Handheld Rocket Tracker

By

Benjamin Olaivar

Maxwell Kramer

Manas Tiwari

Final Report for ECE 445, Senior Design, Spring 2024

TA: Sanjana Pingali

1 May 2024

Project No. 16

Abstract

The Handheld Rocket Tracker seeks to make a more affordable and convenient method of recovering a rocket after an amateur rocket launch. In this two-part system, a beacon is placed inside the rocket, which transmits its gps position. A handheld tracker receives that signal, and guides the user to the beacon, acting as a compass, however instead of pointing north, it points to the beacon. Unfortunately the handheld device was not successfully completed on a PCB, which led to ergonomic issues, however the core functionality of the device was successful.

Contents

1. Introduction

The Illinois Space Society is an amateur rocketry team here on campus, which launches a rocket 1-2 times a semester. Arguably the most important part of these launches is the recovery of the rocket. The team has committed considerable time and money into this project, and wants to retrieve their investment. Launches can reach altitudes of 60,000, where the rocket will deploy a parachute and drift back to the ground. It's nearly impossible to track a rocket with eyesight at this altitude (see Figure 1 for the rocket POV). Commercially available trackers exist, however they're typically clunky, expensive and unintuitive, making it inconvenient to use.

To solve this issue, we have designed a Handheld Rocket Tracker, which consists of 2 parts: First is the beacon module, which is placed inside the rocket and transmits its GPS location at all times. Second is the handheld tracker, which receives this transmission, and guides the user to the beacon. Our solution attempts to make a more affordable and intuitive method of tracking amateur rockets.

Figure 1: Rocket POV at altitude of ~10,000ft. Source: Illinois Space Society 2021 October Launch [1]

1.1 High Level Requirements

1. Successfully transmit positional and state data from the beacon to the handheld tracker, and handheld tracker should successfully transmit commands to the handheld beacon. See outlined in the "Packet Breakdown" under Software Design.

2. Have the capability for the user to switch the frequency of both the beacon and the handheld tracker via user input on the handheld tracker device.

3. Accurately show the distance from the beacon within 5 meters, and point the user in the correct direction of the beacon within 5 degrees. This information should be shown via the screen in the User Interface, and behave similar to a compass, however pointing towards the beacon instead of pointing North.

2 Design

The high-level design and subsystems for the beacon and handheld tracker are shown in Figure 2, and their physical concepts can be seen in Figure 5. The beacon, seen in Figure 4, has 4 primary subsystems: mcu, sensor, communication, and power. The mcu subsystem is responsible for handling communication between subsystems, and making any necessary calculations. The sensor subsystem, consisting of a gps module, is responsible for collecting positional data for its respective device. The communication subsystem handles communication between the beacon and the tracker. This communication is done using the LoRa communication protocol within the 433MHz band. Finally, the power subsystem takes a variable voltage input of 3.7 to 4.2V, and steps it down to 3.3V, which powers the entire device.

Note the handheld system, seen in Figure 5 is identical to the beacon, however with the addition of the user interface subsystem. The user interface consists of buttons and a screen, which take inputs from the user to navigate the menu, and guide the user to the beacon.

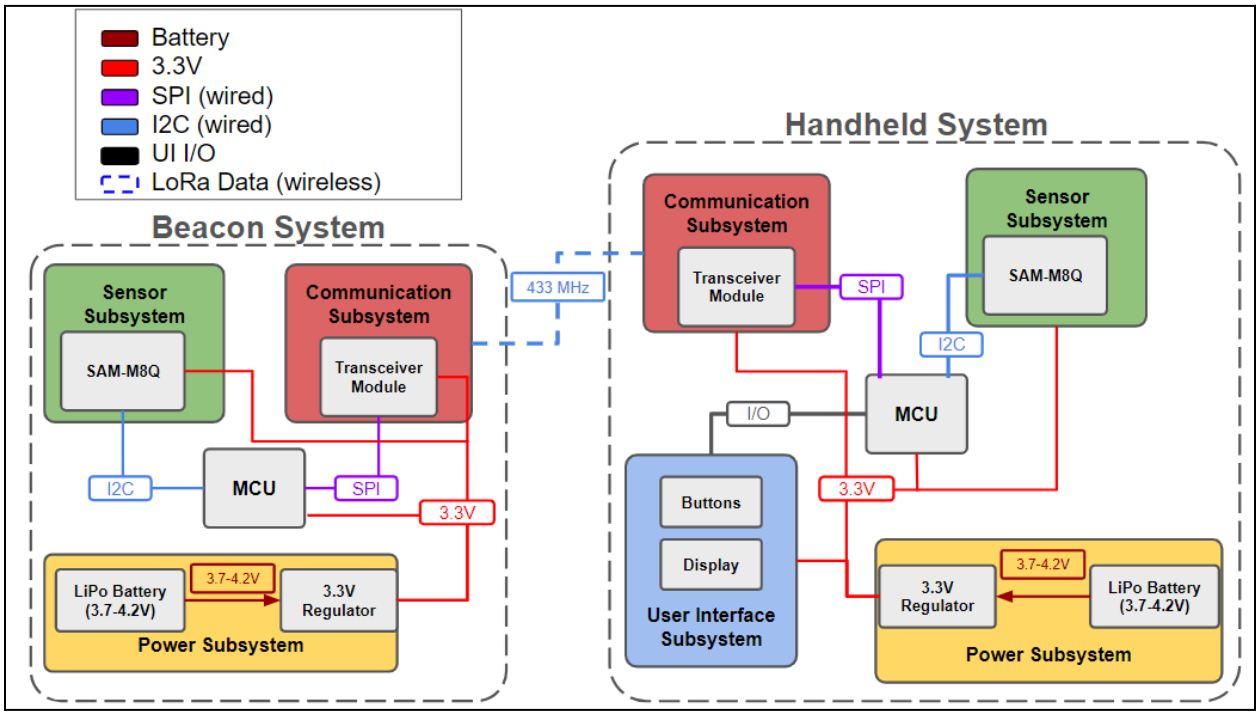

Figure 2: Subsystem Block Diagram

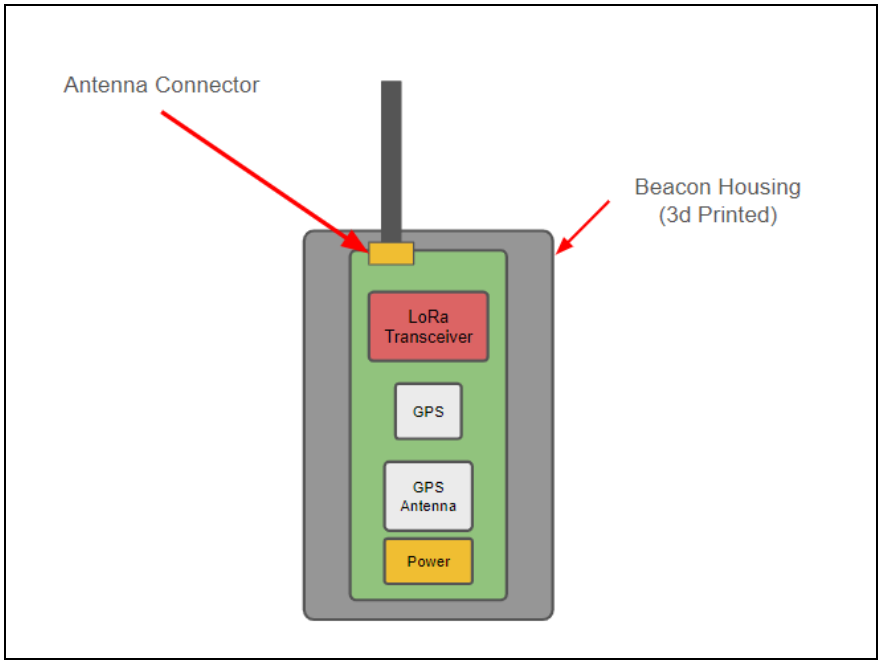

Figure 4: Beacon Concept Design

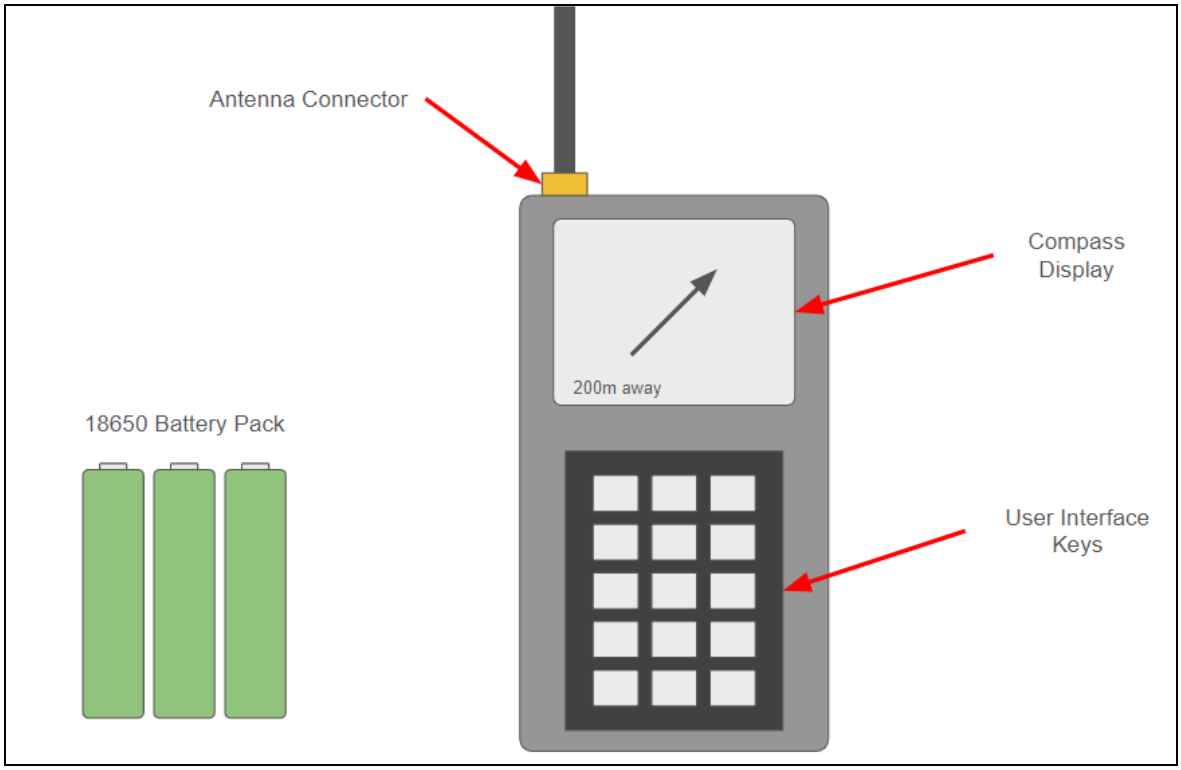

Figure 5: Handheld Tracker Concept Design

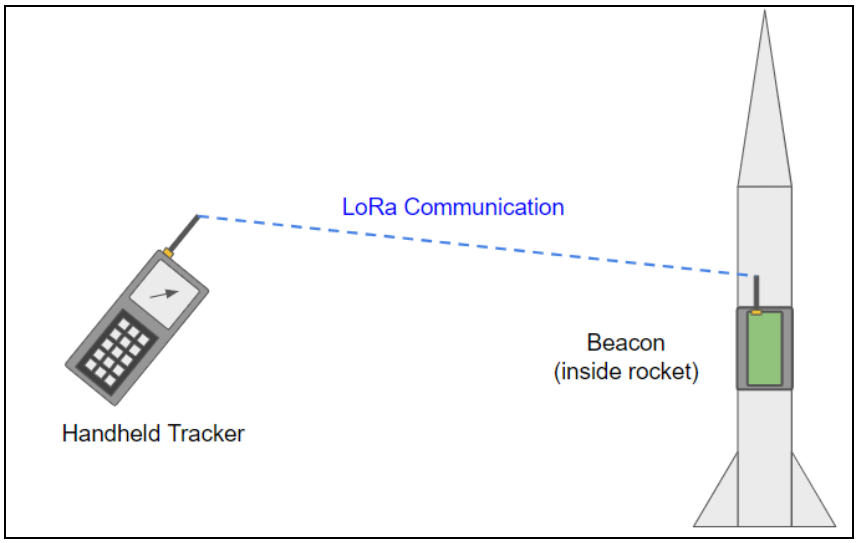

Figure 6: General Overview Diagram

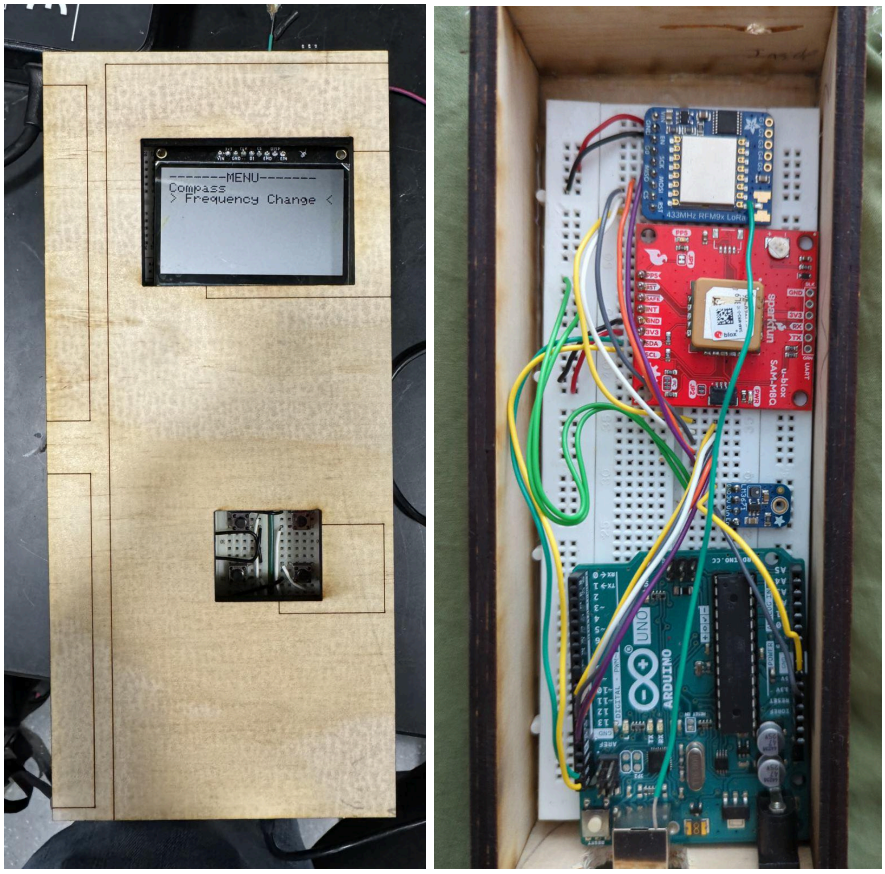

Figure 7: Final Demo Design

2.1 MCU Subsystem

The MCU subsystem is responsible for managing communication between the various subsystems, as well as making relevant calculations. While the beacon and handheld tracker are very similar, the MCU on each has slightly different responsibilities.

2.1.1 Beacon MCU

On the beacon, the MCU is responsible for receiving positional data from the sensor subsystem in the form of Longitude and Latitude via I2C, as well as receiving commands from the handheld tracker. Upon landing, the parachute, which is still deployed, may get caught in the wind and drag the rocket. To account for this, the beacon updates its gps coordinates every 3 seconds, and transmits the new data to the handheld tracker via the Communication Subsystem. Additionally, the user may need to change the transmission frequency of the tracker. This isn't uncommon, considering the narrow band that LoRa operates in, and the high volume of teams at rocket launches, which may be using the same frequency to track their own rockets. The MCU listens for a "Change Frequency" command, and changes frequencies according to user input. Further explanation of frequency changes is detailed in the Communication Subsystem description.

2.1.2 Tracker MCU

The MCU on the handheld tracker has many of the same responsibilities, however with a few notable differences. The handheld device has its own gps, which updates its position as the user walks around. It compares its own GPS coordinates with the coordinates received from the beacon, and calculates the distance and angle between the two devices. Details of how this information is displayed are detailed in the User Interface Subsystem description, however the basic equation to calculate the angle between the two points is seen below.

$$\theta = arctan(\frac{|\Delta Latitude|}{|\Delta Longitude|})$$

To ensure an accuracy of 5 meters, the TinyGPSPlus library was chosen to calculate the distance between the two devices, as seen in Figure 8 below. It was chosen not to use simple trig for this variable, as it could get increasingly less precise as the distance between the beacon and the tracker grew, hence the use of an external library.

```
/* static */
double TinyGPSPlus::distanceBetween(double lat1, double long1, double lat2, double long2){
// returns distance in meters between two positions, both specified
// as signed decimal-degrees latitude and longitude. Uses great-circle
// distance computation for hypothetical sphere of radius 6372795 meters.
// Because Earth is no exact sphere, rounding errors may be up to 0.5%.
// Courtesy of Maarten Lamers
            double delta = radians(long1-long2);
            double sdlong = sin(delta);
            double cdlong = cos(delta);
            lat1 = radians(lat1);
```

```
  lat2 = radians(lat2);
  double slat1 = sin(lat1);
  double clat1 = cos(lat1);
  double slat2 = sin(lat2);
  double clat2 = cos(lat2);
  delta = (clat1 * slat2) - (slat1 * clat2 * cdlong);
  delta = sq(delta);
  delta += sq(clat2 * sdlong);
  delta = sqrt(delta);
  double denom = (slat1 * slat2) + (clat1 * clat2 * cdlong);
  delta = atan2(delta, denom);
  return delta * 6372795;
}
```

Figure 8: TinyGPSPlus Distance between two coordinates function [2]

2.1.3 Chip Selected and Challenges Faced

To accomplish this, we chose to use the Atmega328p, which is the same chip as the arduino uno. We chose this chip because of its simplicity and ease of use, however this ended up coming back to hurt us. This chip only has 2KB of RAM, which is incredibly restrictive, especially considering the fact that our screen requires a minimum of 1KB to operate. This small memory size became even more apparent as we incorporated our GPS module, which was made by SparkFun. The SparkFun gps library [3] was designed to be a “cover all” for all of their GPS devices. This means it wasn’t really meant to be efficient, it was just meant to get the job done. This was a problem for us because we were running out of memory. The initial SparkFun library took up 76% of our total RAM. To solve this problem, we rewrote the GPS library, removing any unused global variables and functions, and deleting debug prints. By doing this, we got the GPS down to only 42% of our memory, which allowed us to use both the GPS and the display at the same time.

2.2 Sensor Subsystem

The Sensor subsystem is responsible for gathering positional data from the gps module and sending it to the MCU via I2C. Conveniently, we have found a sensor that has everything we need built into 1 unit: the SAM-M8Q. The SAM-M8Q has a builtin GPS module and antenna for communicating with local satellites, so no external antenna was needed. The SAM-M8Q breakout can be seen in Figure 9.

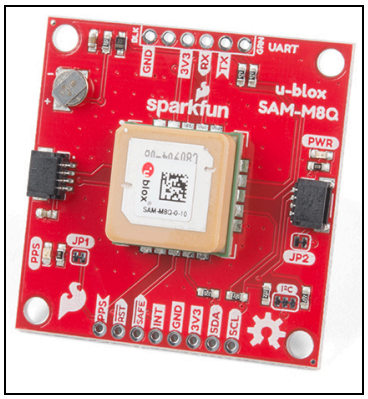

Figure 9: SAM-M8Q breakout board with built-in ceramic antenna [4]

2.3 Communication Subsystem

The communication subsystem is the system responsible for handling communication between the beacon and the tracker. Without this two-way communication we would not be able to track the rocket itself as we would lack comparable GPS data. This subsystem consists of two identical parts. Each is a LoRa RFM96 radio, one located on each of the beacon and tracker. The beacon will generally give the tracker its GPS location while the tracker will, on user request, change both itself and the beacon's operating frequency. The radios can operate between 433 and 434.8 MHz and can operate well between 0.5 to 5 miles [5]. The radios interact with the MCU using SPI. It should be noted that all data being sent needs a valid FCC license attached due to the operating frequencies being non-general purpose wavelengths.

2.3.1 Beacon Communication

The beacon's LoRa module has one transceiver action and one receiver action to perform.

For transceiving, the beacon will collect GPS data and package it within the Sensor and MCU subsystems. After the data is ready for transmission the LoRa code library will be used. Within it are three functions that package data for transmission. These are the beginPacket(), write(), and endPacket() functions which collectively in sequence create and send out a packet of data [6]. Once the data is packaged it will be transmitted on the current frequency set for the beacon. This occurs once every three seconds in order to not overload the tracker with incoming data.

For receiving, every cycle of the main code a parsePacket() is called from the LoRa library. This parses the data into a form that can be manipulated and returns the bytes it received, or 0 if no data was found. The beacon will specifically only ever receive a packet containing the value of a new requested frequency from the user via the tracker. This frequency is extracted and checked for its value to be in the mentioned valid range. If yes then we stop accepting new frequencies for 5 seconds. This reason will be discussed in the next section for the tracker portion, Once this is validated we set the internal frequency and begin transmitting GPS data again.

2.3.2 Tracker Communication

The tracker's LoRa module also has one transceiver purpose and one receiver purpose.

For transceiving, the tracker will only do this for when a new frequency is requested by the user. When this happens we transmit the new requested frequency once every second for 5 seconds. This is because we want to make sure that the beacon doesn't miss the singular request and so we send redundant requests. This is why, as mentioned above, we halt accepting new frequencies for 5 seconds after a valid is received. Testing showed that many requests would get jumbled and cause the beacon to switch to an unwanted frequency. The data is packaged the same way as mentioned above within the beacon.

For receiving, the tracker will use the same parsePacket() LoRa function discussed before. On a valid packet being found, its contents are read into a dedicated struct locally and saved in other variables for the MCU's use.

2.4 Power Subsystem

The power subsystem is responsible for intaking the voltage available from a power source and converting it to a usable voltage for every other subsystem. This system is present in identical forms in both the beacon and the tracker. The system has two main parts. The original design consisted of 18650 LiPo batteries and a LM3671 3.3V buck converter.

The LiPo batteries would, based on its charge, provide 4.2-3.2V to the input of the buck converter [7]. The batteries would've been removable and would allow end users to charge them themselves and swap when needed. The batteries were not, however, usable in the final form of the project. Instead they were replaced by the 5V output line of an arduino uno dev board, which received its power separately. This will be discussed in depth within the power subsystem verification section later on in this report.

The LM3671 buck converter was responsible for converting input voltage to 3.3V output for the rest of the systems in both the beacon and tracker. This was chosen over a linear regulator for concerns about overheating during rocket launch. This specific one was chosen for its ability to handle up to 600 mA of current draw [8]. This draw is well above expected draw from all the other systems and therefore is ideal for our purposes.

2.5 User Interface Subsystem

This subsystem facilitates user interaction with our project. It allows the user to access relevant data pertaining to tracking the beacon (i.e. the distance and direction), and takes in user inputs where and when required. It consists of a display to show data relevant to the user, and push buttons to enable navigation and provide inputs.

The code to display text onto the screen is written in C++, and uses helper functions from the Adafruit_GFX.h and Adafruit_SSD1306.h libraries to do so. In addition, the push buttons are

polled continuously in a loop to detect user action. The logic of printing menu and sub-menu options along with the logic to navigate through them are also coded in C++.

2.5.1 Data Display

The screen used for our project is a 400 x 240 pixels Adafruit Sharp Memory Breakout display. It defaults to displaying the current menu, and switches to displaying other sub menus as per the user's directions.

The distance and direction to the beacon (computed by the MCU) is displayed within the Compass Menu. The distance is displayed as text, while the direction of the beacon is shown in a compass-like format, where the needle would be pointing towards the beacon. See Figure 10 for more information.

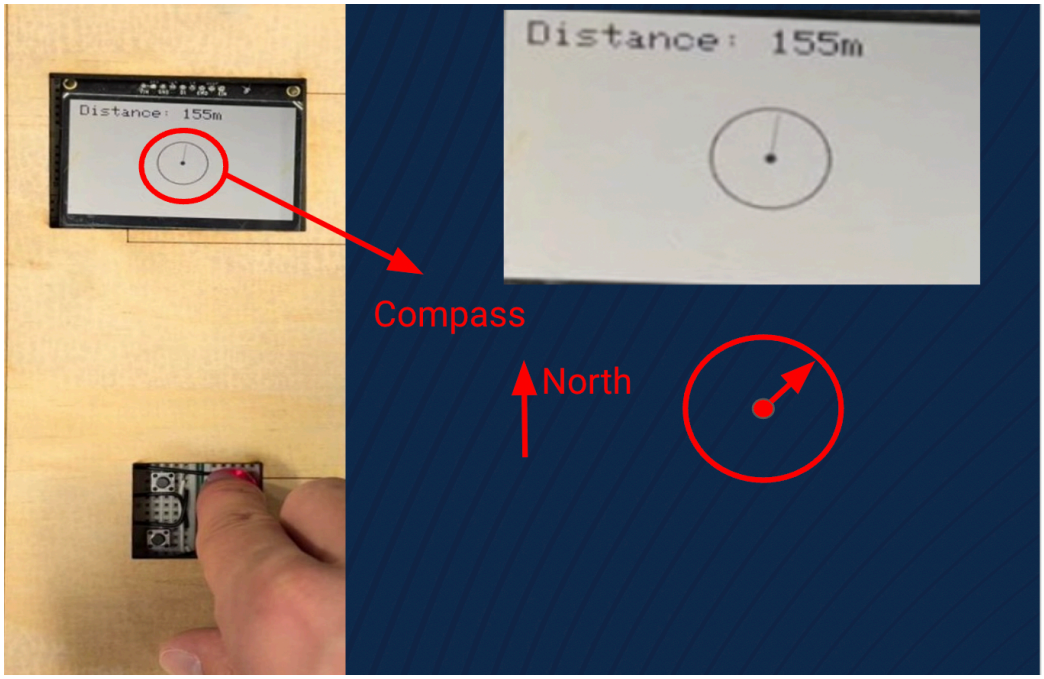

Figure 10: Compass Menu displaying Distance and Direction to the Beacon

The display screen is crucial in most of our verification procedures for the rest of our subsystems. Hence, it was of utmost importance that it worked correctly during our project's progress.

During our project, we needed to resolve multiple issues regarding the display. The first issue arose when the screen we had initially chosen took up more than 50% of our chip's memory. Our solution at the time was to switch to one with a smaller size and resolution. However, we later realized that the new screen chosen would not be physically capable of displaying the direction to the beacon with a precision of 2 degrees. This led us to making a

change in our high level requirements to go from displaying the direction with a precision of 2 degrees to 5 degrees instead.

During final rounds of testing, we noticed that the screen was chipped. This caused the screen to not display anything at all, and forced us once again to make a switch to the 400x240 Adafruit display. This switch required us to rewrite major portions of the UI code, due to the fact that this screen used a different set of drivers (Adafruit_SharpMem.h instead of Adafruit_SSD1306.h). We ordered the same screen, but it did not reach in time before the final demo.

2.5.2 Push Buttons

The last component of the UI subsystem deals with allowing the user to provide inputs in order to navigate to the desired menu to view its relevant information. This was implemented with 4 buttons corresponding to Up, Down, Select (Enter) and Main Menu selection (see Figure 11). Menu navigation is achieved using these buttons, with UI code running in parallel to ensure the correct information is being displayed as requested by the received inputs.

The “up” and “down” buttons are mostly used to change the highlighted option. This behavior differs in the Frequency Change menu, where they are used to increment/ decrement the displayed frequency by 0.1 MHz. The “enter” button signifies confirmation to enter the highlighted menu, or to confirm an action (such as setting the frequency). The “main menu” button is used to quickly navigate back to the default main menu display.

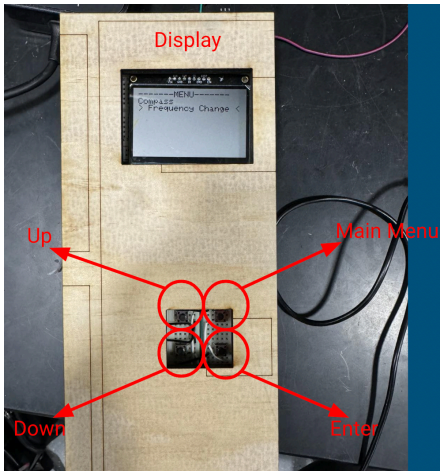

Figure 11: Push Buttons and their functionality

3. Design Verification

3.1 MCU Verification

The verification process for the MCU was primarily focused on getting a response from the chip on the PCB. During our testing, we unfortunately couldn't get the MCU on the PCB to respond to our programmer. We first soldered the MCU and external 16 MHz clock onto the PCB, and attempted to upload to the board using Arduino as ISP. Unfortunately during this process, we never got a response from our chip. Testing showed that the external 16 MHz clock was not oscillating. After discovering this, we instead used the internal 8MHz clock. Again, we got no response from the MCU. Unfortunately we still got no response from the chip. In the end we decided to omit the PCB in order to focus on fulfilling the high level requirements of our project.

3.2 Sensor Verification

To ensure accurate distance calculations between the beacon and tracker, we required this GPS to be accurate within 5 meters. To validate this requirement, we compared the coordinates received by the GPS to some known locations on campus. We used the coordinates of the spire on Eng quad as a baseline. We powered on the GPS, and compared the received coordinates of the gps to the spire, and made sure our location was accurate.

3.3 Communication Verification

The communication subsystem has two main areas of verification. The first is making sure beacon transceiving and tracker receiving are working together. The second is making sure beacon receiving and tracker transceiving are working together.

3.3.1 Beacon TX to Tracker RX

To verify a beacon to tracker transmission of the beacon's GPS data, we have three steps. First, we have beacon print its GPS data it has and send it out as a packet every three seconds. This print out occurs on a connected serial port from the arduino uno the code is housed in. Second, we print out any received packets and their contents on a second serial port for the tracker when it sees any incoming data. This is to verify data is consistent between the devices. Finally, since we had UI working at the time we began RXTX testing, we had the MCU use the data as intended and check if the UI gives proper distance and direction. If yes, then the beacon to tracker data transmission is working.

3.3.2 Tracker TX to Beacon RX

To verify a tracker to beacon transmission, we do the same testing plan as described in 3.3.1 above with some small changes. We do add one additional step at the end regarding matching the frequencies. After the beacon receives the new frequency and begins operating at it we need to check that communication is still active, mainly to see if both devices got the same new frequency. Therefore we hold the user in the current menu and wait to see if a packet of data

is received by the tracker. If it does then they both are at the new frequency. Otherwise, different frequencies have been applied and we therefore have failed.

3.4 Power Verification

Power verification was short but slightly complicated by the need to forgo the PCB. With the project running on a breadboard system with the arduino uno dev board, we could not rely on battery power. The dev board needs 12V-5V input, lower than the 4.2V maximum of the LiPo batteries. To work around this, the arduino was powered via usb connector to a laptop that was needed for serial port printout for verification. The 5V output pin of the dev board was then fed into the input of the buck convertor. This convertor then supplied its intended 3.3V output to the rest of the subsystems and was successful in powering them without any issues.

We also debugged the buck's themselves. The first one we received was determined to have an internal error as power supply and voltmeter testing revealed no output voltage. Replacements ordered did work as intended on the first try.

3.5 User Interface Verification

Testing of the display screen was largely a straightforward process. We first needed to ensure that the screen was working. This was done by powering the screen and displaying some text on it. Once its functionality was confirmed, we tested it further by displaying the distance and directional data, and other details as and when required.

Testing of the push button was done by manually pressing the buttons, and displaying different text for each corresponding push button. Each button was pressed independently several times to ensure their expected behavior.

Menu navigation was tested by providing inputs, and visually verifying that the menu navigation and selection were working as expected.

4. Costs

4.1 Parts

Table 1: Parts Costs

Part	Manufacturer	Individual Cost ($)	Quantity	Total Cost ($)
LM3671 3.3V Buck Converter	Adafruit	$4.95	2	$9.90
Adafruit RFM96W LoRa Radio Transceiver	Adafruit	$19.95	2	$39.90
Monochrome 0.96" 128x64 OLED Graphic Display	Adafruit	$17.50	1	$17.50
ABLS-16.000MHZ-B2-T CRYSTAL 16.0000MHZ 18PF SMD	Abracon LLC	$0.45	2	$0.90
CONREVSMA002-G CONN RP-SMA RCPT R/A 50 OHM PCB	TE Connectivity Linx	$3.74	2	$7.48
Sam M8Q GPS	Sparkfun	$42.95	2	$85.90
Switch Tactile SPST	Shruter INC	$0.26	20	$5.20
RF ANT 433MHZ WHIP TILT	RF Solution	$5.69	2	$11.38
ATMega 328p	Microchip Technology	$2.74	2	$5.48
10K OHM Resistor	Stackpole Electronics	$0.10	10	$1.00
1K OHM Resistor	Stackpole Electronics	$0.10	10	$1.00
22pF Capacitor	Murata Electronics	$0.10	10	$0.29 (Bulk order)
0.1uF Capacitor	Murata Electronics	$0.10	10	$0.19 (Bulk order)
4.7uF Capacitor	Murata Electronics	$0.12	5	$0.60
10uF Capacitor	Murata Electronics	$0.10	10	$0.19 (Bulk order)
Total	~	~	~	**$186.91**

4.2 Labor

With an hourly rate of $50 per hour, as used and approved of in this project's design document, and assuming 15 hours per week over 15 weeks, we can get the following labor cost.

$50/hr * 15 hr/week * 15 weeks * 3 members * 2.5 = $84,375

4.3 Schedule

Table 2: Initial Schedule

Week	Task
February 26th - March 4th	Order parts for prototyping (Max)
	Start prototyping with existing components (All)
	Research Transceiver communication (Manas)
	Start PCB design (Max & Ben)
March 4th - March 11th	Begin 3D print designing (Ben)
	Successfully establish transceiver communication (Manas & Ben)
	Finish 1st iteration PCB design (Max & Ben)
	PCB Order (All)
March 11th - March 18th	Print first versions of 3D printed case prototypes (Max & Ben)
	Finish the baseline transceiver communication code (Manas)
	Finish baseline user interface menu (Ben)
	Range testing with wire antennas (All)
March 18th - March 25th	Finalize 3D prints (Max & Ben)
	Prototype user interface menu, controlling with Arduino Uno (Ben)
	Revisions to PCB (Max)
	Revisions to User interface software (Ben)
	PCB Order (All)
March 25st - April 1st	Revisions to PCB (Max)
	Revisions to User Interface software (Ben)
	PCB Order (All)
	Range testing with 1st ordered antennas (All)
April 1st - April 8th	Revisions to 3D design (Max)
	PCB Order (All)
	Finalize 3D prints (Ben)
	Order new antennas (if necessary) (All)

April 8th - April 15th	Range testing with new antennas (if necessary) (All)
	Fix existing bugs (All)
April 15th - April 21st	Finalize Assembly (All)
	Fix existing bugs (All)
April 22nd	Final Demo (All)

5. Conclusion

5.1 Accomplishments

The project met all the high level requirements which were outlined before. We verified that all our subsystems were working as expected individually, and when put together as a single unit.

We ensured the functionality of the project in expected conditions of amateur rocket launches, and close to complete functionality in less than ideal conditions.

The long range viability of our project and accuracy of the data recorded and displayed makes it immediately viable for amateur rocket tracking. The cost of about $250, including shipping for parts and profit markup, makes it a much more affordable option compared to other similar devices in the market.

This project has several complexities, and we ran into multiple issues while building it. This required our group to learn and deal with challenges in an efficient manner, which would be applicable in the industry. Besides, learning about RF communication (being a new topic for most project members), understanding the vitality of economically using memory on chips, learning new ways of debugging the physical board etc. are all highlights of what we managed to accomplish as learning milestones.

5.2 Uncertainties

The outdoor nature of the project brings uncertainties around its performance in inclement weather conditions. The GPS sensors we currently use work best in ideal weather conditions, and don't work as well in cloudy/ rainy conditions. We would have preferred using better GPS sensors that would be more resistant to poor performance in less than ideal weather.

The difficulties in uploading code onto the actual MCU chip is a factor to potentially look into in more depth. During debugging, we noticed that the internal clock of the chip was not running, which was causing the failure of programming the chip. We are unsure as to why this occurred. We also could look into alternative methods of programming in order to work around this problem.

The screen chipping required us to change our high level requirements, and required a change in UI libraries used to code the functionality of printing to the screen. We are unsure of the cause of the chipping, and needed to simply replace it given the timing of our demo.

5.3 Ethical considerations

This group was careful in considering the ethical and unethical applications of our project. The one main item we wished to address was the misuse of our tracker system for personal espionage and similar unconsented tracking of objects or people. IEEE's Code of Ethics Section 1.1 states we are responsible for the following. "To strive to comply with ethical design and sustainable development practices, to protect the privacy of others, and to disclose promptly factors that might endanger the public or the environment" [9]. We have some design limitations and intended features that prevent such misuse. The tracker requires line of sight or low levels of obstruction between the target and user alongside outside use. Therefore use in dense areas or indoors renders the tracking inactive. Furthermore, the 433-434.8 MHz operating frequencies, as discussed earlier, need a FCC license number to be included in data transmission. This requires a malicious user to provide their license or use an illicit number, making them easy to ID by authorities.

5.4 Future work

Upgrading to a more powerful microprocessor with better storage capabilities would be the first step to improving our project. It would add the capability of adding more features and subsystems (magnetometers, accelerometers etc.) to our current project, and enhance our project's serviceability to the user.

Switching our current GPS sensors to one of higher quality would make our project's performance in inclement weather a lot stronger. While amateur rocket launches wouldn't take place in non-ideal weather conditions, it doesn't rule out the possibility of them occurring during tracking, and it would be helpful if our project could maintain its functionality during these conditions.

Improving the ergonomics of our tracker in order to make it more comfortable for the user to hold it for long periods of time would be another priority to incorporate. This would also include switching the casing from wood to a more durable material.

Incorporating physical antennas in our current design instead of the currently used wired antennas would be an excellent way to improve the quality and range of communication between the tracker and the beacon.

Sources

[1] “Illinois Space society Spaceshot Launch,” YouTube, https://www.youtube.com/watch?v=8wJdRGztPJ4&t=61s (accessed May 1, 2024).

[2] Mikalhart, “Mikalhart/tinygpsplus: A new, customizable Arduino NMEA Parsing Library,” GitHub, https://github.com/mikalhart/TinyGPSPlus (accessed May 1, 2024).

[3] Sparkfun, “SparkFun_Ublox_Arduino_Library/src/sparkfun_ublox_arduino_library.cpp at master · Sparkfun/SparkFun_Ublox_Arduino_Library,” GitHub, https://github.com/sparkfun/SparkFun_Ublox_Arduino_Library/blob/master/src/SparkFun_Ublox_Arduino_Library.cpp (accessed May 1, 2024).

[4] M. #167436, Greggler, M. #873985, and M. #985133, “SparkFun GPS breakout - chip antenna, Sam-M8Q (Qwiic),” GPS-15210 - SparkFun Electronics, https://www.sparkfun.com/products/15210?gad_source=1&gclid=Cj0KCQjw0MexBhD3ARIsAEI3WHJYFJ1ye5tECrM2Z-IMW8Q5sZYL2penmC-ZT9jH7FYt2s6aO2gbxPEaAkd9EALw_wcB (accessed May 1, 2024).

[5] Hoperf, https://www.hoperf.com/uploads/RFM96W-V2.0_1695351477.pdf (accessed May 1, 2024).

[6] Sandeepmistry, “Arduino-Lora/SRC at master · Sandeepmistry/Arduino-Lora,” GitHub, https://github.com/sandeepmistry/arduino-LoRa/tree/master/src (accessed May 1, 2024).

[7] “Lithium-ion Battery DATA SHEET Battery Model : LIR18650 2600mAh.” Available: https://www.ineltro.ch/media/downloads/SAAItem/45/45958/36e3e7f3-2049-4adb-a2a7-79c654d92915.pdf

[8] “LM3671/-Q1 2-MHz, 600-mA Step-Down DC-DC Converter Datasheet.” https://cdn-shop.adafruit.com/product-files/2745/P2745_Datasheet.pdf (accessed Feb. 20, 2024).

[9] IEEE, “IEEE Code of Ethics,” *ieee.org*, Jun. 2020. https://www.ieee.org/about/corporate/governance/p7-8.html (accessed: Feb. 22, 2024)