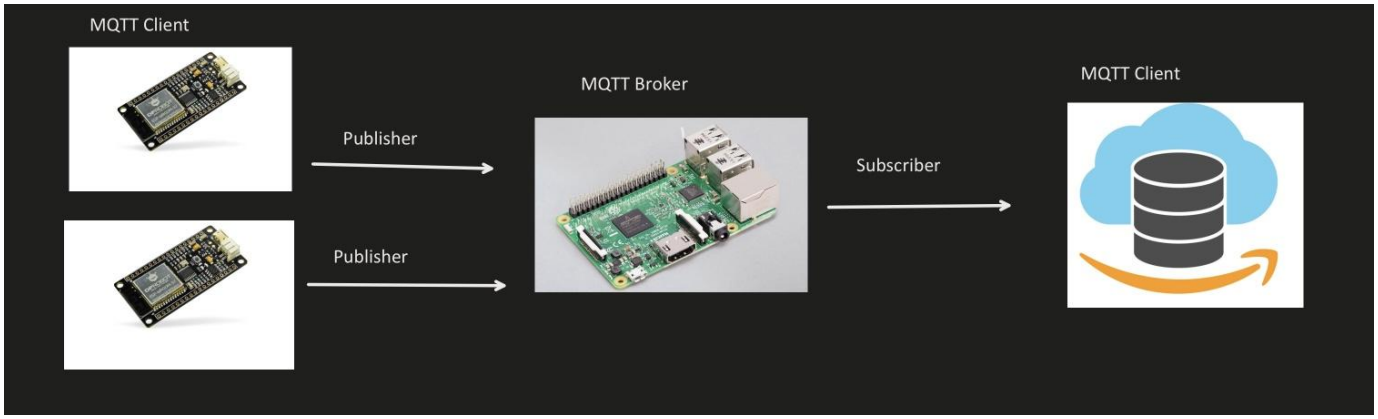
We ordered our second round PCB, and I started working on my individual progress report. Currently, I am trying to come up with an outline for how the progress report will look like and what the content will be. Not much physical progress has been made. Most of the work that I've done has been more on the research side. Below is an image of how I will organize the individual progress report.

Contents

Contents	**2**
1. Introduction	**3**
1.1 Team Project Overview	3
1.2 General Overview of Individual Responsibilities	3
2. Design	**4**
2.1 Considerations	4
2.1.1 Power	4
2.1.2 Network Protocol	5
2.1.3 Button/Motion Sensor to IoT Device	6
2.1.4 IoT device to AWS Server	7
3. Testing	**8**
4. Conclusion	**9**
5. References	**10**

11 - 1 - 2022

Today I continued working on my individual progress report. I will outline the key takeaways from the individual progress report here. I summarized my contributions to building the new charging subsystem on our PCB. I extensively explained the research I conducted about the MQTT network protocol and how it relates to the project. The entire explanation can be summarized with the following image and brief description:



The MQTT protocol uses the subscriber, publisher, and broker model. The publisher will post data to the broker who acts as a middle man between the subscriber and publisher. The subscriber will then poll the broker looking for data that the publisher sends. For the purposes of our project, the ESP32 would be the Publisher, the Pi would be the broker, and the AWS server hosting our website would be the subscriber. Note that the image above contains 2 publishers the reason for this is two fold. As per our high level requirements, we are making two working versions of our PCB to show that our system is horizontally scalable. Two, to show that this system works with multiple publishers. Theoretically, each publisher would be a working version of a PCB on each machine at the arc. They would each be publishing machine availability to a different topic to the broker. The server would then contain code to loop and collect data from each topic.

I also wrote about the code that will be written on the ESP32 to communicate availability information to the Raspberry Pi. Below is an image of the pseudocode

```
while system_is_alive:
    if button_pressed:
        while(start_5_min_timer):
            LED_ON
            publish_machine_in_use
            if button_pressed:
                LED_OFF
                publish_machine_not_in_use
                break
    else motion_sensor_detects_movement:
        while(start_5_min_timer):
            LED_ON
            publish_machine_in_use
            if button_pressed:
                LED_OFF
                publish_machine_not_in_use
                break
```

Within this code, is the fail safe if a user does not press a button when they start working out. Also, if the user doesn't press the button after working out, there is a timer that turns off the led. Note, that every time the LED turns on or off this information is eventually sent to to our website.

The last bit of information that I included in my individual project report was the command to turn the raspberry pi into a MQTT broker. Below is the command to do so.

```
sudo systemctl enable mosquitto
```

11 - 7 - 2022

We met as a group to work on our PCB because we received both designs: one where we use the TP4056 module from Amazon and where we designed our own charging circuit that mimics the TP4056 Amazon module. The group split up the work based on the subsystems. Currently, my role is to figure out how to convert the Pi into a MQTT broker. Also, I am trying to figure out how to test publishers sending messages and subscribers receiving them. Ultimately, my goal is to learn everything about the MQTT process and to witness this with my Pi and MacBook. This is because eventually we will be writing code that will need to connect to the Pi to send MQTT messages. In order for that to happen, it will need to find the broker on the same network as it. Currently, I am trying to get this process to work on the Illini network, guest or net. I am currently following this guide: https://pimylifeup.com/raspberry-pi-mosquitto-mqtt-server/. These steps use the Pi as a subscriber, broker, and publisher. The raspberry pi localhost acts as a broker. These are the steps that I followed:

```
sudo apt install mosquitto mosquitto-clients
```

```
mosquitto_sub -h localhost -t "mqtt/pimylifeup"
```

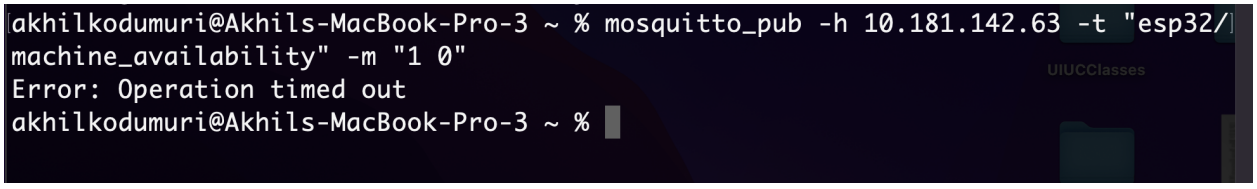
```
mosquitto_pub -h localhost -t "mqtt/pimylifeup" -m "Hello world"
```

However, when I substitute localhost with the IP address of the Pi and I try to send a message from my mac to Pi (which would simulate the PCB sending a message to the Pi). However, I keep getting the following:

Recreating ERROR:

On Pi (broker/subscriber)

On Mac (publisher)



Debug:

1. I already made sure that both the Pi and my Mac were both connected to the same network.
2. I tried to ping from my macbook to pi and works and from to pi both **do not work**

```
akhilkodumuri@Akhils-MacBook-Pro-3 ~ % ping 10.181.142.63
PING 10.181.142.63 (10.181.142.63): 56 data bytes
Request timeout for icmp_seq 0
Request timeout for icmp_seq 1
ping: sendto: No route to host
Request timeout for icmp_seq 2
^C
--- 10.181.142.63 ping statistics ---
4 packets transmitted, 0 packets received, 100.0% packet loss
akhilkodumuri@Akhils-MacBook-Pro-3 ~ %
```

11 - 10 - 2022

I continued to debug the error I am getting when trying to send an MQTT message from my Macbook to Pi. I did more research and I believe the issue has to to with the firewall on UIUC network

Possible Solution: https://courses.engr.illinois.edu/cs424/fa2016/mp/init.pdf

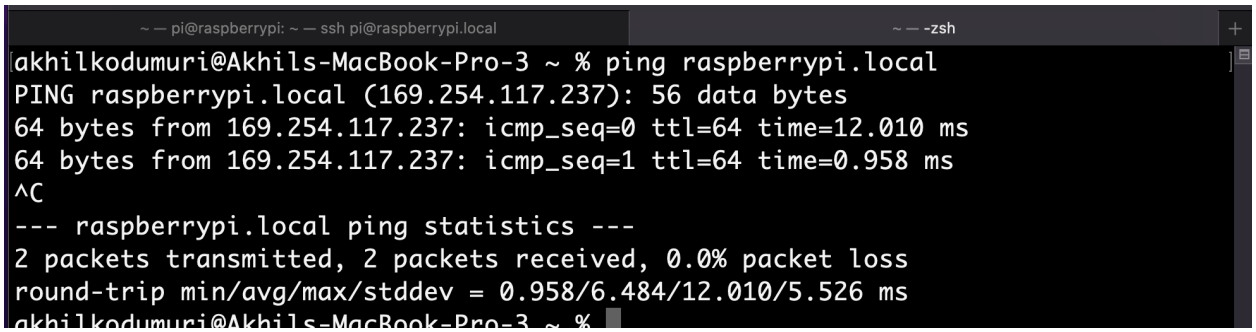
In order to get around this error, I will connect both my MacBook and Pi to my own personal hotspot from my phone.

Solution: Personal hotspot

Now I am able to send messages from my macbook and pi

Solution Images:

```
akhilkodumuri@Akhils-MacBook-Pro-3 ~ % ping 10.181.142.63
PING 10.181.142.63 (10.181.142.63): 56 data bytes
Request timeout for icmp_seq 0
Request timeout for icmp_seq 1
ping: sendto: No route to host
Request timeout for icmp_seq 2
^C
--- 10.181.142.63 ping statistics ---
4 packets transmitted, 0 packets received, 100.0% packet loss
akhilkodumuri@Akhils-MacBook-Pro-3 ~ %
```



Now I can ping between my mac and raspberry pi

```
akhilkodumuri@Akhils-MacBook-Pro-3 ~ % mosquitto_pub -h raspberrypi.local -t "es
p32/machine_availability" -m "Hello World"
akhilkodumuri@Akhils-MacBook-Pro-3 ~ %
```

```
pi@raspberrypi:~ $ mosquitto_sub -h raspberrypi.local -t "esp32/machine_availabi
lity"
Hello World
```

11 - 13 - 2022

Today the group started writing the code for the esp32 broker. Unfortunately, I was sick for a lot of this process , however, to summarize what was done:

1. Code for the ESP32 was written that implements a mqtt message send whenever a button is pushed
2. Code for implementing a PIR motion sensor fail safe whenever a user forgets to press the button before/after using an ARC machine.
3. A general outline for the website

11 - 15 - 2022

As I was still sick the group went to Mock demo without me.

11 - 29 - 2022

Because of my sickness, my group was able to receive an extension and perform our final demo on Friday (12-3-2022). But before we start finishing the rest of the soldering and website code, we met to work on what we will actually demo. We worked on creating a demo sheet that we will give the TAs and professor during the demo. We included RVs, High level requirements, and what are code accomplishes.
High Level Requirements:
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.

11 - 30 - 2022

We met today to finish soldering and add features to our website. Today we worked adding a website framework. We already have a website that shows machine availability but we are making a homepage and email subscriptions. We used the flask framework to create a website. We also used the SMTP (Server Mail Transfer Protocol) for our email subscriptions. When using the SMTP, we are able to utilize a remote server as a mail server. The mail account that we send from will be my own personal account.

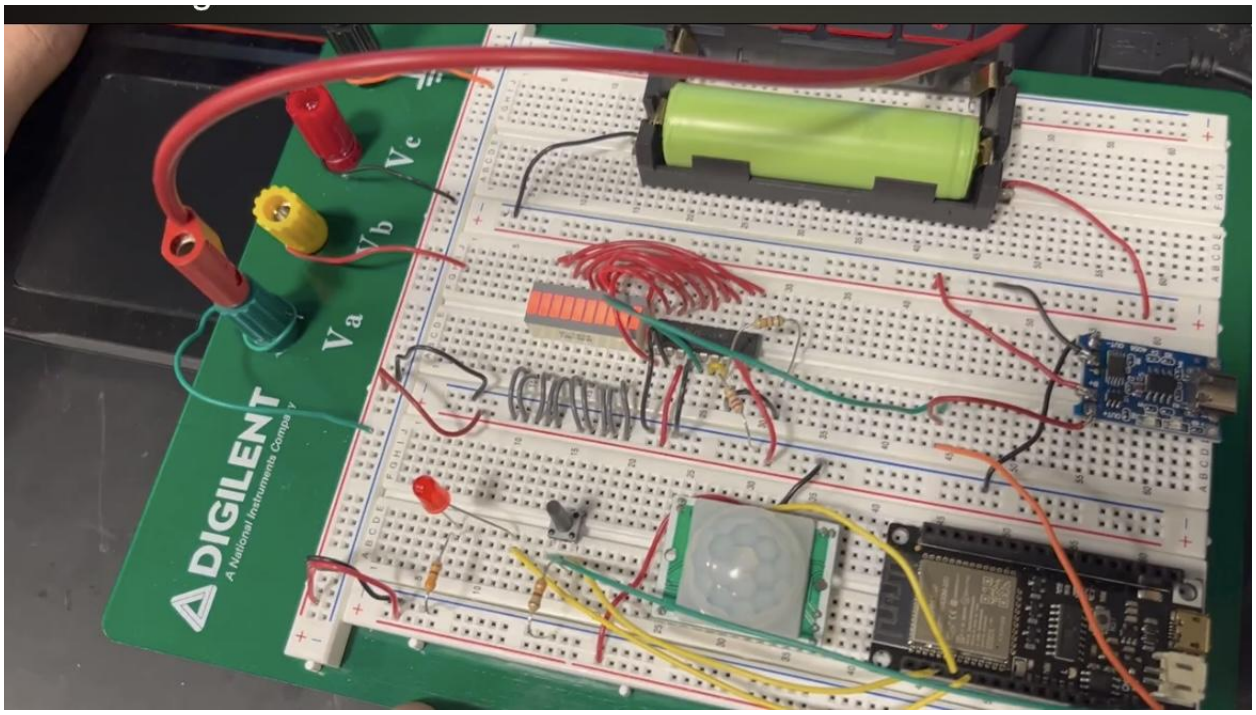
INSERT Pictures of Website

12 - 1 - 2022

ERROR
We unfortunately found another error with our PCB. For some reason, there is not enough current to turn the Firebeetle on. We suspect the following reasons for this: since we are using a dc-dc voltage step up converter, when voltage is increased, the current will decrease. Thus, since our current also contains a lot of internal resistance, this will also decrease the total amount of current in the circuit.

DEBUG:

As shown in the image below, an external power source is plugged into the breadboard.



With this setup, the firebeetle was able to turn on and all of our subsystems were able to work properly. With this, it can be inferred that not enough current was being fed into the system.

12 - 2 - 2022

Since our demo is tomorrow, we also performed a full scale test of everything that we created. Below is the rundown.

Testing Everything

When a button is pressed the following message should be sent to the Raspberry Pi, “1 0” communication that Machine Id 1 is in use.

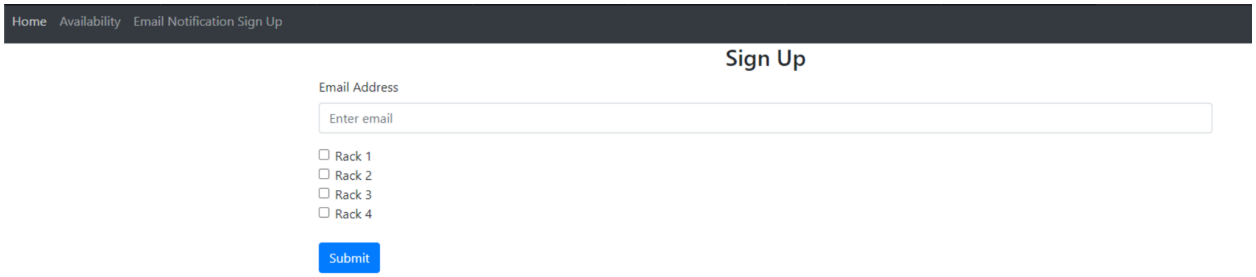
Proof that ESP32 message is received at Raspberry Pi can be seen below.

At Pi:

```
1 0
```

Now after some time, our fail safe should be triggered because movement has not been detected notice, in the image below, there is a “1 1” signalling that machine 1 is now available.

Below is an image of our email subscription service.

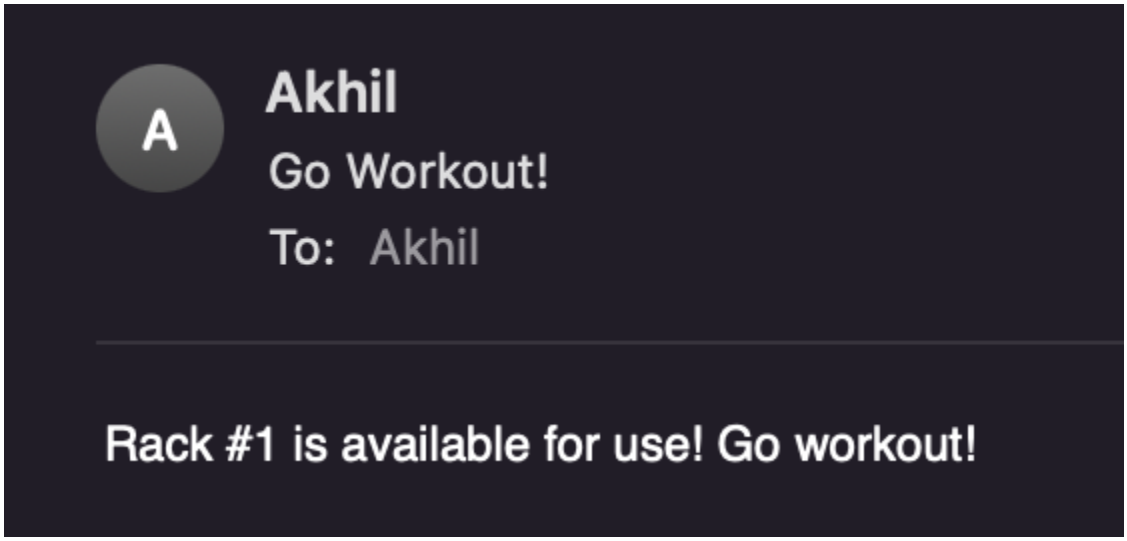


Below is an image depicting how this will appear on our website.

At Website:

Station ID	Availability
1	Available
2	Available
3	Taken
4	Available

Email notifications



The above image should appear if you inserted an email address.

12 - 4 - 2022

This is the last entry since we just conducted our demo. The demo went well and we were able to effectively communicate everything about our project. The final part of this course consists of a final presentation in which I will refer to this document for content.