Wireless Bike Computer

Introduction

This project aims to build a simple but versatile wireless bike computer based on two microcontroller modules.

- It measures wheel cadence and computes basic cycling metrics such as speed, trip distance, trip average speed, trip time and total distance.
- It provides a touchscreen LCD interface for viewing metrics and changing settings.
- It can connect to an Android phone over Bluetooth in order to receive location data and display maps.
- It is battery-powered and rechargeable through USB-C.

The idea behind this project came from the desire to build an open-source bike computer that is more affordable than commercial solutions such as Garmin devices, while still providing the main features needed in practice.

This project is useful both as an educational embedded systems project and as a proof of concept for a customizable and low-cost bike computer. For the author, it is also an opportunity to work on low-power firmware, wireless communication, UI rendering, persistent storage, and sensor integration in a single system.

General Description

The system is split into two main modules:

- **Wheel module** - mounted on the bike fork; measures wheel revolutions using a Hall effect sensor and a magnet attached to a spoke.
- **Main module** - mounted on the handlebars; receives the wheel data, computes user-visible metrics, manages the touchscreen UI, and communicates with the Android phone.

The two modules communicate wirelessly using **ESP-NOW**. The main module communicates with the phone using **Bluetooth Low Energy (BLE)**.

System Overview

The project contains the following hardware and software components:

1. Wheel Module

The wheel module is designed to be low power and is responsible for measuring how long each wheel revolution takes.

Hardware components:

- Seeed Studio XIAO ESP32-C6
- US5881 unipolar Hall effect sensor
- magnet attached to a wheel spoke
- 5V DC-DC boost converter
- LiPo battery

Software responsibilities:

- configure wake-up sources for the ULP (ultra low-power) core
- detect Hall sensor triggers
- measure the time interval between wheel revolutions
- buffer measurements in RTC memory
- wake the HP (high-performance) core when data must be transmitted
- send data to the main module using ESP-NOW

2. Main Module

The main module is the central processing and UI unit.

Hardware components:

- Seeed Studio XIAO ESP32-S3
- 2.8” LCD module with ILI9341 controller
- XPT2046 touchscreen controller
- SD card reader
- LiPo battery

Software responsibilities:

- receive wheel cadence data from the wheel module
- compute metrics such as speed and distance
- display metrics and menus using LVGL
- store persistent data using NVS (non-volatile storage)
- connect to an Android phone over BLE
- load map tiles from the SD card and render them on screen

3. Android Phone

The Android phone acts as an external data source for the maps feature.

Responsibilities:

- pair with the bike computer over BLE
- provide current location data to the main module

Module Interaction

The high-level interaction flow is the following:

1. The wheel module detects wheel revolutions using the Hall effect sensor.
2. It buffers timing data locally.
3. At predefined moments, the wheel module transmits the buffered data to the main module over ESP-NOW.
4. The main module receives the packet, computes user-facing metrics and updates the LCD interface.
5. When the user opens the maps tab, the main module connects to the Android phone over BLE, receives location data, loads the necessary map tiles from the SD card, and displays the current position.

Block Diagram

Hardware Design

Bill of Materials

1. Wheel Module

- 1x Seeed Studio XIAO ESP32-C6
- 1x US5881 unipolar Hall effect sensor
- 1x 10uF 50V X7R capacitor
- 1x 100nF 50V X7R capacitor
- 1x DC-DC 5V boost converter
- 1x 3.7V 500 mAh LiPo battery (902030)
- 1x JST 2.0 cable connector
- 1x 10k ohm pull-up resistor

2. Main Module

- 1x Seeed Studio XIAO ESP32-S3
- 1x 2.8" LCD module: ILI9341 LCD + XPT2046 touch controller + SD card reader
- 1x 10uF 50V X7R capacitor
- 1x 100nF 50V X7R capacitor
- 1x 3.7V 2000 mAh LiPo battery (103450)
- 1x JST 2.0 cable connector

Hardware Architecture

Wheel Module

The wheel module is mounted on the bike fork. A magnet attached to a wheel spoke passes near the Hall effect sensor once per wheel revolution. The sensor output is connected to the microcontroller, which measures the time between triggers.

Because the US5881 operates at a voltage higher than the 3.3V rail used by the ESP32-C6, a DC-DC boost converter is used to provide 5V to the sensor. This is not ideal from a power-consumption perspective, but it was chosen because the sensor is easy to source and integrates well in this proof-of-concept design.

The ESP32-C6 uses both its high-performance core and its ultra-low-power core in order to reduce overall energy consumption.

A 10uF bulk capacitor is used for stabilizing the 3.3V rail. A 100nF decoupling capacitor is used for the US5881 sensor, as per its datasheet.

Main Module

The main module is mounted on the handlebars. It contains the display, touch interface, SD card access, wireless communication, and persistent metric storage using ESP-IDF NVS.

The LCD, touch controller, and SD card reader all use the SPI bus. To save pins, they share the same SPI clock, MOSI, and MISO lines, while each device has its own chip-select line.

The display module is powered via the 3.3V rail, which is stabilized using a 10uF bulk capacitor. A 100nF decoupling capacitor is placed between the display module's VCC and GND pins.

Pinout

Wheel Module Pinout

ESP32-C6 Pin	Function
GPIO0	Hall effect sensor OUT

Main Module Pinout

ESP32-S3 Pin	Function
GPIO1	Capacitive touch wake pin
GPIO2	Touch CS
GPIO3	LCD backlight PWM
GPIO4	LCD D/C
GPIO5	LCD Reset
GPIO6	LCD CS
GPIO7	SPI CLK (shared)
GPIO8	SPI MISO (shared)
GPIO9	SPI MOSI (shared)
GPIO43	SD Card CS
GPIO44	Free

Pinout Design Notes

- The LCD, touchscreen controller, and SD card reader share a single SPI bus, but individual chip-select lines are used.
- GPIO3 is used as the LCD backlight pin. Since it is also a strapping pin on the ESP32-S3, the external circuit connected to it must not force an invalid boot state. If abnormal boot behavior is observed, this pin assignment should be reconsidered or the external circuit should be adjusted so it does not interfere with the required boot strapping state.
- GPIO43 and GPIO44 are typically associated with UART functionality. In this project, GPIO43 is assigned to the SD card chip-select line, therefore the debugging and console output through these pins will not be available.
- GPIO1 is used as a wake-up input for user interaction after the device enters sleep mode.

Electrical Schematic

Software Design

Development Environment

The firmware is developed using:

- **ESP-IDF** as the main framework
- **Visual Studio Code** with the ESP-IDF extension
- C and C++ as the primary programming languages
- standard ESP-IDF build tools for flashing, monitoring, and configuration
- **Android Studio**, **Gradle**, and **Kotlin** for the Android application
- **Python** for offline map-tile conversion tooling
- **QGIS** and **MapTiler** for preparing raster map tiles

Third-Party Libraries / Components

- **ESP-IDF drivers and services**
  - GPIO
  - SPI
  - NVS (Non-Volatile Storage)
  - BLE
  - Wi-Fi / ESP-NOW
  - sleep and power-management APIs
  - SD card support
- **LVGL**
  - used for user interface rendering
- **Squareline Studio**
  - used for designing the UI
- **Android Bluetooth LE and Location APIs**
  - used by the phone application to scan for the bike computer and send current coordinates
- **Python Pillow / NumPy**
  - used by the map conversion tool to convert raster tiles to RGB565

Wheel Module Firmware

The wheel module uses both the HP core and the ULP core of the ESP32-C6.

Workflow

1. The HP core boots and configures the wake-up sources for the ULP core, then goes to deep sleep.
2. The ULP core can wake up either:
   1. periodically, from a timer
   2. from the Hall sensor GPIO trigger
3. If the wake-up source is the Hall sensor:
   1. the ULP core measures the interval since the previous wheel revolution
   2. it stores the value in RTC memory

4. If the wake-up source is the timer:
   1. the ULP core updates a timeout counter
   2. if several timer wake-ups occur without wheel movement, the wheel is assumed to have stopped
   3. in that state, periodic transmission can be reduced and the timer source can be disabled until motion is detected again
5. After enough intervals have been buffered, or after a predefined timeout, the ULP core signals the HP core.
6. The HP core wakes up, reads the shared data from RTC memory, builds a packet, and sends it to the main module using ESP-NOW.
7. The HP core then returns to deep sleep.

Data Sharing

The ULP and HP cores communicate through shared variables placed in RTC memory. These variables contain:

- buffered revolution intervals
- metadata such as counters or status flags

Advantages of this approach

- lower power consumption than keeping the HP core active all the time
- efficient use of the ESP32-C6 low-power capabilities
- reduced wireless transmissions by batching multiple measurements

Main Module Firmware

The main module firmware is responsible for computation, rendering, storage, and communication.

Responsibilities

- receive packets from the wheel module using ESP-NOW
- compute bike metrics:
  - current speed
  - trip distance
  - trip average speed
  - trip time
  - total distance
- show metrics through a touchscreen interface
- allow the user to change settings such as:
  - wheel circumference
  - metric / imperial units

- store persistent metrics in non-volatile storage
- connect to an Android phone over BLE
- load and render map tiles from the SD card

UI Design

The user interface is implemented with **LVGL** and designed in **Squareline Studio**. The UI contains the following main screens:

- **Metrics tab**
  - current speed
  - trip distance
  - trip average speed
  - trip time
  - total distance
- **Maps tab**
  - current user position
  - visible map tiles rendered on screen
- **Settings tab**
  - wheel circumference
  - unit system
  - brightness
  - sleep timeout
  - reset trip button
  - other future settings

Data integrity

The ESP-NOW packets contain a sequence number and cumulative wheel-module counters. The main module uses the sequence number to discard duplicate or stale packets. The cumulative counters make the main module logic simpler: once a packet is accepted, the firmware can compute the delta in wheel rotations and moving time relative to the previous accepted packet.

This approach keeps the main module tolerant of occasional packet loss and avoids complex packet reordering logic. Distance and moving time are derived from accepted wheel-module data, while speed uses the most recent valid wheel periods.

Storage Strategy

Some values, such as total distance or trip statistics, need to survive resets and power cycles. These values are stored in **NVS**.

To avoid excessive write frequency and premature flash wear:

- values are first buffered in RAM
- the persistent storage is updated only at a predefined interval, or when necessary
- ride state is also saved when settings are changed or when the trip is reset

Maps Feature

The maps feature is designed as follows:

- the user enters the Maps tab
- the user presses the pairing button and the main module starts BLE advertising
- the phone provides current location data
- the firmware determines which map tiles are needed
- the tiles are loaded from the SD card
- the tiles are rendered together with the current position marker

The tile system is based on the **slippy map** format.

To reduce CPU overhead:

- map tiles are preprocessed offline
- instead of storing PNG files that require runtime decoding, tiles are stored in a raw format such as **RGB565**
- this allows direct rendering to the display

The map viewport is implemented as an LVGL canvas placed inside the Squareline placeholder container. The canvas size is read at runtime from the UI object, so the viewport can be resized later in Squareline Studio without changing the rendering code. The current position marker is drawn in the center of the viewport, while the map tiles move underneath it.

The firmware uses a fixed zoom level. The SD card path is organized as:

- `/sdcard/maps/<zoom>/<x>/<y>.rgb565`

The RGB565 files are generated offline by a Python CLI tool. The tool receives an input slippy-tile directory, the original image format, and an output directory, then preserves the tile hierarchy while converting each tile to raw RGB565.

To avoid redundant rendering work, the map is redrawn only when the projected location changes by at least a small pixel threshold. The canvas buffer is allocated from PSRAM because the full RGB565 viewport requires a relatively large contiguous buffer.

Wireless Coexistence

The XIAO ESP32-S3 has a single 2.4 GHz radio, so **BLE** and **ESP-NOW** must share the same radio hardware. The firmware relies on radio coexistence support in order to allow both features to operate in the same system.

Power Management

After a predefined period of inactivity the main module enters deep sleep, which in turn powers off most of the components, including the GPIO (which power the display).

The timeout is reset by wheel packets, BLE location activity, and user interaction. The device can be woken up through the dedicated touch input.

Android Application

The Android application is intentionally simple. Its role is not to display the ride metrics, but to act as a bridge between the phone location provider and the ESP32-S3.

The app:

- scans for the bike computer BLE service
- connects as a BLE central/client
- subscribes to connection state changes
- receives GPS/network location updates from Android
- writes coordinates to the ESP32-S3 GATT characteristic
- runs the transfer logic in a foreground service so location updates can continue while the screen is off

The Android UI exposes a single state-dependent action button:

- **Start scanning** when disconnected
- **Stop scanning** while scanning
- **Disconnect** when connected

The app also shows the current connection state, the last phone coordinates, and a short log useful during development and testing.

Algorithms and Data Structures

The main algorithms and data structures used by the project are:

- **interval buffering**
  - the wheel module stores recent wheel-revolution intervals in a circular buffer in the LP-core firmware
- **timeout-based motion detection**
  - repeated timer wake-ups without sensor events indicate that the bike has stopped, reducing unnecessary HP-core wake-ups and radio transmissions
- **metric computation**
  - speed is computed from wheel circumference and revolution interval
  - distance is computed from the number of wheel revolutions
  - ride time is moving time, based on wheel-module movement data rather than wall-clock time

    - metric values are stored internally in metric units, then converted for display if needed
- **speed smoothing**
    - the displayed speed uses a lightweight smoothing factor to avoid sudden jitter from individual wheel periods
- **sequence and cumulative-counter validation**
    - received packets are accepted only if they advance the last accepted packet state
    - cumulative wheel counters are used to compute deltas safely
- **tile selection**
    - determine the subset of map tiles that cover the current visible screen area
    - compute slippy-map tile coordinates from latitude, longitude, and fixed zoom level
- **deferred persistent writes**
    - maintain dirty flags and update NVS at controlled intervals
- **BLE write queue**
    - the Android app keeps one active BLE write and one pending latest location, retrying if Android does not accept or complete a write

Source Structure / Implementation Notes

The repository is split by module and by responsibility:

- `modules/common/`
    - contains the shared ESP-NOW packet definition used by both firmware projects
- `modules/wheel_module/`
    - ESP32-C6 firmware for the low-power wheel sensor node
    - contains the HP-core application and the LP-core ULP program
- `modules/main_module/`
    - ESP32-S3 firmware for the display, UI, storage, ESP-NOW receiver, BLE location service, and map renderer
- `android-app/`
    - Kotlin Android companion app used to scan, connect, and transmit phone coordinates over BLE
- `tools/`
    - helper tools, including the raster-tile conversion CLI

The main module firmware is divided into smaller layers:

- `constants/`
    - board pin assignments, hardware configuration, and application constants
- `hardware/`
    - SPI bus setup, ILI9341 display driver integration, XPT2046 touch integration, and SD card mounting
- `app/`
    - application orchestration, UI callbacks, UI presenter, persistent storage, ride metrics, BLE location service, and map rendering
- `components/ui/`
    - Squareline Studio generated LVGL UI component

The firmware follows an event-driven structure. ESP-NOW receive callbacks, BLE callbacks, timers, and UI callbacks post events to the main application event loop. The application task then updates ride metrics, persistence state, UI state, and map rendering from one place. This avoids doing heavy

work directly inside interrupt-like or stack callback contexts.

For the main module, the UI presenter converts internal data to user-visible labels. Ride metrics remain unit-system agnostic internally and are stored using metric units such as millimetres, microseconds, and kilometres per hour. Unit conversion and formatting are handled when values are presented in the UI.

The map renderer avoids runtime PNG/JPG decoding on the ESP32-S3. Raster tiles are prepared on the computer, converted to raw RGB565, copied to the SD card, and streamed by the firmware. Missing tiles are rendered as a neutral background color so the screen remains usable even if the SD card does not contain every tile around the current position.

The Android application uses a foreground service because Android may stop ordinary background work when the phone screen is off. The service owns scanning, GATT connection state, location updates, write retry logic, and the persistent notification, while the activity only displays state and exposes the action button.

Results and Conclusions

The final system implements the main goals of the project:

- the wheel module measures wheel revolutions while keeping the ESP32-C6 HP core asleep most of the time
- the main module receives wheel packets over ESP-NOW and computes speed, trip distance, average speed, moving time, and total distance
- settings and ride state are persisted across power cycles using NVS
- the touchscreen UI provides metrics, maps, and settings screens
- display brightness and sleep timeout are configurable
- the main module can connect to an Android phone over BLE and receive live coordinates
- offline raster map tiles can be converted to RGB565, copied to the SD card, and rendered on the display with a centered location marker

The project also demonstrates several important embedded-system trade-offs:

- batching wheel measurements reduces radio wake-ups and improves power efficiency
- storing cumulative wheel counters in transmitted packets makes the receiver more robust and simpler
- moving map decoding offline is much more practical than decoding compressed image formats on the microcontroller
- PSRAM is important for framebuffer-like map rendering on the ESP32-S3
- BLE and ESP-NOW can coexist, but the firmware must be careful about radio configuration, connection state, and callback workload

The result is a functional proof-of-concept wireless bike computer. It is not yet a polished commercial device, but it validates the complete data path: wheel sensing, low-power buffering, wireless transfer, metric computation, persistent state, phone-assisted location, and local map display.

Future improvements could include:

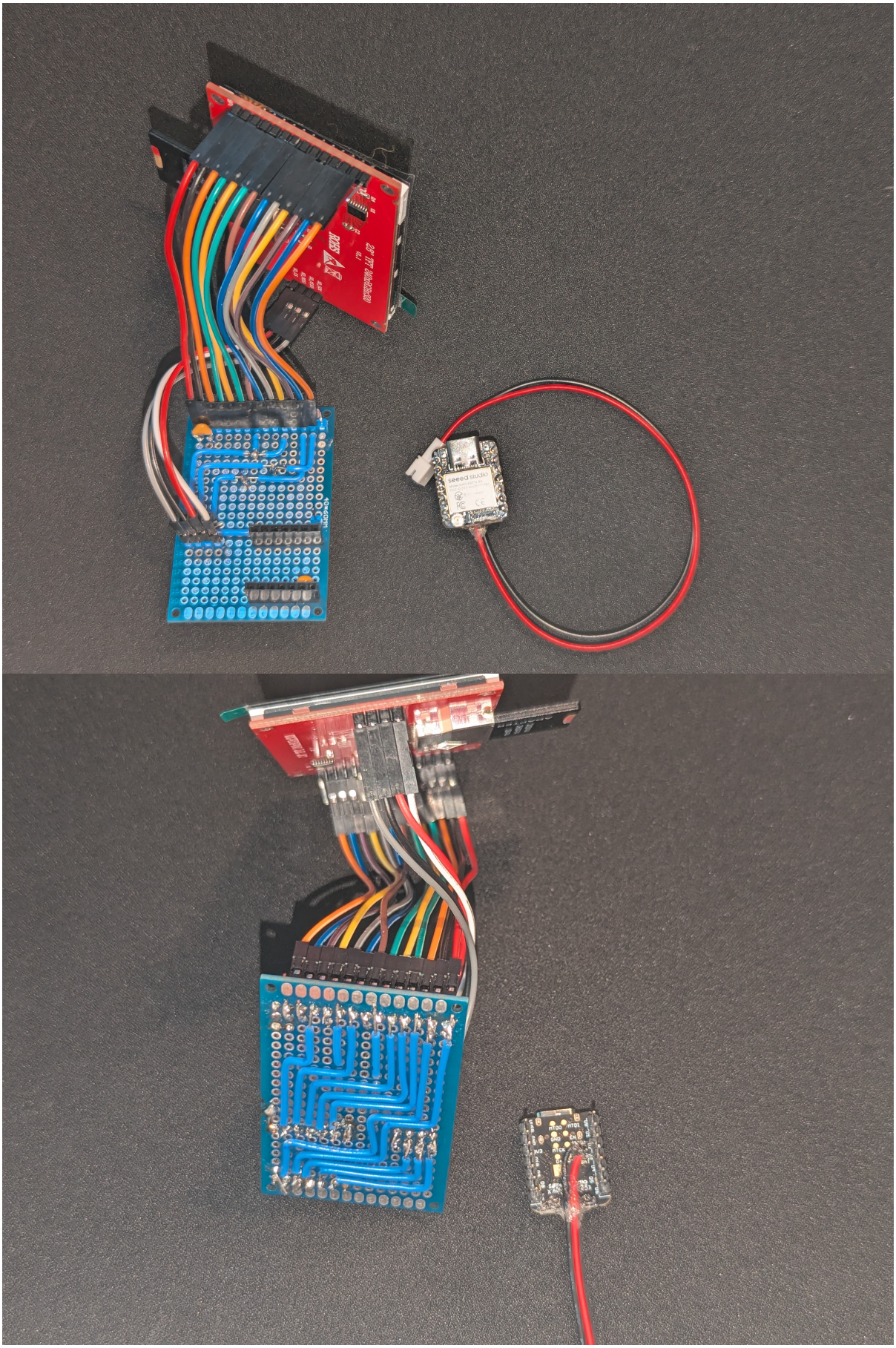
- a custom PCB instead of wiring modules manually
- a more compact enclosure and better weather protection
- battery measurement and charging-state display
- more map zoom levels and route/navigation features
- improved BLE reconnection behavior and phone compatibility testing
- additional ride statistics and data export

Images

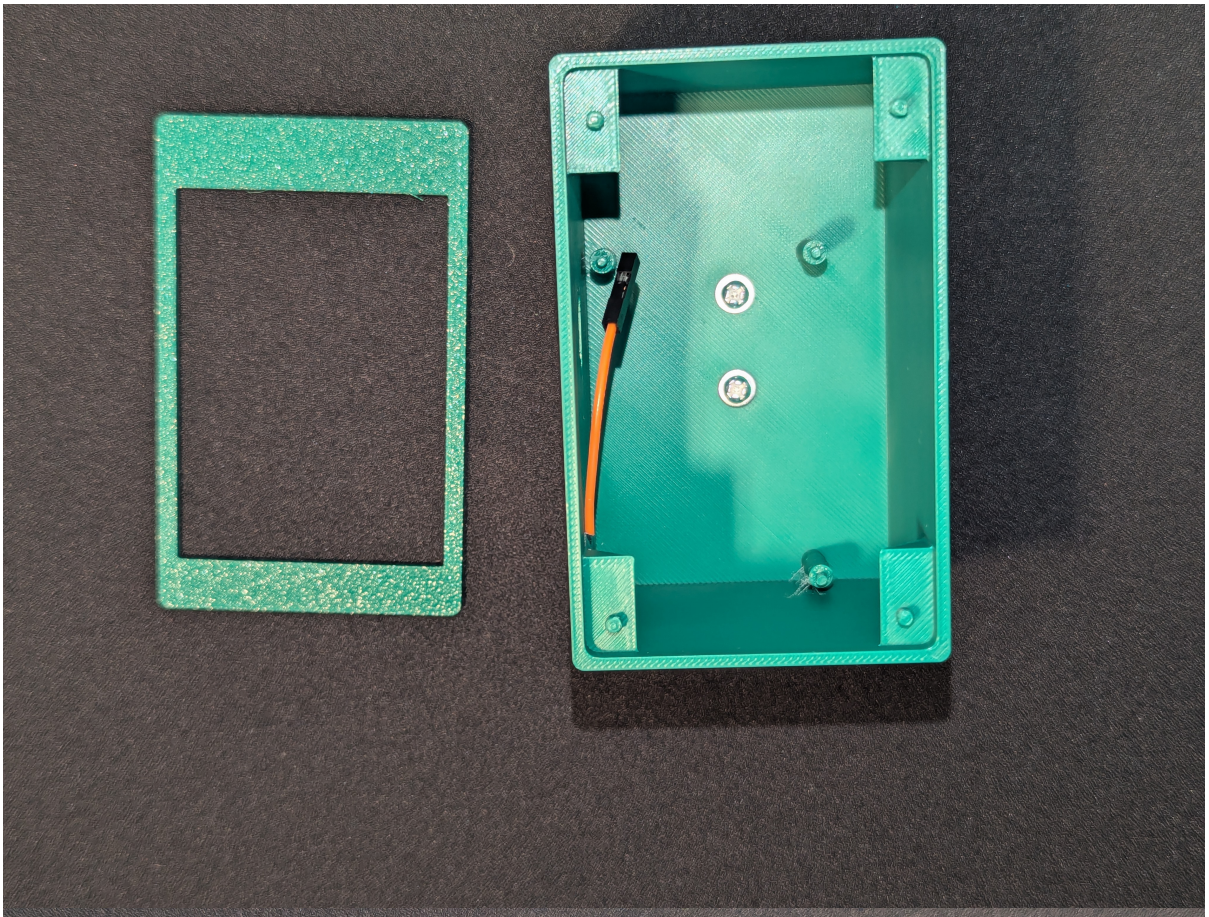
Main Module

Assembled

Board



Case



UI

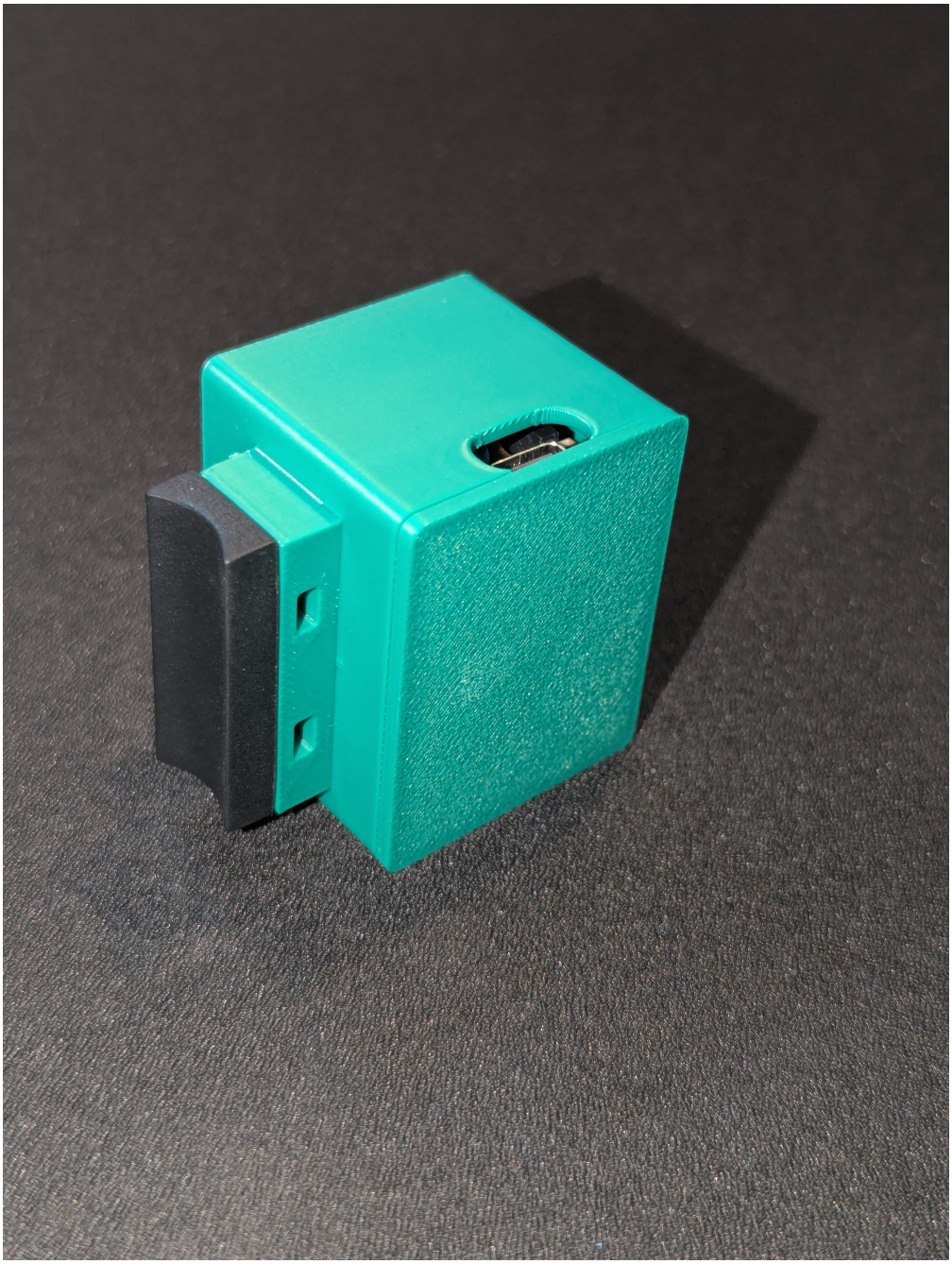
Metrics Screen

Maps Screen

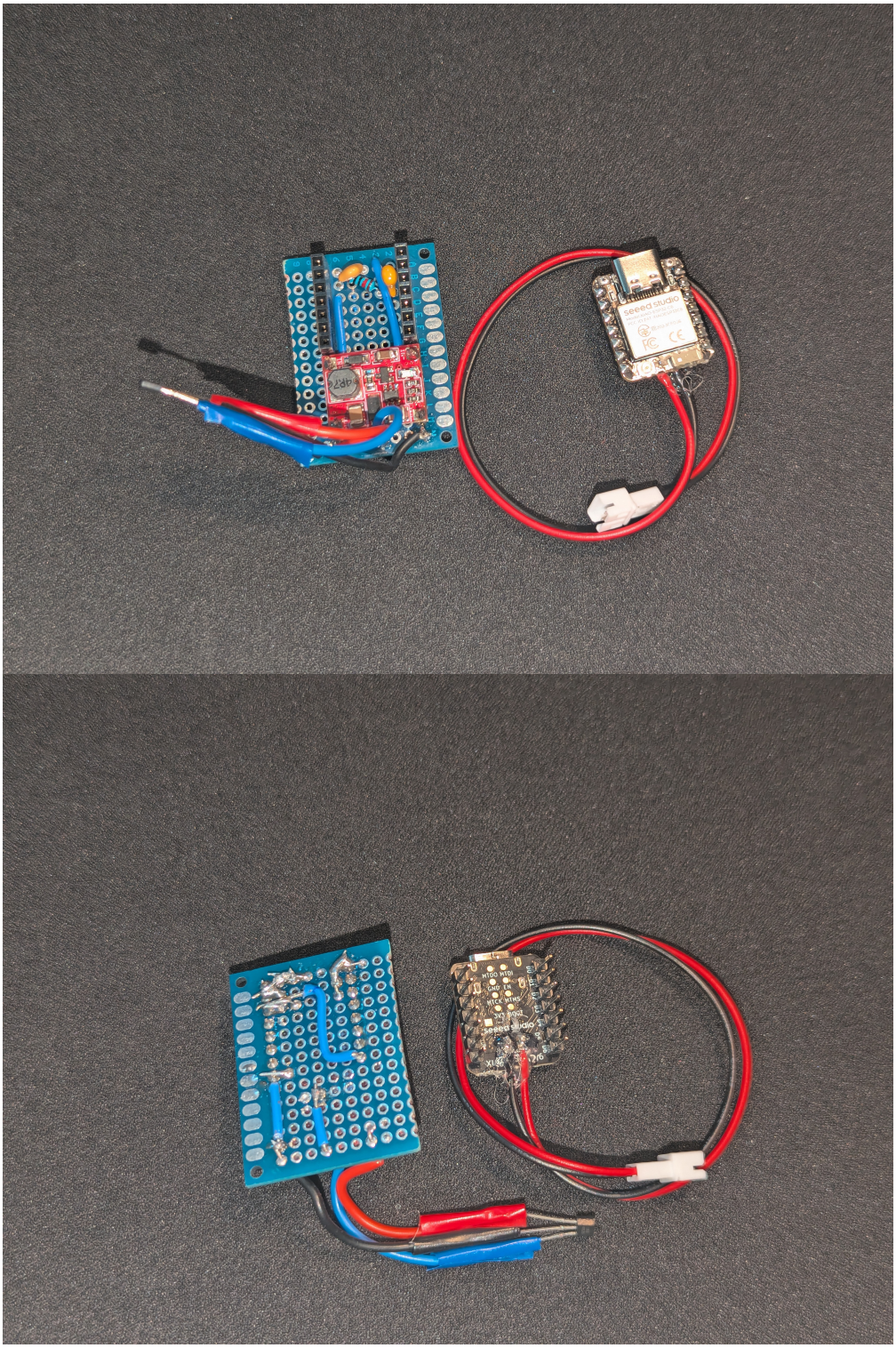
Settings Screen

Wheel Module

Assembled



Board



Case

Download

Github Link

Bibliography / Resources

Below is a list of the main resources used for the project.

Hardware Resources

- Seeed Studio XIAO ESP32-C6
- XIAO ESP32-C6 Wiki
- Seeed Studio XIAO ESP32-S3
- XIAO ESP32-S3 Wiki
- US5881 Hall Effect Sensor
- ILI9341 + XPT2046 + SD Card display module
- DC-DC 5V Boost Converter

Software Resources

- ESP-IDF
- LVGL Documentation
- Squareline Studio
- OpenStreetMap
- Slippy Map Tile Format
- QGIS
- MapTiler
- MapTiler Self-Hosted Maps Documentation
- Pillow Documentation
- NumPy Documentation
- Android Bluetooth LE Documentation
- Android BLE Background Operation
- Android Foreground Service Types
- Kotlin Documentation

Export to PDF