
Portable LoP Tracker

Technical Report

1922268

Contents

1	Definition	1
1.1	Introduction	1
1.1.1	Abstract	1
1.1.2	Aim	2
1.1.3	Scope	2
1.1.4	Objectives	2
1.2	Literature Review	3
1.2.1	LoRa	3
1.2.2	Alternative products	3
2	Development	5
2.1	Design	5
2.1.1	Specification	5
2.1.2	Hardware	7
2.2	Development	9
2.2.1	Initial function	9
2.2.2	Power tester	14
2.2.3	Intermediate steps	15
2.2.4	Battery test	17
2.2.5	Error investigation	20
2.2.6	Case design	21
2.2.7	Distance test	22
3	Completion	23
3.1	Results	23
3.2	Conclusions	23
3.2.1	Assessment	23
3.2.2	Improvements	23
4	References	24
A	Purchasing lists	i
B	Data	v
B.1	Power Consumption	v
B.2	Distance	vii
C	Code	ix
C.1	Power scripts	ix
C.2	Distance scripts	xvi
C.3	Microcontroller sketches	xviii

C.4 SBC scripts	xxiv
D Errors	xxxiv

List of Figures

2.1	System overview flowchart	8
2.2	GPS hardware serial output	10
2.3	LoRa communications with GPS location	13
2.4	Power meter results	15
2.5	Battery charge and discharge curves	19

List of Tables

2.2	Non-standard format example - 230127.csv sample	18
A.1	Preliminary hardware list	i
A.1	Preliminary hardware list (cont.)	ii
A.2	Updated hardware list	ii
A.2	Updated hardware list (cont.)	iii
A.3	Antennae list	iv
B.1	Power meter recordings	v
B.2	Discharge data sample - 230213.csv	vi
B.4	Charge data sample - 230214.csv	vi
B.6	Distance Test 1 sample - first_test.csv	viii
B.8	Distance Test 2 sample - second_test.csv	viii

Glossary

GPGBA Global Positioning System Fix Data. 17

GPRMC Recommended minimum specific GPS/Transit data. 10

LoRa Long Range (Radio). iii, 1, 2, 3, 8, 9, 12, 13, 14, 15, 17, 18

PGTOP Antenna advisor. 10, 11

PMTK Proprietary data transfer protocol. 11

U.FL Miniature RF coaxial connector, aka I_PEX, AMC, AMCC, UMCC. 16

Acronyms

CSS Chirp Spread-Spectrum. 3

CSV Comma Separated Value. 17, 18, 19

GNSS Global Navigation Satellite System. 17

GPIO General Purpose Input Output. 9

GPS Global Positioning System. iii, 2, 8, 9, 10, 11, 12, 13, 14, 17, 20

HAT Hardware Attached on Top. 9

I²C Inter-Integrated Circuit. 8

IDE Integrated Development Environment. 10

NMEA National Marine Electronics Association. 10, 11

OLED Organic Light-Emitting Diode. 12

PCB Printed Circuit Board. 16

PPS Precise Positioning Service. 17

RTK Real Time Kinematic. 17

SBC Single Board Computer. 8

SPI Serial Peripheral Interface. 8, 12

SSH Secure Shell. 18


USB Universal Serial Bus. 20


Chapter 1

Definition

1.1 Introduction

1.1.1 Abstract

Many pet owners, particularly cat owners, allow their pets to freely roam the outdoors. However, it is impossible to determine what the pet is up to, or more pertinently, where they are and whether there is anything dangerous nearby . It is also not unusual for pets to go missing, and without some method of tracking location, it is impossible to determine where they are until they show up by themselves. Should a pet be lost or stuck, this becomes increasingly unlikely.

To counteract this, solutions for location tracking pets exist. Many of these solutions encounter one drawback or another . This study concerns investigating and developing a novel solution to the problem of pet tracking using LoRa radio modulation.

This technique allows for highly efficient radio transmissions, gaining the large distances with minimal power. In effect, it allows the coverage of the roaming range of a cat. The difficulty lies in ensuring the radio module can be powered and protected from the elements.

Development followed steps of getting logical functions that make up the project to work yada yada

Results were mid

conclusion is it's a cool bit of kit and  we're gonna mass produce it now

This document is the feasibility study for a LoRa based cat tracker. This project will enable pet owners to allow their pets to roam free where they otherwise would not be able to do so (i.e. they live in an apartment and cannot install a catflap). This has wide reaching potential, with 26% of UK households owning 11 million cats, of which 63% spend time outdoors [1]. Furthermore, the market size for global pet wearables was \$2 billion in 2022 [2], proving high demand for such a product. Finally, the intended small form-factor and large tracking range make this product potentially applicable in many other situations besides the one it is designed to cater for.

1.1.2 Aim

The ultimate goal is a portable GPS tracker that makes use of LoRa to update location information on a periodic basis. The solution will need to monitor the current location of a moving target for a significant portion of the day and work within a certain radius, perhaps 500 m of a stationary base.

The focus is on cats in particular, so care may need to be taken to ensure the tracker is ergonomically suited and safe. Due to the wider possible uses of the tracker, portability may also need to be considered.

As a product designed for use by the average pet owner, the setup must be suitable for use in a home and not especially difficult for the user to get all required parts working. A solution that works 'out of the box' is better suited.

1.1.3 Scope

Some pets have particularly independent spirits, and perhaps no more renowned for free-roaming prowess is the domestic cat. Cats are also particularly well known for going missing for days at a time - even years [3] -, if they return at all. This project intends to design and develop a product that allows cats to express their natural behaviour [4] by allowing roaming, while providing their owners up-to-date information of their location if, for any reason, it needs to be known.

To achieve this, the cat needs to be fitted with a tracking transmitter of some sort for the period that they are outdoors. This tracker will transmit location on a regular basis, and will be weather-proof. The transmit location will then be received by a permanent fixture at the owner's home, after which it can be processed and made presentable, so the cat can be easily located.

1.1.4 Objectives

This project consists of the following objectives to see completion.

1. Obtain ethical approval for experimentation using cats.
2. Design hardware and software for the tracker.
 - (a) Purchase required hardware.
 - (b) Develop circuit.
 - (c) Program the microcontrollers.
 - (d) Design casing and harness.
3. Manufacture the product.
4. Test reliability and safety.

1.2 Literature Review

1.2.1 LoRa

LoRa radio modulation is a technique derived from CSS, a combination of chirp signals and spread-spectrum technology. Spread-spectrum is a method by which a radio signal has a wider bandwidth due to spreading in the frequency domain, which brings with it benefits such as resistance to noise and reduced spectral flux density (making it more power efficient). Chirp signals are a method by which the signal frequency is increased or decreased with time.


Semtech currently holds the patent [5].

LoRa uses license-free radio bands. In the UK, this is covered by EU868.

chirp spread spectrum uses wideband linear frequency modulated chirp pulses to encode information.

a chirp is a sine signal whose frequency increases or decreases over time

1.2.2 Alternative products

Investigating existing patents, there appears to be a device that performs a similar function published in China [6]. However, it makes no mention on how the tracker operates besides the use of LoRa and discusses more the construction of the casing. A similar patent can be found [7], which again primarily discusses the construction of the tracker casing. However, from the title it can be deduced that the tracker is Bluetooth  based.

The intent of this project is not completely novel. Trackers, in particular for house pets (dogs and cats) and farm cows are commonplace [8, 9], however these are often based on IoT or LoRaWAN specifically. The project attempts to introduce a new solution to this existent problem.

Products that achieve similar functionality to the one proposed in this project exist. They can be broken down into two types:

- LTE based - provides completely remote coverage and is only limited by mobile (LTE) network range. These often come with an upfront cost followed by a subscription fee.
- RF based - only useable within a limited range. These are often smaller and lighter, but often come with more limited functionality.

The product of this project falls into the latter category, however it benefits from a greatly enhanced range. LoRa is capable of reliably transmitting over 3km, even in dense suburban areas [10]. In comparison, the Tabcat V2 quotes a range of 152m [11]. While that may be suitable for many owned cats, this does not cover the maximum range an owned cat may roam (278m [12]), or the typical range of a feral cat (1253m [13]). Furthermore, a single base will be capable of receiving multiple signals, supporting owners with multiple cats (35% of cat-owning households [1]). This product also does not bear reliance on LTE networks, being RF based, and therefore is suitable to use in regions where infrastructure is lacking. Furthermore, LTE type solutions send and store data to a private server [14], raising concerns with security and data protection (i.e. an individual's house can be identified from collected location data).

Another style of product uses privately managed networks, built from users of the product (Apple Air Tags, Tile, etc). This suffers from the same infrastructure problem, except perhaps to a greater degree due to reliance on distributed consumer use of the product to create the network.

Chapter 2

Development

2.1 Design

The system specifications will be defined and hardware purchasing will be informed. The process of developing the system will be outlined.

2.1.1 Specification

The design will need to follow a number of specification points for success.

Portable

With the primary target being cats, the product cannot be a significant burden to wear. Portability in this case means three things:

1. Small form-factor.
2. Lightweight.
3. Battery powered.
4. Equippable.

Small form-factor The product needs to be small enough to fit on a cat. Given the likelihood for an outdoor cat to be squeezing through tight spaces, the product needs to be small enough to not get caught and become a restriction.

The smallest form-factor will be aimed at, and designed in such a way that it can integrate in a seamless fashion with the cat's wearable (be it collar or harness).

Lightweight There is no clear answer for what an appropriate weight for the product would be given great range in the weight of a cat. The range easily covers 3 kg to 6 kg for healthy pet cats [15].

The answer to how much weight a cat can carry for long periods is even less clear. Anecdotal references suggest a value of around 10 % is easily manageable. With that in mind, an aim for half that at 5 % of the minimum likely weight is a reasonable value, such that the weight is appropriate for any cat.

This puts the desired weight for the tracker at ≤ 150 g, subject to testing to validate.

Battery powered The key factor of portability is that the product can operate without a direct connection to an outlet for power. This means that the product requires some form of portable power.

With weight, convenience, and environmental considerations in mind [16], the most appropriate type of battery will be rechargeable lithium-ion. This may raise some safety concerns due to the high likelihood of flammability of rechargeable battery cells [17].

Equippable Some method to attach the product to a cat needs to be devised. If it is small enough, this may be on a collar. Alternatively, attached to a vest may also work. This is very dependant on the dimensions the product takes in the end. Furthermore, if the product takes a prominent, protruding shape, it needs to detach easily so as not to be a restriction for the animal.

The primary focus will be on ensuring it cannot be caught in anything to begin with, as the tracker coming loose defeats its purpose.

Suitability

This product is designed for use by the average pet owner. Therefore, no specialised knowledge on how to operate any equipment can be expected. Furthermore, the system should be installable and maintainable using skills and hardware the average household can be expected to own, with some allowances for a small amount of DIY work.

Installable If any hardware requires installation, it must be in as simple a manner as possible, with only basic (manual) tools. The expectation will be that installation should be, at most, as difficult as a simple piece of furniture from a DIY store.

Compliance Compliance for consumer products needs to be met. If certification is not explicitly sought (for example, in the case third-party verification is required), the product must still meet the standards outlined such that it can be certified when necessary.

As a product developed in the UK, UKCA guidelines will be used.

Reliability

The

Stable


Fault resistant

Safety



Repairable

Reporting

The following  specification list will aid in guiding the design process and help evaluate the success of the final design.

- Portable
 - Small form factor (50mm × 70mm × 10mm).
 - Lightweight (<40g).
 - Battery powered with enough capacity for a day of roaming (~8 hours).
- Suitable for home/consumer use
 - Installable with tools a consumer could reasonably be expected to own.
 - Compliant to regulations for consumer products.
- Reliable
 - Stable - minimal downtime, if any.
 - Resistant to software and hardware failures.
 - Repairable without great inconvenience or requiring specialised tools.
- Suitable Reporting
 - Regular updates - updates every minute or better.
 - Capable of reporting in any range between 5m and 1000m.
 - Suited for an urban environment.
- Safe
 - Not a fire hazard.
 - Not an electrocution hazard.
 - Will not cause burns.
 - Comfortable to wear.
 - Compliant with health, safety, data protection, and environmental regulations.

The system can be broken down into two independent parts - the 'transmitter' (attached to the cat) and 'receiver' (stationary at the owner's home). Each subsystem has an independent process of operation. The minimum requirements are subsequently outlined in fig. 2.1.

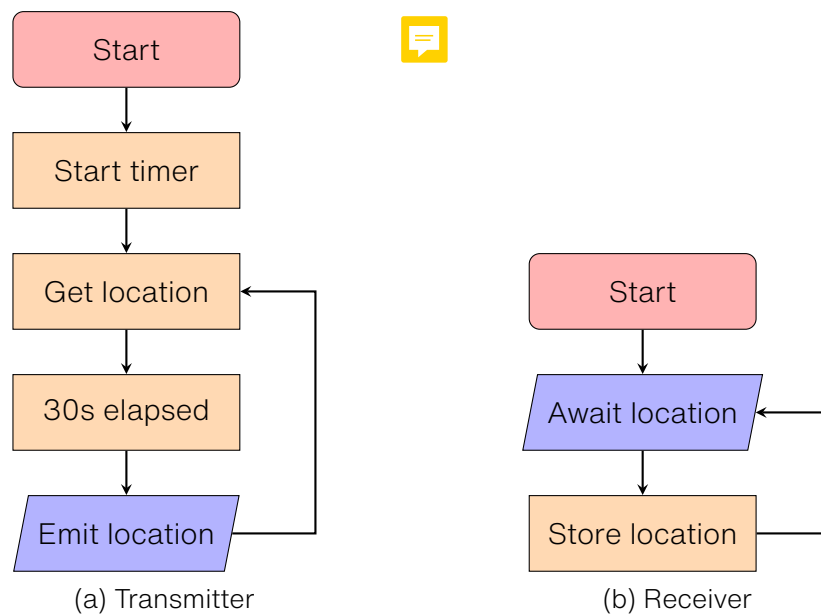
2.1.2 Hardware

Given the specification list, some hardware requirements can be defined to inform purchasing decisions. The transmitter needs to be portable, therefore each component is size, weight, and power limited. The receiver, on the other hand, is designed to be stationary and has no technical size or power limitations. However, it has to be sized within reason for a product designed for home use.

The following component list will help inform what specific components are required.

- Transmitter

Figure 2.1: System overview flowchart



- GPS module for location.
- LoRa module for transmission.
- Small form-factor antenna to aid transmission.
- Microcontroller for general computation.
- Receiver
 - LoRa module for transmission.
 - Microcontroller or SBC for computation.

There are many possible configurations to produce the sets of hardware required from parts available across different vendors. With this in mind, solutions that incorporate built-in features will be prioritised.

Some decisions were made to narrow down the diverse range of options, given hardware availability and technological experience. The microcontroller was limited to a Raspberry Pi 3 as that was readily available to . The microcontroller was limited to about Adafruit's Feather range, due to prior experience and small form-factors provided. Additionally, the Feather boards are stackable and therefore work well together¹. Some antenna options will also be included. Ceramic antennae have the smallest form-factor, however they may prove difficult to mount due to small contact points compared to more traditional wire antennae. The receiver will also require a large gain antenna to pick up weak transmissions.

A preliminary hardware set was produced (table A.1). This list consists of two sets of hardware for the transmitter, hardware required regardless of set, and some antenna options.

¹The RFM95W LoRa chipset uses SPI and may limit bandwidth on the bus for other hardware in personal experience. mentioned in dev too. no refs.. Therefore, an alternative set of hardware using the I²C bus will be produced in case it proves necessary.

The radios have been selected to ensure they are able to communicate with each other using the same modulation method. Conveniently, they each use an RFM95W module [18].

talk about having to ensure the radios will interface with each other

section break?

Upon review, this list had to be adjusted to primarily target hardware from approved vendors. With the vendor adjustment made, some of the initial hardware had to be substituted for more expensive alternatives. This made board options simpler by limited the feasible solution to a Feather M0 with bundled LoRa chip. Additional accessories were also included, along with a more extensive set of antenna and battery options to choose from.

break again?

The final first round purchase list was sent for ordering.

this section is incomplete. figure out the final first order list and tabulate it. discuss omissions and the final cost of products. figure out where to slot in the antenna ordering section, because it happens some ways down. also worth elaborating on the antenna options first proposed.

talk about choosing the pi hat instead of the extra LoRa board and wiring in. mention that the new gps uses hardware serial so no clashes.

mention the gps uses GPMMOPA6H module somewhere

2.2 Development

2.2.1 Initial function

Hardware

Development begins with putting the hardware together and testing functionality.

The Feather (M0 LoRa) and GPS Featherwing are soldered together. They are designed to stack, and therefore each pin can be soldered directly using included headers. For now, additional hardware like batteries and antennae are not necessary, as only basic function will be tested.

The Pi HAT is similarly configured to slot onto the GPIO pins available on the Pi 3 directly. No additional work is required to interface the hardware.

GPS

With the boards soldered together, basic operation can be checked with the blink sketch [19]. This sketch will simply switch the inbuilt LED on and off every second, and can be used to verify that the board can accept and run sketches.

Having proven basic function, the next test concerns the function of the specific hardware. The GPS module will be focused on to begin with.

The initial test sketch as per the documentation [20, 21] is loaded using the Arduino IDE [22]. This produces output as in fig. 2.2.

```
$GPRMC,224029.000,A,5223.00,N,00133.00,W,0.91,44.23,140123,0.0,A*
$PGTOP,11,2*6E
$GPRMC,224034.000,A,5223.00,N,00133.00,W,1.34,39.87,140123,0.0,A*
$PGTOP,11,2*6E
$GPRMC,224039.000,A,5223.00,N,00133.00,W,1.37,42.17,140123,0.0,A*
$PGTOP,11,2*6E
$GPRMC,224044.000,A,5223.00,N,00133.00,W,1.22,66.38,140123,0.0,A*
$PGTOP,11,2*6E
$GPRMC,224049.000,A,5223.00,N,00133.00,W,0.50,24.15,140123,0.0,A*
$PGTOP,11,2*6E
$GPRMC,224054.000,A,5223.00,N,00133.00,W,0.45,354.34,140123,0.0,A*
$PGTOP,11,2*6E
$GPRMC,224059.000,A,5223.00,N,00133.00,W,1.25,74.13,140123,0.0,A*
$PGTOP,11,2*6E
$GPRMC,224104.000,A,5223.00,N,00133.00,W,0.31,12.66,140123,0.0,A*
```

Figure 2.2: GPS hardware serial output with sensitive location data censored, showing GPRMC and PGTOP sentences.

By default, the GPS module outputs many standardised NMEA sentences, the most important of which to begin with is the GPRMC sentence.

```
1 $GPRMC,224029.000,A,5223.00,N,00133.00,W,0.91,44.23,140123,0.0,A*40
```

Listing 2.1: NMEA GPRMC sentence example

The output format is not immediately obvious. The listing 2.1 can be broken down as follows:

GPRMC	Message ID
223029.00	Time in UTC (22:30:29.00)
A	Validity - Active/Void
5223.00,N	Latitude (52°23.00" North ²)
00133.00,W	Longitude (001°33.00" West)
0.91	Speed (knots)
44.23	Tracking angle
140123	Date (14 th January 2023)
A	Positioning mode indicator [23]
	A Autonomous D Differential E Estimated (dead reckoning) M Manual input N Data not valid
40	Checksum (XOR of all bytes between \$ and *)

The output location was verified to be correct. Obtaining a fix while indoors proved very

²The longitude and latitude appear to be in decimal degrees at a glance and will therefore return errors when attempting to plot with this expectation. Furthermore, it is specifically degrees minutes and not degrees minutes seconds as one may expect.

difficult. The GPS module uses coin cell to save fix data, which would have helped obtain a fix outdoors and then move indoors to observe the data.

The PGTOP sentence characterises the antenna connection, a non-standard command specific to the FGPMMPA6H module [24].

```
1 $PGTOP,11,2*6E
```

Listing 2.2: NMEA PGTOP sentence example

Listing 2.2 can be broken down as follows:

PGTOP	Message ID
11	Command ID
2	Antenna status
	1 Active antenna shorted
	2 Using internal antenna
	3 Using external (active) antenna
6E	Checksum

The antenna status is not particularly useful, especially so at this stage of the development. Therefore it can be disabled. Many of the commands to program the device (for example, changing the sentence output frequency) can be found in the PMTK reference document³ [25]. Alternatively, the Adafruit GPS library [26] has support for some commands. These commands are not documented and can only be found hardcoded into the header file of the library [27].

Regarding the antenna, it can be disabled in two ways, via serial⁴ :

```
1 #define GPSSerial Serial1
2 GPSSerial.println("$PGCMD,33,0*6D");
```

Listing 2.3:

or using the library:

```
1 Adafruit_GPS GPS(&GPSSerial);
2 GPS.sendCommand(PGCMD_NOANTENNA);
```

Listing 2.4:

The library provides some useful functions and makes the code less verbose and more readable, however it may prove to include significant enough overhead to impact the battery life. If that happens to be the case the library can be removed as essential function can be programmed to the GPS chip directly via serial.

³With a battery, the PMTK commands are retained and do not need to be presented to the module on power-on. Otherwise, settings must be reconfigured on every boot.

⁴This must be the serial connection to the GPS device from the main board, and cannot be accessed directly via serial monitor. In order to send commands on the fly via serial monitor, the code must be configured to then forward those commands to the correct serial connection (in the case of listing 2.3, 'serial1').

Pi LoRa

The LoRa bonnet library as provided by Adafruit requires Circuit Python to be installed on the Pi [28, 29]⁵.

With required software, libraries, and fonts installed, the example code [30] can be loaded. This checks that the radio is present and the OLED displays correctly.


Finally, the radio function script can be loaded [31]. This will await incoming LoRa packets, but can also send data using the buttons.

Feather LoRa

LoRa transmission with this board relies on the RadioHead library [32] in order to transmit LoRa packets, installation details for which can further be found in the documentation [33].

The transmission test script can be loaded onto the Feather [34] once required software has been installed. This will transmit a packet every second and await a reply. For the time being, the Pi will not automatically send a reply, so the Feather will output the “error text”. This is of little consequence as the Pi primarily only needs to work as a receiver with no two-way communication⁶.

LoRa with GPS

Provided one-way communication is  working as expected⁷, the next step is to integrate LoRa with GPS on the Feather. This stage is where communication collision issues have occurred in previous projects [self cite?], however as the GPS module uses the second serial connection and the LoRa chip uses SPI, no such issues should occur.

For streamlined testing, the library will be used to strip the data sent to only latitude, longitude, and packet number (incremented every time GPS location is fixed to indicate that the location is up-to-date). The various segments of data then need to be placed together in a structure. The public member function `send()` of the `RH_RF95` class [36] takes two arguments, `data (uint8_t*)` and `len (uint8_t)`. This means that the data to be sent needs to be consolidated into a single structure, with the pointer to the structure and its length sent to the function. The Arduino compiler is built on processing AVR-C and therefore suffers when handling strings and performing conversion and concatenation natively, which can lead to complex and verbose code. The IDE also supports C++ classes and function, among which include the `String` class and `+` concatenating function [37, 38]. While the class may prove problematic on embedded hardware due to memory fragmentation on the heap, for the time being it is sufficient where the primary goal is functional and readable

⁵The installation documentation concerns both RFM69 and RFM9X radios. This project uses the RFM95W radio, also referred to as LoRa.

⁶The test script on the Pi does in fact allow output by pressing one of the buttons. This can be used to test communications and will be picked up by the Feather if this is tested.

⁷Both example scripts are set at 915.00 MHz and will work by default. This needs to be adjusted to 868.00 MHz for use in the UK [35].

code. Two additional benefits are that the length is also easily attainable as the method `.length()` on the class, and also conversion to an array of type `char` is supported. Listing 2.5 shows how the body of the code appears.

```

1 void loop() {
2     //buffer gps location
3     char c = GPS.read();
4     //interpret whether new location obtained and parse
5     if (GPS.newNMEAreceived()) {
6         if (!GPS.parse(GPS.lastNMEA())) {
7             return;
8         }
9     }
10    //execute every 5 seconds
11    if (millis() - timer > 5000) {
12        timer = millis();
13        //increment packet counter if new fix obtained
14        if (GPS.fix) {
15            count++;
16        }
17        //produce concatenated string
18        String output = String(count) + ": " + String(GPS.latitude,4) + ","
+ String(GPS.longitude,4);
19        length = output.length();
20        //create variable send to hold packet once converted to type char
21        char send[length];
22        output.toCharArray(send, length);
23        Serial.println(send);
24        rf95.send((uint8_t *)send, length);
25    }
26 }

```

Listing 2.5: Transmission body code

The output can then be observed in section 2.2.1, showing LoRa and GPS working together.

```

54: 5223. ,133.
55: 5223. ,133.
56: 5223. ,133.
57: 5223. ,133.
58: 5223. ,133.
59: 5223. ,133.
60: 5223. ,133.
61: 5223. ,133.
62: 5223. ,133.
63: 5223. ,133.

```

(a) Feather transmission logs

```

pi@pi:~/Project $ python3 radio.py
RX: 57: 5223. ,133.
RX: 58: 5223. ,133.
RX: 59: 5223. ,133.
RX: 60: 5223. ,133.
RX: 61: 5223. ,133.
RX: 62: 5223. ,133.
RX: 63: 5223. ,133.
RX: 64: 5223. ,133.
RX: 66: 5223. ,133.
RX: 67: 5223. ,133.
RX: 68: 5223. ,133.

```

(b) Pi reception logs

Figure 2.3: LoRa communications with GPS location

2.2.2 Power tester

In order to determine the appropriate battery capacity to use, it is helpful to test the power consumption of the transmitter over the expected period of use. Greater capacity batteries are heavier and larger, two factors that need to be minimised. [what about fire hazard?](#)

By recommendation, a KCX-017 USB tester was purchased to measure the current draw of the transmitter under normal function. The included manual was poorly translated and difficult to follow, however a more informative version was found online [39]. No details further details about the manufacturer or any other products they may produce were found.

Test specification

The outline of the test is to determine the appropriate battery required for the module. The tester has various functions available, however only the capacity recording is relevant. The tester will be plugged into a USB outlet, and the transmitter will be plugged into the tester. The transmitter will be plugged into the tester and will operate from thereon. Recordings will be taken every thirty minutes for four hours as a preliminary measure of function. The Pi will be running to record that the transmitter is operating as expected.

Two tests will be performed:

1. The transmitter will be housed indoors so that it is unable to obtain a GPS lock. This will likely lead to the greatest power consumption.
2. The transmitter will be placed near a window so that it is able to update GPS location. This will simulate normal operation.

According to the documentation, the LoRa board draws 40 mA when the radio is actively listening, and draws 130 mA for 70 ms when transmitting at peak gain [33]. The GPS module draws 25 mA when tracking and 20 mA during navigation [20].

This gives a rough total of 65 mA typical draw in the conservative case. Considering the transmitter needs to operate for eight hours [reference spec](#), this suggests a battery capacity requirement of 520 mAh.

Results

Upon commencing the first test, the receiver registered no data. The cause was to simply remove the line `while (!Serial) delay(10);` as it prevents the code executing until the serial interface is connected so output can be monitored. In the case of this test, serial output was not required, however the line still stalled the code.

With the change made and output occurring as expected, the meter did not appear to register any changes. The meter was checked to be working by plugging a mobile phone and it began registering current draw as expected. Thereafter, the meter was left and checked every hour. The first change only occurred after three hours.

The consumption incremented only after three hours. It was left overnight and checked again, at this point registering 8 mA h. The complete set of recordings can be found in table B.1. The data is plotted below using code in listing C.1.

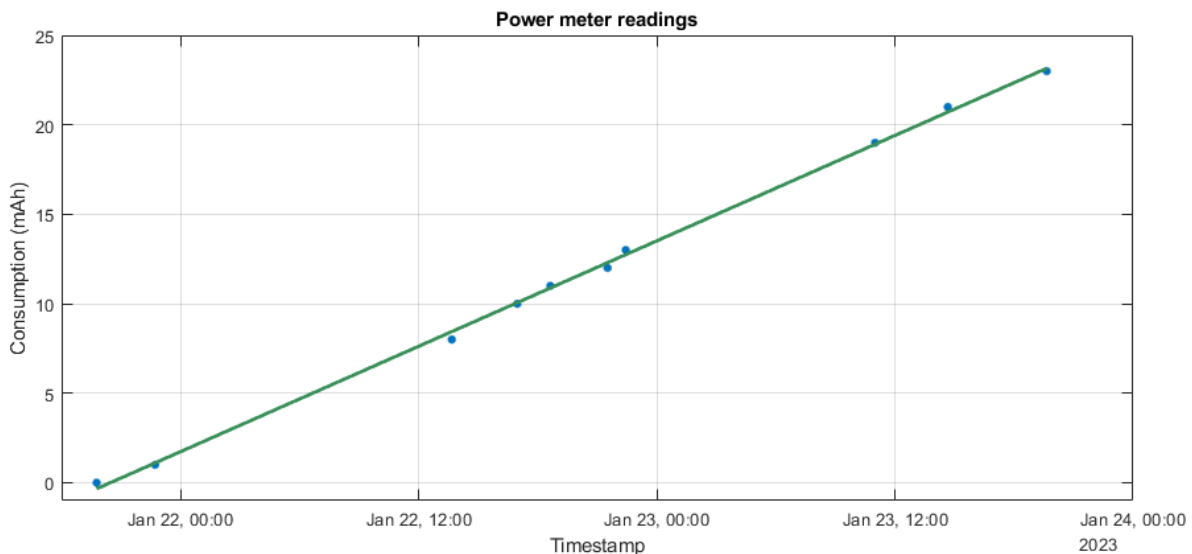


Figure 2.4: Power meter results

The results suggest that, during normal operation, the transmitter requires around only 0.5 mA. This is far too low to be accurate, and in turn, likely means the meter does not have sufficient resolution to measure the device. The meter does indicate a linear trend in power draw suggesting predictability in normal use, however given the poor accuracy of the data, the validity of this cannot be determined.

2.2.3 Intermediate steps

The power meter results were disappointing, however the expected power capacity requirement yielded a useful estimate (520 mA h). By repurposing a 3.7 V 500 mA h quadcopter battery, the transmitter can be tested on functionality directly. The LoRa board requires a JST jack [33]. By investigating Adafruit's other products, this appears to be a JST-PH connector specifically [40].

After performing the header change operation, the polarity was checked to be correct and then the transmitter was powered on using the battery and checked to be functioning correctly.

Code issues

During such a function test, the code on the receiver end crashed twice, producing an error such as in listing 2.6.

```
1 Traceback (most recent call last):
2   File "/radio.py", line 72, in <module>
```

```

3     packet_text = str(prev_packet, "utf-8")
4 UnicodeDecodeError: 'utf-8' codec can't decode byte 0xae in position 0:
    invalid start byte

```

Listing 2.6: Program error message

Due to the module being powered up and disconnected abruptly multiple times while setting up the battery, the suspicion was that transmission was interrupted midway, leading to the receiver attempting to parse an unfinished and corrupted packet. However, given the time window of less than 100 ms for transmission, the fact this occurred twice suggests this theory may not be likely.

In order to test the how long the transmitter is able to function, the code needs to be improved to be more resilient and stable. The example code [31] has been updated to safely handle the exception. While the cause of the error has not been determined or investigated further at this point, the code will now attempt to parse the packet in a `try/except` block. The first handles the `UnicodeDecodeError` directly, while the second handles anything else that may occur, in case there are other issues that have not been uncovered. It will indicate that an unexpected error has occurred and continue execution from thereon. The modified code can be found in listing C.10.

Antenna

For the time being, neither module makes use of an antenna as none are available to use at the moment. Part of the purchasing list included ceramic and strip antennae for the transmitter only. The ceramic antennae cannot be used as they require some form of breakout, usually on a PCB, and further currently require a U.FL adapter to interface with the board. The strip antennae use a very delicate cable between the strip itself and the connector, and were therefore easily sheared by one *Felus catus* who found it unattended during operation.

A spare whip antenna was available to use with the Pi, however its frequency was unknown. Upon casual inspection, it may be appropriately sized. To investigate further, the appropriate antenna length needs to be determined.

The wavelength can be found by:

$$\lambda = \frac{v}{f}$$

where: λ = wavelength

v = velocity, which in this case will be c , the speed of light

f = frequency

For the device operating at 868.0 MHz, this appears like the following:

$$\begin{aligned}\lambda &= \frac{c}{868.0 \text{ MHz}} \\ &= 0.345 \text{ m}\end{aligned}$$

The antenna whip measures between 21 cm and 25 cm, although it is impossible to say exactly how long it is without opening it up. If this antenna is centred near 868 MHz, it may be a five-eighth whip, given $\frac{5}{8}\lambda = 21.5 \text{ cm}$. This suggests it is appropriate to use until the antenna specified in the purchasing list arrives.

2.2.4 Battery test

The purpose of the test is to determine the length of time the transmitter can continuously operate under normal conditions, i.e. obtaining current GPS location and transmitting it via LoRa. As outlined in section 2.2.2, there is an expectation of roughly eight hours of function. This will now be investigated to determine whether this is the case.

The transmitter code will need to be updated to include information about the current status of the battery. The voltage level of the battery indicates whether it is within operating range, which can be measured by reading the analogue value of pin A7. At 4.2 V, capacity as it maximum, whereas the board will cease function around 3.2 V. The exact calculation can be found in the board documentation [33], which has been adapted in listing C.5.

After ascertaining that the calculation was working correctly, the code has been included in listing C.8. The output string has also been modified to include the battery data, altitude, and fix quality. GPS fix takes the following values [41]:

- 0 No fix (no signal)
- 1 Position fixed with GNSS satellites
- 2 Differential GPS
- 3-5 PPS⁸/RTK fix
- 6 Estimated location

The latter two required GPGLL data, which can be included with the library command `PMTK_SET_NMEA_OUTPUT_RMCGGA`. Finally, field separators were replaced with commas as part of preparation for the data to be stored in CSV format.

Test parameters

Two tests will be conducted.

1. Discharge - The board will operate under normal conditions. The battery will be charged to maximum capacity (4.2 V), before being allowed to discharge completely. Throughout that time, the board will be transmitting location data and battery level. The receiver will log the time of packets and store it in a data structure.

⁸Encrypted military/government use

2. Charge - After a complete discharge, the time taken to charge to maximum will be recorded in a similar fashion.

The way the Pi is operated will need to be changed. Normally, an SSH session is terminated when the client disconnect or times out, and any programs run during that session are closed. This was not a problem with the power meter test as the receiver only needed to indicate that transmitter was working, while the data itself was recorded externally. In this case, the Pi is recording all data and needs to be running without interruption for at least eight hours. Linux comes with a utility named `screen` that allows a virtual terminal session to be started that will not be closed⁹.

Issues

The tests were executed without issue and a brief study of the results indicated they were within expectation. However, some issues cropped up.

C treats strings as an array of type `char` and determines the end of the string with the 'null' character¹⁰ `\0`. C++ carries this over. Thereby, when using the C++ `String` class, this behaviour remains. As it is treated as part of the string, it is furthermore included in the length calculation and therefore transmitted over LoRa. The receiver would then interpret the characters of the packet in `utf-8`, and translate the null character as `\x00`. The existence of null characters caused difficulty when attempting to interpret the file in CSV format.

The eventual solution was to perform a small amount of processing on the receiver end. The inbuilt `rstrip()` method on a string type allows a specific character (in this case, `\x00`) to be removed from the string. This solution was determined after no amount of attempting to apply the same method to remove all nulls from a complete file worked.

Additionally, supplementary information like errors were logged to console with extra diagnostic information instead of stored in the file. Beforehand, this information would have to be stripped as the error line was non-standard for the CSV format. An example is shown in table 2.2

Date	Time	Packet	Longitude	Latitude	Voltage
23/01/27	20:27:27	3040	0	0	4.15
23/01/27	20:27:32	3041	0	0	4.15
23/01/27	20:27:35	unicode parse error			
23/01/27	20:27:32	3041	0	0	4.15
23/01/27	20:27:37	3042	0	0	4.157
23/01/27	20:27:42	3043	0	0	4.15

Table 2.2: Non-standard format example - 230127.csv sample

⁹An alternative would be to run the process and move it to background. The benefit of `screen` is that logging messages are easier to view when printed to terminal.

¹⁰In 8-bit hexadecimal representation, `\x00`.

This would cause errors for the CSV interpreter due to mismatched field lengths.

Finally, plotting temporal data types proved to be problematic. To simplify this, two changes were made. Firstly, the Date and Time fields were collated into one field in the standard format MATLAB expects¹¹. Secondly, a field was added for recording time relative to Unix epoch (i.e. now completely numeric and linear) in case plotting with such a field proves simpler.

The tests were then performed a second time.

Results

Data samples of the results can be found in table B.2 and table B.4. The resulting plot is shown in fig. 2.5 using code in listing C.3.

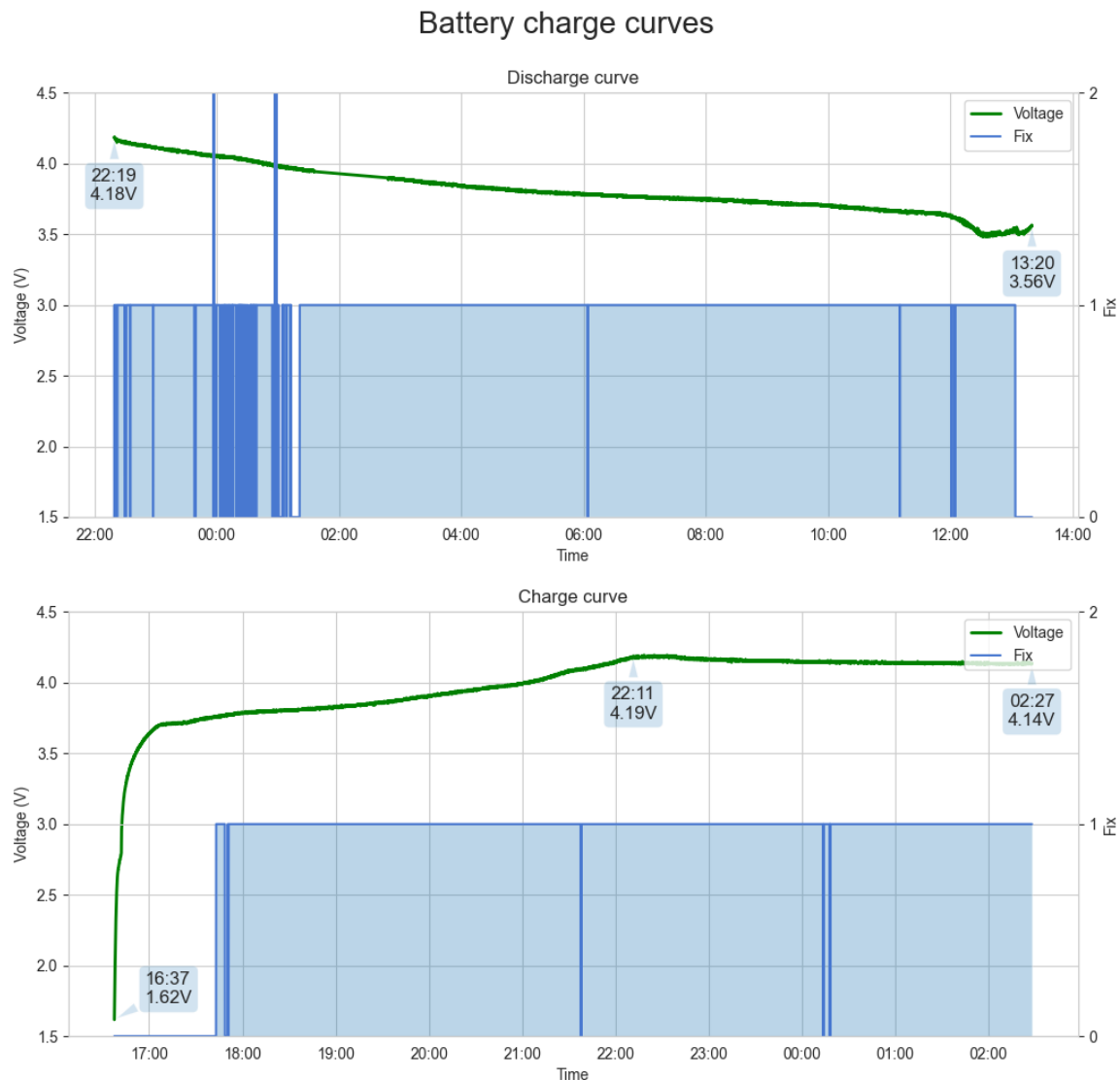


Figure 2.5: Battery charge and discharge curves

¹¹Format: 1st February 2023 at 1:54pm as "01-02-23 13:54:00".

The plots show the battery voltage level and whether the GPS has a fix.

Discharge Recording starts with a full battery, indicated by 4.2 V. Operation continues as expected, with GPS location fixed for much of that time. On two occasions, the fix was recorded at a value of 6¹². GPS was not unavailable for long enough to indicate whether not having a fix has any significant impact on battery consumption. Overall, the data indicates the total useable span of the device is fifteen hours.

Charge Notably, the voltage at the point of charging is much lower than than was last recorded when transmission during discharge cut off. The board does in fact continue to draw power. This ceases when the protection circuitry of the battery cuts supply entirely. This leads to some parasitic draw in the elapsed time. GPS location fix did not have a noticeable effect upon charging. Total time to charge to full capacity was around five hours and thirty minutes.

Remarks The discharge results were surprising. The estimate calculation section 2.2.2 returned an expectation of eight hours of function. Despite being a conservative estimate, the experimentally found value was almost double this. Furthermore, additional powersaving functions like putting the radio to sleep when not in use were not used, meaning the operating time of the device can be further increased. **transmission set at 5 seconds, which is better than the spec**

The time required to charge the module to maximum capacity was greater than expected. The slow charging time may come down to a few reasons. The charging regulator built into the Feather board is current limited to protect the circuitry [33]. It is further limited by the maximum current a USB 2.0 connector can draw (100 mA) [42]. This can be negotiated to be greater [43], however, and certainly many modern devices, particularly mobile phones, utilise some form of rapid charging.

The data collected suggests reliability. Any moment-to-moment fluctuations will be smoothed out due to the operating time of the module so further tests may not be necessary. Furthermore, it is not advisable to repeatedly charge and discharge lithium cells from maximum to minimum as doing so reduces the lifespan of the cell.

2.2.5 Error investigation

Further insight may be gained on the corrupted packets¹³. The errors recorded before the discharge test was performed can be found in listing D.1. It is notable that there are fewer errors here than in previous tests. The only significant change was the use of antennae during transmission. To elaborate on the error message, many of the bytes in the byte array cannot be decoded as they are all invalid utf-8, not just the start byte. The structure should

¹²Reference in section 2.2.4.

¹³Section 2.2.3.

follow 8-bits per character, yet some are greater (`\x90w`, line 13), use invalid hexadecimal (`\xb8B`, line 13), or are entirely not present (`\x89\22Rs`, line 4).

- Invalid hexadecimal - This appears due to the way Python `bytearray` is used. Where the default decoding (`utf-8`) failed, the array is encoded as `ASCII` and printed to console. This leads to non-hexadecimal characters appearing, and likely some of the missing characters also.
- Not 8-bits - If, for example, during transmission part of a byte is lost, the remainder of the packet will fail to decode correctly and appear in different lengths and otherwise non-standard form.

Solution There is no simple solution to what is essentially the question of how to ensure data integrity during transmission in a noisy environment. The `RadioHead` library supports packet confirmations with the `RHReliableDatagram` class. This does not validate the packet integrity¹⁴, but ensures that the packet has arrived. This solution alone would require some reverse engineering of the `recvfromAck` method in order to use it on the receiver end, however can be used to request retransmission (i.e. fail to acknowledge the packet) should it arrive corrupted.

An alternative would be to utilise some form of forward error correction, like parity checking. This has the limitation of increased complexity and computational load on the transmitter, and is further limited by being able to correct only one error. To adequately cover the entire transmission, the packet size (and subsequently, radio power draw and processor time) would increase.

For the time being, the errors are few enough in number that they can simply be omitted. Moving the device to production would require a revisit of handling data integrity.

2.2.6 Case design

designed enclosure for board.
 no thought to cat attachments at the moment, simply protecting delicate parts.
 include section for battery and enclosure for board. ufl holes, lid.
 first printn failed due to extruder block. second print success.
 >pics
 made some updates to the design.
 included usb slot. included power cable routing, and reduced the size slightly.

¹⁴A single bit being flipped may still resolve to a valid packet, even if the data is no longer accurate.

2.2.7 Distance test

attempted to get a static ip for the pi but IT stopped replying to me. :) attempted to find a suitable place to set up the pi given criteria. decided to set up in an office overlooking the carpark. > illustration of setup map?
> test setup.
discovered the ufl holes were too small. taped the antenna out caveman style.
gps tracker on phone failed for second test (show?)
took some time to finally understand the format. wrote a script to perform conversion. plotted in qgis.

Chapter 3

Completion

3.1 Results

3.2 Conclusions

3.2.1 Assessment

points met 
points not met and why fitting to cat pcb design

3.2.2 Improvements

use the button on the pi to send a start command to turn the transmitter transmission on. get the display to show info like "no gps" to warn user.
the board supports underclocking, so you can squeeze out more power - see the docs for more tips
employ manual string manipulation of the lora packet and don't use C++ strings due to memory problems it may cause.
the jst connector was coming loose. thought it almost shorted at one point.
negotiate higher usb current draw, consider something like qualcomm quickcharge.

Chapter 4

References

- [1] Cats Protection. 'Cats Report 2022'. In: (2022). Accessed 20 October 2022. URL: <https://www.cats.org.uk/media/11908/cats-report-2022-uk.pdf>.
- [2] Research and Makets. 'Global Pet Wearable Market Size, Share & Industry Trends Analysis Report By Application, By Technology, By Regional Outlook and Forecast, 2022 - 2028'. In: (May 2022). URL: <https://www.researchandmarkets.com/reports/5615579>.
- [3] Claire Davis. 'Missing cat reunited with owner after 11 years on the street'. In: *BBC News* (6th Mar. 2020). Accessed 20 October 2022. URL: <https://www.bbc.co.uk/news/uk-england-birmingham-51738246>.
- [4] Cats Protection. 'Cats Protection Essential Guide 12. Indoor and outdoor cats'. In: (). Accessed 20 October 2022. URL: https://www.cats.org.uk/media/1023/eg12_in_door_and_outdoor_cats.pdf.
- [5] Olivier Bernard André Seller. 'Wireless communication method'. US9647718B2. Semtech. 2015-09-09.
- [6] Jiang Pengtao. 'Positioning tracker based on LORA wireless communication technology'. CN210090660. 13th June 2019.
- [7] Chen Xifa. 'BLE LoRa positioning tracker shell'. CN209911563. Ltd. Shenzhen Gatintel-ligent Technology Co. 29th Mar. 2019.
- [8] Danco Davcev et al. 'IoT agriculture system based on LoRaWAN'. In: *2018 14th IEEE International Workshop on Factory Communication Systems (WFCS)*. 2018, pp. 1–4. DOI: 10.1109/WFCS.2018.8402368.
- [9] Sivaranjini Sivaraman, Ahmad Anwar Zainuddin and Krishnan Subramaniam. 'Advances in Technology for Pet Tracker Sensing Systems'. In: *2021 International Conference on Green Energy, Computing and Sustainable Technology (GECOST)*. 2021, pp. 1–4. DOI: 10.1109/GECOST52368.2021.9538744.
- [10] Aloÿs Augustin et al. 'A Study of LoRa: Long Range & Low Power Networks for the Internet of Things'. In: *Sensors* 16.9 (2016). ISSN: 1424-8220. DOI: 10.3390/s16091466. URL: <https://www.mdpi.com/1424-8220/16/9/1466>.
- [11] Tabcat. *Tabcat Cat tracker device*. Accessed 7 October 2022. URL: https://tabcat.com/shop/tabcat-v2/?sscid=a1k6_6pjj5.
- [12] Hugh J Hanmer, Rebecca L Thomas and Mark D. E. Fellowes. 'Urbanisation influences range size of the domestic cat (*Felis catus*): consequences for conservation'. In: *Journal of Urban Ecology* 3.1 (Dec. 2017). ISSN: 2058-5543. DOI: 10.1093/jue/jux014. eprint: <https://academic.oup.com/jue/article-pdf/3/1/jux014/22305829/jux014.pdf>. URL: <https://doi.org/10.1093/jue/jux014>.

- [13] Jeff A. Horn et al. 'Home range, habitat use, and activity patterns of free-roaming domestic cats'. In: *The Journal of Wildlife Management* 75.5 (2011), pp. 1177–1185. DOI: <https://doi.org/10.1002/jwmg.145>. eprint: <https://wildlife.onlinelibrary.wiley.com/doi/pdf/10.1002/jwmg.145>. URL: <https://wildlife.onlinelibrary.wiley.com/doi/abs/10.1002/jwmg.145>.
- [14] Tractive. *Privacy Policy of Tractive GMBH*. Accessed 20 October 2022. URL: <https://assets.tractive.com/static/legal/en/privacy-policy.html>.
- [15] Ellen Kienzle and Katja Moik. 'A pilot study of the body weight of pure-bred client-owned adult cats'. In: *British Journal of Nutrition* 106.S1 (2011), S113–S115. DOI: 10.1017/S0007114511001802.
- [16] George E. Blomgren. 'The Development and Future of Lithium Ion Batteries'. In: *Journal of The Electrochemical Society* 164.1 (Dec. 2016), A5019. DOI: 10.1149/2.0251701jes. URL: <https://dx.doi.org/10.1149/2.0251701jes>.
- [17] Yuqing Chen et al. 'A review of lithium-ion battery safety concerns: The issues, strategies, and testing standards'. In: *Journal of Energy Chemistry* 59 (2021), pp. 83–99. ISSN: 2095-4956. DOI: <https://doi.org/10.1016/j.jechem.2020.10.017>. URL: <https://www.sciencedirect.com/science/article/pii/S2095495620307075>.
- [18] HOPERF. *RFM95/96/97/98(W) - Low Power Long Range Transceiver Module V1.0*. HOPE MICROELECTRONICS CO.,LTD. 2/F, Building 3, Pingshan Private Enterprise Science and Technology Park, Lishan Road, XiLi Town, Nanshan District, Shenzhen, Guangdong, China, 2006.
- [19] Scott Fitzgerald, Arturo Guadalupi and Colby Newman. *Blink.ino*. 3rd Oct. 2022. URL: <https://github.com/arduino/arduino-examples/blob/main/examples/01.Basics/Blink/Blink.ino>.
- [20] Limor Fried. *Adafruit Ultimate GPS featherwing*. Accessed 24 October 2022. URL: <https://learn.adafruit.com/adafruit-ultimate-gps-featherwing?view=all>.
- [21] Matt G and Limor Fried. *GPS_HardwareSerial_EchoTest.ino*. 8th July 2022. URL: https://github.com/adafruit/Adafruit_GPS/blob/master/examples/GPS_HardwareSerial_EchoTest/GPS_HardwareSerial_EchoTest.ino.
- [22] Arduino. *Arduino Pro IDE (alpha preview) with advanced features*. Accessed 20 October 2022. URL: <https://blog.arduino.cc/2019/10/18/arduino-pro-ide-alpha-preview-with-advanced-features/>.
- [23] NovAtel. *GPS. GPS specific information*. Accessed 26 February 2023. Dec. 2022. URL: <https://docs.novatel.com/OEM7/Content/Logs/GPRMC.htm#NMEAPositioningSystemModeIndicator>.
- [24] G.top. *FGPMMOPA6H GPS Standalone Module Data Sheet*. Accessed 23 October 2022. GlobalTop Technology Inc. No.16 Nan-ke 9th Rd, Science-Based Industrial Park, Tainan, 741, Taiwan, R.O.C., 2011. URL: <https://cdn-shop.adafruit.com/datasheets/GlobalTop-FGPMMPA6H-Datasheet-V0A.pdf>.
- [25] G.top. *PMTK command packet*. Accessed 26 October 2022. GlobalTop Technology Inc. No.16 Nan-ke 9th Rd, Science-Based Industrial Park, Tainan, 741, Taiwan,

- R.O.C., 24th Sept. 2012. URL: https://cdn-shop.adafruit.com/datasheets/PMTK_A11.pdf.
- [26] Rick Sellens et al. *Adafruit_GPS*. Accessed 26 February 2023. 2022. URL: https://github.com/adafruit/Adafruit_GPS.
- [27] Rick Sellens. *Adafruit_PMTK.h*. 19th Jan. 2020. URL: https://github.com/adafruit/Adafruit_GPS/blob/master/src/Adafruit_PMTK.h.
- [28] Melissa LeBlanc-Williams. *CircuitPython on Linux and Raspberry Pi*. Accessed 27 October 2022. URL: <https://learn.adafruit.com/circuitpython-on-raspberrypi-linux/installing-circuitpython-on-raspberry-pi>.
- [29] Kattni Rembor. *Adafruit Radio Bonnets with OLED Display - RFM69 or RFM9X*. Accessed 24 October 2022. URL: <https://learn.adafruit.com/adafruit-radio-bonnets?view=all>.
- [30] Brent Rubell and Eva Herrada. *rfm9x_check.py*. Accessed 27 February 2023. 30th Sept. 2021. URL: https://github.com/adafruit/Adafruit_Learning_System_Guides/commits/main/pi_radio/rfm9x_check.py.
- [31] Brent Rubell and Eva Herrada. *radio_rfm9x.py*. Accessed 27 February 2023. 30th Sept. 2021. URL: https://github.com/adafruit/Adafruit_Learning_System_Guides/blob/main/pi_radio/radio_rfm9x.py.
- [32] Mike McCauley. *RadioHead Packet Radio library for embedded microprocessors*. Accessed 28 October 2022. URL: <https://www.airspayce.com/mikem/arduino/RadioHead/>.
- [33] Limor Fried. *Adafruit Feather M0 Radio with LoRa Radio Module*. Accessed 20 October 2022. URL: <https://learn.adafruit.com/adafruit-feather-m0-radio-with-lora-radio-module?view=all>.
- [34] Limor Fried, Scott Barnes and Ha Thach. *Feather9x_TX.ino*. Accessed 27 February 2023. 8th Mar. 2022. URL: https://github.com/adafruit/RadioHead/blob/master/examples/feather/Feather9x_TX/Feather9x_TX.ino.
- [35] Ofcom. 'IR 2030 - UK Interface Requirements. Licence Exempt Short Range Devices'. In: (Apr. 2021). Accessed 23 October 2022. URL: https://www.ofcom.org.uk/_data/assets/pdf_file/0028/84970/ir-2030.pdf.
- [36] Mike McCauley. *RH_RF95 Class Reference*. Accessed 28 October 2022. URL: https://www.airspayce.com/mikem/arduino/RadioHead/classRH_RF95.html.
- [37] Arduino. *What is Arduino?* Accessed 27 February 2023. URL: <https://www.arduino.cc/en/Guide/Introduction>.
- [38] Arduino. *Language Reference*. Accessed 27 February 2023. URL: <https://www.arduino.cc/reference/en/>.
- [39] KCX. *Power Bank capacity tester User Manual*. Accessed 20 January 2023. KCX. URL: <https://www.solarbotics.com/wp-content/uploads/kcx-017-power-bank-testing.pdf>.
- [40] Adafruit. *JST PH 2-Pin Cable - Female Connector 100mm*. Accessed 30 January 2023. URL: <https://www.adafruit.com/product/261>.

- [41] ArcGIS. *GpsFixStatus Enumeration*. Accessed 03 March 2023. 2011. URL: <https://help.arcgis.com/en/arcgismobile/10.0/apis/arcgismobile/api/ESRI.ArcGIS.Mobile~ESRI.ArcGIS.Mobile.Gps.GpsFixStatus.html#:~:text=The%20fix%20status%20indicates%20the,of%20the%20location%20being%20reported..>
- [42] Jan Axelson. *USB Complete: The Developer's Guide, Fifth Edition*. Complete Guides Series. Lakeview Research LLC, 2015. ISBN: 9781931448284.
- [43] FTDI. *Simplified Description of USB Device Enumeration*. Accessed 03 March 2023. Future Technology Devices International Ltd. 28th Oct. 2009. URL: https://www.ftdichip.com/Support/Documents/TechnicalNotes/TN_113_Simplified%20Description%20of%20USB%20Device%20Enumeration.pdf.

Appendix A

Purchasing lists

Table A.1: Preliminary hardware list












Part	Description	Quantity	Price per unit	Supplier	SKU	Web link
Common requirement						
SX1262 LoRa HAT	LoRa radio for the Raspberry Pi	1	28.10	PiHut	WAV-16806	Link 
500mAh LiPo battery		1	6.00	PiHut	105189	Link 
Transmitter Set 1						
Challenger RP2040	microcontroller with lora radio	1	21.5	PiHut	104944	Link 
GPS Featherwing	gps module (uart)	1	24.6	PiHut	ADA3133	Link 
Transmitter Set 2 - I ² C alternative						
Feather RP2040	microcontroller	1	11.4	PiHut	ADA4884	Link 
LoRa Featherwing	lora radio (i2c)	1	19.5	PiHut	ADA3231	Link 
GPS breakout	gps module (i2c)	1	29.7	PiHut	PIM525	Link 
uFL connector	to get an antenna onto the lora board	5	0.8	PiHut	ADA1661	Link 
Antenna Options						
W3013	ceramic	2	1.48	Mouser	673-W3013	Link 
PIOV008NRAA-100	strip with ufl	1	1.87	Mouser	523-PIOV008NRAA-100	Link 
318020612	Outdoor antenna - N male	1	30.8	Mouser	713-318020612	Link 

Table A.1: Preliminary hardware list (cont.)

Part	Description	Quantity	Price per unit	Supplier	SKU	Web link
	amazon option with included cable and adapter	1	30.99	Amazon		Link ↗
CAB.951	N female - SMA male, 1m	1	17.13	Mouser	960-CAB.951	Link ↗
	much cheaper amazon option	1	11.98	Amazon		Link ↗
Pigtail Antenna	only if none of the other options work	2	4.9	PiHut	105078	Link ↗
Raspberry Pi	it's okay I actually own a Pi3	1	219.99	Amazon		Link ↗

Table A.2: Updated hardware list









=:

Part	Description	Quantity	Price per unit	Supplier	SKU	Web link
Transmitter						
3178	Feather M0 LoRa	1	30.76	Mouser	485-3178	Link ↗
3133	GPS Featherwing	1	21.96	Mouser	485-3133	Link ↗
	Pimoroni equivalent	1	24.6	Pimoroni	ADA3133	Link ↗
Receiver						
3231	LoRa Featherwing	1	18.68	Digikey	1528-1741-ND	Link ↗
	Equivalent	1	19.5	Pimoroni	ADA3231	Link ↗
	Bonnet OLED alternative	1	31.8	Pimoroni	ADA4074	Link ↗
	+ headers	1	2.1	Pimoroni	COM0003	Link ↗
Accessories						
2258	Pi Case	1	7	Mouser	485-2258	Link ↗
	Equivalent	1	7.8	Pimoroni	ADA2258	Link ↗
4410	lipo (jst) charger	1	5.24	Mouser	485-4410	Link ↗

Table A.2: Updated hardware list (cont.)

Part	Description	Quantity	Price per unit	Supplier	SKU	Web link
2830	stacking headers 1st level	1	1.1	Mouser	485-2830	Link
	Equivalent	1	1.1	Pimoroni	ADA2830	Link
2886	stacking headers base	1	0.836	Mouser	485-2886	Link
	Equivalent	1	1.2	Pimoroni	ADA2886	Link
Battery						
4714	JST PH Female jumper	1	0.836	Mouser	485-4714	Link
RS PRO battery	LiPo 1.8Ah	1	10.86	RS	144-9405	Link
Alternative	1.2mAh, doesn't require jumper or housing	1	9.9	Pimoroni	BAT00044	Link
PHR-2	JST PH Female connector housing	1	0.66	RS	820-1466	Link
Antenna						
ANT-868-CPA	ceramic	1	2.97	Mouser	712-ANT-868-CPA	Link
PIOV008NRAA-100	strip with ufl	1	1.87	Mouser	523-PIOV008NRAA-100	Link
318020612	Outdoor antenna - N male	1	30.8	Mouser	713-318020612	Link
CAB.951	N female - SMA male, 1m	1	17.13	Mouser	960-CAB.951	Link
CONUFL001-SMD-T	uFL connector	5	0.625	Mouser	712-CONUFL001-SMD-T	Link
CASMA-UFL-1	uFL to SMA F	2	8.71	Mouser	125-CASMA-UFL-1	Link
	Equivalent	2	3.9	Pimoroni	ADA851	Link
Additional Antennae						
W3214	ceramic	1	2.08	Mouser	673-W3214	Link
M620720	ceramic	1	1.73	Mouser	581-M620720	Link
W3013	ceramic	1	1.48	Mouser	673-W3013	Link

Table A.3: Antennae list

Description	Part	Name	Price	Supplier	SKU	Web link
Original antenna:	318020612	Outdoor antenna - N male	30.8	Mouser	713-318020612	Link 
Suggested replacement:	318020690	5.8dBi antenna	41.24	Farnell	4060414	Link 
Cheaper alternative:	318020708	3dBi antenna	24.72	Farnell	4060419	Link 
Lora antenna	211140-0100	0.3dBi at 868, 38mm	1.7	Farnell	3498957	Link 
Additional lora antenna	1002289F0-AA10L0200	1.8 dBi	3.95	Farnell	3407000	Link 
gps antenna	9000440		1.79	Farnell	3407009	Link 
additional gps antenna if within budget	APKD1507G2-0100S		6.83	Farnell	3924367	Link 
alternative selected lora	206764-0100		3.31	Farnell	2885764	Link 

Appendix B

Data

B.1 Power Consumption

Table B.1: Power meter recordings

Timestamp	Consumption (mAh)
21/01/2023 19:46	0
21/01/2023 22:43	1
22/01/2023 13:40	8
22/01/2023 16:58	10
22/01/2023 18:38	11
22/01/2023 21:31	12
22/01/2023 22:26	13
23/01/2023 11:00	19
23/01/2023 14:40	21
23/01/2023 19:39	23

Table B.2: Discharge data sample - 230213.csv

Timestamp	Datetime	Packet	Longitude	Latitude	Altitude	Fix	Voltage
1676326772.56411	13-02-23 22:19:32	11	5223.████	133.████	87.10	0	4.183
1676326777.5640159	13-02-23 22:19:37	12	5223.████	133.████	87.10	1	4.183
1676326782.5647342	13-02-23 22:19:42	12	5223.████	133.████	87.10	0	4.189
1676326787.564926	13-02-23 22:19:47	12	5223.████	133.████	87.10	0	4.183
...							
1676380803.0929441	14-02-23 13:20:03	10188	5223.████	133.████	69.10	0	3.564
1676380808.0946279	14-02-23 13:20:08	10188	5223.████	133.████	69.10	0	3.557
1676380813.0988088	14-02-23 13:20:13	10188	5223.████	133.████	69.10	0	3.564
1676380818.1010823	14-02-23 13:20:18	10188	5223.████	133.████	69.10	0	3.564

Table B.4: Charge data sample - 230214.csv

Timestamp	Datetime	Packet	Longitude	Latitude	Altitude	Fix	Voltage
1676392659.3032475	14-02-23 16:37:39	0	0.0000	0.0000	0.00	0	1.617
1676392664.3044226	14-02-23 16:37:44	0	0.0000	0.0000	0.00	0	1.695
1676392669.3057065	14-02-23 16:37:49	0	0.0000	0.0000	0.00	0	1.772
1676392674.305263	14-02-23 16:37:54	0	0.0000	0.0000	0.00	0	1.830
...							
1676428057.1960237	15-02-23 02:27:37	6260	5223.████	133.████	87.50	1	4.131
1676428062.1964362	15-02-23 02:27:42	6261	5223.████	133.████	87.50	1	4.131
1676428067.197315	15-02-23 02:27:47	6262	5223.████	133.████	87.60	1	4.131
1676428072.1974156	15-02-23 02:27:52	6263	5223.████	133.████	87.60	1	4.137

B.2 Distance

Listing B.1: Distance Test 1 sample - 2023-02-17_17_Feb_2023_1_41_52_pm.kml

```
1 <gx:MultiTrack>
2   <altitudeMode>absolute</altitudeMode>
3   <gx:interpolate>0</gx:interpolate>
4   <gx:Track>
5     <gx:coord>-1.56132623 52.38261212 62.51</gx:coord>
6     <when>2023-02-17T13:44:48Z</when>
7     <gx:coord>-1.56123128 52.38268469 127.56</gx:coord>
8     <when>2023-02-17T13:45:10Z</when>
9     <gx:coord>-1.56129532 52.38279329 126.44</gx:coord>
10    <when>2023-02-17T13:45:14Z</when>
11    <gx:coord>-1.56138629 52.38286398 127.49</gx:coord>
12    <when>2023-02-17T13:45:20Z</when>
13    <gx:coord>-1.56153087 52.38289276 127.94</gx:coord>
14    <when>2023-02-17T13:45:31Z</when>
15    <gx:coord>-1.56166255 52.38283815 128.35</gx:coord>
16    ...
17  </gx:Track>
18 </gx:MultiTrack>
```

Table B.6: Distance Test 1 sample - first_test.csv

Datetime	Latitude	Longitude
16/02/23 23:33	52'22.958	-1'33.76
16/02/23 23:33	52'22.959	-1'33.7594
16/02/23 23:33	52'22.9565	-1'33.7566
16/02/23 23:33	52'22.9561	-1'33.756
...		
16/02/23 23:51	52'22.9482	-1'33.7032
16/02/23 23:51	52'22.9463	-1'33.7058
16/02/23 23:51	52'22.9463	-1'33.7058
16/02/23 23:51	52'22.9463	-1'33.7058

Table B.8: Distance Test 2 sample - second_test.csv

Datetime	Latitude	Longitude
17/02/23 0:48	52'22.9409	-1'33.7407
17/02/23 0:48	52'22.9409	-1'33.7407
17/02/23 0:48	52'22.9409	-1'33.7404
17/02/23 0:48	52'22.9409	-1'33.7403
...		
17/02/23 1:16	52'22.9502	-1'33.7284
17/02/23 1:16	52'22.9497	-1'33.73
17/02/23 1:16	52'22.9497	-1'33.7307
17/02/23 1:16	52'22.9502	-1'33.7317

Appendix C

Code

C.1 Power scripts

Listing C.1: Power meter graphing script

```
1 ds = [  
2     datetime("21-Jan-2023 19:46:00")  
3     datetime("21-Jan-2023 22:43:00")  
4     datetime("22-Jan-2023 13:40:00")  
5     datetime("22-Jan-2023 16:58:00")  
6     datetime("22-Jan-2023 18:38:00")  
7     datetime("22-Jan-2023 21:31:00")  
8     datetime("22-Jan-2023 22:26:00")  
9     datetime("23-Jan-2023 11:00:00")  
10    datetime("23-Jan-2023 14:40:00")  
11    datetime("23-Jan-2023 19:39:00")  
12 ];  
13  
14 con = [  
15     0  
16     1  
17     8  
18     10  
19     11  
20     12
```

```

21     13
22     19
23     21
24     23
25 ];
26
27 dsv = datenum(ds); % convert to number type for line of best fit calculations
28
29 coeff = polyfit(dsv,con,1); % line of best fit coefficients
30 lobfx = linspace(min(dsv), max(dsv), 1000); % x data points
31 lobfy = polyval(coeff, lobfx); % y data points
32
33 lobfx = datetime(lobfx, 'ConvertFrom', 'datenum'); % convert back to date type
34
35 figure(1);
× 36 plot(ds,con, '.', 'markersize', 15);
37 grid on;
38 hold on;
39 plot(lobfx, lobfy, 'LineWidth', 2, 'Color', '#388f58');
40 ylabel("Consumption (mAh)");
41 xlabel("Timestamp");
42 title("Power meter readings");
43 ylim([-1,25])
44 hold off;

```

Listing C.2: Power consumption graphing script v1

```

1 import csv
2 import matplotlib.pyplot as plt
3 from datetime import datetime as dt
4 import seaborn as sns
5 sns.set_style("whitegrid")

```

```

6
7 blue, = sns.color_palette("muted", 1)
8
9 timestamps_1 = []
10 level_1 = []
11 fix_1 = []
12
13 with open('Tests\\Battery\\data\\230213.csv', 'r') as csv_file:
14     reader = csv.reader(csv_file)
15     for row in reader:
16         timestamps_1.append(dt.strptime(row[1], '%d-%m-%y %H:%M:%S'))
17         level_1.append(float(row[7]))
18         fix_1.append(int(row[6]))
19 with open('Tests\\Battery\\data\\230214-old.csv', 'r') as csv_file:
20     reader = csv.reader(csv_file)
21     for row in reader:
22         timestamps_1.append(dt.strptime(row[1], '%d-%m-%y %H:%M:%S'))    #13-02-23 22:19:32
23         level_1.append(float(row[7]))
24         fix_1.append(int(row[6]))
25
26
27 fig, (ax1, ax2) = plt.subplots(2, 1)
28 fig.subplots_adjust(hspace=0.5)
29
30 ax1.plot_date(timestamps_1, level_1, fmt='g', lw=2, tz='UTC', xdate=True, ydate=False, label = "Battery level")
31
32 # plt.xticks(rotation = 25)
33 # plt.xlabel('Timestamps')
34 # plt.ylabel('Voltage (V)')
35 # plt.title('Battery discharge')
36 # plt.grid()

```

```
37 # plt.legend()
38
39
40 ax1.plot(timestamps_1, fix_1, color=blue, label = "GPS fix quality")
41 ax1.fill_between(timestamps_1, 0, fix_1, alpha=.3)
42
43
44 # plt.grid()
45
46
47 timestamps_2 = []
48 level_2 = []
49 fix_2 = []
50
51 with open('Tests\\Battery\\data\\230214.csv', 'r') as csv_file:
52     reader = csv.reader(csv_file)
53     for row in reader:
54         timestamps_2.append(dt.strptime(row[1], '%d-%m-%y %H:%M:%S'))
55         level_2.append(float(row[7]))
56         fix_2.append(int(row[6]))
57
58 # plt.subplot(2, 1, 2)
59 ax2.plot(timestamps_2, level_2, color = 'g', lw=2, label = "Battery level")
60
61 # plt.xticks(rotation = 25)
62 # plt.xlabel('Timestamps')
63 # plt.ylabel('Voltage (V)')
64 # plt.title('Battery charge')
65 # plt.grid()
66 # plt.legend()
67
```

```

68 ax2.plot(timestamps_2, fix_2, color=blue, label = "GPS fix quality")
69 ax2.fill_between(timestamps_2, 0, fix_2, alpha=.3)
70
71 # plt.grid()
72
73 # plt.suptitle("Battery charge curves")
74
75 plt.show()

```

Listing C.3: Power consumption graphing script v2

```

1 import pandas as pd
2 import matplotlib.pyplot as plt
3 import seaborn as sns
4 from matplotlib.dates import DateFormatter
5 from datetime import datetime as dt
6
7 sns.set_style("whitegrid")
8
9 blue, = sns.color_palette("muted", 1)
10
11 def legend_merge(plot1, plot2, loc, axis1, axis2):
12     handle1, label1 = plot1.get_legend_handles_labels()
13     handle2, label2 = plot2.get_legend_handles_labels()
14     axis1.legend(loc=loc, handles=handle1+handle2, labels = label1 + label2)
15     axis2.get_legend().remove()
16
17 #read data into dataframes
18 # merging two csv files
19 df_discharge = pd.concat(map(pd.read_csv, ['Tests\\Battery\\data\\230213.csv', 'Tests\\Battery\\data\\230214-old.csv'
20     ]), ignore_index=True)
21 df_discharge['Datetime'] = pd.to_datetime(df_discharge['Datetime'], format='%d-%m-%y %H:%M:%S')

```

```

21
22 df_charge = pd.read_csv('Tests\\Battery\\data\\230214.csv', parse_dates=['Datetime'], dayfirst=True)
23
24 # subplot with shared x
25 fig, ax = plt.subplots(nrows=2, ncols=1, figsize=(10, 6))
26 ax00 = ax[0]
27 ax01 = ax[0].twinx()      #twin the axis
28 ax10 = ax[1]
29 ax11 = ax[1].twinx()
30
31 #first subplot
32 d_level = sns.lineplot(data=df_discharge, x='Datetime', y='Voltage', color='g', lw='2', ax=ax00, label='Voltage')
33 d_fix = sns.lineplot(data=df_discharge, x='Datetime', y='Fix', ax=ax01, color=blue, label='Fix')
34
35 #styling
36 legend_merge(d_level, d_fix, 0, ax00, ax01)
37 ax01.fill_between(df_discharge['Datetime'], 0, df_discharge['Fix'], alpha=0.3)
38 #limits
39 ax00.set_ylim(1.5,4.5)
40 ax01.set_ylim(0,2)
41 ax01.set_yticks([0,1,2])
42 ax00.set_title('Discharge curve')
43
44 #second subplot
45 c_level = sns.lineplot(data=df_charge, x='Datetime', y='Voltage', color='g', lw='2', ax=ax10, label='Voltage')
46 c_fix = sns.lineplot(data=df_charge, x='Datetime', y='Fix', ax=ax11, color=blue, label='Fix')
47
48 #styling
49 legend_merge(c_level, c_fix, 1, ax10, ax11)
50 ax11.fill_between(df_charge['Datetime'], 0, df_charge['Fix'], alpha=0.3)
51 #limits

```



```

52 ax10.set_ylim(1.5,4.5)
53 ax11.set_ylim(0,2)
54 ax11.set_yticks([0,1,2])
55 ax10.set_title('Charge curve')
56
57 #figure styling
58 ax00.set_ylabel('Voltage (V)')
59 ax10.set_ylabel('Voltage (V)')
60 ax00.set_xlabel('Time')
61 ax10.set_xlabel('Time')
62 fig.suptitle(t='Battery charge curves', ha='center', size='20')
63 fig.tight_layout(pad=2.0)
64 # x time formatting
65 fmt = DateFormatter("%H:%M")
66 ax10.xaxis.set_major_formatter(fmt)
X 67 ax00.xaxis.set_major_formatter(fmt)
<
68
69 # annotations
70 ax00.annotate('13:20\n3.56V', xy=(dt(2023,2,14,13,20,18), 3.564), xycoords='data',
71               xytext=(0, -20), textcoords='offset points',
72               size=12, ha='center', va="top",
73               bbox=dict(boxstyle="round", alpha=0.2),
74               arrowprops=dict(arrowstyle="wedge,tail_width=0.5", alpha=0.2))
75 ax00.annotate('22:19\n4.18V', xy=(dt(2023,2,13,22,19,32), 4.183), xycoords='data',
76               xytext=(0, -20), textcoords='offset points',
77               size=12, ha='center', va="top",
78               bbox=dict(boxstyle="round", alpha=0.2),
79               arrowprops=dict(arrowstyle="wedge,tail_width=0.5", alpha=0.2))
80
81 ax10.annotate('02:27\n4.14V', xy=(dt(2023,2,15,2,27,52), 4.137), xycoords='data',
82               xytext=(0, -20), textcoords='offset points',

```

```

83         size=12, ha='center', va="top",
84         bbox=dict(boxstyle="round", alpha=0.2),
85         arrowprops=dict(arrowstyle="wedge,tail_width=0.5", alpha=0.2))
86 ax10.annotate('16:37\n1.62V', xy=(dt(2023,2,14,16,37,39), 1.617), xycoords='data',
87             xytext=(20, 20), textcoords='offset points',
88             size=12, ha='left', va="center",
89             bbox=dict(boxstyle="round", alpha=0.2),
90             arrowprops=dict(arrowstyle="wedge,tail_width=0.5", alpha=0.2))
91 ax10.annotate('22:11\n4.19V', xy=(dt(2023,2,14,22,11,30), 4.189), xycoords='data',
92             xytext=(0, -20), textcoords='offset points',
93             size=12, ha='center', va="top",
94             bbox=dict(boxstyle="round", alpha=0.2),
95             arrowprops=dict(arrowstyle="wedge,tail_width=0.5", alpha=0.2))
96
97 plt.show()

```

XVI

C.2 Distance scripts

Listing C.4: Location format converter

```

1 import re
2 import csv
3
4 # take a coordinate as a string and split it into degrees and minutes
5 def splitter(coordinate_string):
6     match_result = re.search("(\\d{2})\\.\\.*", coordinate_string) #match minutes
7     return [coordinate_string[0:match_result.span()[0]], match_result.group(0)] #return [degrees, minutes]
8
9 #reformat coordinate into standardised system
10 def coord_rewrite(split_coord, direction):
11     d = ''

```

```

12     if direction == 'W' or direction == 'S':
13         d = '-'
14     return d + split_coord[0] + " " + split_coord[1]
15
16 with open('Tests\\Distance\\data\\230217-edit.csv') as csv_file:
17     with open('Tests\\Distance\\second_test.csv', 'w+', newline='') as write_file:
18         reader = csv.reader(csv_file)
19         writer = csv.writer(write_file)
20         writer.writerow(['datetime', 'latitude', 'longitude']) #headers
21         line_count = 0
22         for row in reader:
23             line_count += 1
24             #row0      row1      row2      row3      row4      row5      row6      row7      row8      row9
25             #timestamp  datetime  packet  lat      dir      long     dir      alt      fix      volt
26             datetime = row[1]
27             lat = row[3]
28             lat_dir = row[4]
29             long = row[5]
30             long_dir = row[6]
31
32             if line_count == 1:      #headers
33                 # print(f'{datetime}\\t{lat}\\t{long}')
34                 continue
35             # elif line_count == 5:
36             #     break
37             else:
38                 # print(f'{datetime}\\t{coord_rewrite(splitter(lat), lat_dir)}\\t{coord_rewrite(splitter(long),
39                 long_dir)})')
40                 writer.writerow([datetime, coord_rewrite(splitter(lat), lat_dir), coord_rewrite(splitter(long),
41                 long_dir)])

```

C.3 Microcontroller sketches

Listing C.5: Example battery voltage

```
1 #define VBATPIN A7
2
3 void setup() {}
4
5 void loop() {
6     float measuredvbat = analogRead(VBATPIN);
7     measuredvbat = (measuredvbat*6.6)/1024;
8     Serial.print("VBat: "); Serial.println(measuredvbat);
9 }
```

Listing C.6: Example GPS by library

```
1
2 #include <SPI.h>
3 #include <RH_RF95.h>
4 #include <Adafruit_GPS.h>
5
6 #define GPSSerial Serial1
7 Adafruit_GPS GPS(&GPSSerial);
8
9 #define GPSECHO false
10
11 uint32_t timer = millis();
12
13 void setup() {
14     while (!Serial)
15         ;
16     Serial.begin(115200);
```

```
17 GPS.begin(9600);
18 // GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_RMCGGA);
19 GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_RMONLY);
20 // GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_ALLDATA);
21 // GPS.sendCommand(PMTK_SET_NMEA_UPDATE_1HZ);
22 GPS.sendCommand(PMTK_SET_NMEA_UPDATE_200_MILLIHERTZ); // 10 second update time
23 // GPS.sendCommand(PGCMD_ANTENNA);
24 }
25
26 void loop() {
27     char c = GPS.read();
28     // if (GPSECHO) {
29     //     if (c) {
30     //         Serial.print(c);
31     //     }
32     // }
33
34     if (GPS.newNMEAreceived()) {
35         // Serial.print(GPS.lastNMEA());
36         if (!GPS.parse(GPS.lastNMEA())) {
37             return;
38         }
39     }
40     if (millis() - timer > 2000) {
41         timer = millis();
42
43         if (GPS.fix) {
44             String output = String(GPS.latitude,4) + GPS.lat + ", " + String(GPS.longitude,4) + GPS.lon;
45             Serial.println(output);
46         }
47     }
```

48 }

Listing C.7: Example GPS by serial

```
1 // Test code for Ultimate GPS Using Hardware Serial
2 // (e.g. GPS for Leonardo, Flora or FeatherWing)
3 //
4 // This code shows how to test a passthru between USB and hardware serial
5 //
6 // Tested and works great with the Adafruit GPS FeatherWing
7 // -----> https://www.adafruit.com/products/3133
8 // or Flora GPS
9 // -----> https://www.adafruit.com/products/1059
10 // but also works with the shield, breakout
11 // -----> https://www.adafruit.com/products/1272
12 // -----> https://www.adafruit.com/products/746
13 //
14 // Pick one up today at the Adafruit electronics shop
15 // and help support open source hardware & software! -ada
16
17
18 // what's the name of the hardware serial port?
19 #define GPSSerial Serial1
20
21
22 void setup() {
23   // make this baud rate fast enough to we aren't waiting on it
24   Serial.begin(115200);
25
26   // wait for hardware serial to appear
27   while (!Serial) delay(10);
28
```

XX

```

29 // 9600 baud is the default rate for the Ultimate GPS
30 GPSSerial.begin(9600);
31 }
32
33
34 void loop() {
35     if (GPSSerial.available()) {
36         char c = GPSSerial.read();
37         Serial.write(c);
38     }
39 }

```

Listing C.8: LoRa with GPS

```

1
2 #include <SPI.h>
3 #include <RH_RF95.h>
4 #include <Adafruit_GPS.h>
5
6 #define RFM95_CS      8
7 #define RFM95_INT     3
8 #define RFM95_RST     4
9
10 #define VBATPIN  A7
11
12 #define GPSSerial Serial1
13 Adafruit_GPS GPS(&GPSSerial);
14
15 #define RF95_FREQ 868.0
16
17 RH_RF95 rf95(RFM95_CS, RFM95_INT);
18

```

XXI

```
19 uint32_t timer = millis();
20 int count = 0;
21 int length = 0;
22
23 void setup() {
24     // make this baud rate fast enough to we aren't waiting on it
25     Serial.begin(115200);
26
27     pinMode(RFM95_RST, OUTPUT);
28     digitalWrite(RFM95_RST, HIGH);
29
30     // wait for hardware serial to appear
31     // while (!Serial) delay(10);
32
33     // manual reset
34     digitalWrite(RFM95_RST, LOW);
35     delay(10);
36     digitalWrite(RFM95_RST, HIGH);
37     delay(10);
38
39     while (!rf95.init()) {
40         Serial.println("LoRa radio init failed");
41         Serial.println("Uncomment '#define SERIAL_DEBUG' in RH_RF95.cpp for detailed debug info");
42         while (1);
43     }
44
45     if (!rf95.setFrequency(RF95_FREQ)) {
46         Serial.println("setFrequency failed");
47         while (1);
48     }
49 }
```


XXiii

```
50 rf95.setTxPower(23, false);
51
52 // 9600 baud is the default rate for the Ultimate GPS
53 GPS.begin(9600);
54
55
56 GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_RMCGGA);
57
58
59 GPS.sendCommand(PMTK_SET_NMEA_UPDATE_200_MILLIHERTZ); // 5 second update time
60
61
62
63
64 }
65
66 void loop() {
67     float measuredvbat = analogRead(VBATPIN);
68     measuredvbat = (measuredvbat*6.6)/1024;
69     char c = GPS.read();
70
71     if (GPS.newNMEAreceived()) {
72         // Serial.print(GPS.lastNMEA());
73         if (!GPS.parse(GPS.lastNMEA())) {
74             return;
75         }
76     }
77
78     if (millis() - timer > 5000) {
79         timer = millis();
80     }
```

```

81     if (GPS.fix) {
82         count++;
83     }
84     String output = String(count) + "," + String(GPS.latitude,4) + "," + String(GPS.longitude,4) + "," + String(GPS.
altitude) + "," + String(GPS.fixquality) + "," + String(measuredvbat, 4);
85     length = output.length();
86     char send[length];
87     output.toCharArray(send, length);
88     Serial.println(send);
89     rf95.send((uint8_t *)send, length);
90
91 }
92
93
94 }

```

xxiv

C.4 SBC scripts

Listing C.9: Example test

```

1 # SPDX-FileCopyrightText: 2018 Brent Rubell for Adafruit Industries
2 #
3 # SPDX-License-Identifier: MIT
4
5 """
6 Wiring Check, Pi Radio w/RFM9x
7
8 Learn Guide: https://learn.adafruit.com/lorawan-for-raspberry-pi
9 Author: Brent Rubell for Adafruit Industries
10 """
11 import time

```

XXV

```
12 import busio
13 from digitalio import DigitalInOut, Direction, Pull
14 import board
15 # Import the SSD1306 module.
16 import adafruit_ssd1306
17 # Import the RFM9x radio module.
18 import adafruit_rfm9x
19
20 # Button A
21 btnA = DigitalInOut(board.D5)
22 btnA.direction = Direction.INPUT
23 btnA.pull = Pull.UP
24
25 # Button B
26 btnB = DigitalInOut(board.D6)
27 btnB.direction = Direction.INPUT
28 btnB.pull = Pull.UP
29
30 # Button C
31 btnC = DigitalInOut(board.D12)
32 btnC.direction = Direction.INPUT
33 btnC.pull = Pull.UP
34
35 # Create the I2C interface.
36 i2c = busio.I2C(board.SCL, board.SDA)
37
38 # 128x32 OLED Display
39 reset_pin = DigitalInOut(board.D4)
40 display = adafruit_ssd1306.SSD1306_I2C(128, 32, i2c, reset=reset_pin)
41 # Clear the display.
42 display.fill(0)
```

```

43 display.show()
44 width = display.width
45 height = display.height
46
47 # Configure RFM9x LoRa Radio
48 CS = DigitalInOut(board.CE1)
49 RESET = DigitalInOut(board.D25)
50 spi = busio.SPI(board.SCK, MOSI=board.MOSI, MISO=board.MISO)
51
52 while True:
53     # Clear the image
54     display.fill(0)
55
56     # Attempt to set up the RFM9x Module
57     try:
58         rfm9x = adafruit_rfm9x.RFM9x(spi, CS, RESET, 915.0)
59         display.text('RFM9x: Detected', 0, 0, 1)
60     except RuntimeError as error:
61         # Thrown on version mismatch
62         display.text('RFM9x: ERROR', 0, 0, 1)
63         print('RFM9x Error: ', error)
64
65     # Check buttons
66     if not btnA.value:
67         # Button A Pressed
68         display.text('Ada', width-85, height-7, 1)
69         display.show()
70         time.sleep(0.1)
71     if not btnB.value:
72         # Button B Pressed
73         display.text('Fruit', width-75, height-7, 1)

```

```

74     display.show()
75     time.sleep(0.1)
76     if not btnC.value:
77         # Button C Pressed
78         display.text('Radio', width-65, height-7, 1)
79         display.show()
80         time.sleep(0.1)
81
82     display.show()
83     time.sleep(0.1)

```

Listing C.10: Example radio

```

1 # SPDX-FileCopyrightText: 2018 Brent Rubell for Adafruit Industries
2 #
3 # SPDX-License-Identifier: MIT
4
5 """
6 Example for using the RFM9x Radio with Raspberry Pi.
7
8 Learn Guide: https://learn.adafruit.com/lora-and-lorawan-for-raspberry-pi
9 Author: Brent Rubell for Adafruit Industries
10 """
11 # Import Python System Libraries
12 import time
13 # Import Blinky Libraries
14 import busio
15 from digitalio import DigitalInOut, Direction, Pull
16 import board
17 # Import the SSD1306 module.
18 import adafruit_ssd1306
19 # Import RFM9x

```

XXViii

```
20 import adafruit_rfm9x
21
22 # Button A
23 btnA = DigitalInOut(board.D5)
24 btnA.direction = Direction.INPUT
25 btnA.pull = Pull.UP
26
27 # Button B
28 btnB = DigitalInOut(board.D6)
29 btnB.direction = Direction.INPUT
30 btnB.pull = Pull.UP
31
32 # Button C
33 btnC = DigitalInOut(board.D12)
34 btnC.direction = Direction.INPUT
35 btnC.pull = Pull.UP
36
37 # Create the I2C interface.
38 i2c = busio.I2C(board.SCL, board.SDA)
39
40 # 128x32 OLED Display
41 reset_pin = DigitalInOut(board.D4)
42 display = adafruit_ssd1306.SSD1306_I2C(128, 32, i2c, reset=reset_pin)
43 # Clear the display.
44 display.fill(0)
45 display.show()
46 width = display.width
47 height = display.height
48
49 # Configure LoRa Radio
50 CS = DigitalInOut(board.CE1)
```

XXIX

```

51 RESET = DigitalInOut(board.D25)
52 spi = busio.SPI(board.SCK, MOSI=board.MOSI, MISO=board.MISO)
53 rfm9x = adafruit_rfm9x.RFM9x(spi, CS, RESET, 868.0)
54 rfm9x.tx_power = 23
55 prev_packet = None
56
57 while True:
58     packet = None
59     # draw a box to clear the image
60     display.fill(0)
61     display.text('RasPi LoRa', 35, 0, 1)
62
63     # check for packet rx
64     packet = rfm9x.receive()
65     if packet is None:
66         display.show()
67         display.text('- Waiting for PKT -', 15, 20, 1)
68     else:
69         # Display the packet text and rssi
70         display.fill(0)
71         prev_packet = packet
72         try:
73             packet_text = str(prev_packet, "utf-8")
74         except UnicodeDecodeError:
75             print('parse error')
76         except:
77             print('other error')
78         display.text('RX: ', 0, 0, 1)
79         print('RX: ' + packet_text)
80         display.text(packet_text, 25, 0, 1)
81         time.sleep(1)

```

XXX

```

82
83     if not btnA.value:
84         # Send Button A
85         display.fill(0)
86         button_a_data = bytes("Button A!\r\n","utf-8")
87         rfm9x.send(button_a_data)
88         display.text('Sent Button A!', 25, 15, 1)
89     elif not btnB.value:
90         # Send Button B
91         display.fill(0)
92         button_b_data = bytes("Button B!\r\n","utf-8")
93         rfm9x.send(button_b_data)
94         display.text('Sent Button B!', 25, 15, 1)
95     elif not btnC.value:
96         # Send Button C
97         display.fill(0)
98         button_c_data = bytes("Button C!\r\n","utf-8")
99         rfm9x.send(button_c_data)
100        display.text('Sent Button C!', 25, 15, 1)
101
102
103    display.show()
104    time.sleep(0.1)

```

Listing C.11: First test

```

1 import busio
2 from digitalio import DigitalInOut, Direction, Pull
3 import board
4 import adafruit_rfm9x
5 from datetime import datetime
6

```


XXI.

```

7 # Create the I2C interface.
8 i2c = busio.I2C(board.SCL, board.SDA)
9
10 # Configure LoRa Radio
11 CS = DigitalInOut(board.CE1)
12 RESET = DigitalInOut(board.D25)
13 spi = busio.SPI(board.SCK, MOSI=board.MOSI, MISO=board.MISO)
14 rfm9x = adafruit_rfm9x.RFM9x(spi, CS, RESET, 868.0)
15 rfm9x.tx_power = 23
16
17 prev_packet = None
18
19 # timestamped log file for incoming packets
20 file = open((datetime.now()).strftime("%d%m%y.txt"), 'w+')
21 print("Start")
22
23 try:
24     while True:
25         packet = None
26         packet = rfm9x.receive()
27         if packet is not None:
28             prev_packet = packet
29             try:
30                 # create timestamp
31                 now = datetime.now()
32                 date_time = now.strftime("%y/%m/%d,%H:%M:%S")
33
34                 packet_text = str(prev_packet, "utf-8")
35                 text = (date_time + ", " + packet_text)
36             except UnicodeDecodeError:
37                 # log timestamp of invalid packet

```

```

38         print(date_time + ", " + "unicode parse error", file=file, flush=True)
39         # log accepted packet
40         print(text, file=file, flush=True)
41 # gracefully exit on ^C command
42 except KeyboardInterrupt:
43     print("\nTerminating")
44     file.close()
45     raise SystemExit

```

Listing C.12: Battery test

```

1
2 # Import Blinky Libraries
3 import busio
4 from digitalio import DigitalInOut, Direction, Pull
5 import board
6 # Import RFM9x
7 import adafruit_rfm9x
8 from datetime import datetime
9 import time
10
11 # Create the I2C interface.
12 i2c = busio.I2C(board.SCL, board.SDA)
13
14 # Configure LoRa Radio
15 CS = DigitalInOut(board.CE1)
16 RESET = DigitalInOut(board.D25)
17 spi = busio.SPI(board.SCK, MOSI=board.MOSI, MISO=board.MISO)
18 rfm9x = adafruit_rfm9x.RFM9x(spi, CS, RESET, 868.0)
19 prev_packet = None
20 file = open((datetime.now()).strftime("%y%m%d.csv"), 'w+')
21 print("Start")

```

XXX:
||

XXXXIII

```
22
23 try:
24     while True:
25         packet = None
26         packet = rfm9x.receive()
27         if packet is not None:
28             prev_packet = packet
29             try:
30                 now = datetime.now()
31                 date_time = now.strftime("%d-%m-%y %H:%M:%S")
32                 timestamp = str(time.time())
33
34                 packet_text = (str(prev_packet, "utf-8")).rstrip('\x00')
35                 text = (timestamp + "," + date_time + "," + packet_text)
36
37             except Exception as e:
38                 print(date_time)
39                 print(e)
40                 print("data" + str(prev_packet))
41                 print(text, file=file, flush=True)
42 except KeyboardInterrupt:
43     print("\nTerminating")
44     file.close()
45     raise SystemExit
```

Appendix D

Errors

Listing D.1: Decoding error

^|XXX

```
1 Start
2 13-02-23 19:50:11
3 'utf-8' codec can't decode byte 0x9f in position 0: invalid start byte
4 bytearray(b'\x9f\x88\x11\xde\x9c\xfb\x89\\22Rs\n\x95\x9a\x95')
5 13-02-23 20:12:48
6 'utf-8' codec can't decode byte 0xe3 in position 1: invalid continuation byte
7 bytearray(b'=\xe3\xc1\x90\xef\x8b\x04.\xc4i\xa9\xbe\xab\x07\xb2q\xc24\xbd\xec\xc1\xb54H`\x9c#k\xaeG\x92\x0f\x17\
   xdd\xaf\r\x00\xa5V \x7f\xb0\xaa\xb0\xb1t6;d\x8c\xc1\xd8\xe6]\xef\x15d\xb9\x0c\xae\x89R\xdbTF\x0b\xa5%=o\x96')
8 13-02-23 20:49:27
9 'utf-8' codec can't decode byte 0xc2 in position 0: invalid continuation byte
10 bytearray(b'\xc2\xed\xa9\xff\x8c\xcf\xfa`X\xcf\x88I\xa5\xa6\x15\x7f\x9b\xa7~c\xbf\xac')
11 13-02-23 20:51:02
12 'utf-8' codec can't decode byte 0xb2 in position 0: invalid start byte
13 bytearray(b'\xb2\r\x9aLqR\x90w\xb8B\xfd\xb8\xd8\x9b\xc2\xb5U\x1a\x9e;\xda\x02v\xdb\x95`-&\xae+\x92\xd2\x99\xd5\
   xa2\x16\x04\x1b\xa0<\t2\x9e\xdc3g$\xf5I\xe6\x13\x8aJ\xfa\x85\xb3v+\xa9\x03He\x16\xb8\xad\xdf')
14
```