Smart Bike Light Project

34346 Networking Technologies and application development for Internet of Things (IoT)

May 15 2025

s241750	Corinne Fogarty Draper
s233360	Mariana Sousa Pinho Santos
s215518	Nipun Fernando
s243363	Paul Martin Künnapuu
s221601	Sia Jæger Linde
s205073	Vandad Kolahi Azar

Group 8

Contents

1 Introduction

In Denmark, cycling is a popular form of transportation across all age groups. However, school-aged children faces the issue of forgetting to turn on their bike light or charge the bike light. This is a huge safety and security issue. In Denmark, it is estimated that 2200 accidents could be avoided if these bike lights were automated and more reliable. In addition, bikes disappearance and theft are also huge issues. Bike theft alone costs Danes 250 million DKK annually, which means that the need for integrated geolocation is crucial. [1]

To address these issues, the project aims to develop a Smart Bike Light that will improve safety for both cyclists and everyone else on the road. The Smart Bike Light combines automatic light activation and deactivation, monitoring of battery status as well as geolocation tracking, all enclosed in a custom 3D-printed case.

Users are able to obtain these real-time data remotely through a mobile application, such as the location of the bike and the status of the battery. This will be possible with the use of LoRaWAN. By combining these features, the bike light aims to reduce accidents, discourage theft and enhance the overall biking experience.

2 Overview of our project

To create a solution that addresses the issue discussed, we defined a set of specifications. These are shown in Table 1.

Specification	**Function**
ESP32-based microcontroller	Core unit handling processing and communication
Accelerometer-based motion detection	Detects movement for automatic light control
Light sensor (LDR)	Ensures the light only activates in low light
Manual button control	Allows user to manually switch the light on or off
Auto-off after 30 seconds	Conserves power when the bike is inactive
GNSS data	Provides precise geolocation data at trip start and stop and upon request
WiFi scanning	Provides geolocation data frequently during the trip
LoRaWAN communication	Wireless communication
Geolocation and battery data transmission	Enables tracking and system monitoring
Low-battery warning	Alerts the user through sound on 20% and 10% battery level
Shut-off switch	Completely shut-off the system for long battery life between charges
Rechargeable battery system	Enables off-grid, sustainable operation
USB charging (TP4056)	Convenient recharging of battery
3D printed enclosure	Protects electronics; mounts easily on bike
Field-test ready design	Prototype validated in real-world use

Table 1: Requirement Specifications and Functions

In a high-level overview, the system that we created to meet our requirements is schematised on Figure 1.

This diagram can be divided into 2 parts: the components that are physically inside the bike light enclosure – represented by the green box –, and those that are not. Besides being a broad overview of the system, the diagram also provides information on the user interface for the bike light. In section section 3 we will delve into the details of each component.

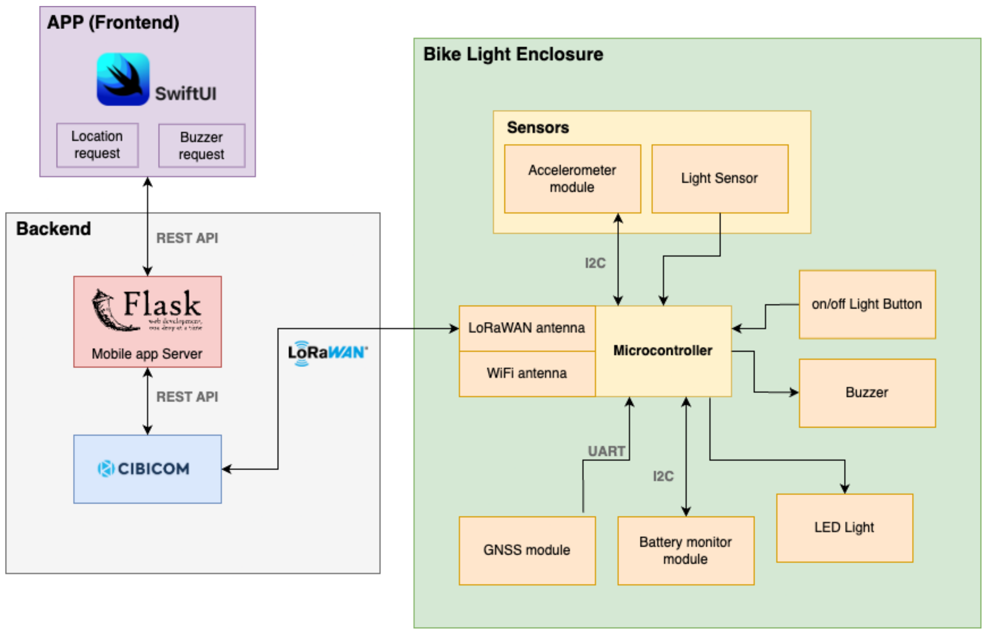

Figure 1: Block diagram of system overview.

2.1 Components inside enclosure

The microcontroller, shown in the middle of the green box, takes the role of the controller of the system, communicating with the peripheral devices. For this purpose, we used the Heltec board [2], which uses an ESP32C3 + SX1262 chip, and has integrated LoRa and WiFi capabilities.

To fulfill our requirements, the microcontroller communicates with an accelerometer module, a battery monitor module, and a GNSS module. The protocols used for communication are highlighted in the diagram. All of these components are hidden from the user inside the case. On the other hand, there are a number of elements exposed to the user: an on/off switch to power off the system, a button to override the automatic light switching, the light itself, a buzzer, a light sensor, and a charging port.

Communication to the outside is allowed through the use of the LoRa capabilities of the Heltec board, enabled by the LoRa antenna. The WiFi antenna, on the other hand, is used solely for geolocation purposes.

2.2 Components outside enclosure

Outside of the enclosure we have the online services or digital counterpart to our solution. Perhaps the most important is the Cibicom [cite] application. Cibicom works as an intermediate between LoRaWAN and WiFi. Not only does it send and receive data from the microcontroller, through LoRa, but it also allows this to be done through a web server that connects to it. A breadboard diagram was also created to illustrate the system layout. While it uses a generic ESP32 instead of the specific Heltec module (due to part availability in the Fritzing program), it still provides a clear overview of all components. The wiring may be imprecise, as the diagram was not the focus after we chose to proceed with a different microcontroller. (Appendix B Figure 2)

In the scope of this project, we developed an iOS application backed by a Flask web server.

3 System components and implementation

Having explored how the different components make up the system as a whole, in this section, we will delve into the specifics of the different parts of our solution and explain the reasoning behind our implementation choices. Table 2 shows a summary about the hardware components that we used for our project.

Component	Function	Reason
SG90	Small button	Manual activation or deactivation of the bike light
Photocell	Photoresistor	Detects light levels to enable automatic night-time activation
ADXL345	Motion sensor	Detects movement to trigger light activation and inactivity for auto shut-off
NEO-6M-0-001	GNSS module	Obtains precise GPS data from satellites
MAX17048	Battery measurement	Monitors real-time battery level and sends data to ESP32
TP4056	USB charging module	Charges lithium battery safely via micro-USB
MCP1826S-1802E-AB	Voltage regulator	Ensures stable voltage from battery to components
Lithium Battery 3.7V	Power supply	Portable energy source for the entire system
MPD BH-18650	Battery holder	Physically holds and connects the lithium battery

Table 2: Hardware components used

In the remaining of this section, we'll focus on the hardware used: why we chose it, and how we used it for our goal.

3.1 Battery management

We chose the MAX17048 fuel gauge for our smart bike light because it provides accurate, real-time battery state-of-charge (SoC) readings using Maxim's ModelGauge algorithm, which outperforms traditional analog ADC monitoring. Analog voltage readings can be misleading during load changes, while the MAX17048 internally samples open-circuit voltage with a 12-bit ADC and computes SoC without needing a sense resistor or complex Coulomb counting. Its compact size, ultra-low power consumption, and I2C interface make it ideal for space-constrained, battery-powered designs. The ESP32 communicates with the gauge by reading cell voltage and SoC percentage registers, allowing the microcontroller to retrieve precise battery data without performing learning cycles or additional processing—greatly simplifying the design while enhancing user experience.[3]

3.2 Light control

3.2.1 Accelerometer

The accelerometer used is the ADXL345 by analog devices. This accelerometer is a good choice for our application, as it is compact and has many features that help lower power consumption, which is important for IoT devices. The accelerometer can communicate using I^2C or SPI communication. We chose to use I^2C as it uses less wires and less power, which benefits our compact low power design. The interrupt pin is used to send an interrupt to the ESP32 when movement is detected. This removes the need for constant poling, which reduces power consumption. The accelerometer is set to full bit resolution for precision. This does not increase power consumption. As we do not need to be checking for movement too often, the accelerometer can always run in low power mode and run at the lowest rate. In low power mode, the lowest data rate is 12.5 Hz. At this data rate data is sent every 80 ms which is completely satisfactory for this application. This reduces current consumption to 40 microAmps, which is great for low power consumption. To facilitate programming, the `DFRobot_ADXL345` [4]. library was used. All accelerometer information was referenced from the manufacturer's datasheet [5].

3.2.2 Photo-resistor

The photo-resistor used is the Photosensitive Resistance LDR Sensor by Arduino. This is a simple resistor with a value that varies between 0.5 kohm and 0.5 Mohm depending on light [6]. The darker it is the higher the resistance. As the microcontroller can't measure the resistance itself, the photo-resistor is put in series with a 10 kohm resistor. The photo-resistor is connected to 3.3V and the resistor to ground. The point between the LED and resistor is connected to an analog pin. The pin essentially reads the voltage across the 10 kohm resistor. If the value is high, then the the lighting is bright. If the value is low, then it is dark. The threshold was determined by trial and error.

The 10 kohm resistor was chosen as its value falls in the operation range of the photo-resistor, which gives a good spread of analog values to measure. It is also large enough to reduce current, which reduces power consumption.

3.2.3 LED

The LED used is a typical white 10 mm LED. To protect the LED from pulling too much current, which can cause damage, the LED is in series with a 220 ohm resistor. This value offers a compromise between current consumption and brightness.

3.3 Geolocation

To obtain the geolocation of the bike light, two different techniques were employed: using a GNSS module to obtain precise location data, or scanning WiFi networks nearby and sending their information to an external API, such as Google Maps API, which will provide us with approximate coordinates.

3.3.1 GNSS module

We used the NEO-6M-0-001 [7] peripheral to obtain GPS data from satellites. It communicates with the microcontroller through UART, sending messages in the NMEA language, commonly used for GPS information. This data is converted into coordinates using the `TinyGPS++` [8] library.

This device allows us to geolocate the bike light with great precision. However, a few disadvantages could be pointed out about this method:

- It is very power intensive, and the peripheral does not offer any sleep mode or low energy option.
- From the moment it is powered on, it normally takes a long time until it provides useful GPS data.
- It does not work well indoors. Even though it is not a problem for the final product, it makes it difficult to test it during development.

From these, we'd like to highlight the fact that it takes a long time to produce valid data. Had we implemented a way to force shut-down the device – such as through the use of a transistor –, then we'd still have to keep the device on for a long time before it could be useful.

3.3.2 WiFi scanning

Since the Heltec board we used integrates WiFi capabilities, we used it to obtain approximate geolocation data. With the help of the `WiFi` [9] library, we could obtain information about the networks within reach. This includes the Basic Service Set Identifier (BSSID), the Received Signal Strength Indicator (RSSI), and the WiFi channel of each found network. This information, concerning the first 14 networks found, is sent through LoRaWAN to the Cibicom application. The web server will be responsible to communicate with the Google Maps API [10] to obtain actual approximate coordinates.

This method is significantly less power consuming than the GNSS module. However, there are a few disadvantages that can also be listed:

- It does not provide as accurate results as the GPS module.
- It only works if there are WiFi networks within reach, which might not be the case when the user is cycling in remote locations.

4 External communication and mobile application

The communication between the bike light and the external world is powered through LoRa. All the messages exchanged have the following format:

`operation code (1 byte)`	`data ? data : ∅ (up to 114 bytes)`

There are a total of 5 different operation codes employed in our solution, described in Table 6.

Operation code (HEX)	Sender → Receiver	Function	Data
10	Bike light → Mobile app	send GPS location	encoded coordinates
20	Bike light → Mobile app	send WiFi scanning data	encoded WiFi networks data
30	Bike light → Mobile app	send battery percentage	battery percentage
01	Mobile app → Bike light	request buzzer sound	none
20	Mobile app → Bike light	request GPS location	none

Table 3: Different message operations in our project.

4.1 Data encoding

All the messages sent by the Heltec board contain data that needed to be encoded into a number represented in hexadecimal notation.

4.1.1 GPS location

After decoding the NMEA sentences, we obtained the latitude and the longitude in the Decimal Degrees (DD) format, which consists of a simple decimal floating point number. Both the latitude and the longitude values vary between -90 and 90. The first step was to decide how many bytes were necessary to encode each coordinate. With 3 bytes, we would have an approximate precision of 0,00001, which is an appropriate measure for our problem. In Equation 1 we calculate this precision by dividing our range by the amount of numbers one can represent with 3 bytes. The conversion from each of the coordinates value into a 3 bytes value is shown on Equation 2: first we normalize the values so that they are always positive, we divide them by the largest possible value obtained, and then divide by the maximum value that 3 bytes can code. The decimal places are truncated so that we're left with an integer that can be directly put in a message.

$$180/(2^(8*3)) = 0,000010729 \approx 0,00001 \tag{1}$$

$$normalisedvalue = floor((((coordinatevalue + 90.0)/180.0) * 16777215.0)) \tag{2}$$

The final message compiles both the latitude and the longitude by concatenating each 3-bytes, such as described below.

`0x10`	`normalised latitude (3 bytes)`	`normalised longitude (3 bytes)`

4.1.2 WiFi scanning

The WiFi scanning data is much easier to encode into a number that can be sent over LoRa. The BSSID corresponds to 6 bytes, so it can be coded directly into the message. The absolute value of the RSSID corresponds to just 1 byte, as well as the channel, and so these were simply appended to the message. The message concatenates these values for 14 different networks, in sequence, such as schematised below:

`0x20`	`BSSID (6 bytes)`	`abs(RSSI) (1 byte)`	`channel (1 byte)`	`BSSID (6 bytes)`	`...`	`...`
	network 1			network 2		...

4.1.3 Battery level

We obtained a float value of the battery percentage directly from the peripheral. Because accuracy is not so important, we just ignored the fractional part and sent the integer part, which fits into 2 bytes.

`0x30`	`battery level in percentage (2 bytes)`

4.2 Web server

The server was built using Python and Flask, acting as the core communication layer between the mobile application, the LoRaWAN backend (Loriot), and external services such as the Google Geolocation API. It exposes four main endpoints:

- **/send-to-lorawan**: Accepts POST requests from the app containing hex-encoded commands (e.g., `"01"` to trigger the buzzer, `"02"` to request GPS location). The payload is forwarded to the Loriot backend using their REST API.
- **/loriot-uplink**: Receives uplink data from Loriot whenever the device sends information. The server parses the hex payload to determine its type:
  - `0x10`: GPS location (decoded from 3-byte normalized coordinates).
  - `0x20`: WiFi scan results (BSSID, RSSI, channel).
  - `0x30`: Battery level (as a percentage).

  If WiFi data is received, the server uses the Google Geolocation API to determine the device's approximate coordinates.
- **/location**: Returns the most recent location as a JSON object, either from GPS or WiFi-based estimation.
- **/battery**: Returns the latest known battery percentage as a JSON response.

Parsing is handled through isolated functions (`parse_gps_data()`, `parse_wifi_data()`, etc.) and all responses are logged for debugging. The use of both GPS and WiFi allows for flexible geolocation. The Flask server provides a minimal, clean REST API that efficiently bridges the device, backend, and frontend.

4.3 Mobile app

The mobile application was developed in SwiftUI for iOS. Swift was chosen due to the team's prior experience and the need for rapid prototyping. The app is divided into two primary tabs: **Location** and **Status**.

Status Tab This view displays the battery level and includes two buttons:

- **Request Buzzer**: Sends the hex payload `"01"` to the server, which forwards it to the device via Loriot.
- **Request Location**: Sends the hex payload `"02"`, prompting the device to perform a GPS or WiFi scan.

Battery information is fetched from the `/battery` endpoint. If the battery is below 20%, the displayed value turns red; otherwise, it remains in the default color.

Location Tab This view shows a map with a marker indicating the current position of the bike. It fetches coordinate data from the `/location` endpoint and updates the map accordingly.

Implementation Notes All communication with the server is handled using `URLSession` and standard JSON encoding/decoding. Since the app was built as a prototype, no authentication or error handling mechanisms were included. The primary goal was to validate system functionality with a lightweight, responsive interface.

In short, the app serves as a simple control and monitoring tool for the Smart Bike Light system, allowing users to remotely request status updates and trigger specific functions.

5 System behaviour and battery consumption analysis

Our bike light has 2 different modes of operation: automatic and manual. It has also 2 different states: active, and shut-down.

5.1 Shut-down state

This state is achieved by physically using the switch on the side of the bike light. This will disconnect the battery from the board, effectively powering off the device and all its peripherals. This allows the battery to be preserved for a long time.

5.2 Active state

This is the state that describes the normal functioning of the bike light, when it is on. To better describe the behaviour of the system, we will delve into each part separately.

5.2.1 Light control

The light is controlled by an accelerometer and a photo-resistor in Automatic Mode, which is the default mode. In this mode, the light will turn on if low lighting and movement are both detected. If better lighting or no motion is detected for 30 seconds, the light will turn off. If the exterior button is pressed, the light turns on and enters Manual Mode. In this mode, the sensors do not affect the light, which will only turn off once the button is pressed again. This will put the light back into Automatic Mode.

5.2.2 Geolocation

As explained, we have two different methods of obtaining geolocation. As such, we had to decide on how to use these two different techniques. Considering the different power consumption rates, we decided on the following:

GNSS module : The GPS module is only used to obtain data in two situations: (1) every time that the LED turns on; and (2) every time the user requests it through the application.

WiFi scanning : Given its lower power requirements, this method was used to provide geolocation information more often during a trip. WiFi scanning is performed once every 10 minutes of a trip (assuming that a trip corresponds to the period when the light is on without interruption).

5.2.3 Battery Monitor

To alert the user about low battery conditions the battery level is read from the MAX17048 sensor every 60 seconds and transmitted to the backend via LoRaWAN every 120 seconds. When the battery level drops below the alarm threshold of 30%, a buzzer is activated to alert the user of the low battery percentage. The low-battery alarm consists of a sound pattern lasting approximately 1.2 seconds (3 x 400 ms). Additionally, the buzzer can be utilized by the user to find the bike, by using a remote buzzer activation via the app, where the buzzer plays a 5-second tone.

5.3 LoRaWAN transmission

Upon powering on the system, it will try to connect to the Cibicom network. We decided to use LoraWAN class C so that the board doesn't restart every time after sending a message, even though it is not very power efficient. After each message is sent, there will be a time cycle time where no messages will be sent, and therefore it is not guaranteed that the messages will be sent at exactly the time described. The loop function of the code we programmed in the Heltec board contains a `deque` structure that holds ordered information about which messages are scheduled to be sent.

5.4 Battery Management and Life

Power–budget scenario

Our current firmware keeps the ESP Heltec module in full-run mode and leaves the GPS permanently powered, resulting in only a marginal difference between "active" and "idle" current. Under continuous use, this has a battery life of just $\sim$ 11.7 hours. To understand what we could gain, we built a theoretical power budget that assumes the ESP32 enters modem-sleep and the GPS is power-gated. As shown in **Appendix C Battery-life power budget** (Table 5), that change would lower the idle draw from about 212 mA to 22 mA. The estimate assumes the ESP32 contributes $I_{\text{ESP32}} \approx 20$ mA in modem-sleep, with only the ADXL345, MAX17048, and regulator quiescent currents remaining, for a total $I_{\text{sleep}} \approx 22.09$ mA.

With 1 hour active and 23 hours sleep per day the 24-h average is:

$$\bar{I} = \frac{1 \cdot 212.92 + 23 \cdot 22.09}{24} \approx 30.04 \text{ mA}.$$

For a 2500 mAh Li-ion cell the runtime is:

$$t_{\text{life}} = \frac{2500}{30.04} \approx 83 \text{ h } (\sim 3.4 \text{ days}).$$

Caveat: The modem-sleep data comes from the ESP32-C3 datasheet. The Heltec *Wireless Shell V3* back-plane may draw several additional mA of quiescent current, so the computed 3.4-day endurance is a best-case projection.

6 3D case

The enclosure of our bike light has to fulfill several roles: it must securely house and protect the components of the project, be small and convenient to not be too bulky on a bike, allow the user to comfortably handle tactile controls and be easy and comfortable to place and adjust on the handlebars of the bike. An added goal of the prototype was for it to be easily openable to inspect components and update the software of the program. For the enclosure we decided upon a rectangular box solution which fit the necessary components in two layers, separated by a 3D printed internal framework to securely keep components in place. The dimensions were selected around the provided solderable bread boards, the necessary space for wires, antennas and external components and finally a recycled CATEYE bike light lense.

The components were modelled using SolidWorks and 3D printed in PLA plastic using both Ultimaker 2+ and BambuLabs A1 FDM printers. The main body was printed in 3 separate parts, two of which were glued (for improved print orientation and thereby part strength optimization) and modelled to snap fit together with the final section. Holes for charging port access, buttons, light sensors and the power switch were later implemented with hand tools.

To allow the comfortable mounting and dismounting of the bike light to the handlebars an adjustable system based on neodymium magnets was developed. The current system works as a prototype, however further attachment methods such as a turn lock, or stronger/more magnets would have to be implemented for the solution to be implemented in a real-world environment.

7 Reflection, Challenges & Future Work

With a team of six with very diverse academic and technical backgrounds have been both a challenge and a very valuable learning experience. A big, initial challenge was getting started on the project. We started by defining the scope and assigning responsibilities, as well as trying to figure out what components would be needed. In the early stages, it has been difficult to visualize the complete system, but as we progressed, the group gained a better understanding of the requirements, and things started to fall into place.

The biggest issue that we had was due to the use of the Heltec board: only one of these boards were provided per group, and this meant that all the other team members had to work on their own different boards. When the time came to integrate the work of every person together, we were faced with a huge challenge. We could hardly find any documentation for this microcontroller, and we had difficulty understanding how the pin numbering worked. This made the project frustrating and time-consuming, and didn't allow us to explore more interesting aspects of developing an IoT project, such as smart solutions to preserve battery.

Battery supply & life : While we initially added a TP4056 charger to the PCB, we only discovered after assembly that the Heltec ESP32 module already contains an IP5306 battery-management IC. The external TP4056 is therefore redundant and may even interfere with the on-board charger, but the hardware was already locked and could not be reworked. This late discovery underscores the importance of reviewing a module's built-in power-management features early in the design process.

Currently, our battery lasts only 11.7 hours due to the absence of low-power optimizations. Since power efficiency is critical for IoT devices, future improvements should focus on enabling low-power modes. This includes integrating a MOSFET to disconnect power to the GPS module when inactive, as it is one of the most power-hungry components. Additionally, leveraging the low-power capabilities of the Heltec ESP32 module, such as modem-sleep, would have the most significant impact on battery life. Together, these changes could extend the battery life toward the 82 hours estimated in our theoretical low-power model.

Light control : The current code used for controlling the light is not power friendly. There was a better version of the code in production that would pole the analog read of the photo-resistor only once per second, and would have the accelerometer in auto-sleep mode, so that it would only wake up after an activity interrupt, and go back to sleep after 30 seconds of inactivity. Sadly, due to many hardware problems related to the Heltec board, this code was not able to be finished or tested.

Furthermore, we noticed – already too late in the project –, that the pin we used for the button is internally assigned to a `LoRa_RST` pin (as shown in Appendix E). This means that the pin will be internally assigned, and it's value will change automatically, without our control. Effectively, this means that the manual mode will be activated/deactivated internally by the board. Because we found this problem so late in the project, and because we did not have any other usable pins available, we could not fix this issue.

Geolocation tracking : Sadly we did not have time to address how to save battery life when using the GNSS module. An easy option would be to use a trasistor to power-off the module when not used.

Bike case : Future improvements would be to implement rubber gaskets and/or silicone sealant, along with possibly potting the electronics in epoxy to further weatherproof the enclosure and product.

8 Conclusion

This project gave us the opportunity to develop an innovative solution to bike lighting, and allowed us to explore different areas within IoT development. This included not only working with different peripheral devices – including both software development and hardware design –, but also to explore 3D printing and soldering. In the end, we created an interesting prototype that works. In the process, we learned many things, and understand what went wrong and what could be improved.

References

[1] D. Electro, "34346 networking technologies and application development for iot: Project work slides," 2025, lecture slides, DTU.

[2] Heltec Automation, "Heltec automation – international lora/lorawan/meshtastic/iot devices manufacturer," 2025, accessed: 2025-05-22. [Online]. Available: https://heltec.org/

[3] Adafruit, "Adafruit_max1704x: Arduino library for max1704x fuel gauge," https://github.com/adafruit/Adafruit_MAX1704X/tree/main, 2025, accessed: 2025-05-24.

[4] DFRobot, "Dfrobot_adxl345: Arduino library for adxl345 accelerometer," https://github.com/DFRobot/DFRobot_ADXL345, 2025, accessed: 2025-05-24.

[5] Analog Devices, Inc., *ADXL345: Small, Thin, Low Power, 3-Axis ±2g/±4g/±8g/±16g Digital Accelerometer*, Analog Devices, 2009, revision D. [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/ADXL345.pdf

[6] Ardustore.dk, "Photosensitive resistance ldr sensor," https://ardustore.dk/produkt/photosensitive-resistance-ldr-sensor, 2025, accessed: 2025-05-24.

[7] ——, "Ublox neo-6m gps satellit module," 2025, accessed: 2025-05-22. [Online]. Available: https://ardustore.dk/produkt/ublox-neo-6m-gps-satellit-module

[8] M. Hart, "Tinygpsplus: A new, customizable arduino nmea parsing library," https://github.com/mikalhart/TinyGPSPlus, 2025, accessed: 2025-05-22.

[9] Arduino Project, "Wifi library," https://docs.arduino.cc/libraries/wifi/, 2025, accessed: 2025-05-22.

[10] Google LLC, "Google maps platform documentation," https://developers.google.com/maps, 2025, accessed: 2025-05-22.

Appendices

A Work distribution

Task	Corinne	Mariana	Nipun	Paul	Sia	Vandad
ESP32 programming	33%	33%	33%	0%	0%	0%
Light sensor	100%	0%	0%	0%	0%	0%
LoRaWAN	0%	90%	0%	0%	0%	10%
Geo-location	0%	90%	0%	0%	0%	10%
Soldering	10%	10%	0%	70%	10%	0%
Accelerometer	100%	0%	0%	0%	0%	0%
CAD design & 3-D printing	20%	0%	0%	80%	0%	0%
LED	80%	0%	0%	20%	0%	0%
Buzzer	0%	0%	50%	0%	50%	0%
App development	0%	10%	0%	0%	20%	70%
Backend server	0%	10%	0%	0%	0%	90%
Battery / charging	0%	0%	80%	10%	10%	0%
Report	16%	16%	16%	16%	16%	16%
Presentation	16%	16%	16%	16%	16%	16%

Table 4: Division of responsibilities among team members.

B Breadboard diagram

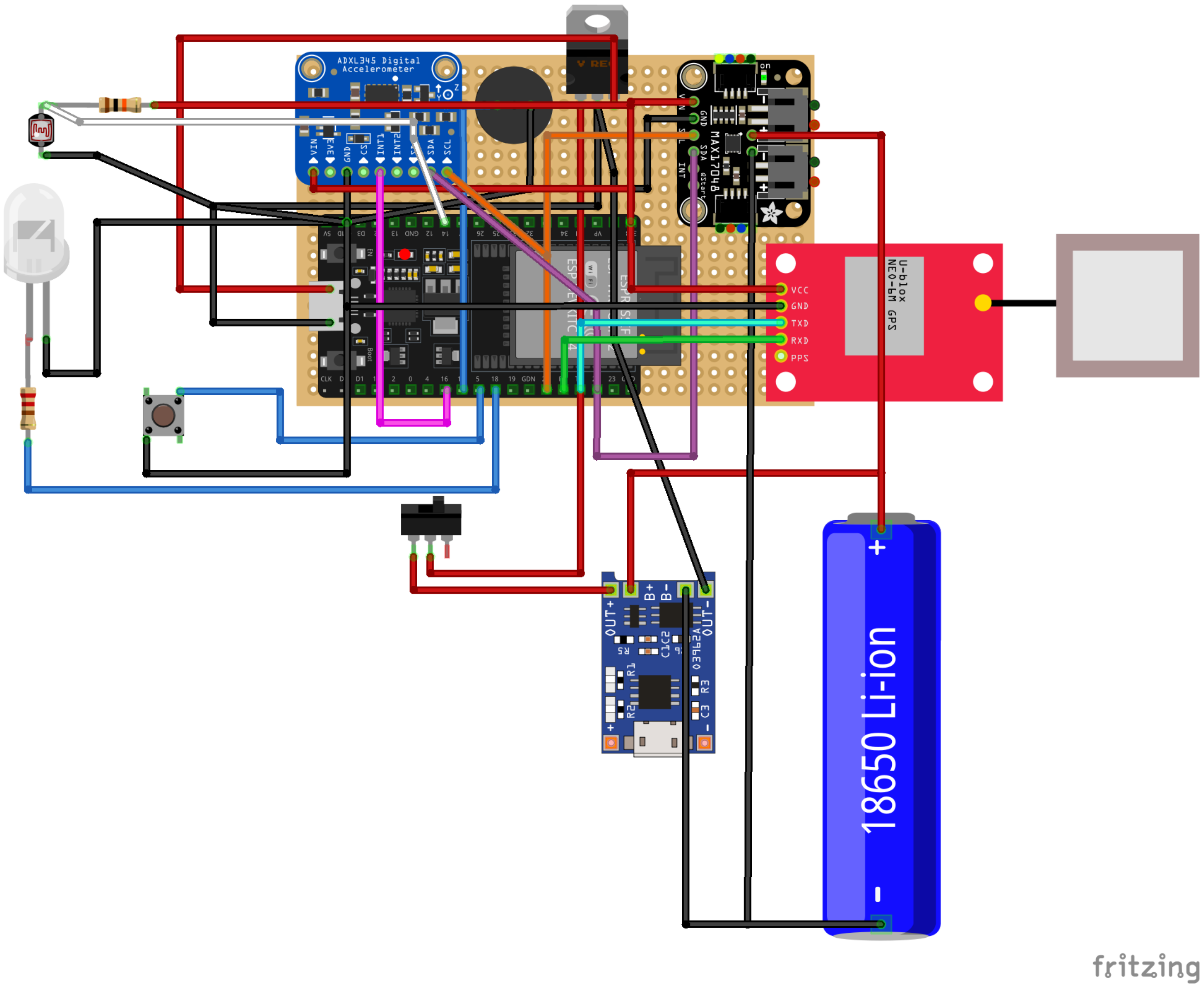

Figure 2: Breadboard setup showing the key components.

C Battery-life power budget

Component	Assumptions (Active)	Active current (mA)	Assumptions (Modem)	Modem current (mA)
ESP32 (Active) + LoRa & Wi-Fi	Draws 150 + 1.97 + 0.02 mA continuously	151.99	ESP32 modem sleep	20
GPS NEO-6M (Always On)	Draws ∼39 mA constantly	39.0	GPS off via MOSFET	0.0
ADXL345 Accelerometer	Draws 40 µA = 0.04 mA	0.04	Draws 40 µA = 0.04 mA	0.04
MAX17048 Fuel Gauge	Draws 50 µA = 0.05 mA	0.05	Draws 50 µA = 0.05 mA	0.05
LED (15 min/30-min trip)	(15/30) × 3.3 mA = 1.65	1.65	LED off	0
Regulator overhead (10% of subtotal)	10% of subtotal before overhead	20.185	Fixed quiescent draw (2 mA)	2
Total	Sum of all components incl. regulator loss	**212.915**	Sum of ESP32 + sensors	**22.09**

Table 5: Active-mode vs. modem-sleep power budget used in the battery-life estimate.

D GitHub repository

All project files are archived at:

```
https://github.com/immarianaas/iot-bling-bling.git
```

E Pinout

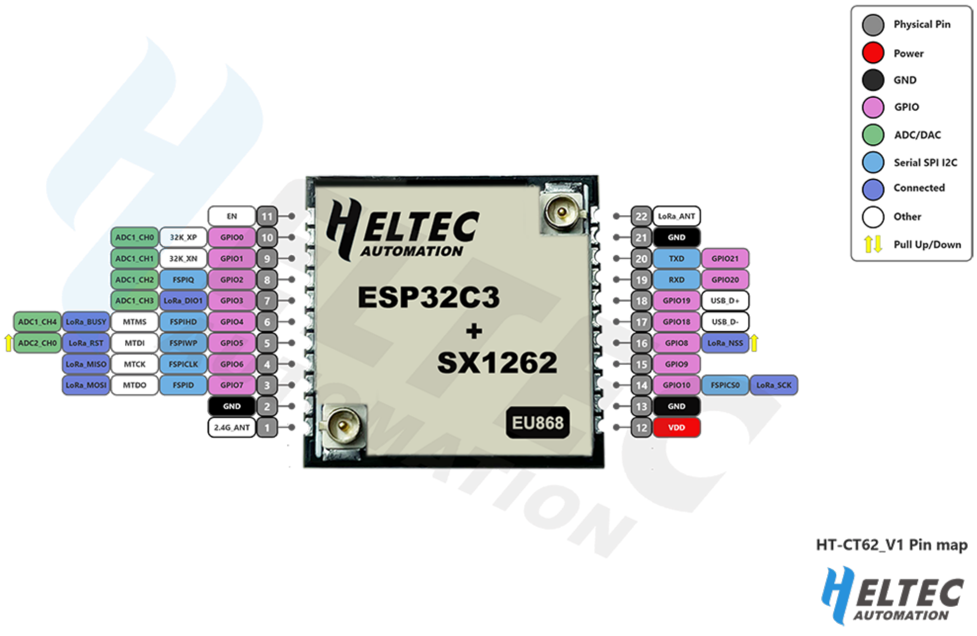

Figure 3: ESP32C3 pinout diagram, obtained from: `https://heltec.org/project/ht-ct62/`

ESP32C3+SX1262 pin	Pin of the peripheral device
2	SCL of both the Battery monitor and Accelerometer
9	SDA of both the Battery monitor and Accelerometer
19	Accelerometer interrupt
0	TX of the GNSS module
3	RX of the GNSS module
18	Buzzer
8	LED
8	LED
4	Photoresistor sensor
5	Button

Table 6: Connections between the peripherals and the ESP32C3+SX1262 chip that we used in our project.