LoRaWAN Asset Tracker and Emergency Button

William Lorenzen
UF Electrical Engineering
Gainesville, FL
williamlorenzen@ufl.edu

Justin Nagovskiy
UF Electrical Engineering
Gainesville, FL
jnagovskiy@ufl.edu

Maria Barrera
UF Electrical Engineering
Gainesville, FL
maria.barrera@ufl.edu

Abstract - This project consists of a LoRaWAN-based IoT system for agricultural applications. The system serves a dual purpose: an asset tracker for farm machinery and a personal emergency button for farm workers. It consists of a LoRaWAN radio, a GPS module, a Bluetooth LE module, input buttons, a rechargeable battery, and an OLED display. The system can periodically acquire its location and broadcast it along with emergency data across the LoRaWAN network to allow for tracking. The location data is received by a LoRaWAN gateway and server and is viewable on a geolocation map. A mobile device can be used by a user to communicate directly to the system through Bluetooth LE.

Keywords: LoRaWAN, Emergency Button, Asset Tracker, GPS, IoT, Agriculture.

I. BACKGROUND

A. Previous Semester's Work

This IoT4Ag project builds upon the work of the Fall 2023 LoRaWAN team, consisting of Nathan Rosenberger, Guillermo Cadima, and Manuel Agurcia. This team set up an SX1303 Raspberry Pi LoRaWAN gateway and configured it on The Things Network (TTN), which is a LoRaWAN network server. They then assembled an end node consisting of a Heltec Wireless Stick, Argon Microcontroller, and soil sensor, which they also configured on TTN. Utilizing the LMIC node software, they established protocols for LoRaWAN uplink messages from their end nodes to the LoRaWAN gateway, which allowed them to view the end node's soil sensor values on TTN.

In contrast with the Fall 2023 team's LoRaWAN environmental sensor, we were tasked with creating a LoRaWAN Asset Tracker and Emergency Button: a device that transmits its GPS location to a server so that it can be plotted on a geolocation map and that contains an emergency button which can be used by agricultural workers to call for help. By making use of the Fall 2023 team's SX1303 gateway and using their Heltec-LoRaWAN Gateway-TTN pipeline as the basis for our implementation, we were able to build upon this previous team's work. Most notably, we integrated the ThingSpeak IoT analytics platform into our project, which allowed us to create a custom GUI for users to view end node data.

B. LoRa and LoRaWAN

LoRaWAN (Long Range Wide Area Network) is a networking protocol designed for low-power and wide-area applications where battery-operated devices are wirelessly connected to the Internet to create IoT (Internet of Things) networks [1]. LoRa (Long Range) defines the physical layer part of the communication protocol that LoRaWAN is built upon. It specifies the device-to-infrastructure parameters that make interoperability between manufacturers seamless [1]. Our project follows the architecture of the system shown in Figure 1 where LoRaWAN is used to establish communication between the end nodes and the gateway and between the gateway and the network server.

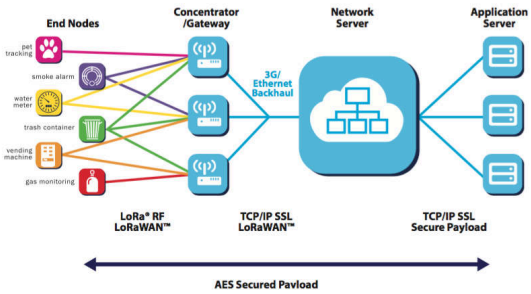

Figure 1. LoRaWAN in a sensor network [2].

II. INTRODUCTION

The agricultural sector has a critical need for the implementation of safety measures as it is currently one of the most dangerous industries in the United States with fatality rates of 23.5 per 100,000 workers [3]. Additionally, the nature of this work requires the use of

extensive farmland and equipment, which makes it difficult to watch and keep track of all equipment. After talking to those with firsthand experience, we found that they had a difficult time keeping track of all their equipment and a lot goes missing when it is time to do inventory. Another prevalent issue present in agricultural settings that makes safety measures difficult to implement is the lack of reliable Wi-Fi and cellular connection. This presents a threat to the personal safety of workers because a poor or non-existent wireless connection could leave them stranded.

To confront these challenges, our group developed a multifunctional device that can work as a personal emergency button as well as an asset tracker. To ensure a reliable connection in areas with limited network infrastructure, we incorporated LoRaWAN communication. As an asset tracker, the device will periodically broadcast its location, which will be visualized on a map. As an emergency button, the device will also broadcast its current location and send a text alert to relevant parties when an emergency signal has been sent.

The system's architecture is designed for seamless operation. It begins with the device transmitting GPS and various other data points via LoRaWAN to the TTN server through a gateway. From there, the data is processed and decoded before being forwarded to the ThingSpeak server. At this stage, the server monitors predefined values to trigger and dispatch alerts. Finally, the Python GUI comes into play. Extracting the data from ThingSpeak, it crafts an intuitive and user-friendly interface, presenting the GPS data in a visually appealing map format, alongside comprehensive device details for easy access and analysis.

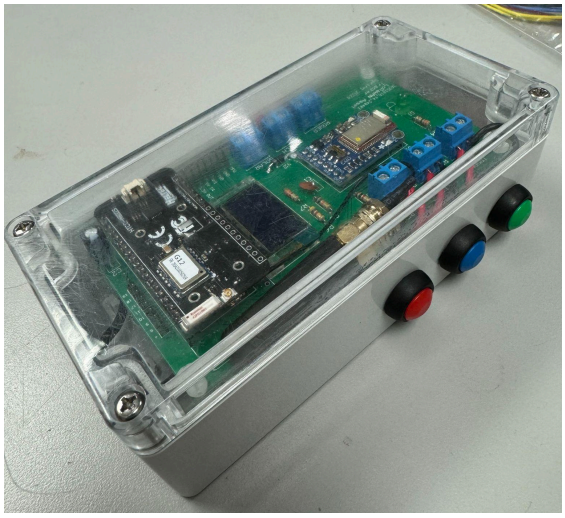

Figure 2. Finished device prototype.

III. METHODS

A. Quincy Site Visit

On February 29, the LoRaWAN team traveled with Professor Eisenstadt and Professor Stapleton to the North Florida Research and Education Center in Quincy, Florida. At the test site, we were given a tour of the facility by Dr. Small and were given a new perspective on the desired features for our end node by both Dr. Small and the site's manager.

Originally, we intended for the only input to the end node to be the emergency button, but Dr. Small expressed interest in having multiple modes through which the end user could interact with the device, including via buttons and Bluetooth. Additionally, Dr. Small desired there to be a website where the locations of the end nodes could be viewed, as well as the past locations of the end nodes, as this would be useful to see if a tractor or other piece of equipment had covered an entire field. Other features that were deemed to be useful were to add polygons for different fields on the website's geolocation map, which would allow users to better identify the project and task that a piece of equipment was being used for, and to have an emergency signal sent if an end node went outside of the site's perimeter.

This site visit played a pivotal role in the development of our project as we centered our design based on many of the end user's desires, as expressed by Dr. Small. The preference for a website led us to build an easy to use Python GUI that displayed the data from each of the end nodes. The desire for more methods of communication with the end node led us to add a Bluetooth Low Energy module to it, which allowed message codes to be sent in uplinks to the server. This site visit proved crucial in steering our project's development, as we adapted our design to align closely with the specific needs and suggestions brought forth by Dr. Small and the team in Quincy.

B. Software

This project encompasses various software components. The first aspect is the Heltec code that processes and transmits all the data. Subsequently, the front-end of the server on TTN, receives, decodes, and forwards this data to the backend server hosted on ThingSpeak, an online tool designed for data visualization and analysis through MATLAB. Within this framework, we analyze the data coming from The Things Network (TTN), sort it into separate channels per device, and send necessary alerts based on programmed triggers. Finally, a Python GUI provides a user-friendly

interface for visualizing the GPS data retrieved from ThingSpeak on a map, along with easy access to additional information concerning emergencies, geographical boundaries, device battery status, and messages.

1. Heltec Code

To program the Heltec AB02S, we utilized Heltec's Cubecell Series Arduino Library. By analyzing the example projects in the library, we learned how to initialize and operate all of the development board's peripherals, including the OLED display, Air530Z GPS, GPIOs, and ADCs. The AB02S's Arduino integration greatly simplified the software development process, making it intuitive to collect end node data with the Heltec itself, format that data for proper communication, and send an uplink with the end node data, which could then be decoded with the server's payload formatter.

To ensure proper communication between the end node and TTN, each end node had to be configured with a unique device ID (devEUI) and encryption key (appKey) on TTN, which were then used to initialize these respective Over The Air Activation (OTAA) variables in our Heltec code. Additionally, we had to change the default LoRaWAN channelmask such that the device operated on Sub-Band 2 of the 902-928 MHz US LoRaWAN band, as this is the Sub-Band used by TTN. With the LoRaWAN initializations and settings configured in the Arduino IDE, we were then able to edit and build upon the existing LoRaWAN example script such that it suited our application.

In our Heltec code, after the end node's LoRaWAN parameters were initialized and the device was connected to the TTN server, we configured the device to collect our end node data, such as longitude, latitude, emergency status, and battery voltage. The data was then formatted into individual bytes and transmitted to the server. After this data was transmitted, the device went into low power mode for 15 seconds, after which it would exit low power mode, collect data once again, and send it to the server in a loop that persisted while the device was powered on. Button inputs with corresponding interrupt service routines and OLED display messages were added to the code over time, and made the end node more user friendly. Finally, communication protocols for an Adafruit BLE module were integrated into the script, which allowed a user to send messages to the device via the BluefruitConnect App's UART feature.

The Adafruit BLE module was chosen for our application due to its integration with Arduino and because it had a dedicated iOS app that would allow users to easily interface with it. A dedicated button for turning Bluetooth communication on and off was added to our end node, ensuring that the Air530Z GPS and BLE had their functionalities carried out in separate switch blocks to avoid unexpected behavior, as these two peripherals shared UART pins (TX2 and RX2) on the Heltec.

With Bluetooth support on our end node, we were able to add code to our script that checked if a UART message received from the BluefruitConnect App matched a particular code word, and if so, changed a message flag variable to a corresponding value. This message flag was then sent to the server, transmitted to a particular field on ThingSpeak using TTN's payload formatter, and mapped to its respective message in our Python GUI, thus allowing end users to use their phone to send uplink messages to the server.

Additionally, we configured our end device to decode downlink messages from TTN, and if a particular downlink code was received, a corresponding message was displayed on the end node's OLED for ten seconds. These messages could be sent by agricultural managers and viewed by field workers several kilometers away. The combination of Bluetooth uplink messages and TTN downlink messages supported by our end node can facilitate long range communication via LoRaWAN in rural regions that do not have cellular connection, potentially enhancing the efficiency and safety of agricultural operations by ensuring timely and effective communication between management and field personnel.

2. The Things Network

The next phase involved configuring the reception aspect within The Things Network (TTN): the LoRaWAN network that provided the tools necessary to create our IoT application. With the previous LoRaWAN team having set up an SX1303 Raspberry Pi gateway and configured it on TTN, we needed to configure our AB02S end nodes on TTN such that they could communicate with this gateway, and thus the TTN server.

The first step was to register the Heltec boards on separate applications to establish dedicated communication channels between the server and each end node. The setup included registering the device by selecting the specific board, the frequency band of operation, and generating various keys to connect. After this registration was completed and the Heltec successfully connected to the server, we began to see the transmission of data.

This data was made up of 14 bytes that encapsulate the integer, fraction, and sign components of latitude and longitude, alongside the emergency signal, battery voltage, and message code. The entirety of this data was parsed in the payload formatter scripted in JavaScript. For each device, the payload formatter separated, decoded, and stored the information in respective variables, subsequently directed to registered integrations. Integrations serve as distinct endpoints for data transmission from TTN, and in our case, we registered a ThingSpeak webhook that received the data for each device in separate channels.

3. Thingspeak

ThingSpeak, as mentioned before, was the back-end of the server where data underwent analysis. TTN transmitted all information from individual devices into separate channels within ThingSpeak, which offered great flexibility for data analysis through customizable MATLAB code. For example, something we implemented early on was MATLAB's geoplot function to visually map GPS data, plotting latitude and longitude coordinates onto a map.

The primary function of ThingSpeak, however, was to monitor the status of the emergency and boundary flags. We set up a reaction code that would monitor if either of these flags were triggered (set to 1), in which case ThingSpeak would send an email alert detailing which device is having an emergency along with its current position, or that a device is out of bounds. This email would then be forwarded as a text message to designated recipients of these alerts.

Another feature that we set up on ThingSpeak related to the aforementioned boundary flag, is polygons. These are arbitrary shapes representing specific plots of land. Polygons define boundaries and we used them to simulate potential project areas in Quincy. They created a visual boundary that can be monitored on the map and also serve to identify when the device was out of bounds and update the boundary flag accordingly.

Finally, the last purpose of ThingSpeak in our project was to provide access to this data for reanalysis in a much better and more user-friendly Python GUI.

4. Python GUI

Despite ThingSpeak's numerous advantages, the IoT analytics platform had several major limitations that would reduce the scalability and efficacy of our project and the work of future semesters. Chief among these was that each ThingSpeak channel was limited to a maximum of eight fields. This meant that we did not have enough fields to send GPS data and emergency status data for even three devices, much less the tens of devices that could eventually be deployed on the Quincy site. Additionally, the MATLAB Visualizations on ThingSpeak, which we originally intended to use for geolocation mapping, produced very small plots that could not be interacted with by the end user and that were difficult to discern useful information from, as can be seen in Figure 3.

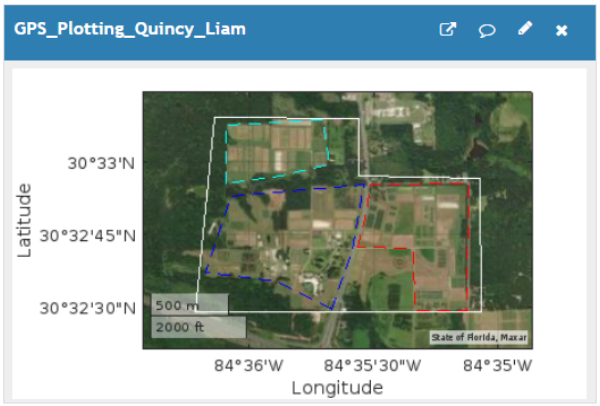

Figure 3. MATLAB Visualization of Quincy test site on ThingSpeak.

To combat the scalability issue and to create a better end user experience for viewing end node data, we decided to alter our TTN-Thingspeak integration such that the data from each end node was sent to a dedicated application on TTN, which subsequently sent data to a dedicated channel on ThingSpeak. This way, each device was able to send eight fields of data to the backend server, which was sufficient for our project and would allow future teams the bandwidth to send additional data from our end nodes, such as agricultural sensor data. The problem with this approach was that it made viewing the data from the different end nodes difficult, as they appeared on different channels. To address all the aforementioned issues, we decided to create a Python GUI which seamlessly integrated all the end node data into one location and allowed for a better end user experience.

To get our end node data to the Python GUI from ThingSpeak, we utilized the ThingSpeak API, which allowed us to collect the field data and timestamps from each of our public channels. Once we had our end node data, we were able to integrate it into a Python GUI by utilizing the CustomTkinter and TkinterMapView libraries, which allowed us to plot the current and previous locations of the end nodes on a geolocation map. This library further allowed us to display all the end node data from each ThingSpeak device channel in a visually appealing manner and in an

easy to use menu system. The Python GUI can be seen in Figure 4.

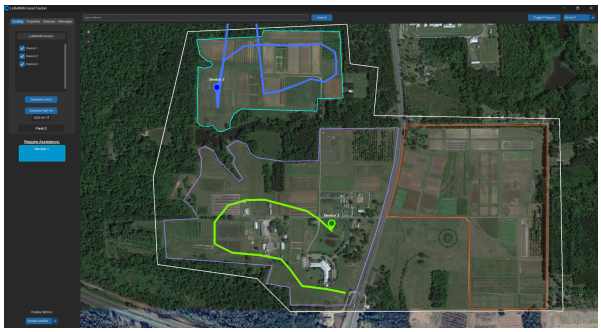

Figure 4. LoRaWAN Asset Tracker GUI.

Within the GUI, the end user is able to see the charge percentage, time since data was last sent to the server, emergency status and boundary status flags, and all the current day's received messages for each end node that has been properly configured with a public ThingSpeak channel. The GUI also allows the user to snap to the latest GPS location of any end node, toggle the GeoMap polygons representing different areas on and off, mark the latest GPS location of specified devices, and plot the path of specified devices on a particular day. Keeping scalability in mind while implementing the GUI, we made sure that with edits to only a few lines of code, the GUI can display the information of significantly more end nodes, as can be seen in Figure 5.

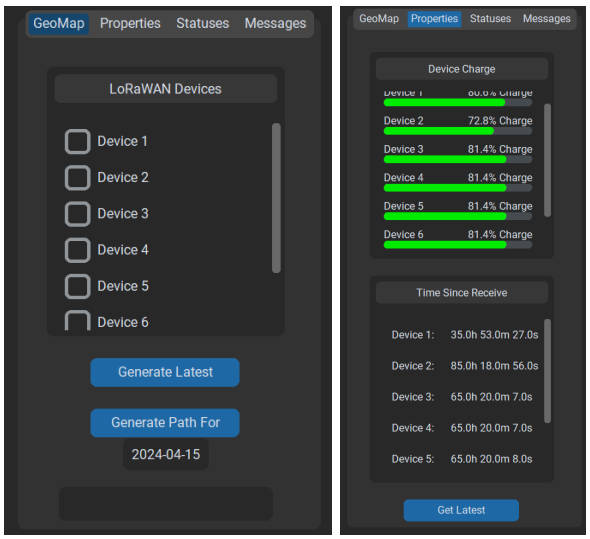

Figure 5. Scalability of LoRaWAN Asset Tracker GUI.

C. Hardware

The hardware can be grouped into 6 subsystems: the Heltec CubeCell, the Adafruit Bluefruit BLE, the power management circuit, the battery monitoring circuit, the input buttons, and status LED. In addition, a PCB joins the subsystems together. The entire system is also contained within a waterproof plastic box.

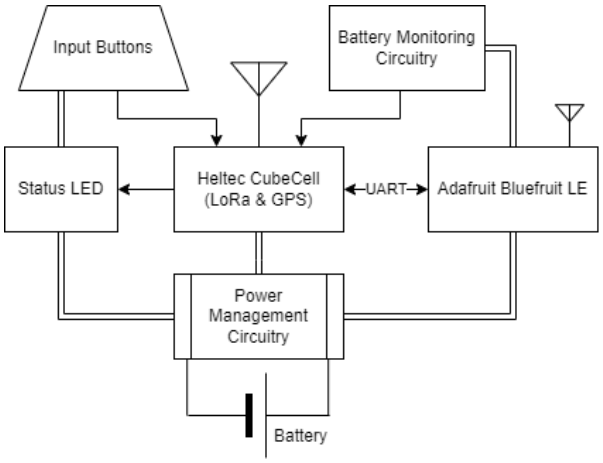

Figure 6. Hardware block diagram.

1. Heltec CubeCell

The Heltec CubeCell AB02S is a development board that is responsible for much of the system's functionality. It consists of the microcontroller, GPS module with antenna, LoRa module, and an Organic Light-Emitting Diode (OLED) display, along with other circuitry, all integrated on one board. It interfaces with each of the other subsystems in the project. The display can be unmounted from the CubeCell, and in this project is mounted directly on the PCB. A detachable antenna is used for LoRa communication. A block diagram of the hardware is shown in Figure 6.

2. Adafruit Bluefruit LE

The Adafruit Bluefruit LE is a Bluetooth Low Energy (BLE) module allowing communication with a mobile phone. This is used to notify the system that a mobile device is near. The Bluefruit communicates with the CubeCell asynchronously through UART. This communication is full-duplex and happens through a transmit (TXO) and receive (RXI) channel. The Bluefruit module has additional pins for hardware flow control that are unutilized in this project. Bluetooth operates in the 2.4 GHz ISM band in the United States.

3. Power Management Circuit

The power management subsystem powers the remaining subsystems. Energy is stored in a rechargeable 5 V, 2.4 A Li-ion or LiPo power bank/battery. The system also has functionality for solar power. This is not required and unused in this project, but the design accommodates use of the solar panel

input pin on the CubeCell through a screw terminal. In case a solar input is used, the CubeCell has internal circuitry for charging the battery through solar power. A 1N5819 power diode prevents current flow to the battery when the system is powered via USB (such as when it is being reprogrammed). A single-pole, single-throw switch turns the system on and off by connecting and disconnecting the power source, respectively. Bypass capacitors ensure low noise at the voltage inputs to relevant subsystems.

4. Battery Monitoring Circuit

The battery monitoring circuit is a resistor network used to read the voltage on the battery. It is connected to an ADC module on the CubeCell. This voltage can be used to estimate the charge level on the battery. The ground of the resistor network is connected to a GPIO pin such that it can be set to a high impedance state when the circuit is not in use to minimize power dissipation through the resistors. A diagram of the circuit is seen in Figure 7.

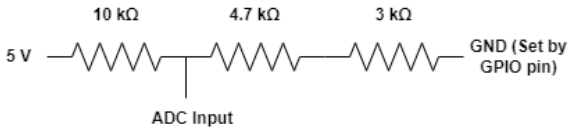

Figure 7. Battery monitoring circuit.

5. Input Buttons

The input buttons allow a user to interface with the system. There are three buttons: Toggle, Select, and Emergency. The Toggle button toggles the display on/off, the Select button turns the BLE communication protocol on/off, and the Emergency button allows the user to send an emergency signal to the server, which results in email and text message alerts of the device's emergency status. The buttons are mounted to the side of the box enclosing the system, making them easily accessible for the user. The buttons are also color-coded for easy identification.

6. Status LED

The status LED illuminates in green when the system is on. It gives the user visual feedback that the system is working when the power is enabled.

7. PCB

The PCB serves to connect all subsystems together into one unit. Its size is exactly 2.5 in. by 5 in. and it has holes for mounting in the enclosing box. Two copper planes—one for 5 V and the other for ground—are poured in the top and bottom layers, respectively. All routing on the PCB was done manually (i.e., no auto-route). Bypass capacitors are placed close to their corresponding voltage inputs. Screw terminal footprints are aligned with identical spacing between each terminal. Footprints for ceramic capacitors use a 100 mil pitch while those for electrolytic capacitors use a 200 mil pitch. A header allows for external access to power pins and the UART pins for communication between the

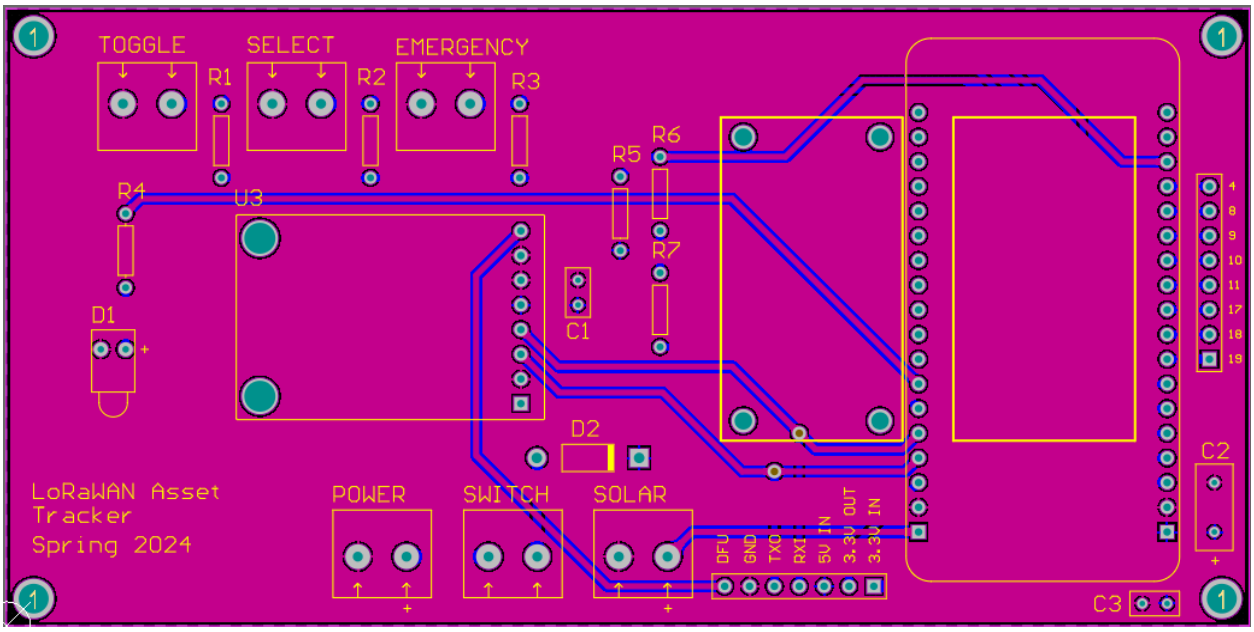

Figure 8. Top view of PCB layout of system. CubeCell footprint is shown on right side.

Bluefruit and CubeCell. Another header (8-pin) is connected to unused pins of the CubeCell, allowing them to be accessed in the future if needed to add functionality.

The CubeCell footprint is placed on the PCB such that the GPS antenna is facing outwards. This ensures that the GPS signal is not blocked by the copper planes of the PCB. Since the display is on the opposite side of the CubeCell as the GPS module, it needs to be unmounted from the CubeCell to allow it to still be visible. To give it support, it is mounted directly to the PCB. Since the display is held to the CubeCell by a set of four screws, it can easily be removed. Space was allocated and precise measurements were made for the display on the PCB. Screw holes were aligned with the display's location on the CubeCell. A top view of the PCB is shown in Figure 8.

8. Enclosure

The enclosure is a waterproof gray polycarbonate box from Bud Industries, Inc. The box is large enough to fit all the required parts of the system. The PCB is mounted in the box with four plastic standoff connectors and screws. There is ample space under the mounted PCB for the battery, which is secured in place by a velcro strap. Four holes are drilled into the sides of the box for the three buttons and power switch. The cover of the box is transparent, allowing the user to see the display and status LED through the box, and is secured to the box with a set of screws. The buttons and switch are mounted securely to the box with washers and hex nuts and built-in plastic clips, respectively. They can further be waterproofed with some sort of waterproof seal, such as caulk.

A limitation of this design is that removing the battery requires removing the PCB mounting screws. However, since the battery needs to be recharged infrequently, this is a minor concern. Another limitation is that the LoRa antenna is not securely mounted within the enclosure. This is also a minor concern since there is little room for the antenna to move around into the box given the little space between it and the wall of the box.

9. Bill of Materials

The hardware components along with their costs can be seen in Table 1. The total estimated cost comes out to $88.20 per system. Some of the associated costs assume the quantities of each part being purchased are for multiple systems. This is the case with the PCB which is purchased in quantities of at least 5, and the cost per system is the total PCB cost divided by 5.

IV. CONCLUSION

Our project presents a comprehensive LoRaWAN-based IoT system tailored for agricultural applications that bridges critical safety and efficiency gaps in the sector by integrating features such as asset tracking for farm machinery and a personal emergency button for workers. We leveraged several systems and software to create a robust network and ensure reliable communication.

We created code for the device to transmit data, code for the frontend server on TTN to receive, decode, and retransmit this data, and code for the backend server on ThingSpeak to receive the decoded data, analyze it, and send alerts. Additionally, we implemented a Python GUI that extracts data from ThingSpeak and creates various visualizations for the data including a geolocation map, device status, device charge, land plots, and much more, for the user to have a simple, seamless experience interfacing with all the data. Lastly, we integrated Bluetooth capability, facilitating seamless

Item	Quantity	Designator	Description	Value	Manufacturer or Retailer	Manufacturer Part Number	Cost
1	1	U1	CubeCell GPS-6502	902-928 MHz	Heltec Automation	HTCC-AB02S	$25.90
2	1	U3	Bluefruit LE UART Friend		Adafruit	2479	$17.50
3	1	D1	Green LED		*Generic*		$0.29
4	1	D2	Schottky Power Diode		STMicroelectronics	1N5819	$0.38
5	2	C1, C3	Ceramic Through-Hole Capacitor	100 nF	*Generic*		$0.44
6	1	C2	Electrolytic Through-Hole Capacitor	100 uF	*Generic*		$0.24
7	4	R1, R2, R3, R4	Axial Through-Hole Resistor	1 kΩ	*Generic*		$0.40
8	1	R5	Axial Through-Hole Resistor	10 kΩ	*Generic*		$0.10
9	1	R6	Axial Through-Hole Resistor	4.7 kΩ	*Generic*		$0.10
10	1	R7	Axial Through-Hole Resistor	3 kΩ	*Generic*		$0.10
11	1	P1, P2	Male Pin Header		*Generic*		$0.80
12	6		Screw Terminals		Pololu Electronics		$3.00
13	3		Waterproof Momentary Push Button Switch		Apiele		$4.25
14	1		Waterproof Round Rocker Switch		Tesorrio		$3.40
15	1		Rechargeable Power Bank	5000mAh/5V/2.4A	Miady		$17.99
16	1		IP68 NEMA 6P Box with Clear Cover		Bud Industries, Inc.	PN-1332-AC	$11.65
17	1		Printed Circuit Board		JLCPCB		$1.66
						Total Cost	**$88.20**

Table 1. Bill of materials for system.

communication between the server and end node via a dedicated cell phone app.

In the end, we created a reliable, scalable, and robust device that can revolutionize the agricultural sector. As this project progresses, with deployment, testing, and fine-tuning, its transformative potential will usher in tangible benefits for both farmworkers and operational efficiency.

V. FUTURE WORK

Our end node and GUI were designed to be flexible and to allow future LoRaWAN teams to easily add to our hardware and software. With female header pins on the PCB connected to the Heltec's GPIOs, future teams will be able to add environmental sensors or other peripherals to the end nodes and edit the Heltec's software accordingly. By utilizing our Heltec-TTN-Thingspeak-Python GUI pipeline and editing it to align with their project's goals, they will be able to get a head start on the software side of their project and be able to focus more on their hardware.

An area of focus for the future LoRaWAN teams, assuming that they will build upon our Asset Tracker and Emergency Button project, is to miniaturize the end node, which will allow it to be worn by field workers. Currently, our end node is far too bulky to be worn and is more conducive to being mounted on expensive farm equipment. Miniaturization of the device will necessitate use of LiPo batteries to power the device, rather than the large, 5V power banks currently used. Additionally, more work is required to produce accurate battery percentages on the Python GUI. This battery percentage is achieved via linear interpolation of the assumed voltage ceiling and floor of the power bank using the battery voltage that is sent to the server. However, these voltage ceilings and floors were merely estimated, and further testing is required to identify the voltage of the end node's battery at full charge versus minimal charger. We ensured that such changes are easily configured in our GUI code.

VI. FURTHER RESOURCES

For additional details, including our project documentation, PCB design files, and source code, please visit our GitHub page:
https://github.com/williamlorenzen/LoRaWAN-Team

VII. ACKNOWLEDGEMENTS

We would like to thank our senior design faculty sponsor, Dr. William Eisenstadt, for his guidance on this IoT4Ag project. We would also like to thank Professor Mike Stapleton for his help with hardware design, Dr. Ian Small for his user interface suggestions, and Eric Liebner for his help with the enclosure and parts.

VIII. REFERENCES

[1] "What is LoRaWAN® Specification," LoRa Alliance®, https://lora-alliance.org/about-lorawan/#

[2] "Lora Architecture - LoRaWAN Tutorial," 3GLTEInfo, https://www.3glteinfo.com/lora/lora-architecture/

[3] U.S. BUREAU OF LABOR STATISTICS, "Census of Fatal Occupational Injuries Summary, 2022," Bls.gov, 2023. https://www.bls.gov/news.release/cfoi.nr0.htm

[4] "geoplot," Plot line in geographic coordinates - MATLAB, https://www.mathworks.com/help/matlab/ref/geoplot.html