Trap Finder: Real-Time Crab Trap Tracking System (CTTS)

ECE499

August 6, 2024

University of Victoria

Group ID: 7

Faculty supervisor: Dr. Hong-Chuan Yang

Co-supervisor (if any): Glenn Jones from Reach Technologies

Team Information:

S. No.	Name	V Number
1	Evan Lee	V00025548
2	Ewan Chatham	V00922053
3	Simon Pollak	V00910939

Acknowledgment

We would like to thank everyone who helped us with this project.

First, we are very grateful to our project supervisor, Dr. Hong-Chuan Yang, for his guidance and support. His knowledge and advice were very helpful. We also want to thank Dr. Sana Shuja, and our teaching assistant, Maryam Ahang, for their support and feedback throughout the semester.

An enormous thank you to Glenn Jones from Reach Technologies, who sponsored and co-supervised our project. His technical and financial support was crucial in completing the project.

We appreciate the Electrical and Computer Engineering Department for funding our project, which allowed us to get the resources we needed. Finally, we want to thank our family and friends for their constant support and encouragement.

Table of Contents

Table of Figures

Executive Summary

Ocean pollution has become an increasingly problematic issue that the world is trying to solve. Of this ocean pollution, ghost gear, such as abandoned fishing traps pose significant harm to both marine life and marine economical industries. Implementing a low-power, real-time GPS tracking system into crab trap buoys, would reduce the chances of losing crab traps to drift, theft and abandonment.

The Real-Time Crab Trap Tracking System is implemented with a Feather M0 microcontroller equipped with a Long Range Radio (LoRa) module for communication and a GPS to track precise locations. The Feather M0 is housed inside the trap's buoy which communicates its location to another Feather M0 onboard a boat when it receives a ping asking for its location. By utilizing a method of request, acknowledgement handshaking from the Boat device to the Trap device, the Trap can maintain a low power "sleep" mode and only awaken periodically to check for requests. This low power mode, paired with solar charging capabilities allows the buoy to stay in the water for longer periods of time reliably.

This project was successfully able to implement a solar powered, real-time crab trap tracking system which can easily be deployed for existing crab traps. It shows the current and last known locations of crab traps in order to reduce the possibility of ghost gear in the ocean.

I Introduction

Trap Finder: Real-Time Crab Trap Tracking System (CTTS) aims to solve the significant problem of lost crab and prawn traps, commonly referred to as "ghost traps." Ghost traps continue to catch marine life indiscriminately, leading to both environmental harm and economic losses [1]. Our project addresses this issue by providing a reliable and eco-friendly solution for tracking traps using solar-powered traps and a boat equipped with LoRa technology and GPS.

The goal of this project is to develop a system comprising solar-powered communication devices attached to trap buoys and a Boat device that communicates with these Trap devices. When a signal is received from the boat, the Trap device will wake up, determine their GPS location, and transmit this data back to the boat, which will display the locations on a chart plotter.

The scope of this project includes designing and integrating both hardware and software components for the Trap and Boat devices, ensuring effective communication using LoRa technology, and integrating solar charging capabilities to keep the traps powered.

This project is significant because it addresses a real-world problem faced by both commercial and recreational crabbers. By reducing ghost fishing, we help protect marine ecosystems and promote sustainable fishing practices.
Our project aligns with several principles of the Engineers and Geoscientists BC (EGBC) Code of Ethics[2]:

- Protecting the public and environment by reducing ghost fishing and promoting sustainable fishing practices.
- Competence by applying our knowledge of embedded systems, solar energy, and communication technologies.
- Sustainability by promoting renewable energy sources in marine operations.

Potential users of our system include commercial and recreational crabbers, government fisheries, and ocean researchers. By providing a reliable and eco-friendly solution for tracking and managing crab traps, our project offers significant economic and environmental benefits.

II Objectives

- To develop the Trap Finder: Real-Time Crab Trap Tracking System (CTTS) consisting of solar-powered Trap device and Boat device.
- To design and integrate solar charging circuits for battery management of the Trap device.
- To ensure reliable communication between Trap and Boat devices using LoRa technology.
- To implement GPS functionality to track the location of crab traps.
- To develop an interface for the boat to display the location data on a chart plotter, initially on a laptop, with potential extension to boat chart plotters.
- Specific to the first progress report:
    a. To assign team member roles and responsibilities.

b. To define anticipated milestones and timelines for each phase of the project.
c. To outline deliverables and success criteria for each team member.

III Design Specifications

Microcontroller and Radio Communications

Adafruit Feather M0 RFM95 LoRa Radio

The microcontroller selected was an Arduino Feather M0 with LoRa module. This component was selected as it covers the need for both a main processor as well as the radio module, removing the need for additional hardware to manage LoRa communication. The frequency used in this project is 915 MHz as it is within a band designated for industrial, scientific and medical (ISM) applications [3] in Canada. The LoRa module has a 10-20 km signal radius which is more than enough for this project's application.

Power

2.5W 5V/500mAh Solar Panels

To power the device over long periods, a 3.7V 1200mAh battery was purchased as it was designed to integrate seamlessly with the Feather M0 via JST connector. The Feather M0 operates on 3.3V logic, but also accepts up to 5V which is down regulated to 3.3 V with a 500mAh peak current output [8].

DC Solar LiPo Charger

The DC Solar was utilised in order to smooth out the inconsistent current outputs from the solar panels. By using this charger (which contains a Voltage Proportional Charge Control (VPCC) and a 4700uF capacitor, it stabilises the voltage and current draw from the panel together with a Schottky diode, preventing current draining back into the panel.

LiPo 3.7V 1200mAh battery

The 2.5 W 5V/500mAh solar panels were chosen to provide maximum current with a sufficient voltage within a reasonable price range. As the DC solar charger accepts voltages of 5-6 V, this solar panel was chosen.

Based on the Arduino Feather's 300uA sleep mode power consumption, these batteries alone would be able to power the feather for up to 166 days [8].

GPS Communications

U-Blox SAM-M8Q

The U-Blox SAM-M8Q GNSS antenna module was selected for this application. As mentioned in the above review of literature, U-Blox's use of GPS, Galileo, *and* GLONASS allows a much more precise location reading than using just one GNSS system [9][10]. Alternatives exist such as the U-Blox NEO M9N module that incorporate BEIDOU, an additional GNSS system, but lack an integrated antenna so would be less convenient to implement.

Enclosure

The enclosure was modelled in SolidWorks and printed at the UVic library to have a specialized design for the electronics housing and solar panel mounts. The angle for the panel mounts was calculated using a solar-angle-calculator [4] which uses your location (latitude) to determine the optimal angle for the panels at each month in the year. This was determined to be about 58 degrees in July. The enclosure however used a ~56 degree incline in order to accommodate the 3D printer's size constraint and space for electronics inside.

While designing this project, we must be mindful of the EGBC Code of Ethics.

- In distinguishing facts from assumptions and opinions [Principle 7], impact of solar energy generation, and power consumption will be disclosed and supported by data collected. The data will be an accurate representation of the performance of this system.

- In accordance with Principle 3, when testing in unfamiliar marine areas, it is important that local laws and enactments are followed, so as to not disturb, disrupt or cause harm to colleagues, citizens or marine life.

- Following Principle 1, this project is designed to be an environmental solution that reduces the impact of ghost gear on the surrounding marine life.

- In holding paramount the safety, health and welfare of the public [Principle 1] GPS devices will be only for its intended purpose of tracking crab traps and will only be disclosed to those with authorization.

IV Literature Survey

With the problem defined, we are investigating 4 major features to review and implement for the Crab Trap Tracking System:

1. Radio Communications
2. Longevity of Deployment
3. GPS Communications
4. User Interface

Radio Communications is concerned with identification of trackers and communication of position data and any other status details. Longevity of deployment refers to how long a module can be deployed either continuously or without maintenance between uses. Traps for crabs and other shellfish can be deployed for mere hours, or in some cases more than a day [7]. GPS Communications refer to the choice of GPS system used to pinpoint the location of the tracking system. The user interface refers to how users will interact with the data, and what medium they will be able to visualize information in. This includes diagnostics like battery life, whether the device is moving more than expected (due to theft or weather conditions), and of course the exact position of the device.

A review of the existing market has been performed to ensure our product has not already been implemented, as well as ensuring our design explores as many solutions as possible for each of the above features. Buoy's for both commercial and recreational trapping/ fishing that offer GPS functionality are relatively limited, usually accompanied by an excessive price-tag as well as inconvenient size for deployment in large numbers. One design, from FullOceans, features SigFox radio protocol for coverage up to 15km from coastlines [6]. Another solution is Satlink's more robust standalone ILL+ buoy, which features solar power and solely uses GPS communication, rather than both radio and GPS[7]. Other products exist that achieve similar means however Satlink and FullOceans designs were most like the small form-factor and IoT/node-based approach we have in mind.

Radio communication for this application has 3 main contenders: LTE, LoRa, and SigFox. LTE requires tapping into existing mobile networks, and as such is not suitable for situations where traps are being deployed offshore or in remote locations with poor cellular coverage. As such, SigFox and LoRa are left as the main options to implement communication from a main boat and the traps that have been deployed previously. Both protocols are compliant with regulations on public radio protocols within Canada. LoRa and SigFox have similar power consumption, and as the product will be deployed on the ocean, penetration loss is not of great concern. As such, the decision will be based on bandwidth, latency, and device range. LoRa has superior bandwidth, a maximum of 50kbps compared to SigFox's 600bps, and the option of lower latency with a Class-C implementation (although utilizing this feature increases power consumption) [8]. On the other hand, SigFox has significantly higher range of up to 40km, while LoRa can transmit at most 20km. Based on the above, we intend to utilize LoRa. The higher bandwidth alone is a compelling reason to choose it to allow for scalability of the project, and the additional range provided by SigFox is not significant when tracking traps that are known to be in a much smaller general vicinity.

Buoy designs can be either battery powered, solar powered, or a combination of the two. Implementation of solar power significantly increases manufacturing cost but allows for a product with near indefinite lifespan (outside of other complications). Alternatively, battery power is extremely affordable and with appropriate design can have a lifespan of months or years. FullOcean's battery powered implementation is an excellent example of this, with a 3-to-8-year lifespan [6]. To ensure deployment to areas with poor sunlight is possible, a battery solution is essential. Solar can be used to reduce the load on the batteries and extend the lifespan of the product between maintenance. This also adheres to EGBC code focusing on protecting the environment. Affordable and compact solar solutions produce 30-100mA in strong sunlight which will be sufficient to support most off-the-shelf microcontrollers.

GPS, Galileo and GLONASS are the most ubiquitous satellite networks used for global positioning. Of the products we investigated, all featured exclusively GPS implementations despite Galileo having superior accuracy, and GLONASS having better performance at extremely high/low latitudes [9]. Overall, we would like to be able to utilize all three of these protocols in tandem. The industry leader U-Blox provides a wide range of modules

supporting GNSS that can interface with GPS, Galileo, and GLONASS to provide a more reliable location reading. Overall, there is extremely limited competition with U-Blox for the consumer market and we were unable to locate even comparable GNSS modules.

V Team Duties & Project Planning

Roles within the team were assigned as follows, however tasks such as planning and testing are fulfilled by the team.

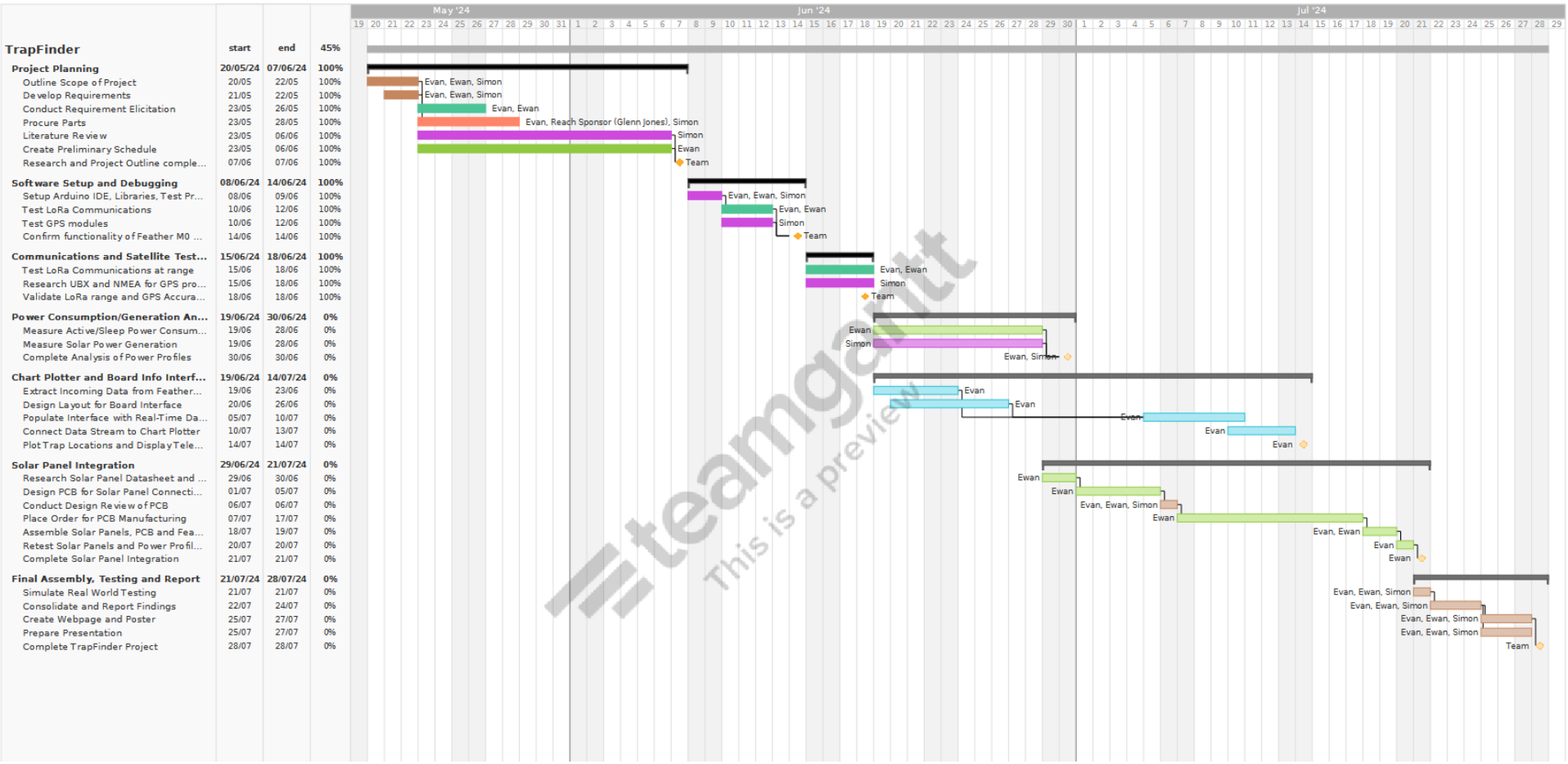
- Evan Lee – Responsible for extracting the data from the boat and displaying GPS location on a map to create a simple User Interface. Designed and printed enclosure and assisted in solar panel integration.

- Ewan Chatham – Responsible completing analysis of power consumption when board is sending/receiving and in sleep mode. Integrating solar panels with 1200mAh LiPo battery and Feather M0 Trap board.

- Simon Pollak – Responsible for GPS signal decoding and formatting, developing code for the ping system between the Boat and Trap devices. Assisted in graphing and visualizing current draw of board when sending and receiving.

A more detailed breakdown of individual deliverables, dependencies and contributions can be seen below in the form of a Gantt Chart.

Challenges such as difficulty in reliability measuring current draw from batteries and solar panels were faced. For example, solar panels that were rated for 5V 2.5 W 500mAh output was only outputting 100mAh even when connected in parallel on a clear sunny day (each panel individually outputted ~100mAh, but when connected in parallel also only outputted 100mAh where it was expected to output ~200mAh). Troubleshooting and testing various challenges took time away from implementing further features. Due to this extra time constraint the Mesh Network was not implemented in favour of further testing.


May '24
Jun '24
Jul '24
TrapFinder
start
end
45%
Project Planning 20/05/24 07/06/24 100%
Outline Scope of Project 20/05 22/05 100%
Develop Requirements 21/05 22/05 100%
Conduct Requirement Elicitation 23/05 26/05 100%
Procure Parts 23/05 28/05 100%
Literature Review 23/05 06/06 100%
Create Preliminary Schedule 23/05 06/06 100%
Research and Project Outline comple... 07/06 07/06 100%
Software Setup and Debugging 08/06/24 14/06/24 100%
Setup Arduino IDE, Libraries, Test Pr... 08/06 09/06 100%
Test LoRa Communications 10/06 12/06 100%
Test GPS modules 10/06 12/06 100%
Confirm functionality of Feather M0 ... 14/06 14/06 100%
Communications and Satellite Test... 15/06/24 18/06/24 100%
Test LoRa Communications at range 15/06 18/06 100%
Research UBX and NMEA for GPS pro... 15/06 18/06 100%
Validate LoRa range and GPS Accura... 18/06 18/06 100%
Power Consumption/Generation An... 19/06/24 30/06/24 0%
Measure Active/Sleep Power Consum... 19/06 28/06 0%
Measure Solar Power Generation 19/06 28/06 0%
Complete Analysis of Power Profiles 30/06 30/06 0%
Chart Plotter and Board Info Interf... 19/06/24 14/07/24 0%
Extract Incoming Data from Feather... 19/06 23/06 0%
Design Layout for Board Interface 20/06 26/06 0%
Populate Interface with Real-Time Da... 05/07 10/07 0%
Connect Data Stream to Chart Plotter 10/07 13/07 0%
Plot Trap Locations and Display Tele... 14/07 14/07 0%
Solar Panel Integration 29/06/24 21/07/24 0%
Research Solar Panel Datasheet and ... 29/06 30/06 0%
Design PCB for Solar Panel Connecti... 01/07 05/07 0%
Conduct Design Review of PCB 06/07 06/07 0%
Place Order for PCB Manufacturing 07/07 17/07 0%
Assemble Solar Panels, PCB and Fea... 18/07 19/07 0%
Retest Solar Panels and Power Profil... 20/07 20/07 0%
Complete Solar Panel Integration 21/07 21/07 0%
Final Assembly, Testing and Report 21/07/24 28/07/24 0%
Simulate Real World Testing 21/07 21/07 0%
Consolidate and Report Findings 22/07 24/07 0%
Create Webpage and Poster 25/07 27/07 0%
Prepare Presentation 25/07 27/07 0%
Complete TrapFinder Project 28/07 28/07 0%
Evan, Ewan, Simon
Evan, Ewan, Simon
Evan, Ewan
Evan, Reach Sponsor (Glenn Jones), Simon
Simon
Ewan
Team
Evan, Ewan, Simon
Evan, Ewan
Simon
Team
Evan, Ewan
Simon
Team
Ewan
Simon
Evan, Simon
Evan
Evan
Evan
Evan
Ewan
Ewan
Evan, Ewan, Simon
Ewan
Evan, Ewan
Evan
Ewan
Evan, Ewan, Simon
Evan, Ewan, Simon
Evan, Ewan, Simon
Evan, Ewan, Simon
Team
teamgantt
This is a preview

VI Design Methodology & Analysis

Design Methodology

Our design methodology for the Trap Finder: Real-Time Crab Trap Tracking System (CTTS) was structured around iterative development and comprehensive testing. Here's a detailed breakdown:

1. Designing:
    - Overall System: We developed a detailed system flowchart for how the Boat and Trap devices communicate.
    - Circuit Design: We specified the components needed, such as solar panels, batteries, LoRa modules, GPS modules, and microcontrollers, and designed the circuit using a breadboard.
    - Enclosure Design: Using SolidWorks, we designed waterproof enclosures for the Trap devices to ensure durability in marine environments.
2. Prototyping:
    - Hardware Prototypes: We built initial prototypes for the Boat and Trap devices. This included 3D printing the enclosure and integrating the circuits with the enclosure.
    - Software Development: We developed the software for the Trap and Boat devices, focusing on reliable communication and GPS tracking based on the flowchart in the designing phase. Additionally, we created a website to display real-time data as a prototype for the boat's chart plotter.
3. Testing and Iteration:
    - Component Testing: Each component was tested individually to ensure proper functionality. For instance, we tested the power consumption of the solar panels and batteries under different conditions.
    - System Integration: We integrated the components into a working system and conducted integration tests to ensure they worked together seamlessly.
    - Website Testing: The website was tested for real-time data display and user interface responsiveness.
4. Optimization:
    - Based on the test results, we optimized the design for better performance and reliability. This included tweaking the ping algorithm and making adjustments to the website display.

Analysis

The analysis involved several key areas:

1. Power Consumption:
    - We calculated the power consumption of the Trap devices under different scenarios (e.g., idle, transmitting, receiving) using a digital multimeter

(DMM). This helped us optimize the solar charging circuits and ensure the Trap device could remain powered indefinitely.

2. Communication Range:
    - We used simulation tools to analyze the communication range of the LoRa modules. This ensured reliable communication between the Trap and Boat devices over the required distances. We tested ranges of 1.5 km and 3.5 km.
3. GPS Accuracy:
    - We conducted GPS accuracy tests. This analysis was critical for ensuring the accuracy of the trap locations. One team member moved around with the GPS module, and we observed and analyzed how fast and accurately the data was updated.
4. System Performance:
    - We analyzed the overall system performance. This helped us identify and fix potential issues; for example, we found a bug in our ping algorithm during the first overall system performance test and fixed the code.
5. Compliance with Standards:
    - Our design and testing followed relevant standards and regulations in the field. We made sure to distinguish between facts, assumptions, and opinions, and clearly stated any assumptions made.

VII Design & Prototype

Final Design

The final design of the Trap Finder system includes the following components:

1. Trap Units:
    - The flowchart below shows the main algorithm of the Trap device's code: The Trap device actively scan for a ping signal and, upon detecting one, validates its Device ID and sends location data periodically until acknowledged. It then enters a sleep mode to conserve power.

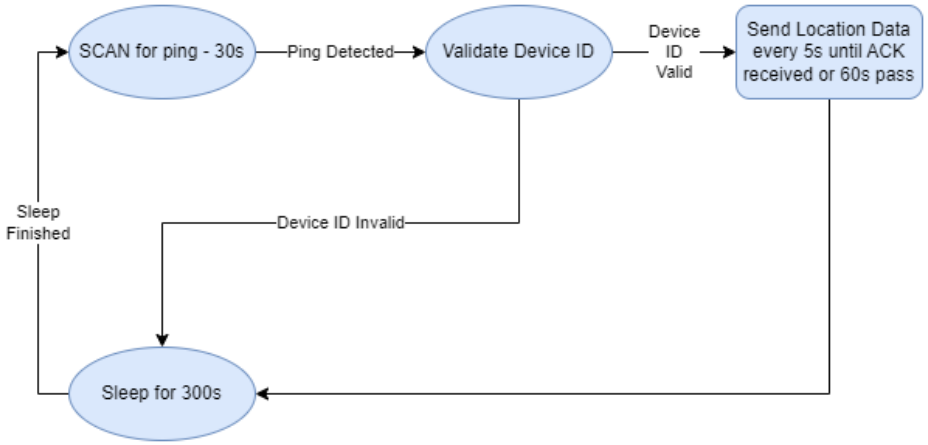


Figure 1: Trap Software Flowchart

- Solar-Powered Trap Device: These are attached to crab trap buoys and are powered by integrated solar charging circuits to ensure continuous operation.
- Components: Each Trap device includes LoRa modules for communication, GPS modules to track trap locations, and microcontrollers to manage the system.
- Enclosure Design: The Trap devices are housed in waterproof enclosures designed using SolidWorks and 3D printed to ensure durability in marine environments.

Figure 2: Enclosure design in SolidWorks

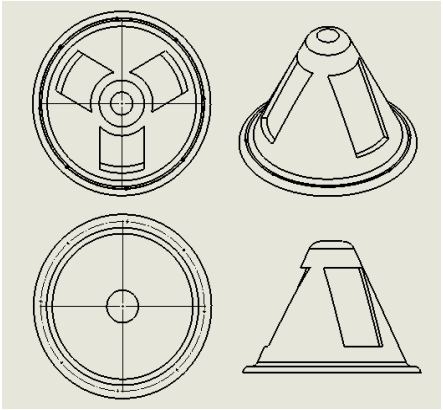


Figure 3: Enclosure drawing in SolidWorks

2. Boat Unit:
   - The flowchart below shows how the Boat communicates with Trap devices: The Boat device periodically pings Trap devices within a specified range and sends an acknowledgment to any responding Trap device. It keeps pinging until a response is received, ensuring reliable communication and tracking of trap units.

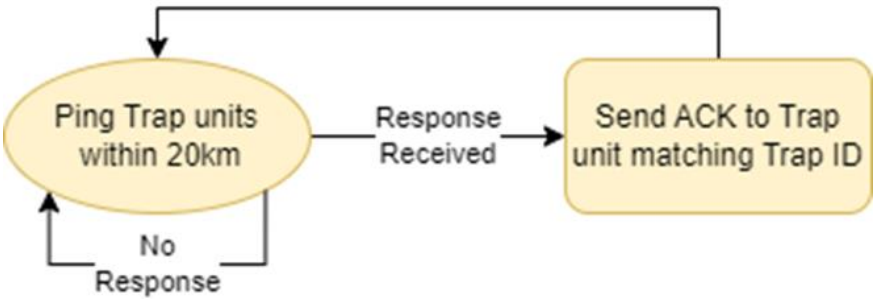


Figure 4: boat Software Flowchart

- Boat Device: This device, onboard a boat, is equipped with a LoRa module to communicate with the Trap devices.
- User Interface: The boat features a user interface developed using a web server. This interface displays Trap device locations and battery status on a laptop initially, with the potential for future integration into boat chart plotters.
    - Real-Time Data Display: A website was developed to display real-time data from the Trap device, including a map for visualizing trap locations and sections for displaying battery status and timestamps.
    - Refresh Mechanism: The website refreshes every 10 seconds to ensure the data displayed is up-to-date.

Addressing Objectives

- Solar-Powered: We successfully designed and integrated solar charging circuits, ensuring the Trap devices remain powered during operation.
- Reliable Communication: The LoRa technology provided robust communication between the Trap and Boat devices, with a tested range sufficient for our needs.
- GPS Functionality: The GPS modules accurately tracked the trap locations, which were displayed on the user interface (website).
- User Interface: The website allowed users to monitor trap locations and battery status in real-time.

Prototype Development

1. Hardware Prototypes:
    - We have built initial prototypes for both the Trap and Boat devices. The Trap device prototypes include the selected components: solar panels, batteries, LoRa modules, GPS modules, and microcontrollers. One Trap device is embedded in the enclosure, as shown in Figure 5. The other Trap prototype module is also shown in Figure 5. The Trap device has the ID of 238, and the other one has the ID of 239.

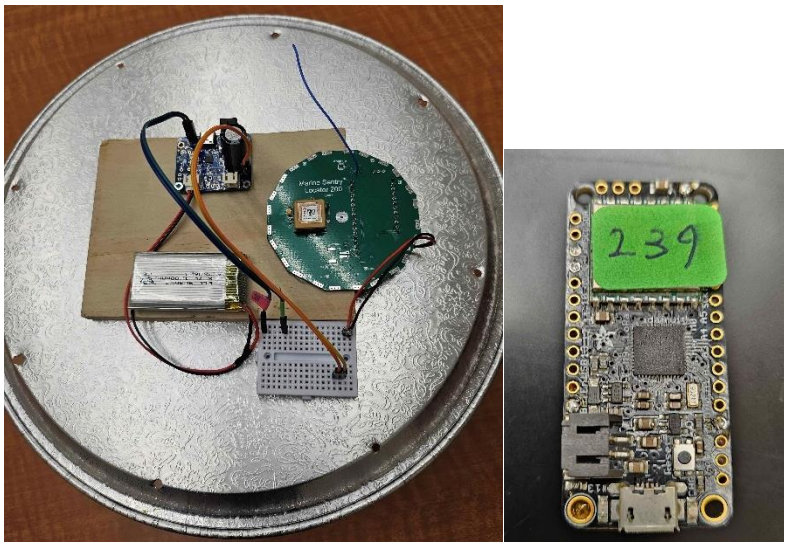


Figure 5: Trap prototypes ID238 (left) and ID239(right)

Figure 6: buoy enclosure prototype

- The boat controller prototype was equipped with a LoRa module and connected to a laptop to serve as the initial user interface.

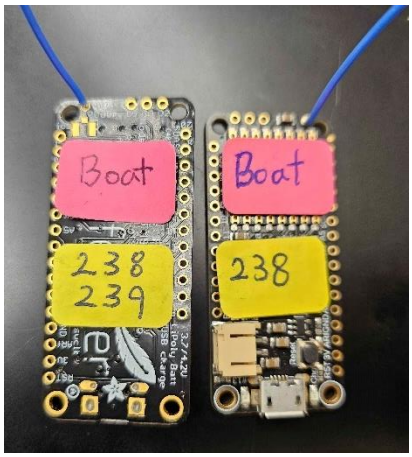


Figure 7: Boat prototypes. left device can communicate with 2 Trap devices while the device on the right can only communicate with one

2. Software Development:
    - The software for the Trap devices focused on reliable communication using LoRa technology and accurate GPS tracking.
    - For the boat, we developed a web server-based interface to display real-time trap data.
    - A website was also created to display trap locations on a map and provide battery status updates.

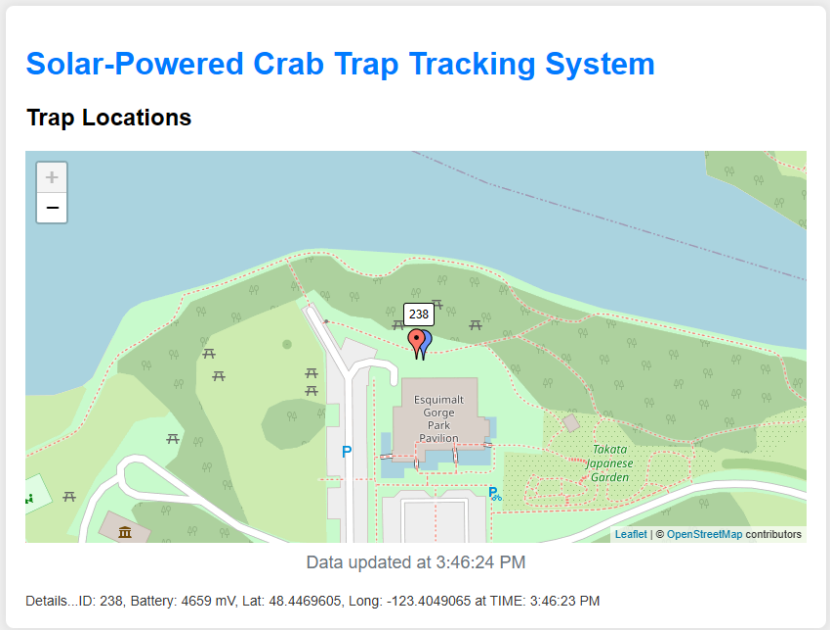


Figure 8: Website to display trap locations, batteries, IDs.

3. Testing and Optimization:
    - Each component was tested individually for proper functionality. For example, we tested the power consumption of the solar panels and batteries under different conditions.

- System integration tests ensured that all components worked together seamlessly.
- The website was tested for real-time data display and user interface responsiveness.
- Based on test results, we optimized the design for better performance and reliability, including tweaking the ping algorithm and making adjustments to the website display.

The final prototype met most of our initial objectives, providing a functional and reliable system for tracking crab traps in real-time using solar-powered Trap devices and a Boat device with an intuitive user interface.

VIII Testing & Validation

Testing

1. Power Consumption Testing:
    - Conditions: We calculated the current to the microcontroller under various scenarios using a multimeter: when the microcontroller is idle, transmitting LoRa data, receiving LoRa data, and receiving GPS data. Additionally, we measured the current generated by the solar panel.

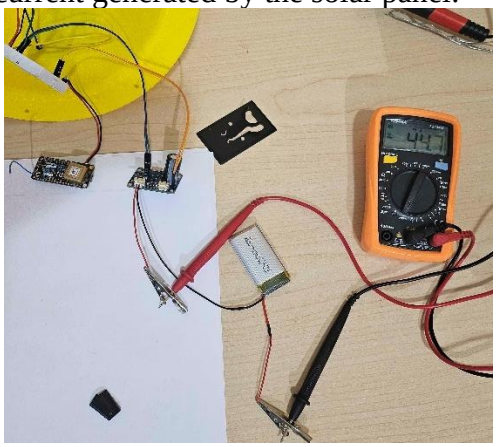


Figure 9: Current measurement setup

The above conditions were analyzed based on the ping-ACK system devised for this project, with 3 different expected power consumption scenarios plotted in Figure 10. These showed a worst case (where the device was receiving a location request but failed to ACK for the entire timeout duration), the expected operation (one location sent, ACK received), and idle operation (no ping received).

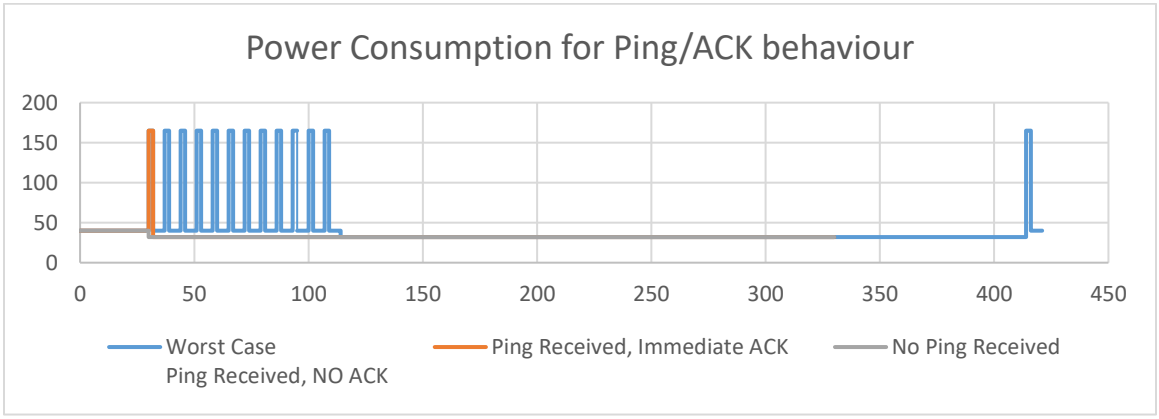


Figure 10: Power Consumption for Ping/ACK behaviour

- Outcome: The panel was able to provide 100mA charging current to the battery with our configuration and charger. An analysis of the power consumption without solar panels showed the results in Table 1. Solar power was analyzed and a charging current of 100mA continuous was found

in both sunny and heavily clouded weather, limited by the charge controller and microcontroller.

Table 1: Power consumption of system

Ping/ACK Case	Average Current Draw(mA)	Battery Life w/o Solar(days)
Worst Case Ping Received, NO ACK	41.5	1.2
Ping Received, Immediate ACK	33.52	1.49
No Ping Received	32.72	1.52

- Based on the average current draw (even in worst case) being significantly under the 100mA solar power generation, and there being negligible passive discharge from the batteries for the short duration they would be deployed in typical trapping scenarios, it was surmised that the system would be self sufficient. This operated on the assumption that the housing had appropriate solar panel angle for the region and that crab trapping would be occurring during regular seasons where daylight was not at a minimum (May-November).

2. LoRa Communication Range Testing:
   - Conditions: We tested the communication range of the LoRa modules at distances of 1.7 km and 3.5 km.
   - Outcome: Both distances maintained reliable communication, verifying the long-distance performance of the LoRa technology.

Figure 11: Map view of communication test locations. Rx is the boat, while Tx(1) and Tx(2) are the Trap devices.

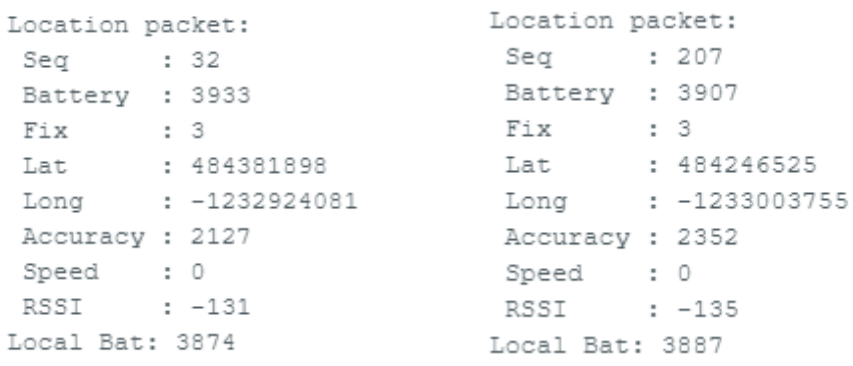


Figure 12: LoRa distance testing results 1.7km(left) 3.5km(right)

3. GPS Accuracy Testing:
    - Conditions: We conducted two tests:
        1. Accuracy Test: We verified the GPS-reported locations at a known latitude and longitude (Engineering Lab Wing main entrance) to check accuracy.
        2. Update Speed Test: A team member moved around with the GPS module, and we observed the update speed of the GPS data.
    - Outcome: The accuracy test showed that the GPS module was very accurate. The update speed test revealed a 40-second delay in data updates.

4. Website Testing:
    - Conditions: We tested the website's ability to display the boat's received data in real-time.
    - Outcome: Initially, the website did not work because the serial port was accessed by another user. After closing the Arduino Serial Monitor, the serial data was successfully displayed on the website.

5. System Integration Testing:
    - Conditions: We tested the overall functionality of the fully assembled system. There are two Trap devices (ID 238 and 239) and two boats: one boat communicates only with Trap device ID 238, and the other communicates with both Trap devices, IDs 238 and 239. According to the flowcharts in the design, the first Boat device should display only the location of Trap 238, while the other Boat can display the locations of both Traps 238 and 239.
    - Outcome: Initially, there were issues with the ping algorithm, but after troubleshooting, the system worked as expected.

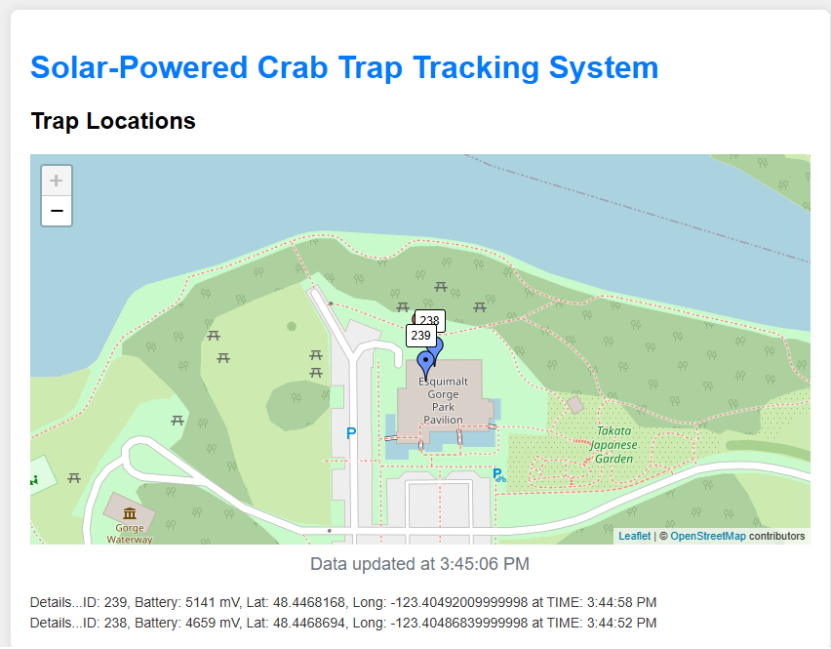


Figure 13: Display for boat that can communicate with 2 IDs. Red pin indicates Boat device. Blue indicates Trap device..

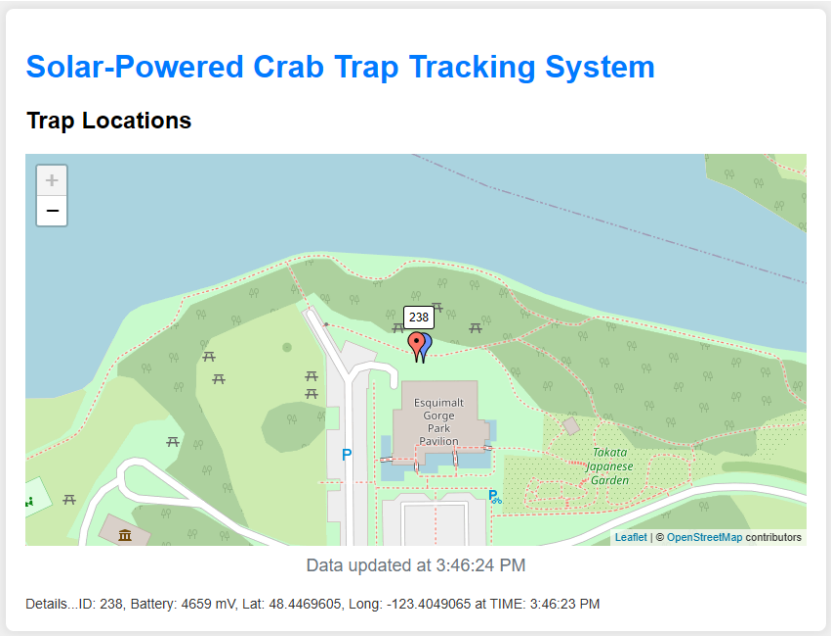


Figure 14: Display for boat that can communicate with 1 ID at the same place.

6. Enclosure Testing:
    - Conditions: We tested the buoy's ability to float in water and its waterproof capabilities by bringing it to Cadboro Bay.
    - Outcome: The buoy floated successfully, and the interior remained dry. Behaviour in tumultuous weather is uncertain so further steps should be taken to ensure waterproofing and the ability to be deployed for extended periods.

Validation

The validation process involved running multiple test sequences to demonstrate the system's ability to meet the desired objectives:

1. Power Efficiency: The power consumption tests validated that the solar charging circuits were efficient, keeping the Trap devices powered indefinitely under typical conditions.
2. Communication Reliability: The range tests confirmed that the LoRa modules provided robust communication over the required distances, both in open water and semi-obstructed environments.
3. GPS Accuracy: The GPS tests validated the accuracy of the location data, ensuring that the Trap devices could be reliably tracked.
4. System Integration: The integration tests demonstrated that the components worked together seamlessly, providing a functional and reliable system.
5. Website Functionality: The website tests confirmed that the user interface was intuitive, and that the real-time data display was accurate and up to date.
6. Durability: The enclosure tests validated that the Trap devices could withstand marine conditions without compromising functionality.

IX Cost Analysis

- Direct Costs
  1. Components and Materials:
     - Solar Panels: $18
     - Solar Panel Charging Circuit: $31.55
     - Batteries: $39
     - GPS Modules: $31.05
     - Microcontrollers: $162.30
  2. Prototyping:
     - 3D Printing Services: $48
  3. Poster Printing:
     - Poster Printing: $52

  Total Direct Costs (excluding sponsored components): $333.90
- Indirect Costs
  1. Labor Costs:
     - Team Hours Worked: 80 hours
       - Average Hourly Rate based on previous co-op salary: $25/hour
       - Total Labor Cost: $2,000

  Total Indirect Costs: $2,000

  Total Project Cost: $2,333.90
- Funding
  1. Sponsorship:
     - Glenn Jones from Reach Technologies: Provided batteries, GPS modules, and microcontrollers (valued at $232.35 for cost calculation)
  2. University Funding: $750

  Total Funding: $982.35
- **Net Cost:** $2,333.90 - $982.35 = $1,351.55

Tentative Pricing & Return on Investment (ROI)

- Tentative Pricing**:**
  - Initial Unit Production Cost: $333.90
  - Selling Price per Unit: $500
- Revenue Calculation assuming Expected Sales in First Year: 50 units**:**
- Total Revenue: 50 units * $500/unit = $25,000
- Profit Calculation:
  - Total Profit: $25,000 - (50 units * $333.90/unit) = $8,305
- ROI Calculation:
  - ROI: (Total Profit / Net Cost) * 100 = ($8,305 / $1,351.55) * 100 = 614%

Cost Reduction Suggestions

1. Bulk Purchasing:
   - Purchasing components in bulk can reduce the cost per unit. For example, buying batteries and other electronic components in larger quantities can lower the overall expense.
2. In-House Manufacturing:
   - Setting up an in-house facility for 3D printing and PCB manufacturing can significantly cut down prototyping and production costs.

By implementing these cost-saving measures, we can further improve the profitability and sustainability of the Trap Finder system.

X Conclusion & Recommendations

Overall, the final prototype met most of our initial objectives. While some features, such as the water quality monitoring system, remain in the future scope, the current system provides a functional and reliable solution for tracking crab traps in real-time using solar-powered Trap devices and a Boat device with an intuitive user interface. Our design methodology and testing process adhered to the EGBC Code of Ethics, ensuring competence, sustainability, and protection of the public and environment throughout the project.

During this process, a main constraint was the limitations of electrical systems deployed in marine environments. While testing was performed to ensure functionality in temperate conditions, rigorous testing was omitted for deployment in violent weather conditions. Aside from this potential shortcoming, the project could be further developed with features discussed previously in this report.

Predominantly, these consist of a) water quality monitoring system to ensure crab trap conditions are ideal, b) cut-line monitoring to better combat ghost gear and theft, c) a mesh network system to expand the tracking range of the Trap device, and d) OpenCPN chart-plotting integration. These items were mainly omitted from the final implementation due to cost constraints, particularly the chart plotting component. Another constraint was the research required to design a water quality monitoring system that could be completely submerged while still integrating with the buoy design.

References

[1] NOAA Marine Debris Program, "Ghosts of Fishing Past," NOAA Marine Debris Blog. Available:https://blog.marinedebris.noaa.gov/ghosts-fishing-past#:~:text=The%20term%20%E2%80%9Cghost%20gear%E2%80%9D%20refers,as%20a%20hazard%20to%20navigation.. [Accessed: 17-Jun-2024].

[2] Engineers and Geoscientists British Columbia (EGBC), "Code of Ethics," EGBC. Available:https://www.egbc.ca/complaints-discipline/code-of-ethics/code-of-ethics. [Accessed: 20-Jun-2024].

[3] ISED Canada, "Canadian Table of Frequency Allocations (2022)," Government of Canada, Aug. 10, 2022. [Online]. Available: https://ised-isde.canada.ca/site/spectrum-management-telecommunications/en/learn-more/key-documents/consultations/canadian-table-frequency-allocations-sf10759. [Accessed: Aug. 3, 2024].

[4] Greenstream Publishing Limited, "Solar Angle Calculator," Solar Electricity Handbook. [Online]. Available: http://www.solarelectricityhandbook.com/solar-angle-calculator.html. [Accessed: Aug. 3, 2024].

[5] Marine Stewardship Council, "Pots and Traps," MSC, [Online]. Available: https://www.msc.org/what-we-are-doing/our-approach/fishing-methods-and-gear-types/pots-and-traps. [Accessed: 01-Jun-2024].

[6] "FullOceans Tracker," *FullOceans*, [Online]. Available: https://www.fulloceans.com/en/home/112-fulloceans-tracker.html. [Accessed: June 22, 2024].

[7] "Satlink ILL+ Buoy," *ITPLAN*, [Online]. Available: https://itplan.co/en/products/satlink-elb3010-ill-buoy/. [Accessed: June 22, 2024].

[8] I. Khan, "Suitability of LoRa, Sigfox and NB-IoT for Different Internet-of-Things Applications," *CORE*, [Online]. Available: https://core.ac.uk/reader/280341760. [Accessed: 05-Jun-2024].

[9] "GPS vs GLONASS: Which is Best for Tracking Applications?," *Symmetry Electronics*, [Online]. Available: https://www.symmetryelectronics.com/blog/gps-vs-glonass-which-is-best-for-tracking-applications/. [Accessed: 01-Jun-2024].

[10] "SAM-M8Q Data Sheet," u-blox, Feb. 2019. [Online]. Available: https://content.u-blox.com/sites/default/files/SAM-M8Q_DataSheet_%28UBX-16012619%29.pdf. [Accessed: Jun. 10, 2024].

Appendix A – ECE499 Project Website

The ECE 499 website component for our project can be found at the following github pages link: https://shimashimo.github.io/ECE499/

Appendix B – EGBC Code of Ethics

A registrant must adhere to the following Code of Ethics:
Registrants must act at all times with fairness, courtesy and good faith toward all persons with whom the registrant has professional dealings, and in accordance with the public interest. Registrants must uphold the values of truth, honesty, and trustworthiness and safeguard human life and welfare and the environment. In keeping with these basic tenets, registrants must:

1. Hold paramount the safety, health, and welfare of the public, including the protection of the environment and the promotion of health and safety in the workplace;
2. Practice only in those fields where training and ability make the registrant professionally competent;
3. Have regard for the common law and any applicable enactments, federal enactments, or enactments of another province;
4. Have regard for applicable standards, policies, plans, and practices established by the government or Engineers and Geoscientists BC;
5. Maintain competence in relevant specializations, including advances in the regulated practice and relevant science;
6. Provide accurate information in respect of qualifications and experience;
7. Provide professional opinions that distinguish between facts, assumptions, and opinions;
8. Avoid situations and circumstances in which there is a real or perceived conflict of interest and ensure conflicts of interest, including perceived conflicts of interest, are properly disclosed and necessary measures are taken so a conflict of interest does not bias decisions or recommendations;
9. Report to Engineers and Geoscientists BC and, if applicable, any other appropriate authority, if the registrant, on reasonable and probable grounds, believes that:
    a. The continued practice of a regulated practice by another registrant or other person, including firms and employers, might pose a risk of significant harm to the environment or to the health or safety of the public or a group of people; or
    b. A registrant or another individual has made decisions or engaged in practices which may be illegal or unethical;
10. Present clearly to employers and clients the possible consequences if professional decisions or judgments are overruled or disregarded;
11. Clearly identify each registrant who has contributed professional work, including recommendations, reports, statements, or opinions;
12. Undertake work and documentation with due diligence and in accordance with any guidance developed to standardize professional documentation for the applicable profession; and

13. Conduct themselves with fairness, courtesy, and good faith towards clients, colleagues, and others, give credit where it is due and accept, as well as give, honest and fair professional comment.

Appendix C – Code Listing

RadioRxFinal.ino

```
#include <SPI.h>
#include <RH_RF95.h>

// I/O Pin definitions for the Feather M0 LoRa Board
#define RFM95_CS 8
#define RFM95_RST 4
#define RFM95_INT 3

// Set the LoRa radio to 915.0 MHz, must match Rx radio's freq!
#define RF95_FREQ 915.0

// Singleton instance of the radio driver
RH_RF95 rf95(RFM95_CS, RFM95_INT);

// Blink LED on receipt or Error
#define LED 13

// Measure battery voltage
#define VBATPIN A7

uint8_t buf[RH_RF95_MAX_MESSAGE_LEN];
uint8_t len = sizeof(buf);
uint8_t deviceList[2];

void error( uint32_t flash)
{
  uint32_t count;

  if ( flash > 0 )
  {
    for ( count=0; count < flash; count++ )
    {
      digitalWrite(LED, HIGH);
      delay(200);
      digitalWrite(LED, LOW);
      delay(200);
    }
  }
  else
  {
    while (1)
    {
      digitalWrite(LED, HIGH);
      delay(50);
      digitalWrite(LED, LOW);
      delay(50);
    }
  }
  return;
}

//Send deviceList packet
void _ping()
{
```

```
  deviceList[0] = 238;   // 0xEE through 0xFE reserved for ID, example with 1 node.
  deviceList[1] = 240;
  //Serial.println("Ping Outgoing");
  rf95.send((uint8_t *)deviceList, 2);
  rf95.waitPacketSent();
  digitalWrite(LED, HIGH);
}

void setup()
{
  pinMode(LED, OUTPUT);
  pinMode(RFM95_RST, OUTPUT);
  digitalWrite(RFM95_RST, HIGH);

  Serial.begin(115200);
//  while (!Serial) {
//    delay(1);
//  }
  delay(100);

  // manual reset
  digitalWrite(RFM95_RST, LOW);
  delay(10);
  digitalWrite(RFM95_RST, HIGH);
  delay(10);

  while (!rf95.init()) {
    //Serial.println("LoRa radio init failed");
    //Serial.println("Uncomment '#define SERIAL_DEBUG' in RH_RF95.cpp for detailed debug info");
    while (1);
  }
  //Serial.println("LoRa radio init OK!");

  if (!rf95.setFrequency(RF95_FREQ)) {
    //Serial.println("setFrequency failed");
    while (1);
  }
  //Serial.print("Set Freq to: "); Serial.println(RF95_FREQ);

  if ( !rf95.setModemConfig(RH_RF95::Bw125Cr48Sf4096)){
    //Serial.println("setModemConfig failed");
    while (1);
  }

//  rf95.setTxPower(23, false);
  // Radio after config: 915.0 MHz, 23dBm, Bw = 125 kHz, Cr = 4/5, Sf = 4096 chips/symbol, CRC on
  digitalWrite(LED, LOW);
}

void loop()
{
  //ping, then wait 10s for reply
  _ping();

  if (rf95.waitAvailableTimeout( 10000 ))
  {
    if (rf95.recv(buf, &len))
    {
      digitalWrite(LED, HIGH);

      // Send a reply, ACK code is 0xFF
      uint8_t data[2] = {0xFF, buf[20]};
      rf95.send( (uint8_t *)data, 2 );
      rf95.waitPacketSent();

      if ( len == 21 )
      {
    //      RH_RF95::printBuffer("Received: ", buf, len);
        uint16_t count = buf[0];
```

```
        uint16_t vBat     = ( buf[2] << 8 ) | buf[1];
        uint8_t fix       = buf[3];
        int32_t longitude = ( buf[7] << 24 ) | ( buf[6] << 16 ) | ( buf[5] << 8 ) | buf[4];
        int32_t latitude  = ( buf[11] << 24 ) | ( buf[10] << 16 ) | ( buf[9] << 8 ) | buf[8];
        uint32_t speed    = ( buf[15] << 24 ) | ( buf[14] << 16 ) | ( buf[13] << 8 ) | buf[12];
        uint32_t accuracy = ( buf[19] << 24 ) | ( buf[18] << 16 ) | ( buf[17] << 8 ) | buf[16];
        uint8_t id        = buf[20];

            Serial.print(id, DEC);
            Serial.print(",");
            Serial.print(vBat, DEC);
            Serial.print(",");
            Serial.print(latitude, DEC);
            Serial.print(",");
            Serial.println(longitude, DEC);
          }
          digitalWrite(LED, LOW);
        } else {
          error(2);
        }
      }
      else {
        error(3);
      }

}
```

RadioTxFinal.ino

```
#include "ArduinoLowPower.h"
#include <SPI.h>
#include <RH_RF95.h>
#include "MS_GNSS.h"
#include "MS_GNSSmsg.h"

// I/O Pin definitions for the Feather M0 Board
#define RFM95_CS 8
#define RFM95_RST 4
#define RFM95_INT 3

// Change to 434.0 or other frequency, must match RX's freq!
#define RF95_FREQ 915.0

// Single instance of the radio driver
RH_RF95 rf95(RFM95_CS, RFM95_INT);

// Blink LED on Tx
#define LED 13

// Measure battery voltage
#define VBATPIN A7

// GNSS Serial Interface
#define PIN_SERIAL1_RX       (0ul)
#define PIN_SERIAL1_TX       (1ul)
#define PAD_SERIAL1_TX       (UART_TX_PAD_2)
#define PAD_SERIAL1_RX       (SERCOM_RX_PAD_3)

typedef enum { SCAN,            // Scan for boat for
               SENDLOC,         // Send Location data every 10s until an ACK is received from the
boat.
               SLEEP            // Sleep for 300s after an ACK is received or after scanning for
the boat's signal.
} DeviceMode_e;

// Global variables
//
```

```
GNSS         gnss( Serial1, 9600 );  // This is SAM default - can go faster but SAM must be
configured for that!
bool         Beacon;          // True - beacon Tx only (no GNSS data)
int8_t       packetId = 0;    // packet counter, we increment per Tx
uint8_t      deviceId = 0xEF;
uint8_t      locationPacket[21];
uint8_t      beaconPacket[1];
uint8_t      buf[32];
uint8_t      len = sizeof(buf);
GNSSmsg      gnssMsg;
UBX          ubxMsg;
DeviceMode_e mode = SCAN;

void error(uint32_t flash)
{
  uint32_t count;
  if ( flash > 0 )
  {
    for ( count=0; count < flash; count++ )
    {
      digitalWrite(LED, HIGH);
      delay(200);
      digitalWrite(LED, LOW);
      delay(200);
    }
  }
  else
  {
    while (1)
    {
      digitalWrite(LED, HIGH);
      delay(50);
      digitalWrite(LED, LOW);
      delay(50);
    }
  }
  return;
}

void SERCOM1_Handler()
{
  Serial1.IrqHandler();
}

bool _scan() //scan for a ping and return true if ping has a matching device ID;
{
  Serial.println("Scanning");
  if(rf95.waitAvailableTimeout(20000))
  {
    if(rf95.recv(buf, &len))
    {
      Serial.print("Got Ping:");
      Serial.println(len, DEC);
      for(int i = 0; i < sizeof(buf)/8; i++)
      {
        Serial.println(buf[i]);
        if(buf[i] == deviceId)
        {
          Serial.println("Got Match");
          rf95.sleep();
          return true;
        }
      }
    }
  }
  return false;
}

//Sends location every 10s until an ACK is received
```

```
bool _sendLocation( uint8_t* pvtMessage, uint16_t vBat )
{
  bool gotAck = false;
  packetId++;
  locationPacket[0] = packetId;    // Will need more bits for this VesselId + DeviceId =
  locationPacket[1] = vBat & 0xFF;
  locationPacket[2] = ( vBat >> 8 ) & 0xFF;
  locationPacket[3] = pvtMessage[20];   // Fix (+Moving+Acc)
  locationPacket[4] = pvtMessage[24];   // Longitude
  locationPacket[5] = pvtMessage[25];
  locationPacket[6] = pvtMessage[26];
  locationPacket[7] = pvtMessage[27];
  locationPacket[8] = pvtMessage[28];   // Latitude
  locationPacket[9] = pvtMessage[29];
  locationPacket[10] = pvtMessage[30];
  locationPacket[11] = pvtMessage[31];
  locationPacket[12] = pvtMessage[60];   // Speed (One byte - knots)
  locationPacket[13] = pvtMessage[61];
  locationPacket[14] = pvtMessage[62];
  locationPacket[15] = pvtMessage[63];
  locationPacket[16] = pvtMessage[40];   // Position Accuracy (one byte = metres)
  locationPacket[17] = pvtMessage[41];
  locationPacket[18] = pvtMessage[42];
  locationPacket[19] = pvtMessage[43];
  locationPacket[20] = deviceId;         //deviceID

  Serial.println("Send location");

  rf95.send((uint8_t *)locationPacket, 21);
  rf95.waitPacketSent();
  digitalWrite(LED, HIGH);

  // Now wait for a reply
  //
  if (rf95.waitAvailableTimeout( 10000 ))
  {
      Serial.print( "wait" );
      //Serial.println( len, DEC );
    // Should be a reply message for us now
    if ( rf95.recv( buf, &len ))
    {
      Serial.println( "rcv " );
      if( buf[0] == 0xFF && buf[1] == deviceId)
      {
        Serial.print( "Got Ack: RSSI: " );
        Serial.println( rf95.lastRssi(), DEC );
        gotAck = true;
      }

    }
    else
    {
      error( 2 );
    }
  }
  else
  {
      error( 3 );
  }
  digitalWrite(LED, LOW);   // Turn off LED before Tx to save battery
  rf95.sleep();
  return gotAck;
}

void setup()
{
  pinMode(LED, OUTPUT);
  digitalWrite(LED, HIGH);
  pinMode(RFM95_RST, OUTPUT);
```

```
  digitalWrite(RFM95_RST, HIGH);

// Uncomment this function if you wish to attach function dummy when RTC wakes up the chip
// LowPower.attachInterruptWakeup(RTC_ALARM_WAKEUP, dummy, CHANGE);

  Serial.begin(115200);

  Serial.println("LoRa TX Test!");

  Uart Serial1( &sercom0, PIN_SERIAL1_RX, PIN_SERIAL1_TX, PAD_SERIAL1_RX, PAD_SERIAL1_TX ) ;

  // starting communication with the GNSS receiver
  gnss.init();
  gnss.configure();   // Required after a power cycle
//  gnss.sleep();       // Sleep the GNSS Receiver

  // manual reset
  digitalWrite(RFM95_RST, LOW);
  delay(10);
  digitalWrite(RFM95_RST, HIGH);
  delay(10);

  if (!rf95.init())
    error( 0 );         // Fatal Error

  if (!rf95.setFrequency(RF95_FREQ))
    error( 0 );         // Fatal Error

//  if (!rf95.setModemConfig(RH_RF95::Bw125Cr48Sf1024))
  if (!rf95.setModemConfig(RH_RF95::Bw125Cr48Sf4096))
    error( 0 );         // Fatal Error

  // TODO Set Tx power based on RSSI
  //
  rf95.setTxPower( 23, false );

  memset( locationPacket, 0, 21 );
  mode = SCAN;      // Start in Beacon Mode - No GNSS, Sleep for 60 sends if No ACK received,
start GNSS if ACK received (may take 30 seconds for an initial fix).
  digitalWrite(LED, LOW);
}

void loop()
{
  if ( mode == SCAN) //scan and
  {
    if(_scan())
    {
      mode = SENDLOC;
    }
    else
    {
      Serial.println("NO ACK");
      delay(10000);
    }
  }
  else if ( mode == SENDLOC)
  {
    Serial.println("Sending Location");
    uint32_t vBat = analogRead(VBATPIN);
    vBat *= 66;    // we divided by 2, then multiply by 3.3V, our reference voltage
    vBat /= 10;    // convert to mV

    // Wait for a GNSS message (either UBX or NMEA) and display the packet info
    // The packet Tx rate is set by the GNSS NavPvt update rate (10 seconds)
    //  FUTURE: Bacl off the update rate when we have no reply from a receiver)
    //
    Serial.println("GNSS: ");
```

```
    switch ( gnss.readMessage( gnssMsg, 15000 )) {
      case GNSSmsg::GNSS_MSG_NONE:
        Serial.println("no msg");
        error( 4 );
        break;
      case GNSSmsg::GNSS_MSG_UBX:
        Serial.println("UBX");
        ubxMsg.setMessage( gnssMsg.getMessage() );
        if (( ubxMsg.getMessageClass() == UBX::UBX_NAV ) && ( ubxMsg.getMessageId() == UBX_NAV_PVT
))
          if(_sendLocation( ubxMsg.getPayload(), vBat ))
          {
            mode = SLEEP;
          }
        break;
      case GNSSmsg::GNSS_MSG_NMEA:
        Serial.println("NMEA");
        break;
    }
  }
  else //mode == SLEEP
  {
    Serial.println("sleep");
    rf95.sleep();
    delay(10000);
    mode = SCAN;
  }

}
```

Web.py

```
#Author: Evan Lee les01004@gmail.com
import serial
import http.server
import socketserver
import threading
import json
import datetime

#to update refresh time, search for "Refresh every" and "setInterval" two variables should be
changed.

SERIAL_PORT = 'COM8'  # Update if COM port changes
BAUD_RATE = 9600

CURRENT_LOCATION = [48.46127508996356, -123.31065910516826]  # UVIC ECS

class SerialDataHandler:
    def __init__(self):
        self.serial = serial.Serial(SERIAL_PORT, BAUD_RATE, timeout=10)
        self.trap_data = {}  # Dictionary to store ID, lat, long, battery, and timestamp
        self.thread = threading.Thread(target=self.read_serial_data_thread, daemon=True)
        self.thread.start()

    def read_serial_data_thread(self):
        while True:
            if self.serial.in_waiting > 0:
                raw_data = self.serial.readline().decode('utf-8').strip()
                try:
                    id_str, vbat_str, lat_str, long_str = raw_data.split(',')
                    trap_id = int(id_str)
                    vbat = float(vbat_str)
                    lat = float(lat_str)
                    long = float(long_str)
                    timestamp = datetime.datetime.now().isoformat()
                    while lat > 100 :
                        lat/=10
                    while long < -1000 :
```

```
                    long/=10
                if (lat == 0 and long == 0):
                    lat=48.446334
                    long=-123.404944

                self.trap_data[trap_id] = {'id': trap_id, 'vbat': vbat, 'lat': lat, 'long':
long, 'timestamp': timestamp}
            except ValueError:
                print(f"Invalid data received: {raw_data}")
            print(f"Received data - ID: {trap_id}, Battery: {vbat}, Lat: {lat}, Long: {long}")



    def get_trap_data(self):
        return list(self.trap_data.values())

serial_handler = SerialDataHandler()

class SimpleHTTPRequestHandler(http.server.SimpleHTTPRequestHandler):
    def do_GET(self):
        if self.path == '/':
            self.send_response(200)
            self.send_header('Content-type', 'text/html')
            self.end_headers()
            html = f"""
            <!DOCTYPE html>
            <html lang="en">
            <head>
                <meta charset="UTF-8">
                <meta http-equiv="refresh" content="10">  <!-- Refresh every 10 seconds -->
                <meta name="viewport" content="width=device-width, initial-scale=1.0">
                <title>Solar-Powered Crab Trap Tracking System</title>
                <link rel="stylesheet" href="https://unpkg.com/leaflet@1.7.1/dist/leaflet.css" />
                <style>
                    body {{
                        font-family: Arial, sans-serif;
                        margin: 20px;
                        padding: 20px;
                        background-color: #f0f0f0;
                    }}
                    h1 {{
                        color: #007bff;  /* Bootstrap blue color */
                    }}
                    .container {{
                        max-width: 800px;
                        margin: auto;
                        background-color: #ffffff;
                        padding: 20px;
                        border-radius: 8px;
                        box-shadow: 0 0 10px rgba(0,0,0,0.1);
                    }}
                    #map {{
                        height: 400px;
                        margin-top: 20px;
                    }}
                    #status_message {{
                        font-size: 18px;
                        margin-top: 10px;
                        text-align: center;
                        color: #6c757d; /* Bootstrap muted color */
                    }}
                    #last_updated {{
                        font-size: 14px;
                        margin-top: 20px;
                        color: #333;
                    }}
                    .trap-update {{
                        margin-top: 5px;
```

```
                }}
            </style>
        </head>
        <body>
            <div class="container">
                <h1>Solar-Powered Crab Trap Tracking System</h1>
                <h2>Trap Locations</h2>
                <div id="map"></div>
                <div id="status_message">Waiting for data...</div>
                <div id="last_updated"></div>
            </div>

            <script src="https://unpkg.com/leaflet@1.7.1/dist/leaflet.js"></script>
            <script>
                let map = L.map('map').setView([0, 0], 2);  // Initial map view

                L.tileLayer('https://{{s}}.tile.openstreetmap.org/{{z}}/{{x}}/{{y}}.png', {{
                    attribution:                          '&copy;                          <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
                }}).addTo(map);

                let currentLocation = [{CURRENT_LOCATION[0]}, {CURRENT_LOCATION[1]}];
                let currentMarker = L.marker(currentLocation, {{
                    icon: L.icon({{
                        iconUrl: 'https://maps.google.com/mapfiles/ms/icons/red-dot.png',
                        iconSize: [32, 32],
                        iconAnchor: [16, 32],
                        popupAnchor: [0, -32]
                    }})
                }}).addTo(map).bindPopup("Your Location");

                let trapMarkers = new Map();
                let allMarkers = [currentLocation];

                function fetchData() {{
                    fetch('/data')
                        .then(response => response.json())
                        .then(data => {{
                            allMarkers = [currentLocation];
                            document.getElementById('last_updated').innerHTML = '';  // Clear
previous updates

                            data.forEach(trap => {{
                                const latLng = [trap.lat, trap.long];
                                allMarkers.push(latLng);

                                if (trapMarkers.has(trap.id)) {{
                                    map.removeLayer(trapMarkers.get(trap.id));
                                }}

                                let trapIcon = L.divIcon({{
                                    html: `<div style="position: relative;">
                                              <div  style="position:  absolute;  bottom:  40px;
left: -5px; background-color: white; padding: 2px 5px; border: 1px solid #000; border-radius: 3px;
font-size: 12px;">${{trap.id}}</div>
                                              <img
src="https://maps.google.com/mapfiles/ms/icons/blue-dot.png" style="width: 32px; height: 32px;"/>
                                           </div>`,
                                    className: ''
                                }});
                                let      marker      =      L.marker(latLng,      {{      icon:
trapIcon }}).addTo(map).bindPopup(`Trap ID: ${{trap.id}}<br>Battery: ${{trap.vbat}} mV`);
                                trapMarkers.set(trap.id, marker);

                                // Update last updated time
                                let updateDiv = document.createElement('div');
                                updateDiv.className = 'trap-update';
```

```
                                    updateDiv.innerText = `Details...ID: ${{trap.id}}, Battery:
${{trap.vbat}} mV, Lat: ${{trap.lat}}, Long: ${{trap.long}} at TIME: ${{new
Date(trap.timestamp).toLocaleTimeString()}}`;
                                    document.getElementById('last_updated').appendChild(updateDiv);
                                });
                                let bounds = L.latLngBounds(allMarkers);
                                map.fitBounds(bounds);
                                document.getElementById('status_message').innerText = "Data
updated at " + new Date().toLocaleTimeString();
                            })
                            .catch(error => {{
                                console.error('Error fetching data:', error);
                                document.getElementById('status_message').innerText = "Error
fetching data";
                            });
                    }}

                    // Fetch data initially and then every 10 seconds
                    fetchData();
                    setInterval(fetchData, 10000);
                </script>
            </body>
            </html>
            """
            self.wfile.write(html.encode('utf-8'))
        elif self.path == '/data':
            self.send_response(200)
            self.send_header('Content-type', 'application/json')
            self.end_headers()
            trap_data = serial_handler.get_trap_data()
            self.wfile.write(json.dumps(trap_data).encode('utf-8'))
        else:
            super().do_GET()

def run_server():
    with socketserver.TCPServer(("", 8000), SimpleHTTPRequestHandler) as httpd:
        print("Serving at port 8000")
        httpd.serve_forever()

# Start the server in a separate thread
server_thread = threading.Thread(target=run_server, daemon=True)
server_thread.start()

# Keep the main thread alive
server_thread.join()
```