HOCHSCHULE
RHEIN-WAAL
Rhine-Waal University
of Applied Sciences

The Talking Mailbox

2907 Sensors and Actuator Networks

Winter Semester 2025/26

Authors:

Justin Julius Chin Cheong
34140
MSE

Abhinav Kothari
33349
MSE

Contents

1 Introduction

1.1 Problem Statement

All Professors and Lecturers have a lot to do and may not always have time to check their mailbox. Imagine how long some letters are left in the mailbox for days just because a professor is busy. On the other hand, checking your mailbox only to find nothing is quite frustrating. What if there was a way that your mailbox could tell you when there is mail? What if you had a talking mailbox?

To solve this problem, we introduce **The Talking Mailbox**. The aim of The Talking Mailbox project is to design and assemble a system that can detect the presence of mail within a mailbox in Building 06 and notify the owner of the mailbox.

1.2 Literature Review

Before developing The Talking Mailbox, various existing smart mailbox solutions were reviewed, considering their sensor technology and communication methods as well as their advantages and shortcomings.

Perhaps the simplest solution is presented in frankenfoamy (2021). A wire connected to the door protrudes out of the box and holds down a flag. Once opened, the flag is released indicating the mailbox has been opened. On the opposite end of the technological spectrum, several commercial smart mailboxes are available. The Fouvin (2025) model uses a passive infrared (PIR) motion sensor to detect mail presence, while Notific (2025) employs a motion sensor[1] attached to the mailbox door. The Fouvin (2025) connects directly to Wifi and the Notific (2025) to LoRaWAN, providing remote notifications through dedicated apps. Similar to the Notific (2025), the InstaView (2024) detects door openings utilising a tilt sensor, while the X-Sense (2025) combines a tilt sensor and an IR motion sensor for enhanced detection. Both of these models communicate with a base station within the home via radio frequency (RF) technology, with the X-Sense (2025) capable of managing multiple mailboxes.

While these solutions certainly have merit, The Talking Mailbox offers some advantages over them and has a degree of novelty. While very cheap and reliable, the frankenfoamy (2021) has no means of remote notification. The Fouvin (2025) and X-Sense (2025) both use IR sensors which may be prone to false positives from environmental heat sources. The InstaView (2024) and X-Sense (2025) require an additional base station device which increases complexity of setup. The Talking Mailbox eliminates these issues as it connects to LoRaWAN directly and uses multiple sensors (that are not IR-based) to mitigate false positives. Most critically, all of these solutions rely on inferring mail presence from door movement or motion within the box, which may not always be accurate. The Talking Mailbox directly detects mail presence using a weight sensor, providing a more reliable solution.

1.3 Project Plan

1.3.1 Schedule

The Talking Mailbox project is planned to be executed over a period of 3 months, starting from October 2025 to January 2026. The project is divided into several milestones as shown in Figure 1. These deliverables and milestones adhere to the requirements set out in Section 1.4.3.

[1] While no literature could be found on the exact technology used, the sensor is likely an accelerometer of some kind

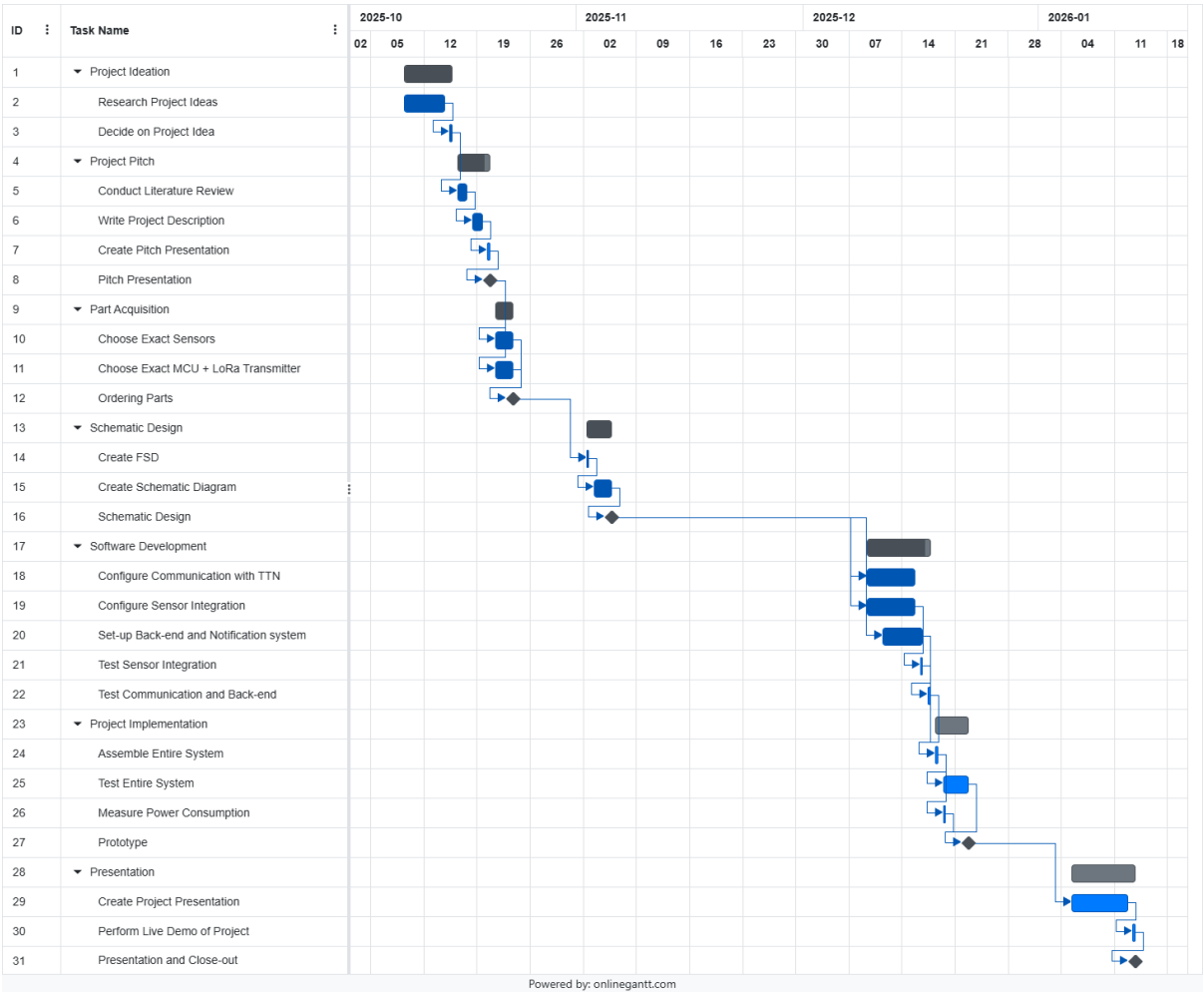


Figure 1: Gantt Chart for The Talking Mailbox Project

1.3.2 Resources

The resources for the project include the budget shown in Section 1.4.3 and all materials purchased as part of the Bill of Materials in Section 3.2.5.

The project team consists of two Mechatronics Engineering Students whose responsibilities are divided as follows:

- **Abhinav Kothari (33349):** Backend Development, Server Setup and Email Notification System.
- **Justin Julius Chin Cheong (34140):** Schematic Design, Sensor integration, Microcontroller programming
- **Both Students:** Component Choice, System Design, System Assembly & Implementation, System Testing, Report Writing and Presentation Preparation.

1.4 System Concept

1.4.1 Functional Requirements

For The Talking Mailbox to be a satisfiable product, the following functional requirements must be implemented:

- It can detect if the mailbox is opened.
- It can detect light as a redundancy for confirming the opening status of the mailbox.

- It can detect whether or not mail is present within the mailbox.
- It can communicate if mail is in the box to a website / dashboard (based on LoRaWAN).
- It alerts the responsible person via email or dashboard upon mail detection.
- It can check the battery status.
- It sends battery status updates to a website at regular intervals.
- It sends a low battery warning to a website when the battery falls below a defined threshold.
- It should run for at least 1 week on a single charge.

1.4.2 Technical Requirements

For The Talking Mailbox to operate and perform its functions, the following technical requirements must be implemented:

- The weight sensor can detect a change in weight of approximately 20 g. This indicates when a piece of mail has been placed within the box.
- The tilt sensor can detect the rotation of the post box lid. This indicates when the lid is opened.
- The LDR can detect the change in light intensity by a defined threshold. This indicates when the lid is opened.
- The transmitter can reliably connect and communicate via the LoRaWAN Gateway.
- The server with which the LoRaWAN communicates can send emails to relevant personnel about the mail.
- The power supply is a battery with a working voltage of 3.1 V to 5 V.
- The enclosure can protect the system within a typical indoor environment (IP 31).
- The system should function at temperatures ranging 0–40°C and humidity 10–90%.

1.4.3 Project Requirements

For The Talking Mailbox project to produce a functional product upon close out, the following project requirements must be met:

- The budget is 100€.
- The project workload is estimated at 100 h.
- The project schedule adheres to the following deadlines:

Pitch:	2025-10-21
Bill of Materials:	2025-10-23
Schematic Design:	2025-11-23
Project Implementation:	2025-12-19
Project Report:	2026-01-05
Project Presentation and Demo:	2026-01-17

2 Theoretical Background

Before starting with actual project, some theoretical framework is required.

2.1 Communication

As for the communication, this project used LoRa as the communication encoding and LoRaWAN as the MAC Protocol.

2.1.1 LoRa

LoRa (**Lo**ng **Ra**nge) is the physical layer and is a modulation technique which allows for wireless communication. It is able to send information long ranges, with relatively less energy. It is derived from Chirp Spread Spectrum (CSS). It encodes information similar to how bats/dolphins communicate. LoRa is used extensively with sensors and actuator projects for the following reasons:

- Low power consumption
  - Transmitting: 10 mA
  - Sleep: 100 nA
- Long range → upto 15 km
- Robust against interferences

There are many more reasons as well, but these are the primary which were kept in mind for selecting it for this project.
LoRa works on a license free frequency range, in Europe this is EU868 (863–870/873 MHz). This will be used in this project (Precisely: 868.1 MHz).

2.1.2 LoRaWAN

LoRaWAN (**LoRa** **W**ide **A**rea **N**etwork) on the other hand is the data link layer on top of LoRa. It defines the communication protocols and architecture. After the initial release in January 2015, many versions have been released, with latest being 1.0.4 (Series 1.0) and 1.1 (Series 1.1) being released in October 2020 and October 2017 respectively. (Yes, 1.0.4 is newer than 1.1). The version used in this project is 1.0.4. Figure 2, shows how LoRa and LoRaWAN differ and work together.

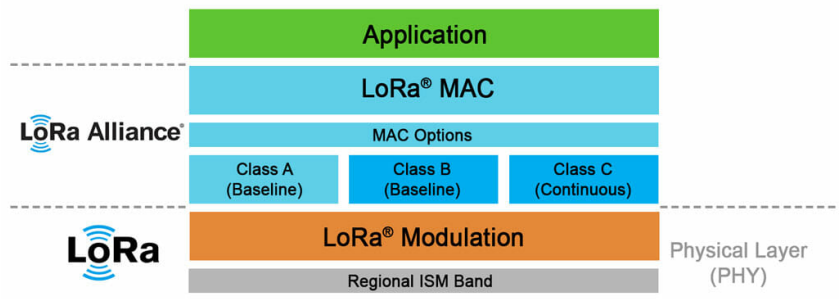


Figure 2: LoRa and LoRaWAN

2.2 Sensors

This project used 3 sensors, their functioning has been briefly described below. This framework is essential to understand the design choices made in this project in Section 3.2.

2.2.1 Load Cells

Load Cells are the primary sensor for this project and should be able to detect the presence of mail. Load cells are used to measure force, and hence can achieve this task. The specific load cell used in this project, is a strain gauge load cell. Strain gauges in the cell are arranged in a way that applying force changes resistance of the gauges in arranged in a wheatstone bridge and hence send out a voltage. This voltage is very small, and hence must be amplified. This amplification is done with the HX711 board, which makes it readable for the microcontroller (ESP32-S3). More specific details regarding these equipment can be seen in Section 3.2.5.

2.2.2 Tilt Switch

Tilt switch is being used to detect the opening of the lid of the postbox. There are multiple types of tilt switches mechanical (rolling ball/ liquid mercury) or electronic (MEMs). The one used in this project is a mechanical rolling ball switch, due to its lower voltage requirement as well as it being a safer option. The functioning can be demonstrated by Figure 3. Whenever the switch is in a specific

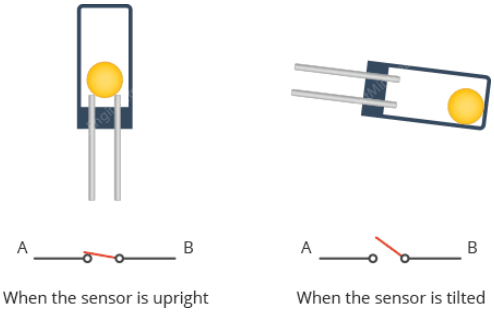


Figure 3: Tilt switch working

orientation the ball allows for contact and hence making an electrical connection, else there is no connection.

2.2.3 LDR

A light dependent resistor is just a resistor which varies its resistance based on the light intensity. This can be detected and hence compared to a threshold to check if the box is open or not.

3 Methodology and Design

3.1 Design Approach

The design approach used in this project was a simple waterfall model. After a deliverable was completed, the next deliverable was started. This can be clearly seen in Figure 1. The only exception to this model was during the Software Development and Project Implementation Deliverables (which are discussed in Section 4.1) where some iterative testing and debugging was employed. The testing methods are discussed in Section 3.3.

3.2 System Design

3.2.1 System Architecture

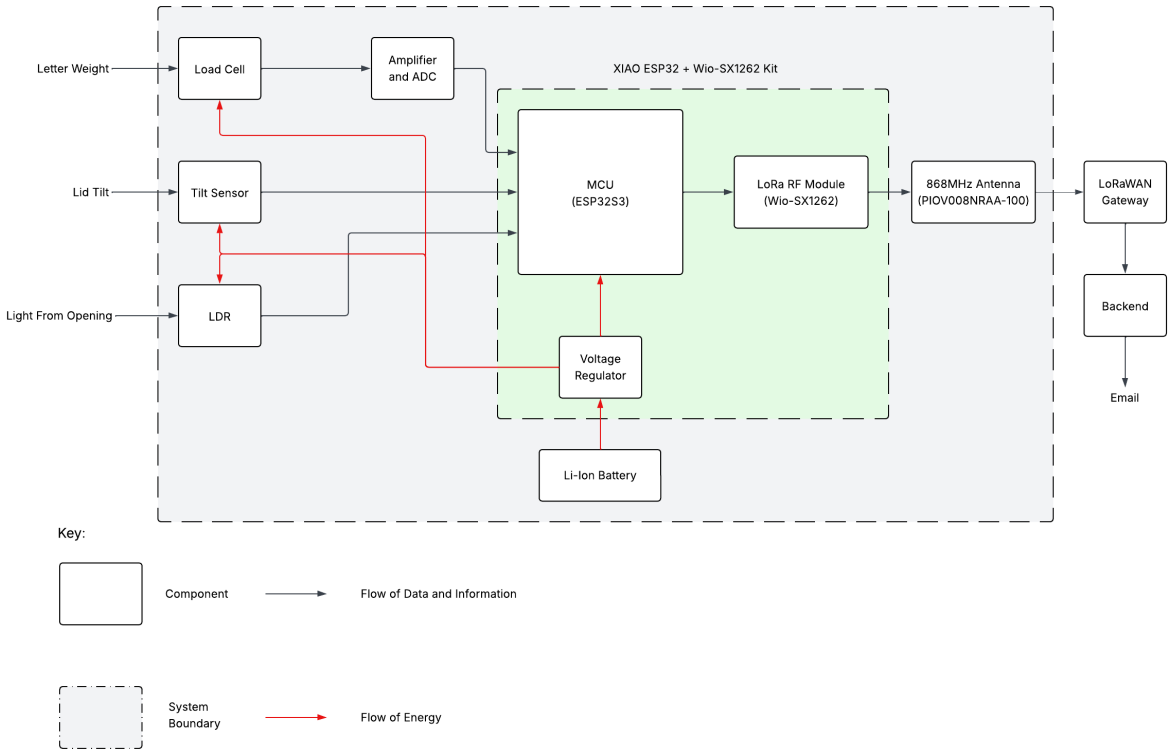


Figure 4: Functional Structure Diagram of the System Architecture

3.2.2 Sensor Selection

The sensors selected for this project were based on the functional requirements defined in Section 1.4.3 as well as the shortcomings discussed in Section 1.2. The general idea was to use multiple sensors to detect the same event, in order to increase reliability as well as a sensor which can detect not only events, but the current status of the mailbox. Hence, the light sensor and tilt sensor combination was chosen to detect the opening of the mailbox, while the load cell was chosen to detect the presence of mail. The reasons for choosing these specific sensors and their principle of operation are discussed in Section 2.2.

3.2.3 Schematic Design

The complete schematic design of the system is shown in Figure 5. The design is quite simple as the light and tilt sensors are digital sensors and connect directly to any GPIO pins on the microcontroller. The load cell on the other hand connects to the HX711 amplifier board, which then has a data line and clock line connecting to the ESP32-S3. The SX1262 LoRa module is connected via SPI, but the

depiction 5 is a bit inaccurate. The kit comes with B2B connection which allows the MCU and LoRa module to be connected directly without any wiring. The pin mapping between the sensors and the microcontroller is shown in Table 1.

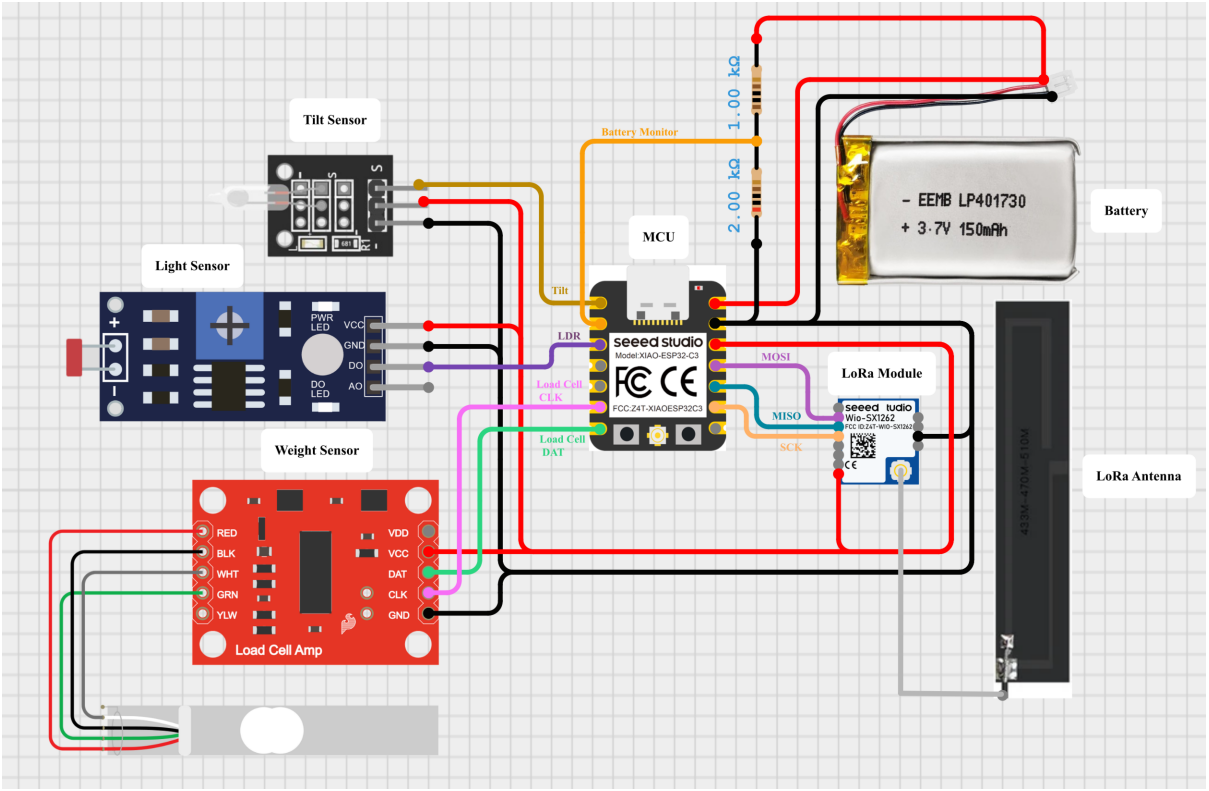


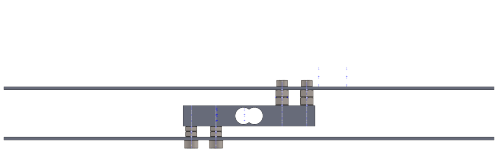
Figure 5: Schematic Diagram of the System

Sensor Pin	ESP32-S3 Pin
Tilt Sensor DO	GPIO1
Battery Monitor	GPIO2
Light Sensor DO	GPIO3
HX711 CLK	GPIO6
HX711 DAT	GPIO43
SX1262 SCK	GPIO7
SX1262 MISO	GPIO8
SX1262 MOSI	GPIO9

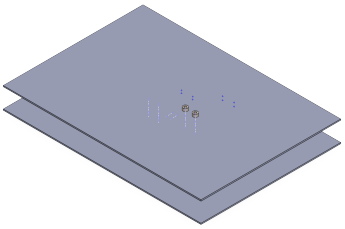
Table 1: Sensor-to-ESP32-S3 Pin Mapping

3.2.4 3D Design

As understood from Section 2.2.1, to reliably detect any weight, the load cell must be allowed to bent as easily as possible. To prevent any mail from "missing" the detection platform, the platform must cover the entire box area. The design which allows us to do this is shown in Figure 6.
The height with the hex nuts, allow the load cell to have clearance to bend during load from the upper platform. The off center screw platform on both platform allow for best readings from the load cell. The size of the upper platform should match the post box size, however the lower platform allows for flexibility, it can be made just large enough to not cause toppling over and allow us to place necessary modules (such as the HX711).
Some other design considerations include the requirement of the platform being rigid and lightweight, to allow load cell to be as sensitive as possible. A good material to use for the platform is wood. The screws should also be flat with the platform, to prevent mail from tearing / getting stuck.



(a) 3D Model of Load Cell Arrangement



(b) Platform for mail detection

Figure 6: 3D Model of detection platform

3.2.5 Bill of Materials

This is an estimate of the materials required to make this project, these are all over estimates, as all components/materials except 2.1 - 2.6 were used from the university stock.

Item	Part	Description	Qty	Notes	Total Price (€)
1.0 Mechanical Components					
1.1	Top Plate	Detection Plate	1	Wood (40x30x0.5cm)	5.00
1.2	Bottom Plate	Base for load cell	1	Wood (15x15x0.5cm)	3.00
1.3	M4 Screw	Top plate screw	2	-	1.00
1.4	M5 Hex Nut	Height spacer nut	4	-	0.50
1.5	M5 Screw	Bottom plate screw	2	-	1.00
1.6	M6 Hex Nut	Height spacer nut	4	-	0.50
1.7	Misc.	Tape, Glue	-	-	0.50
2.0 Electrical Components					
2.1	Load Cell + HX711	Load cell + Amplifier module	1	JOY-IT	6.40
2.2	Tilt Switch	Ball tilt switch	1	IDUINO	0.94
2.3	LDR	Light resistor	1	SERTRONICS	1.35
2.4	Battery	1800 mAh Li-Ion	1	SOLDERED	10.24
2.5	MCU + LoRa	Xiao ESP32 + SX1262	1	Seeedstudio	11.68
2.6	Antenna	Long range antenna	1	Amphenol-SAA	2.69
2.7	1 kOhm resistor	Through Hole	1	YAGEO	0.10
2.8	2 kOhm resistor	Through Hole	1	YAGEO	0.10
2.9	Wires	Jumper wires of different length	-	-	0.30
2.10	Misc.	Breadboard, Wire Sleeves, Solder	-	-	0.50
				Tax (VAT 20%)	9.16
				Grand Total (€)	**54.96**

Table 2: Combined Mechanical and Electrical Bill of Materials with Total Cost

3.3 Validation Method

Once development of the system was underway, it became essential to validate system incrementally. After each task was completed within the Software Development and Project Implementation deliverables, the component was tested against the requirements outlined in Sections 1.4.1 and 1.4.2.

The following briefly describes the validation methods used specific requirements:

- **Mail Detection:** Various objects ranging from around 20 g to little above 100 g were weighed on a scale and then placed on the load cell platform to check if the load cell could detect the presence of mail and classify them accurately via the serial monitor.
- **Lid Opening Detection:** The tilt switch and LDR were tested by opening and closing the mailbox lid multiple times. Using the serial monitor, the readings from both sensors were observed to ensure they accurately detected the lid status.
- **LoRa Communication:** The LoRa module was tested by sending test messages from the microcontroller to the LoRaWAN gateway. The successful receipt of messages was confirmed via TTN console and Datacake dashboard.
- **Email Notification:** The email notification system was tested by simulating mail detection events and verifying that emails were sent to the designated recipient.
- **Battery Status Monitoring:** The battery monitor was tested by simulating mail detection events and checking the battery level readings on the dashboard and then checking them against a multimeter.
- **Battery Life:** The system was powered by the battery and power consumption was monitored while idle and while sending messages. The readings were used to estimate the battery life.

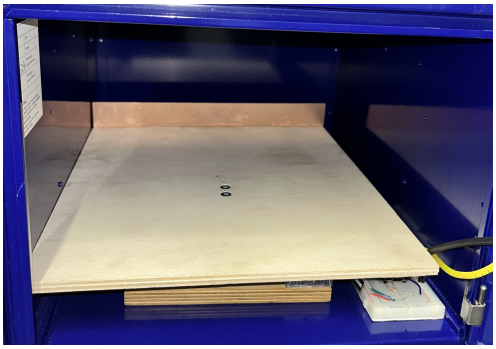
4 Results and Implementation

4.1 Implementation

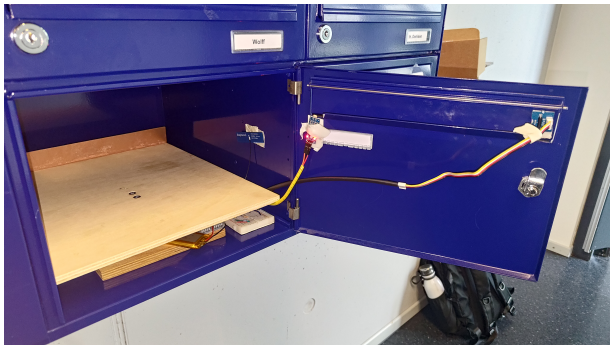
4.1.1 Product Implementation - Hardware Layer

Final Product Assembly

Considering all the design choices made in Section 3.2, the actual implementation of the hardware layer was done as shown in Figure 7.



(a) Front View

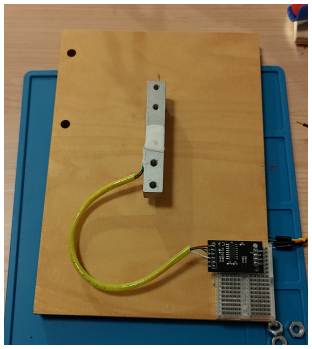


(b) Side View

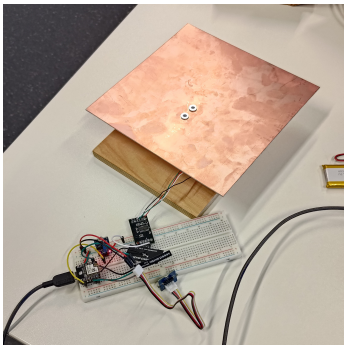
Figure 7: Final Product Inside the Post Box

Prototyping

However, before this complete assembly was constructed, a smaller prototype was made to test the functioning of the sensors and the LoRa communication. This prototype is shown in Figure 8.



(a) Weight Sensor Construction



(b) First Assembled Prototype

Figure 8: Prototype of the System for Testing

4.1.2 Product Implementation - Embedded Software Layer

Using the prototype shown in Figure 8, the embedded software layer was implemented. The code was written in C++ using the Arduino IDE and flashed to the ESP32-S3 microcontroller.

The microcontroller code was structured with modularity in mind and divided into four files:

- **SensorManager.cpp:** This file contains all the functions related to sensor initialization, reading sensor values, and processing the sensor data.
- **SensorManager.h:** This is the header file for SensorManager.cpp, containing function prototypes and necessary libraries and macros.
- **LoRaConfig.h:** This file handles the LoRaWAN communication, initializing required parameters, including libraries, defining functions for LoRa activation and upload.

- **LR-V1.0.ino:** This is the main file which essentially just runs the functions defined in the preceding files in the specific order required.

4.1.3 Backend Implementation - Application Layer

This section covers the implementation of the backend server and application layer, which is responsible for receiving data from The Things Network, decoding the payload, storing the data, and providing a user interface for monitoring the mailbox status. The link with the The Things Network is done using webhooks, which send HTTP POST requests to the python server whenever new data is received from the LoRaWAN device.

Initial setup To set up the backend server, we require three files, which include: 'server.py', '.env', and 'requirements.txt'. Each file servers a specific purpose:

- **server.py:** This is the main server file that contains the Flask application code. It handles incoming requests, decodes the payload, updates the mailbox status, and serves the user interface.
- **.env:** This file contains environment variables such as server port, authentication token, email sender credentials, and recipient email addresses. This allows for easy configuration without hardcoding sensitive information in the code.
- **requirements.txt:** This file lists all the Python libraries required to run the server.

These all need to be placed in the same directory for the server to function correctly. The server can then be started by running the command 'python server.py' in the terminal. The depenendencies, to be added to requirements.txt as shown in Listing 1, can be installed using the command 'pip install -r requirements.txt'.

```
Flask==2.3.3
python-dotenv==1.0.0
```

Listing 1: requirements.txt file contents

These can be installed directly using pip as well, however using requirements.txt makes it easier and more future proof. These steps can be also be avoided, and the bash script from Section 4.1.4 be used directly to install the required libraries if they are missing.

Next a random authentication token must be generated, for Windows 11, the command shown in Listing 2 can be used.

```
$bytes = New-Object byte[] 16; (New-Object System.Security.Cryptography.RNGCryptoServiceProvi
[System.BitConverter]::ToString($bytes) -replace '-'
```

Listing 2: Generate random auth token command

Now you can open the .env file using a text editor of your choice (here edit by MS was used). The following variables must be added as shown in Listing 3:

```
export SERVER_PORT=3000
export AUTH_TOKEN=<token from above>

EMAIL_SENDER=your_email@gmail.com
EMAIL_PASSWORD=your_app_password_here
```

Listing 3: .env file contents

The main contents of server.py are explained in the following sections and full code can be found in the project repository.

Payload Decoder

The LoRaWAN is able to send the bits to The Things Network. However for these to be actually useful to the user they must be decoded and used to represent relevant information for a user, this includes the mail status, the battery and which post box it is. For this first a payload decoder must be made. This is made keeping in mind how bits were encoded in the first place. The decoder can be seen in Listing 4

```python
def decode_mailbox_data(base64_string):
    try:
        raw_bytes = base64.b64decode(base64_string)
        if len(raw_bytes) < 4:
            return None, "Error: Short Data", "red", None

        # 1. Decode ID (Hex to Int logic)
        try:
            device_id = int(f"{raw_bytes[0]:x}")
        except:
            device_id = raw_bytes[0]

        state_byte = raw_bytes[1]
        value_id = raw_bytes[2]
        val = raw_bytes[3]

        # 2. Decode Status (Aligned with new JS Decoder)
        if state_byte == 0x04:
            status, color = "Tampering Alert", "red"
        elif state_byte == 0x05:
            status, color = "Heavy Mail", "#004d40"
        elif state_byte == 0x06:
            status, color = "Medium Mail", "#00897b"
        elif state_byte == 0x07:
            status, color = "Light Mail", "#4db6ac"
        elif state_byte == 0x08:
            status, color = "No Mail", "blue"
        else:
            status, color = f"Unknown: {hex(state_byte)}", "orange"

        # 3. Decode Battery (valueID 0x09)
        battery = val if value_id == 0x09 else None

        return device_id, status, color, battery

    except Exception as e:
        logger.error(f"Decoding error: {e}")
        return None, "Decoding Failed", "black", None
```

Listing 4: Payload Decoder Function

This allows us to correctly identify if a heavy, medium or light package was detected. This information can then be used to update the website to represent the appropriate information and also be included in the mail sent to the user.
The key parts of the decoder are explained below:

- ”state_byte” is used to determine the mail status, this is done by checking the value of the byte and mapping it to the appropriate status message.
- There is return at the start for robustness, in case the payload is too short or empty.
- ”value_id” has been used as a form of future proofing, in case more values are added to the payload in the future.

For actual mapping of device ID to device name and email addresses, a dictionary is used as shown in Listing 5.

```python
    # --- Device Mapping ---
    DEVICE_NAMES = {
        33: "PostBox SAN"
    }
    # --- Email Mapping ---
    DEVICE_RECIPIENTS = {
        33: "email1@gmail.com, email2@gmail.com"
```

```
    }
```

Listing 5: Device mapping method

More can be added as just new rows with the similar syntax just a comma at the end of all rows except the last one. Multiple email addresses can also be added by separating them with a comma within the double quotes.

Frontend Implementation - User Interface

For the actual server, which the user interacts, with a python server was created. This server uses the Flask framework to create a simple web application that displays the status of the mailbox. The server listens for incoming data from The Things Network and updates the mailbox status accordingly. The relevant code snippet is shown in Listing 6

```
@app.route('/', methods=['GET'])
def show_dashboard():
    d = dashboard_data
    html = """
    <!DOCTYPE html>
    <html>
    <head>
        <title>Mailbox Monitor</title>
        <meta http-equiv="refresh" content="5">
        <style>
            body {
                font-family: 'Segoe UI', sans-serif;
                text-align: center;
                padding: 40px;
                background-color: #f0f2f5; }
            .card {
                background: white;
                padding: 40px;
                border-radius: 15px;
                display: inline-block;
                box-shadow: 0 4px 12px rgba(0,0,0,0.1);
                width: 450px; }
            .status-box {
                font-size: 32px;
                font-weight: bold;
                margin: 20px 0;
                padding: 20px;
                color: white;
                border-radius: 10px;
                background-color: {{ d.status_color }}; }
            .battery-indicator {
                font-weight: bold;
                font-size: 18px;
                padding: 5px 15px;
                border-radius: 20px;
                display: inline-block;
                color: white;
                background-color: {{ d.battery_color }}; }
            .meta {
                color: #888;
                font-size: 13px;
                margin-top: 15px; }
            table {
                width: 100%;
                border-collapse:
                collapse;
                margin-top: 20px;
                text-align: left; }
            th, td {
                padding: 10px;
```

```
                border-bottom: 1px
                solid #eee;
                font-size: 14px; }
            .dot {
                height: 10px;
                width: 10px;
                border-radius: 50%;
                display: inline-block;
                margin-right: 5px; }
        </style>
    </head>
    <body>
        <div class="card">
            <h1>Smart Mailbox</h1>
            <h2 style="color:#666">{{ d.device_name }}</h2>
            <div class="status-box">{{ d.status_text }}</div>
            <div class="battery-indicator">{{ d.battery_level }}</div>
            <p class="meta">Last Update: {{ d.timestamp }}</p>
            <h3>Recent Activity</h3>
            <table>
                <tr><th>Time</th><th>Event</th><th>Bat</th></tr>
                {% for event in d.history %}
                <tr>
                    <td>{{ event.time }}</td>
                    <td>
                        <span class="dot"
                            style="background-color: {{ event.color }};">
                        </span>{{ event.status }}</td>
                    <td>{{ event.battery }}</td>
                </tr>
                {% endfor %}
            </table>
        </div>
    </body>
    </html>
    """
    return render_template_string(html, d=d)
```

Listing 6: Flask Server Code Snippet

4.1.4 Issues with static IP and hosting

The server was initally hosted locally, however to provide access to the user over the web, a public IP address was required. This was also meant to be static, so user can always access the server without having to worry about changing IP addresses. This is what led to the use of ngrok, which allows for a static IP address to be used. However, this meant the user had to run multiple scripts to start the server. Which can be annoying for a non technical user.
To combat this issue, a .bat file was created, which runs all the necessary commands to start the server and ngrok tunnel. This allows the user to just double click the .bat file and have everything start automatically. This also helps us solve another issue, which is incase the required python libraries are not installed, the script checks for them and installs them if they are missing. It also hides all ngrok instances, to prevent confusion. The bat script contents can be seen in Listing 7

```
@echo off
title Smart Mailbox Dashboard
color 0A

echo ============================================================
echo           SMART MAILBOX PROJECT LAUNCHER
echo ============================================================
echo.

:: --- STEP 1: INSTALL/UPDATE REQUIREMENTS ---
```

```
echo [1/4] Checking Python libraries...
pip install flask python-dotenv
if %errorlevel% neq 0 (
    color 0C
    echo.
    echo [ERROR] Python or PIP is not installed or not in your PATH.
    echo Please install Python from python.org and try again.
    pause
    exit /b
)
echo Libraries are ready.
echo.

:: --- STEP 2: CHECK CONFIGURATION ---
echo [2/4] Checking configuration file...
if not exist .env (
    color 0E
    echo.
    echo [WARNING] .env file was not found!
    echo I have created a template .env file for you.
    echo Please open ".env", add your passwords, and run this script again.

    :: Create a default .env file
    echo SERVER_PORT=3000> .env
    echo AUTH_TOKEN=BC5FB17A739C64639751B59209E07F88>> .env
    echo EMAIL_SENDER=my-iot-project@gmail.com>> .env
    echo EMAIL_PASSWORD=REPLACE_WITH_APP_PASSWORD>> .env
    echo EMAIL_RECIPIENT=your_personal_email@gmail.com>> .env

    pause
    exit /b
)
echo Configuration found.
echo.

:: --- STEP 3: START NGROK (NEW WINDOW) ---
echo [3/4] Launching Ngrok Tunnel...
:: This opens a separate popup window for Ngrok so it doesn't block the script
start "Ngrok Tunnel" ngrok http --domain=unarithmetically-peppiest-libbie.ngrok-free.dev 3000

:: --- STEP 4: START PYTHON SERVER ---
echo [4/4] Starting Python Server...
echo.
echo ========================================================
echo    Dashboard: https://unarithmetically-peppiest-libbie.ngrok-free.dev
echo    Local:     http://localhost:3000
echo    Status:    RUNNING (Keep this window open)
echo ========================================================
echo.

python server.py

:: If python crashes, keep window open to see error
pause
```

Listing 7: .bat Script to start server and ngrok

However, this is still not an ideal solution, as the user still has to run the .bat file manually, must have python installed on their machine and most importantly run the server continously. A better solution would be to host the server on a local raspberry pi or similar device, which can run the server 24/7 without any user intervention. This would also eliminate the need for ngrok, as the raspberry pi can be given a static IP address.

4.2 Validation Results

5 Discussion

5.1 Product Evaluation

6 Conclusion

6.1 Project Summary

6.2 Future Work / Improvements

While working on The Talking Mailbox project, several areas for potential improvements were identified that could enhance the functionality, reliability, and user experience of the system. These improvements include:

- **Better tampering detection**: While the LDR and tilt sensor provide good and reliable tampering detection, they can be replaced with a single reed swich. This would reduce complexity and power consumption.
- **Improved housing**: Housing of the components can be combined with load cell platform for a better fit and protection of the components.
- **Additional information**: A camera can be added to capture images of the mail inside the box. This would provide visual confirmation of mail presence and enhance user experience. An LED can be used in conjunction with the camera to provide illumination inside the mailbox even when closed.
- **Easier server access**: The server can be deployed on a local raspberry pi or similar device to allow for local access and control. This would eliminate the need for an external hosting service and provide more flexibility.
- **Tampering alert**: A buzzer can be added to the box, to alert surroundings when tampering is detected. This would enhance security and deter unauthorized access.
- **Enhanced power management**: A piezo can also be added to the lid to harvest energy from the opening and closing of the mailbox. This would extend battery life and reduce maintenance.

Acknowledgement

References

Fouvin. (2025). *Mailbox sensor with remote monitoring via tuya app.* Amazon. Retrieved from https://www.amazon.de/Bewegungssensor-Intelligenter-Bewegungsmelder-Heimsicherheit-Fern%C3%BCberwachungs/dp/B0DDGGDMPX/ref=asc_df_B0DDGGDMPX?mcid=4350992a94a03c3eb6a28ecb28a86d00&th=1&psc=1&tag=googshopde-21&linkCode=df0&hvadid=720823234981&hvpos=&hvnetw=g&hvrand=8721252282549905970&hvpone=&hvptwo=&hvqmt=&hvdev=c&hvdvcmdl=&hvlocint=&hvlocphy=9044849&hvtargid=pla-2373176237640&psc=1&hvocijid=8721252282549905970-B0DDGGDMPX-&hvexpln=0 (Accessed: 2025-12-29)

frankenfoamy. (2021). *Automatic mailbox alert.* YouTube. Retrieved from https://youtu.be/3E_R6cfH0M8?si=_cTojZNmOHxOoKMr (Accessed: 2025-12-29)

InstaView. (2024). *Long-range mail arrival indicator device.* Amazon. Retrieved from https://www.amazon.com/InstaView-Delivered-Notification-Long-range-Indicator/dp/B0DPDRX6MG (Accessed: 2025-12-29)

Notific. (2025). *Smart mailbox sensor.* Notific.at. Retrieved from https://notific.at/en (Accessed: 2025-12-29)

X-Sense. (2025). *Smart briefkastensensor sma51.* Bigshopper. Retrieved from https://bigshopper.de/product/x-sense-smart-briefkastensensor-erfordert-sbs50-basisstation-funk-melder-mit-grosser-reichweite-fuer-briefkaesten-briefkasten-alarm-zugestellte-post-sma51-2941520710.htm?gad_source=1&gad_campaignid=22853637764&gbraid=0AAAAAqqSOIYjGM2alA_7gHg2dK6Kmec4W&gclid=Cj0KCQiA6sjKBhCSARIsAJvYcpOqZ7COLWc3ZKFM_2j8uUDxGS53lbsTWrKlkUtC2Gwa3fv-dxQFQzcaAhnSEALw_wcB#description (Accessed: 2025-12-29)

Appendix