Sea Weather Station

Vasileios Zoidis, Iasonas Lamprinidis, Ioannis, Michalainas, Savvas Tzanetis

May 2025

Repository:
github.com/ioannisam/SeaWeatherStation

Abstract

A brief overview of the Sea Weather Station project: goals, key features, and overall system architecture.

Contents

1 Introduction

This project implements a *remote weather station*. The premise is to use multiple Arduino devices to create a network of data nodes. These nodes will monitor temperature, pressure and estimated wave shore impact/height at sea level. Collected data will then be displayed in a web application, to be used by fishermen, researchers or hobbyists.

1.1 Motivation and Objectives

1.2 Report Structure

This report's structure mimics the workflow of our app. We will be explaining from the bottom up the architecture, design choices and development stages of our project.

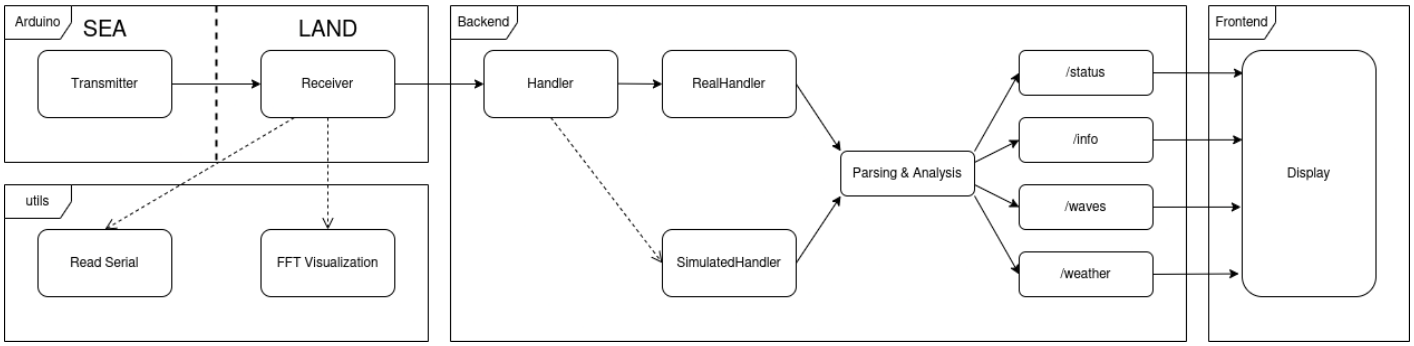

Figure 1: Workflow

2 Arduino

2.1 Hardware Components

2.1.1 Sensor Selection

2.1.2 Wiring and Pinout

2.1.3 Enclosure and Mounting

2.2 Transmitter Module

2.2.1 Data Acquisition Loop

2.2.2 Transmission

2.2.3 Power Management

2.3 Receiver Module

2.3.1 Reception

2.3.2 Serial Output

3 Backend

In this section we discuss the backend's features and capabilities, as well as its architecture. The role of the backend is to capture raw data from the sensor, analyze it, package it and send it to the frontend to be displayed in a meaningful way.

3.1 Data Ingestion

The data ingestion system is designed with a flexible architecture that accommodates both development and production environments through two specialized handlers; `SimulatedHandler` and `RealHandler`. Both handlers expose a consistent interface through the `SerialHandler` facade, which transparently selects the appropriate implementation based on the configured operational mode (`DEMO` or `DEPLOY`). This design pattern enables the application to operate identically regardless of data source, facilitating a smooth transition between development and production environments.

3.1.1 Simulated Handler

The `SimulatedHandler` provides a simulation environment for development and testing purposes. It generates realistic sensor data that mimics actual hardware outputs. The simulation incorporates deliberate variability -employing sinusoidal patterns with randomized amplitude, frequency and noise components- to realistically model sensor behavior. Data generation occurs in a dedicated thread running at 100Hz, maintaining a rolling buffer of 500 samples (equivalent to 5 seconds of historical acceleration data). This approach enabled us to test data processing pipelines, visualization systems and analytical algorithms without requiring physical sensor hardware.

3.1.2 Real Handler

The `RealHandler` establishes and maintains communication with physical Arduino-based sensor hardware via serial connection. It implements several resilience features including connection retry logic with exponential backoff, error handling for disconnection scenarios and automatic reconnection attempts. The handler continuously reads and parses incoming serial data through a dedicated thread, transforming raw string input into structured data objects according to the expected protocol format. Invalid or corrupted data is filtered through parsing logic that validates ranges and formats. Similar to its simulated counterpart, this handler maintains a time-series buffer of recent acceleration readings, enabling applications to analyze trends and patterns in the most recent sensor data.

```
================================================
             INITIALIZING DEPLOY MODE
          USING HARWARE ARDUINO SENSORS

➢ Attempt 1/3
Connection failed: [Errno 2] could not open port /dev/ttyACM0: [Errno 2] No such file or directory: '/dev/ttyACM0'
Retrying in 2s...
➢ Attempt 2/3
Connection failed: [Errno 2] could not open port /dev/ttyACM0: [Errno 2] No such file or directory: '/dev/ttyACM0'
Retrying in 4s...
➢ Attempt 3/3
Connection failed: [Errno 2] could not open port /dev/ttyACM0: [Errno 2] No such file or directory: '/dev/ttyACM0'
Retrying in 8s...

CRITICAL HARDWARE ERROR
Failed to connect to /dev/ttyACM0

DEPLOY MODE FAILED - ARDUINO CONNECTION REQUIRED
Error: Max connection attempts exceeded
Either connect Arduino or restart with DEMO mode
[ble: exit 1]
```

Figure 2: RealHandler Resilience

3.2 Data Processing

3.2.1 Wave Analysis

The wave analysis module implements a frequency-domain approach to extract meaningful metrics from raw accelerometer data. It performs Fast Fourier Transform (FFT) to identify the dominant wave frequency, filtering out low-frequency components (below 0.1 Hz) to eliminate sensor drift and environmental noise. The system converts raw accelerometer readings from LSB units to standard m/s^2 using a calibration factor derived from the MPU6050 sensor's sensitivity at $\pm 2g$ range (datasheet). A physics-based heuristic transforms the acceleration variance into estimated wave height by applying appropriate scaling based on wave frequency, yielding actionable metrics that describe sea state conditions. The module provides a clean interface that handles edge cases such as insufficient data points and returns normalized, rounded values suitable for downstream processing.

3.2.2 Mathematical Background

The shore impact analysis implements established coastal engineering models to translate wave metrics into practical inundation forecasts.[1] The system calculates fundamental wave parameters including period (reciprocal of frequency) and deep-water wavelength using standard gravity-based equations. The core of the analysis employs the Iribarren number (surf similarity parameter) to classify wave behavior as either dissipative or reflective/intermediate, applying the appropriate runup estimation formula for each regime. The model classifies hazard zones based on runup thresholds, with values exceeding 0.9 meters designated as high-risk (VE zone).[2] The final inundation distance is calculated by applying the slope correction factor, providing horizontal reach estimates critical for coastal planning.

3.3 API Endpoints

In this section we analyze the API endpoints available for the frontend to call

3.3.1 Status

The /status endpoint provides essential system telemetry for monitoring the operational state of the device. It returns a JSON payload containing connectivity status, power information (battery percentage), and geolocation coordinates derived from the sensor data stream. The endpoint also includes signal strength metrics (measured in dBm) to facilitate deployment troubleshooting and network quality assessment. By serving as a lightweight health check, this endpoint enables rapid verification of device functionality without interrogating the more resource-intensive data processing pipelines.

3.3.2 Info

The /info endpoint exposes environmental context data captured by the sensor array, specifically temperature (in degrees Celsius) and barometric pressure (in hectopascals).

3.3.3 Waves

The /waves endpoint delivers processed wave characteristics and their projected coastal impact. It retrieves the acceleration time-series buffer from the sensor handler and passes it through the wave analysis pipeline to extract frequency and height parameters. These metrics are then fed into the shore impact model to calculate vertical runup height, FEMA flood zone classification, and horizontal inundation distance based on a predefined slope parameter (set to 0.05 const). This endpoint represents the system's primary analytical output, transforming raw sensor data into actionable coastal flooding information delivered in a standardized JSON format.

3.3.4 Weather

The /weather endpoint enriches the system's data with third-party meteorological information by interfacing with the OpenWeatherMap API. It dynamically constructs API requests using the device's current geolocation coordinates and securely transmits the authentication key from environment variables. The endpoint parses the response to extract wind parameters (speed and direction) along with general weather conditions and human-readable descriptions.

3.4 Test Functions and Utilities

3.4.1 Package Check

The package check utility verifies that all dependencies meet the required version specifications, preventing compatibility issues. It generates a formatted tabular output with clear visual indicators for each package's status: successfully meeting requirements (), outdated versions requiring updates (Too Old), or missing packages that need installation (Not Found).

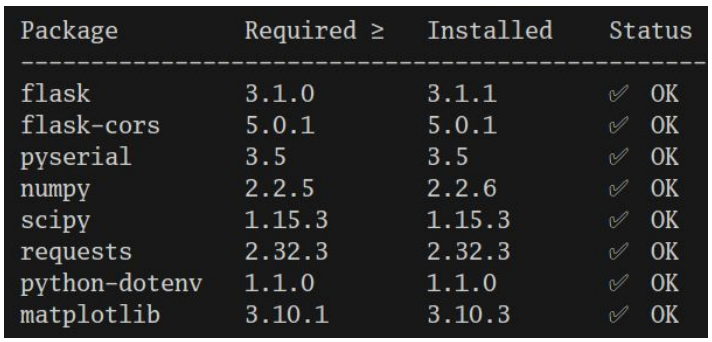

Figure 3: Package Check

3.4.2 Serial Screening

The serial screening utility provides direct access to the raw data stream from connected Arduino hardware, bypassing the application's processing pipelines. It establishes a serial connection to the configured port using the same parameters as the main application (9600 baud) and continuously reads, parses and displays each data frame in real-time. This tool was particularly useful during initial deployment to confirm proper hardware operation before launching the complete application stack.

3.4.3 FFT Visualization

The FFT visualization tool enables detailed analysis of sensor data characteristics through both time and frequency domain representations. It captures a configurable duration of acceleration data (default 30 seconds) using the same SerialHandler interface as the main application, ensuring consistency between visualization and production operation. The tool produces a dual-panel plot showing raw acceleration values (in m/s^2) against time in the upper panel and the corresponding frequency spectrum via Fast Fourier Transform in the lower panel. The visualization highlights the detected dominant frequency with a vertical marker. This dual representation allowed us to correlate time-domain patterns with their frequency components, aiding in algorithm refinement and system tuning.

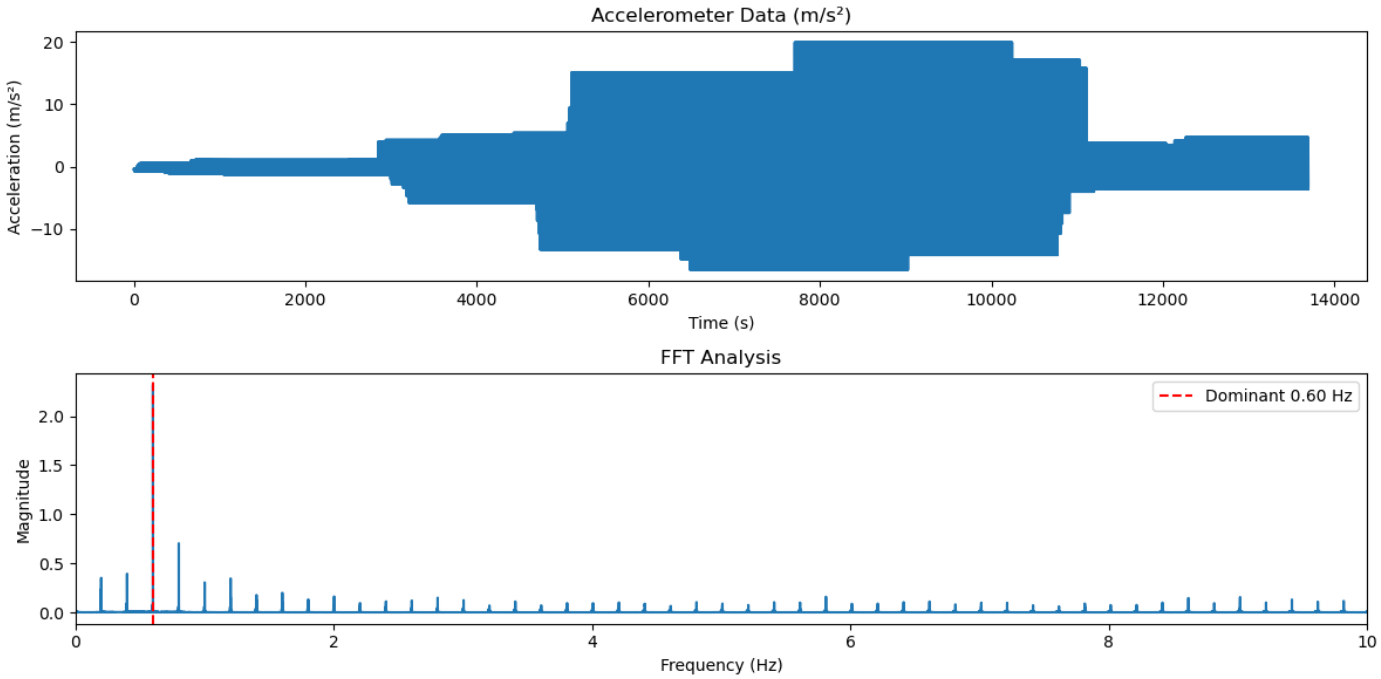

Figure 4: FFT Visualization

3.4.4 Cleaning

The cleaning utility implements system maintenance functionality through a shell script that removes Python cache files. This utility improves development workflow by preventing stale bytecode from causing unexpected behavior after code changes, particularly when performing major refactoring.

4 Frontend

4.1 User Interface Design

4.1.1 Wireframes and Mockups

4.1.2 Look and Feel

4.2 Data Visualization

4.2.1 API Integration

4.2.2 Real-time Data Flow

4.2.3 Real-time Charts and Graphs

4.3 User Interactions

4.4 Testing and Accessibility

4.4.1 Cross-browser Compatibility

5 Conclusions

5.1 Summary of Achievements

5.2 Lessons Learned

5.3 Future Work

Future work includes:

- use solar or aeolian power to cahrge the battery
- proper battery level implementation for the /status endpoint
- adding more sensors
-

A User Manual

How to operate our Weather Station:

1.
2.
3.
4.

B Bill of Materials

Item	Quantity	Price (€)
Arduino Uno R3	1	A
DHT22 Temperature Sensor	2	B
BMP280 Pressure Sensor	1	C
ESP8266 Wi-Fi Module	1	D
Jumper Wires (pack)	1	E
USB Power Cable	1	F
Total		**G**

Table 1: Bill of Materials

C Tools and Frameworks

The following tools and frameworks were used:

- Arduino IDE (Sensor Interaction)
- Python with Flask (Backend API)
- TypeScript with React (Frontend UI)

References

[1] Stockdon, H. F.; Holman, R. A.; Reichmuth, A.; Van Ormondt, M. Estimation of shoreline position and change using airborne LiDAR data. *Coastal Engineering* **53**(3), 2006.

[2] Federal Emergency Management Agency (FEMA). Coastal Floodplain Mapping Report. November 2022.