CART-E: INDOOR TRACKING DEVICE

A Design Project Report

Presented to the School of Electrical and Computer Engineering of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering, Electrical and Computer Engineering

Submitted By

Alga Peng (cp444), Xiangyi Zhao (xz598)

MEng Field Advisor: Professor Joseph Skovira

Degree Date: May, 2023

Abstract

Master of Engineering Program

School of Electrical and Computer Engineering

Cornell University

Design Project Report

Project Title: Cart-E: Indoor Tracking Device

Author: Alga Peng (cp444), Xiangyi Zhao (xz598)

Abstract:

We designed an indoor tracking device Cart-E which is located based on WIFI router addresses. The key features of Cart-E are small, inexpensive, easy to use, and energy-efficient. The area of a Cart-E device is small so that it can be easily attached beneath the cart. Through a website, users can easily search for the available carts in Phillips and Upson and see the history of their movement. It uses open-source information such as WIFI routers' public MAC addresses to ensure accuracy and avoid privacy issues. It is also inexpensive since it only costs around $6 for a PCB design if we order 3000 pieces altogether. We also utilized the deep sleep mode of the electronics to ensure that the energy cost is as small as possible. This technology produces an affordable device with little maintenance requirement, providing detailed information about the object's location in 3D space. This allows the user to be able to retrieve the object fast and easily.

Keywords: Embedded Systems, ESP32, Printed Circuit Board, WIFI Locating

Executive Summary

Our invention originated from the problem to find hidden carts inside Phillips Hall - many people don't feel obligated to return the carts after usage, so it is often difficult to find an available cart at need. The existing locating devices' functionalities don't quite satisfy our needs - they either rely on GPS, which is inaccurate indoors, or rely on Bluetooth, which needs other Bluetooth devices to provide a relative distance and the locating resolution is to a building. In order to solve this missing cart problem for the ECE department, we decided to design and implement a more concise and inexpensive tracking device for in-door object locating - Cart-E. Inspired by the idea of the Campus Congestion & Guide project [2] by Minjung Kwon and Esther In, we plan to utilize the WiFi routers' MAC addresses to track the current location of the cart.

Our device contains an ESP32 (later only the ESP32-WROOM-32E-N8 module of the ESP32) [6] that has a convenient WiFi library to gather the information including name, MAC address, signal strength, and accessibility (public or not) of the surrounding detectable routers. We constructed a table of the recognizable routers in Phillips 1-3 floor and Upson 1-3 floor, which contains each router's MAC address and physical location. When the device performs a scan, it compares the signal strengths of the recognizable routers. A higher signal strength indicates that the device is closer to this router. Therefore, we claim that the device is at the location of the router with the strongest signal. This locating method gives us a granularity of smaller than 5 meters, depending on the distance between two routers in the buildings.

To save energy, we want to only start scanning when the cart is moved. Therefore we utilize the deep sleep mode of ESP32 [9] and a vibration switch to only wake the device up when a vibration is sensed. After waking up, the device will first check if the current location is the same as the last one in its history. If true, it means that this vibration is a false alarm - probably someone just bumped the cart. If not, this cart is actually moving, thus the device will perform the scanning for a minute, record the location of each scan, and send this history to our website. Then the device will go into deep sleep mode and wait for the next vibration.

On our website, users can see the historic location of the carts and search on the maps for an available cart. The history table keeps track of the historical locations of 5 carts. We have the floorplans for Phillips 1-3 floor and Upson 1-3 floor, and the available carts are marked as dots and numbered. Different devices are distinguished by the MAC address of the ESP32. To make our device more portable, reliable, and cost-effective, we constructed a PCB of our design.

Individual Contribution

In this project, we worked closely together and did a lot of research and debugging together. In this section, we detail the high-level contributions of each member of the group.

Alga mainly worked on the AWS server and PCB design. The AWS server part involved setting up the AWS server, adding to the Arduino code to send the final message to the server, writing Python scripts on the server to process the information received, and displaying the information in a visually appealing way. The PCB design part involved heavy searching for appropriate tutorials since we were PCB novices, researching and comparing different components to add to the board. PCB design includes the schematic, physical layout, searching for hardware parts since they are often out of stock, and some soldering when testing the functionality of vibration switch Plan B which was a failed attempt.

Xiangyi mainly worked on Arduino, which involved WIFI mapping and selecting the router address with the maximum signal to send to the server. The WIFI mapping involved writing an Arduino script that displays the name, MAC address, and signal strength of the scannable routers nearby. After all the routers' MAC addresses were documented in Phillips 1-3 and Upson 1-3 floor, she wrote another Arduino script to compare their signal strengths and pack the MAC address with the strongest signal as a message to the server.

WIFI Address Mapping

The essence of our locating functionality lies in Cart-E recognizing nearby WIFI router signals with known locations and determining its location based on the strength of these signals. Therefore, we want to map the physical locations of the routers with their identifications scannable from the ESP32.

We noticed that the WIFI library of ESP32 has a function to scan the MAC addresses of each router [6]. We were expecting the list of MAC addresses that Cornell Information Technology maintains [10] would match the ones we scan, but the list they provided contained a different set of MAC addresses that are for admin-only access to the routers and not scannable from our ESP32. Therefore, we had to do the mapping manually.

We wrote an Arduino script that displays the MAC address and signal strength of the scannable routers nearby. For the signal strength, we transferred it from the Decibal scale to the percentage scale for better intuition. Then we stood beneath every router with an ESP32, recording the MAC address of the strongest signal scanned by the ESP32, typically with a strength of 100%, assuming it's from this router, and labeling the location of this router on the floorplan on our website. Figure 1 below shows this process. We performed this manual mapping for the routers in Phillips 1-3 floor and Upson Hall 1-3 floor, and marked these routers to be recognized. Every time a router's MAC address was matched, we added to the Arduino script to also print the location of this router, such as Phillips 239, next to its original information, so we knew that it was a labeled router and wouldn't again spend time matching its MAC address. Once we have this list of mapping, we can make it public for successors who want to expand the Cart-E product or work on their own projects related to WIFI routers.

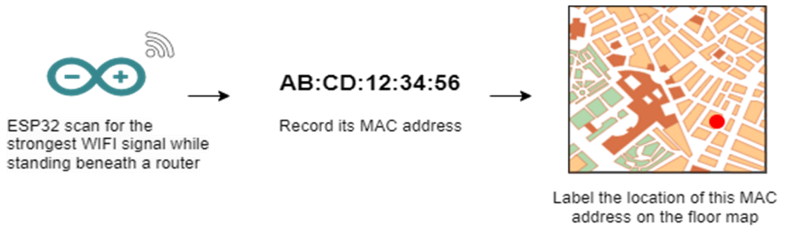

ESP32 scan for the strongest WIFI signal while standing beneath a router
AB:CD:12:34:56
Record its MAC address
Label the location of this MAC address on the floor map

Figure 1. Manual Mapping Process of the Routers' MAC Addresses

When the device performs a scan, it compares the signal strengths of the recognizable routers. A higher signal strength indicates that the device is closer to this router. For now, we claim that the device is right at the location of the router with the strongest signal. The resolution of this locating algorithm depends on the distance between two routers in the buildings. The buildings that we recorded, it's approximately 3-4 meters. If this range proved to be too wide, later improvements can be made to improve the resolution by using not only the absolute strength but also the relative strength of the signals.

To save energy, we want to only start scanning when the cart is moved. Therefore we utilize the deep sleep mode of ESP32 [9] and a vibration switch to only wake the device up when a vibration is sensed. After waking up, the device will first check if the current location is the same as the last one in its history. If true, it means that this vibration is a false alarm - probably someone just bumped the cart. If not, this cart is actually moving, thus the device will perform the scanning for a minute, record the location of each scan, and send this history to our website. Then the device will go into deep sleep mode and wait for the next vibration. Figure 2 below shows this process.

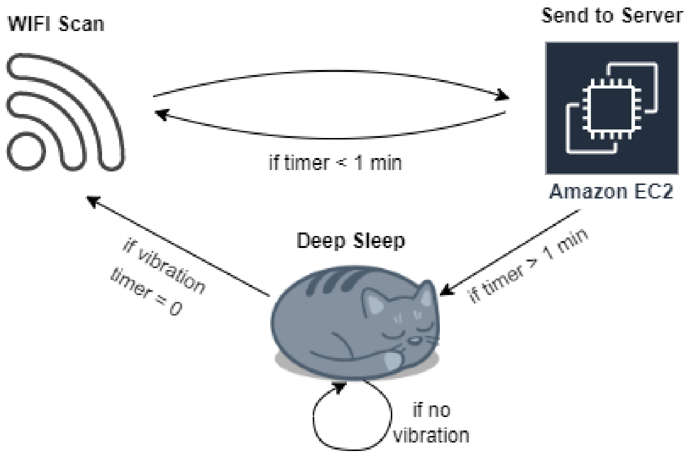


Figure 2. FSM on Entering Deep Sleep Mode and Waking Up on Vibration

AWS EC2 Server

We were fortunate to obtain temporary access to an AWS EC2 server through our friend Owen Deng, a Cornell ECE alumni from the class of 2023.

The server hosts our website and collects information about the location of the Cart. Collected WIFI signals from different carts will be stored on the server and analyzed by Python script. As shown in Figure 3 below, first, we store the WIFI signals from different carts (using Cart MAC address as a unique device ID) in different files. Then we use a Python script to parse the files and retrieve the WIFI address to match the corresponding room. The location information results in another file which will be obtained from the front-end HTML file to display. Our website has two features: it displays a table of all carts' locations and has a 2D floor plan for the user to look at in case they are not familiar with the building.

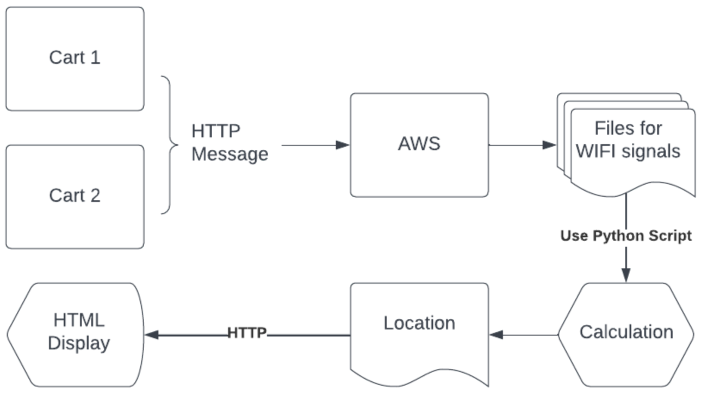


Figure 3. Display Flow

Since we have several devices running simultaneously, we designed the ESP to send a packet of: the maximum WIFI signal, its signal strength, and the device's own MAC address after three valid scans (if no valuable signals, ESP will discard the scan and try again) together to distinguish different devices. By doing so, the WIFI signal and its strength can be categorized and stored in corresponding txt files. Each device has its own WIFI_signal.txt and WIFI_history.txt. For future work, we could potentially store all information in a database for more precise calculations and detailed historical data. For now, each txt file contains only 10 previous signals' information and will only be updated when a new signal of this device comes in. For future work, we could store all previous events and have more throughout the history table.

The WIFI_history.txt is generated by another Python script which reads and parses the data from WIFI_signal.txt and determines which room the WIFI signal address matches. The Python script stores the corresponding location information in the WIFI_history.txt and also marks the time the signal arrived. With the above information, the Python script can update the map.html files and history table.

The way we designed the maps originally made it tricky to display CART information on the floorplans after we have more than one Cart-E device running at the same time. We have a class for Cart, which contains its current location, the current floor plan it is at, and other useful information. Then we create a list to store all devices' Cart objects and update their location based on new WIFI signals. Every time the selection page is opened, a Python script on our server will update every floor plan's corresponding map.html based on all the Carts' locations, making sure that all maps are updated with the newest information.

For the floor plans, a page is designed for users to choose which building and floor they want to look at. As shown in Figure 4 below, each box that is labeled with a number leads to the floor plan of the specific floor in that building. Users can hover over the boxes and choose which one to look at.

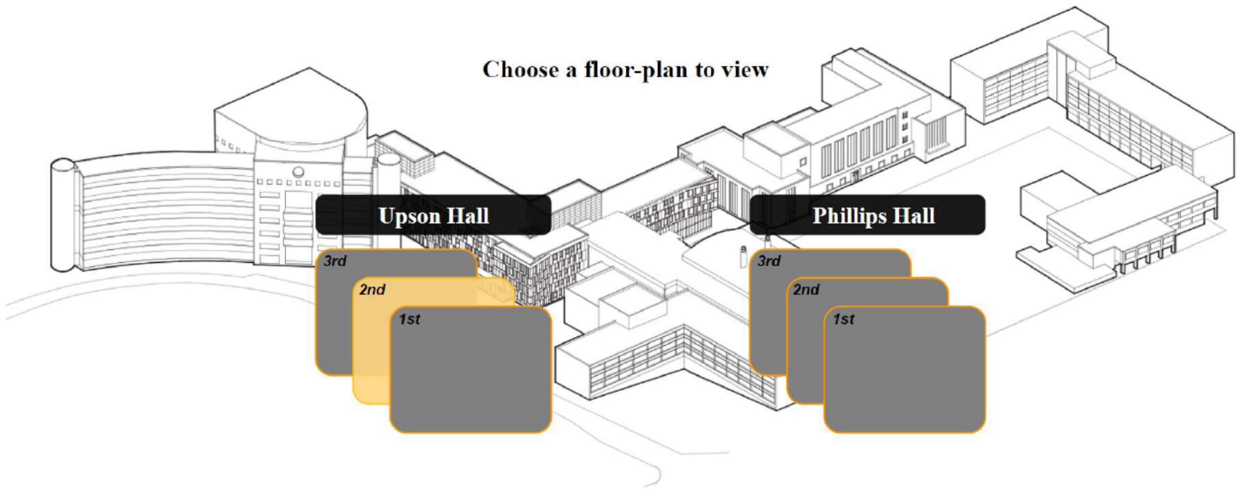


Figure 4. Page to choose from Upson and Phillips Hall

In the figure, the user hovered over Upson's second floor and chose to look at its floor plan. Figure 5 shows the next page, which is a simple floorplan for upson second floor with WIFI routers labeled at different locations. It also shows that two CARTs are currently near two different routers which translates to room 20040-2 and 20041-1 as seen on the map.

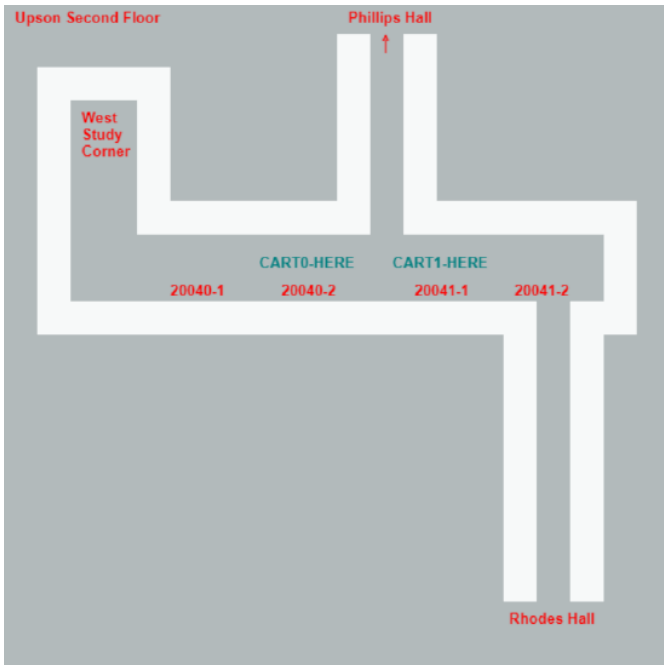
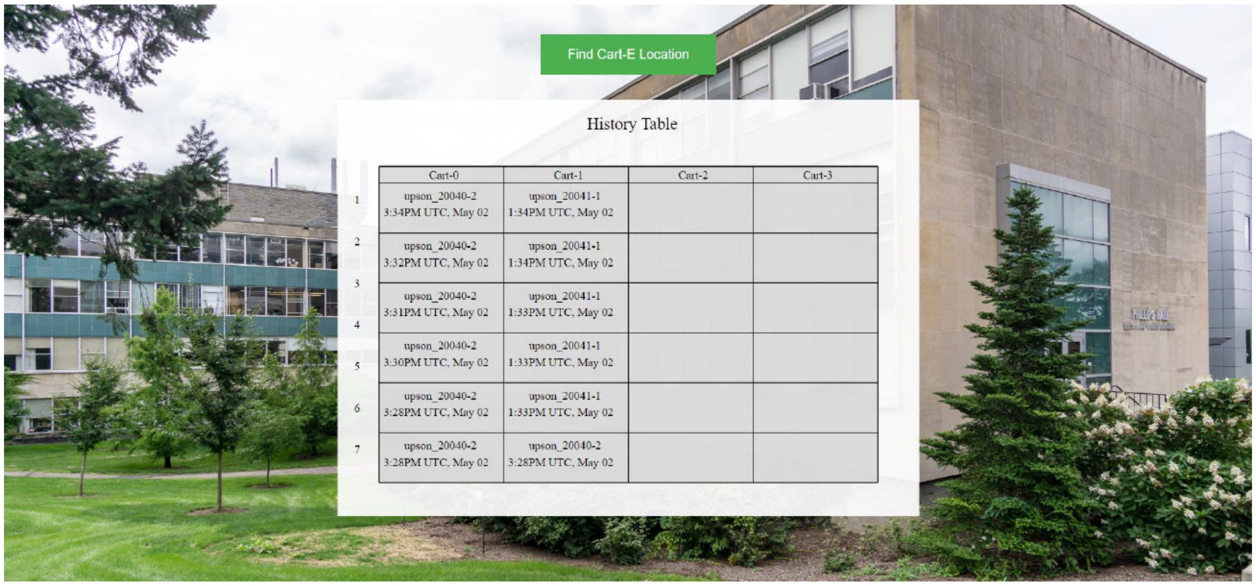


Figure 5. Upson Second Floor Floorplan with the Locations of Two Devices

The history table in Figure 6 contains information about different cart locations and times in the past. The floorplan example on the bottom left corner shows that a cart is currently around Upson 20040-2 and a cart is in Upson 20041-1, which matches the information shown in the previous floor plan.



History Table

	Cart-0	Cart-1	Cart-2	Cart-3
1	upson_20040-2 3:34PM UTC, May 02	upson_20041-1 1:34PM UTC, May 02		
2	upson_20040-2 3:32PM UTC, May 02	upson_20041-1 1:34PM UTC, May 02		
3	upson_20040-2 3:31PM UTC, May 02	upson_20041-1 1:33PM UTC, May 02		
4	upson_20040-2 3:30PM UTC, May 02	upson_20041-1 1:33PM UTC, May 02		
5	upson_20040-2 3:28PM UTC, May 02	upson_20041-1 1:33PM UTC, May 02		
6	upson_20040-2 3:28PM UTC, May 02	upson_20040-2 3:28PM UTC, May 02		
7				

Figure 6. History Table

Prototype-0

We will also take an incremental approach when choosing the hardware we use to build this tracking device. Our prototype-0 uses Adafruit HUZZAH ESP8266 shown in Figure 7 with a cost of $9.95.[7] This is a microcontroller programmable by the Arduino system and can be connected to other components through wires and a breadboard. The entire design costs around $12 with vibrations switches and batteries. This prototype is convenient for testing purposes.

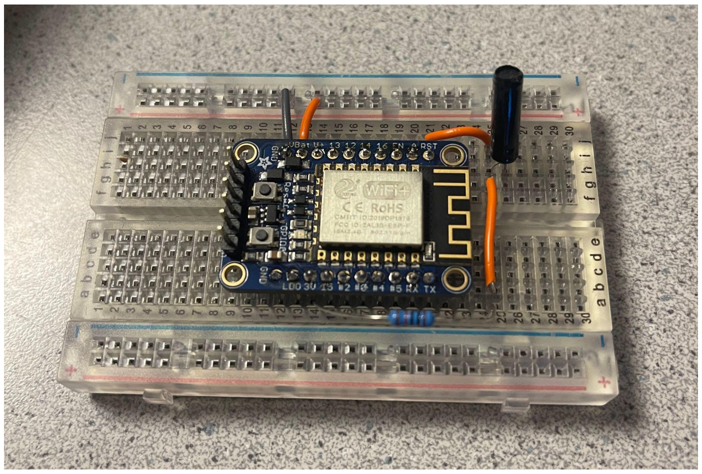


Figure 7. HUZZAH ESP8266 with vibration switch

Our transition from ESP32 HUZZAH to ESP32 Feather was driven by the presence of an advantageous feature: the inclusion of a deep-sleep mode that enables awakening from GPIO pins beyond the limitations of solely relying on reset pins.[8][12] The incessant resetting triggered by vibrations on ESP32 HUZZAH hindered the retention of historical data, impeding the data acquisition process. Conversely, ESP32 Feather ignored the GPIO input caused by vibration signals upon restart, thereby facilitating uninterrupted scanning operations.

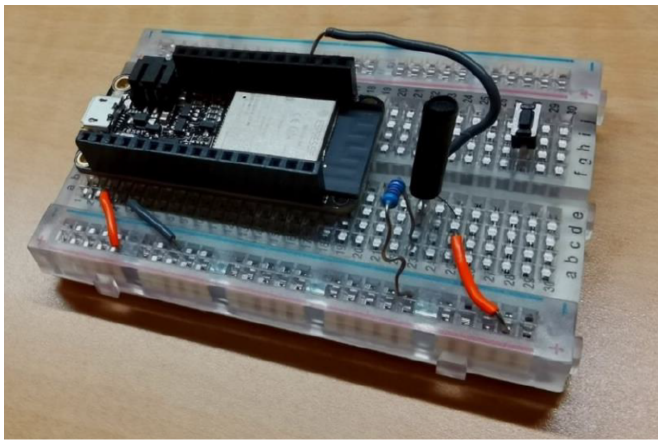


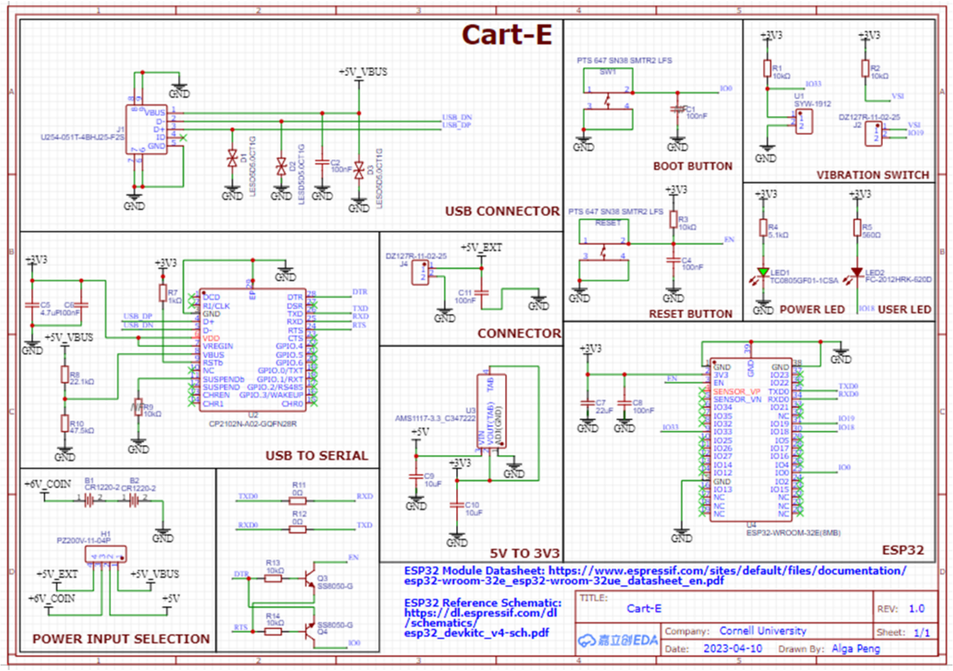
Figure 8. ESP32 Feather with vibration switch

Following the adoption of ESP32 Feather, comprehensive testing was conducted, culminating in the decision to transition towards the integration of a printed circuit board (PCB), which would be optimal for our requirements.

PCB Design

We decided to build our own PCB for several reasons. The rationale behind pursuing the development of a custom PCB design lies in our objective to enhance the professional appearance of our product, aligning it with commercial-grade standards. By employing battery supplies and integrating a vibration switch, our design aims to emulate a cohesive and self-contained unit to replace the utilization of a breadboard and exposed wires, which is not conducive to long-term durability. The justification for adopting a PCB design also stems from cost considerations. With an estimated assembly cost of only $6 per device when produced in quantities, this approach offers a financially advantageous solution.

As shown in the schematic below, Cart-E's PCB contains the following components: an ESP32-WROOM-32E-N8 module, a USB-to-Serial converter, a USE connector, a power input selector, one onboard vibration switch, an attempt to add an external vibration switch connection, two coin batteries, and a 5V-to-3.3V convertor. The ESP32 module is the centerpiece of the design, which requires the USB-to-Serial converter and a USB connector so that we can program it. The power input converter are acting as a buck converter to lower the input voltage to 3.3V for the ESP32 module. The power input selector chooses three possible input power supplies: 5V from an external source, 5V from USB, and 6V from coin batteries. The external 5V is a backup plan if the coin battery design on the board fails to work. It was our first time building a PCB and we were concerned about any kind of mistakes that could occur, so we tried to have backup plans for essential designs. [14][15]



Cart-E
USB CONNECTOR
BOOT BUTTON
VIBRATION SWITCH
USB TO SERIAL
CONNECTOR
RESET BUTTON
POWER LED
USER LED
5V TO 3V3
ESP32
POWER INPUT SELECTION
ESP32 Module Datasheet: https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32e_esp32-wroom-32ue_datasheet_en.pdf
ESP32 Reference Schematic: https://dl.espressif.com/dl/schematics/esp32_devkitc_v4-sch.pdf
TITLE: Cart-E
REV: 1.0
Company: Cornell University
Sheet: 1/1
Date: 2023-04-10
Drawn By: Alga Peng

Figure 9. PCB Schematic [3][4][5]

Our coin battery design failed to function properly. We did not take current requirements into consideration. We discovered that the coin batteries utilized as a power source for our ESP module did not deliver adequate current to sustain its operations effectively. The two 12mm coin batteries in series only support 35mA current while the ESP32 requires at least 250mA.[1][13]

We also have a backup plan for the vibration switches: one vibration switch is directly attached to the PCB but the other is a plugin. To do so, we mounted two holes (one connected to GPIO19, another connected to GND) so that we can plug our own vibration switch into the PCB. To avoid having GPIO19 be active high, we added a two-pin connector. As shown lower right corner of the PCB layout and 3D preview below, the pin header connects VS2, which is connected to a 10K pull-up resistor to 3.3V, and GPIO19. We prepared a pull-up circuit for the second vibration switch which can only be activated if the user chooses to connect these two pins manually.

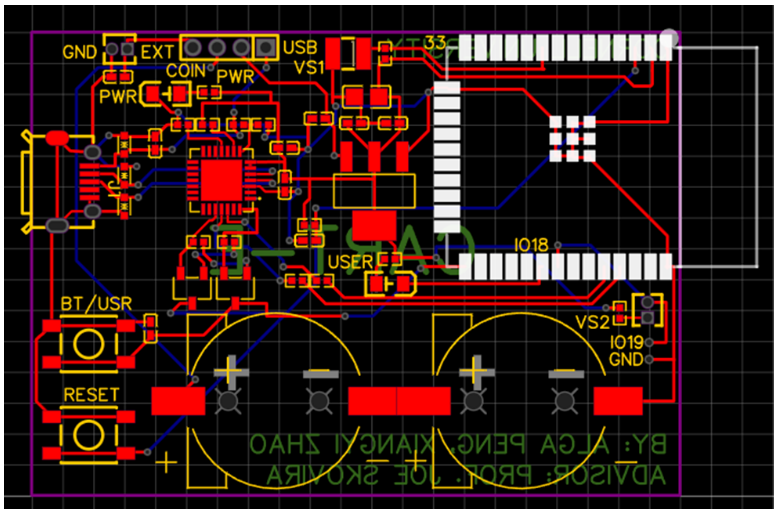


Figure 10. PCB layout after routing

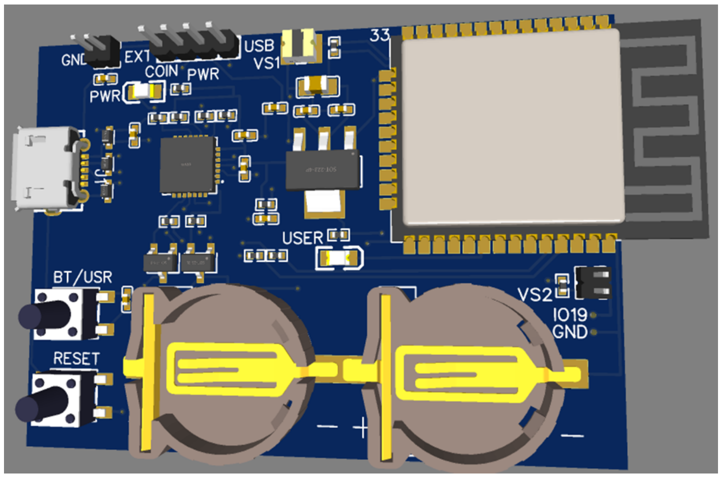


Figure 11. 3D preview of the PCB

It turned out that both Plan A and Plan B for the vibration switch do not work. The mounted vibration switch is extremely sensitive to the point that it will constantly be active and send signals. Our efforts to achieve a noiseless operation proved to be transient, as the device remained in a muted state for only a brief duration upon placing it on a soft, even surface. So in prototype-2, we might consider adding a filter to reduce noise so that it is not as sensitive. The second plan failed due to a miscalculation of the holes' diameter. They were too small to pass through a wire or to solder a wire to it. We then remembered that we placed a header for GPIO 19 and our coin battery had exposed the GND part that we can solder wire to it. So we soldered the GPIO 19 pin and GND pin to both sides of the fast vibration switch and crossed our fingers hoping it works. Weirdly, if we simply print the signal input from the vibration switch, it corresponds to the movement. It reads in 0 when we shake the PCB and reads in 1 if we set it on the surface. It didn't seem to work once we set it to be the wake-up pin. After hours of debugging, we realized that it was because GPIO 19 is not an RTC pin, which means that it does not serve as a wake-up pin. Since in deep sleep mode, only RTC is active and can be used by the ULP co-processor, which is used to wake up the ESP32.[9] So our plan B failed as well and we were forced to temporarily give up our plan to demo using vibration switches as the wake-up source.

To finish up with our demo, we added a timer sleep function in our design. It will constantly scan the WIFI signals for a few minutes, check the signals every 8 scans to see if all 8 times have the same signal (enter deep sleep mode if that's the case because it indicates that the cart is parked in the location for a while), send the signals to our server and enter deep sleep mode. It wakes up after 1 minute and checks if there is a vibration signal (most of the time it is the case). If so, it checks if the first scanned signal is the same as the previously scanned signal which we stored in RTC memory so that it is not flushed away in deep sleep mode.[9] If it is the same meaning that it is still in the same location, the cart can enter deep sleep mode directly.

PCB Testing

Our experimental setup involved employing a portable battery pack to provide power to our device through a USB connection, which can be seen in both figures below. To evaluate the functionality of our multi-device system on AWS, we conducted tests utilizing two devices in unison. We had one person conceal a Cart-E unit while another person held the other unit. The person in search of the hidden device carried a computer equipped with the Cart-E application, continuously monitoring the displayed location on the device and navigating towards the concealed unit as indicated on the map. The successful execution of this procedure was repeated multiple times to ascertain the precision and reliability of our device. In addition, we plan to conduct beta testing at the end of the semester, aimed at further scrutinizing and demonstrating the performance of Cart-E.

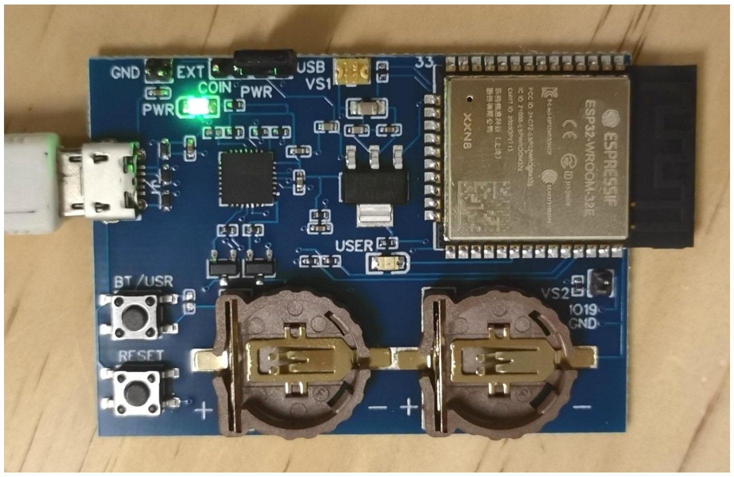


Figure 12. Cart-E Front

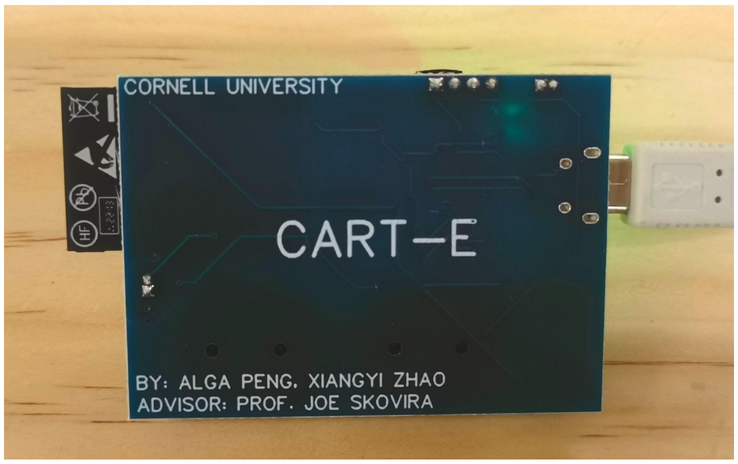


Figure 13. Cart-E Back

Conclusion

Our project has achieved a high level of completion over the academic year. We achieved indoor locating based on WiFi, long battery life using the deep sleep mode and vibration switch wakeup, and a mostly successful PCB design prototype No.1. During the poster session and Beta test, we successfully located the cart using our website and PCB. The remaining issues are: 1. The PCB has the wrong size for holding the coin battery 2. The vibration switch mounted on the PCB is too sensitive 3. The WiFi router MAC address to physical location mapping is limited.

Reference

[1]Admin. “Power Supply for ESP32 with Boost Converter & Battery Charger.” *How To Electronics*, 2 Oct. 2022, how2electronics.com/power-supply-for-esp32-with-boost-converter-battery-charger/#:~:text=The%20power%20required%20by%20ESP32,drawing%20more%20than%20200mA%20current.

[2] *Campus Congestion & Guide* project by Minjung Kwon and Esther In, Campus Conjestion Project of ECE 5725 Spring 2022: https://courses.ece.cornell.edu/ece5990/ECE5725_Spring2022_Projects/Wednesday May 18/Campus Congestion/W_ei53_mk2592/index.html

[3] How to Make Custom ESP32 Board in 3 Hours | Full Tutorial. wwwyoutubecom . [accessed 2023 Apr 20]. https://www.youtube.com/watch?v=S_p0YV JlfU

[4] ESP32WROOM32E & ESP32WROOM32UE Datasheet. https://www.espressif.com/sites/default/files/documentation/esp32 wroom 32e_esp32 wroom 32ue_datasheet_en.pdf

[5] https://dl.espressif.com/dl/schematics/esp32_devkitc_v4 sch.pdf

[6] Arduino IDE: https://learn.adafruit.com/adafruit-huzzah-esp8266-breakout/using-arduino-ide

[7] Huzzah Platform IO: https://docs.platformio.org/en/stable/boards/espressif8266/huzzah.html

[8] ESP8266 Sleep Mode: https://www.electronicshub.org/esp8266-deep-sleep-mode/#Operating_Modes_in_ESP8266_SoC

[9] “In-Depth: Esp32 Deep Sleep & Wakeup Sources: Timer, Touch & External.” *Last Minute Engineers*, 4 July 2022, lastminuteengineers.com/esp32-deep-sleep-wakeup-sources/.

[10] WiFi Router CIT Information: http://mrtg.cit.cornell.edu/

[11] ESP8266 WIFI library & documentation: https://arduino-esp8266.readthedocs.io/en/latest/esp8266wifi/scan-class.html

https://arduino-esp8266.readthedocs.io/en/latest/esp8266wifi/readme.html

[12] “Sleep Modes.” *ESP*, docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/sleep_modes.html. Accessed 19 May 2023.

[13] *CR1220 Lithium Manganese Dioxide - Panasonic*, api.pim.na.industrial.panasonic.com/file_stream/main/fileversion/3637. Accessed 19 May 2023.

[14] “PCB Prototype & PCB Fabrication Manufacturer.” *JLCPCB*, jlcpcb.com/. Accessed 19 May 2023.

[15] “An Easier and PowerfulOnline PCB Design Tool.” *EasyEDA*, easyeda.com/. Accessed 19 May 2023.