Universidade Tecnológica Federal do Paraná – UTFPR
Departamento Acadêmico de Eletrônica – DAELN
Departamento Acadêmico de Informática – DAINF
Engenharia de Computação
Oficina de Integração 3 (ELEX22) – S71 – 2025/2

Technical Report
BeForBike

Bruno Kunnen Ledesma – brunokunnen@alunos.utfpr.edu.br
David Segalle – davidsegalle@alunos.utfpr.edu.br
João Lucas Marques Camilo – joaolucascamilo@alunos.utfpr.edu.br
Lucas Yukio Fukuda Matsumoto – lucmat@alunos.utfpr.edu.br
Viviane Ruotolo – vivianeruotolo@alunos.utfpr.edu.br

November, 2025

Abstract

This Technical Report details the development of the Integrated Performance and Safety Monitoring System for Cycling (BeForBike), a low-cost solution designed to serve cyclists with budgetary constraints. The system addresses the high cost of commercial power meters (typically above US$ 400) and the need for nighttime vehicle safety. The core technical approach relies on the use of load cells integrated into the pedal for power measurement, combined with onboard data processing via Raspberry Pi [1]. For safety, a Retractable Directional Indicator (Blinker) was implemented. The software architecture is multithreaded, ensuring simultaneous collection, processing of GPS [2] and pedal sensor data, and Bluetooth Low Energy (BLE) [3] communication. Data is persisted and visualized in real-time via a handlebar-mounted display, and ride data can be analyzed using the mobile application developed for Android. The report covers the hardware design of the pedal and the central unit, the multithreaded software architecture, the data model, and the performance analysis of the prototype in a controlled environment.

1 Introduction

1.1 Problems

High-performance cycling requires the continuous monitoring of effort metrics, with power (measured in Watts) being the most reliable indicator of effort and performance. Commercial power meter devices, utilize strain gauges [4] in pedals or cranksets to calculate power. However, the market is dominated by equipment that far exceeds the cost of US$ 400, and comprehensive cycling computers easily reach US$ 500. This cost makes performance monitoring technology inaccessible to a large portion of enthusiasts and amateur athletes, particularly in regions of lower median income, including Brazil. Furthermore, current cycling practices present safety concerns to the rider, hand-direction signaling is ineffective in low-light conditions, whereas adding conventional indicators often compromises the bicycle's aerodynamics [5].

1.2 Objectives

The project's budget is rather limited, a total cost even close to what available market solutions deliver is out of the question. Totalling R$ 1200, close to US$ 225, all parts combined end up costing less than the cheapest power meter pedals available in the market. With this budget we propose a fully fledged solution covering most areas of interest a cyclist may have from a cycling computer system.

Our goal is, with this budget, provide a fully fledged solution for cyclists to utilize as their bicycle computer. Firstly, we gather data from the pedals and GPS to show the rider data in real-time, meaning he can track his ride live and have precise measurements for what he is doing. Additionally, in order to track progress this data must be available to the cyclist whenever desired, in the form of checking information for older rides.

Finally, as safety is always a concern when cycling on the road alongside other cars. For this it is important to make an easy to see blinker so the cyclist can better communicate his intentions with drivers on the road.

2 Solutions

Multiple solutions are available for purchase on the internet, they all cover different parts of the system, though a complete solution similar to what BeForBike attempts to present is difficult to find.

2.1 Existing products

Onboard cycling computers can be cheap, models without GPS and very basic software can be bought for close to US$ 50[1], they, however, don't provide gps or a way to integrate with a power meters. Systems that offer both these capabilities are often far more expensive, often well above US$ 400[2]. These more expensive solutions offer a far more appealing product, however, are held back by their price.

[1]A cheap cycling computer: `https://shopee.com.br/product/255247590/8631939258`

[2]Examples can be found at: `https://produto.mercadolivre.com.br/MLB-3579819695-ciclocomputador-garmin-edge-1050-preto-com-gps-wi-fi-_JM` and `https://www.mercadolivre.com.br/gps-garmin-edge-540-010-02694-02-cor-preto/p/MLB27983213`

No really cheap power meter is available commercially, so they all face similar issues to cycling computer, often costing more than most modest income households have available for the hobby[3].

Blinkers are often cheap[4] due to them being a far simpler system than those mentioned above.

2.2 Our solution

To address the constraints and goals established in the objectives, the proposed system[5] is designed to deliver a comprehensive bicycle computer solution while remaining within the strict budget of R$ 1200. The project therefore combines low-cost hardware components with custom firmware and data-processing strategies to replicate the core functionalities of commercial cycling computers and power meter pedals at a fraction of the price.

First, to provide real-time ride monitoring, the system integrates sensor on a pedal, as well as a GPS module capable of processing and displaying real time data. Sensor readings composed of cadence, estimated power output, Calories consumed, speed and travelled path are continuously collected and transmitted to the display interface.

Second, to support post-ride analysis, the device can connect to a mobile phone, during each session, all relevant metrics are recorded in a standardized format, allowing the cyclist to review historical rides through an external mobile platform after the data is automatically exported. This provides long-term progress tracking despite the system's low-cost architecture.

Lastly, to improve safety during road cycling, the project incorporates a high-visibility electronic blinker. The blinker uses an LED strip and a simple interaction mechanism to ensure that turn signals are easily noticeable to surrounding drivers. The safety module improves communication between the cyclist and nearby vehicles, increasing safety without a significant impact in the system's overall cost.

[3] An example can be found at: `https://uaiadventure.com.br/produto/sram-rival-axs-dub-powermeter-2x12-velocidades-48-35t-pedivela-speed/`

[4] An example can be found at: `https://shopee.com.br/product/805294657/41624344577`

[5] Our blog: `https://verbose-pigeon-06a.notion.site/Be-for-Bike-25603987007e8078ab14c2dd9d0d7a4b?pvs=143`

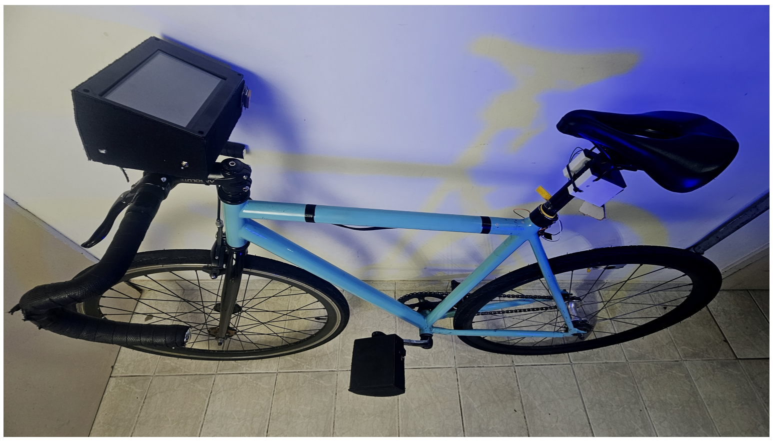


Figure 1: Complete project mounted on bike.

2.3 Requirements

2.3.1 Stakeholders

STKH1 - Cyclists.

STKH2 - Mobile App Users.

STKH3 - Project Team.

STKH4 - University professors.

2.3.2 Stakeholders Needs

STN1 - Cyclists need feedback on performance metrics like power, speed, and cadence.

STN2 - Mobile app users need history of bicycle ride metrics.

STN3 - Project team need clear requirements and testable outputs.

STN4 - University professors need project documentations.

2.3.3 Software Functional Requirements

SFR1 - The bicycle computer screen must display power output of the cyclist.

SFR2 - The bicycle computer screen must display the calories consumed.

SFR3 - The bicycle computer screen must display the current local map.

SFR4 - The bicycle computer screen must display the path travelled on map.

SFR5 - The bicycle computer screen must display the distance traveled.

SFR6 - The bicycle computer screen must display the current speed.

SFR7 - The bicycle computer screen must display the current cadence.

SFR8 - The bicycle computer screen must display the current date and time.

SFR9 - The bicycle computer screen must display the GPS signal status.

SFR10 - The bicycle computer screen must display the smartphone Bluetooth connection status.

SFR11 - The bicycle computer screen must display the crank sensor Bluetooth connection status.

SFR12 - The bicycle computer screen must display the direction of the blinker.

SFR13 - The app must display history of power applied to the crank during the ride.

SFR14 - The app must display total burnt calories of the ride.

SFR15 - The app must display the path travelled on the map.

SFR16 - The app must display history of the altitude.

SFR17 - The app must display the distance traveled.

SFR18 - The app must display history of the speed/cadence.

SFR19 - The app must display the bicycle computer Bluetooth connection status.

SFR20 - The app must be able to control which function will be displayed on the bicycle computer screen.

SFR21 - The app must save information in database.

SFR22 - The app must display a list of rides sorted by date and time.

SFR23 - The app must display history of the cyclist power output.

SFR24 - The app must have a user-friendly interface.

SFR25 - The crank switcher must start a ride when switched on.

SFR26 - The crank switcher, when switched off, can end a ride.

SFR27 - If requirement SFR26 is done, the bicycle computer must store recent data when there is no current bluetooth connection.

SFR28 - The system must send recent data from Raspberry to the app, after the ride is finished.

SFR29 - The bicycle computer must broadcast its BLE device name.

2.3.4 Software Non-functional Requirements

SNFR1 - The bicycle computer must calculate the calories consumed from the force applied to crank.

SNFR2 - The bicycle computer must calculate the traveled distance by GPS.

SNFR3 - The bicycle computer must calculate speed by crank RPM.

SNFR4 - The bicycle computer must calculate cadence and RPM by inertial measurement unit on the crank.

SNFR5 - The bicycle computer must retrieve current date and time from GPS.

SNFR6 - The bicycle computer must estimate path on map by GPS coordinates.

SNFR7 - The database must have timestamp for each entry.

SNFR8 - The app must receive data from bicycle computer through Bluetooth connection.

SNFR9 - The smartphone must allow external app installation.

SNFR10 - The bicycle computer system must take data from multiple Bluetooth devices at once.

SNFR11 - The speed unit must be km/h.

SNFR12 - The calories unit must be kcal.

SNFR13 - The power unit must be Watts.

SNFR14 - The distance unit must be meters.

SNFR15 - The cadence unit must be rpm.

SNFR16 - The battery level unit must be percentage.

SNFR17 - The Raspberry must send data to the app using json format

2.3.5 Hardware Functional Requirements

HFR1 - The handlebar must have a box with GPS module, display and Raspberry Pi.

HFR2 - The bicycle computer on/off switch must switch on/off the bicycle computer system with exception to the pedals.

HFR3 - The left LED switcher must switch on/off the LED strip and stepper motor when pressed.

HFR4 - The right LED switcher must switch on/off the LED strip and stepper motor when pressed.

HFR5 - The LED strip must blink when it is powered on.

HFR6 - The crank switcher must switch on/off the power meter system.

2.3.6 Hardware Non-functional Requirements

HNFR1 - The bicycle computer battery must be rechargeable.

HNFR2 - The crank sensor battery must be rechargeable.

HNFR3 - The bicycle computer must have Bluetooth Low Energy (BLE) connection.

HNFR4 - The crank sensor must have Bluetooth Low Energy (BLE) connection.

HNFR5 - The smartphone must have Bluetooth Low Energy (BLE) connection.

HNFR6 - The bicycle computer must save current date and time in RTC.

HNFR7 - The bicycle computer must have recharging port.

HNFR8 - The crank sensor must have recharging port.

HNFR9 - The bycicle computer circuit must be assembled on universal board.

HNFR10 - The crank sensor circuit must be assembled on universal board.

2.3.7 Mechanical Functional Requirements

MFR1 - The bike must be single geared.

MFR2 - The blinker must be a moveable arrow.

MFR3 - The LEDs strip must be on the blinker.

MFR4 - The blinker must move to left when left button is pressed.

MFR5 - The blinker must move to the right when right button is pressed.

MFR6 - The blinker must stay in center position when the the LEDs are off.

MFR7 - The blinker must be coupled behind the seat.

MFR8 - The power meter system must be coupled on the crank.

2.3.8 Mechanical Non-functional Requirements

MNFR1 - The crank sensor must be calibrated in a wheatstone-bridge strain gauge configuration.

MNFR2 - The load on the crank arm must be extracted from strain gauge sensors.

MNFR3 - The crank sensor must have a protection case.

2.3.9 Anti-requirements

PAR1 - The system must not support multi-gear shifting nor derailleur feedback integration.

PAR2 - The system must not store user data in the cloud.

PAR3 - The system must not allow database data editing after saving.

PAR4 - The power measurement system must not be temperature adjusted.

PAR5 - The app must not display more than one ride at once on the map.

2.3.10 Restrictions

R1 - The system must operate within BLE standard power and communication constraints.

R2 - The crank arm design must accommodate only single gear bikes.

3 Theoretical basis

3.1 Power measurement in cycling

The instantaneous power P (W) (Eq. 1) can be calculated as the product of the pedal speed v (m/s) with the instantaneous force F (N) applied to it. The speed is calculated from cadence and the force comes from the load cells.

$$P = F \cdot v \tag{1}$$

From power and cadence, making use of the fact the bike is fixed gear, we can calculate calories consumed Q (cal) (Eq. 2) from the joules generated, where P (W) is the instantaneous power and t (s) is the time, multiplying by a constant.

$$Q = P \cdot t \cdot \frac{0.23900574}{1000} \tag{2}$$

Cadence can also be used to generate speed (Eq. 4) and distance (Eq. 3) where s (m) is the position, f (rpm) is the cadence and t (s) is the time, utilizing the 44 by 14 teeths cog system and the wheel circunference of 2.37384 meters.

$$s = \frac{f}{60} \cdot t \cdot \frac{44}{14} \cdot 2.37384 \tag{3}$$

$$v = \frac{s}{t} \tag{4}$$

3.2 Using the Fast Fourier Transform (FFT) for cadence

Cadence is crucial for power calculation. The pedal rotation (in a full 360° cycle) can be inferred by analyzing the IMU acceleration data. The pedal movement generates an approximately sinusoidal signal which, when analyzed in the frequency domain via FFT [6], allows for the identification of the fundamental frequency corresponding to the cadence. The accuracy of the FFT algorithm, including the use of windowing to reduce spectral leakage, is a determining factor in the precision of the BeForBike power measurement system.

3.3 Wireless communication: Bluetooth Low Energy (BLE)

Bluetooth Low Energy is a wireless personal area network technology operating in 2.4 GHz radio band, primarily designed for significantly reduced power consumption over classic Bluetooth. Its architecture is defined by a protocol stack, featuring a controller (which handles the physical and link layers for radio management and connection establishment) and a host (which manages higher-level functions, security, and data organization via protocols like ATT, and GATT). At the foundational level, the Generic Access Profile defines the high-level roles: the peripheral (server), which is the discoverable, low-power device that sends advertising packets, and the central (client), which scans for and initiates connections. Once connected, the Generic Attribute Profile (GATT) dictates the data exchange structure, organizing information into Services and Characteristics. Data transfer between the GATT server and the GATT client is primarily managed via write/read operations or, critically for low-latency updates, via notifications, where the server spontaneously pushes data to the client after the client has subscribed to the characteristic.

4 Electronic design

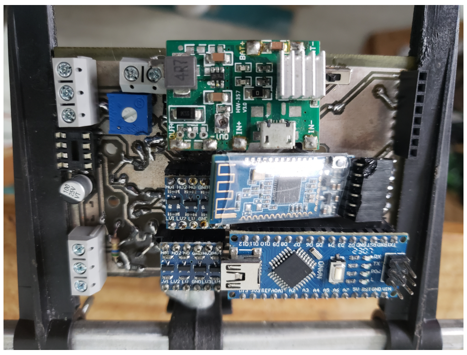
4.1 Schematics drawing

Electronic schematics were drawn using the Computer Aided Design (CAD) tool EasyEDA where there are community contributed parts available.

4.2 PCB Manufacturing

PCBs were made by the team using simple tools due to the high cost of industrial PCB fabrication. A dual side design was chosen to accommodate more components into small area and allow it to fit into small sized boxes as portable devices, as demonstrated in figure 2

The manufacturing process is composed of many steps, they are: Draw schematics in EasyEDA software; Configure PCB trace widths and hole diameters; Run autoroute and reorganize components by minimizing their link distances till all routes are successful; Generate black and white photolite without any component marks, only traces and holes. Back side needs to be generated mirrored; Paint both sides of a double side copper plated phenolite board with black matte spray ink; Load PCB photolite drawing into LaserGRBL and burn the design into both sides of the board, ensuring they are perfectly aligned, using a laser engraving machine; Clean away charred ink by brushing; Etch copper using iron perchlorate till you see no more copper in the unprotected areas; Remove black ink by passing kerosene; Drill vias and holes; Plat all traces with silver nitrate to protect from corrosion; Solder vias using 90° male header pins without plastic, 180° female header pins for the modules and the THD and SMD components; Test all traces with multimeter; Apply varnish mask to protect PCB from scratches and short contact; Place modules and cables into the headers.


(a) PCB shield Arduino Nano

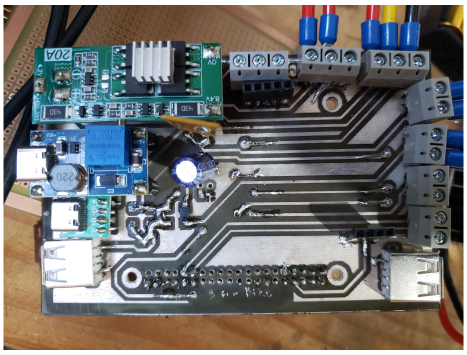

(b) PCB shield Raspberry Pi

Figure 2: PCBs

5 Connection

The system employs a robust BLE architecture to manage data flow between the central unit, a fixed microcontroller and the mobile application. When a ride is active, the Arduino [7] transmits real-time sensor data serially to a dedicated BLE module. This module then uses the Characteristic UUID 0000ffe1-0000-1000-8000-00805f9b34fb to send continuous notifications to the embedded computer. The Characteristic UUID acts as a unique identifier for the specific data stream on the peripheral device, distinct from a MAC address. The embedded computer recognizes this Characteristic UUID to identify and process the incoming real-time data from the crank.

In addition, the embedded computer manages an independent connection with the mobile app, acting as a GATT server, to transfer complete ride files. This connection is established when the embedded computer identifies a device whose BLE Manufacturer Data does not match the fixed identifier of the crank/Nano device. This telemetry data is JSON encoded, Gzip compressed, and transmitted in batches using the Characteristic UUID 12345678-1234-5678-1234-56789abcdef0 via a Write with response procedure. The mobile app, running as an Android Foreground Service, processes the received compressed data in a background executor thread to ensure UI stability, confirming successful database insertion before sending the final GATT response.

6 Peripheral systems

6.1 Pedal

This subsection details the pedal subsystem[6], which is mounted directly on the bicycle. It integrates load cells to measure force and an MPU6050 sensor to track velocity. The primary function of this unit is to collect this sensor data and stream it wirelessly via Bluetooth to our bicycle computer.

6.1.1 Mechanics

The mechanical structure can be separated into four distinct parts, visible in Figure 3 from bottom to top. The first layer, at the bottom, is the electronics protector, a 3D-printed component responsible for protecting the circuit. Above this is the second layer, an "H"-shaped structure used for installing the load cells, which guarantees that force is distributed equally among them. The third layer provides the support for the load cells, allowing them to work properly. Finally, the fourth (top) layer is a box that protects the second and third layers. This top piece also serves as a foot holder, ensuring the applied force is distributed evenly across the load cells below.

[6] Pedal Power Meter code: `https://github.com/ViDaProjects/BeForBike_PowerMeter`

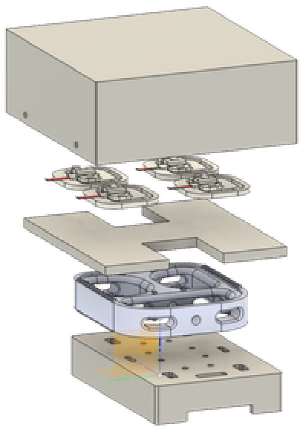


Figure 3: Pedal 3D Model

6.1.2 Electronics

The MPU6050 and Bluetooth module are connected to the Arduino Nano through a bidirectional logic level converter module, allowing them to communicate converting 3.3 V and 5 V. The battery management system (BMS) is a module which connects to the batteries, protecting from overdischarge and allowing the system to be charged with a Micro USB port.

Finally, the load cell portion of the electronic design is installed in a full wheatstone bridge connected to a HX-711 module, which amplifies and serializes the signal for the Arduino to read.

6.1.3 Software

The software implementation uses two primary libraries to collect data from the sensors: a purpose built MPU6050 driver[7] and an HX-711 driver[8]. After initialization, the code enters a main loop that continuously samples data from both the MPU6050 and the load cells. This raw data is then transmitted via Bluetooth to the bicycle computer in JSON form.

Before the main loop could be run, a critical calibration step was required for the load cells. This process establishes an accurate conversion factor from the sensor's raw electrical output to a meaningful unit of weight or force. To do this, we first performed a taring (zeroing) procedure by reading the sensor's baseline value with no load. Subsequently, a known reference mass (1 kilogram of beans) was placed on the pedal. The resulting sensor value (minus the tare value) was then mapped to 1kg, allowing us to calculate a scaling factor that is applied to all future measurements. Taring is performed every time the system is turned on.

[7]Code can be found at: `https://github.com/jrowberg/i2cdevlib`

[8]Code can be found at: `https://github.com/bogde/HX711`

6.1.4 Connection

The Bluetooth communication process relies on two distinct links: a wired serial (UART) connection between the microcontroller and the BLE module, and a wireless link between that module and the Raspberry Pi. First, the microcontroller initializes its serial port at a 9600 baud rate. The Raspberry Pi then connects to the BLE module and, most importantly, subscribes to the characteristic "0000ffe1-0000-1000-8000-00805f9b34fb". Once this is set up, the microcontroller enters a loop where it collects sensor data and sends it over the wired serial connection to the BLE module. The module immediately uses this data to update the value of that specific characteristic, and, consequently, triggers a Notification packet, which is sent wirelessly, delivering the new data to the Raspberry Pi.

6.2 Blinker

The Blinker consists of a torque servomotor [8] and a set of LEDs mounted on a 3D-printed arrow. The servo is coupled to the seatpost, allowing the arrow to remain hidden when not in use. Activation is done by push buttons connected to the Raspberri Pi's GPIOs, using the native interrupt feature to ensure an immediate and asynchronous response.

6.2.1 Mechanics

A structure was designed to be mounted on the back of the seatpost, as shown in Figure 4. This 3D-printed housing features a dedicated recess for the servo motor, which holds an arrow equipped with addressable LEDs. The unit is connected by four insulated cables routed from the handlebar: VCC, GND, a PWM signal (for the servo), and the LED control line.

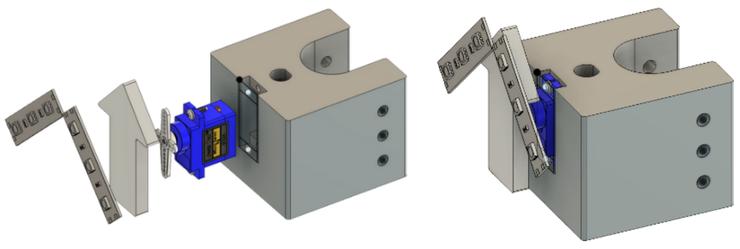


Figure 4: 3D model blinker

As shown by Figure 4 the arrow only points in one direction, this is due to the servo motor system. Whenever the blinker is turned on the servo moves so the arrow is pointed to the desired direction. This movement makes the blinker easier to see as humans are good at identifying moving objects.

6.2.2 Electronics

The Blinker electronics consist of two push buttons, one servomotor and ARGB LED strip connected to Raspberry Pi by cables screwed to terminals. Buttons are powered by 3.3 V and servomotor and LEDs by 5 V.

6.2.3 Software

The control of the Blinker resides in the BlinkerMoverThread and is initiated by the TouchButtonHandler, which utilizes high-priority hardware interrupts (GPIO) for event detection. Upon detecting an event, such as a button press, the Handler temporarily disables interrupts on the corresponding GPIO before sending a signal to the BlinkerMoverThread. This thread then executes the signaling sequence, which involves the servo motor movement (rotation to expose the arrow) and the intermittent activation of the LEDs for a duration of 5 s. Following the completion of this signaling time, the BlinkerMoverThread reverses the servo motion and re-enables the button interrupts, thus completing the signaling cycle.

7 Bicycle Smart Computer

This section describes the software architecture of the Bicycle Smart Computer (BSC)[9], which functions as the system's central unit. The BSC is responsible for managing all real-time data acquisition, processing, and user interaction. It features a graphical user interface (UI) built with the PySide6 [9] library, has bluetooth connections with the pedal and the mobile application, and utilizes dedicated serial connections to integrate a GPS module, a real time clock and the communication with the blinker system.

The following sections detail the structure of these individual threads and the communication mechanisms, such as queues and signals, that are used to ensure synchronization and data integrity.

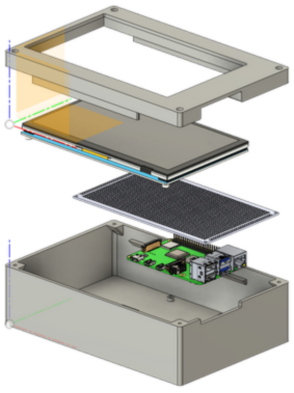
7.1 Electronics

The PCB of Raspberry Pi shield has two USB A power supply ports, one for the Raspberry Pi and one for the display, and two USB C ports, one merged with a step up to charge batteries, along with a BMS to protect them from overcharging and overdischarging and one directly connected to 5 V power trace to serve as backup DC source. The battery is composed of two packs in parallel, of two Li-Ion 18650 cells in series, totallizing 8.4 V that is fed into a voltage regulator to step down to 5 V. There are female 180° headers to plug and use the GPS and RTC modules and to attach the Raspberry Pi.

[9]Bicycle Smart Computer code: `https://github.com/ViDaProjects/BeForBike_BSC`

7.2 Mechanics

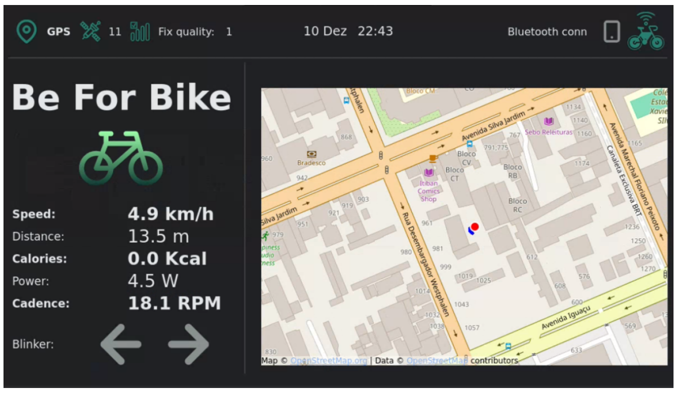
The components are protected by an enclosure (box) designed to cover and secure all internal elements. A custom-designed lid seals the enclosure while providing a clear viewing aperture for the display. The internal components, specifically the Raspberry Pi, the Raspberry Pi shield and the display, are organized in a stacked arrangement, as depicted in Fig. 5a. Furthermore, the enclosure incorporates dedicated pass-through holes to facilitate the use of zip ties or other fasteners, allowing the assembly to be securely mounted to the handlebar.



(a) 3D Handlebar box

7.3 Interface functions

The central unit features a display running an embedded user interface (UI) developed using the PySide6 library. This UI shows key metrics in real time, as depicted in Fig. 6. The information presented includes speed (km/h), distance (m), Calories (kcal), power (W), cadence (rpm), the blinker direction status, current time, Bluetooth connection status with both the cellphone and the Arduino, the quantity of satellites caught for GPS precision, and an integrated GPS map.



GPS 11 Fix quality: 1 10 Dez 22:43 Bluetooth conn

Be For Bike

Speed: 4.9 km/h
Distance: 13.5 m
Calories: 0.0 Kcal
Power: 4.5 W
Cadence: 18.1 RPM
Blinker:

Avenida Silva Jardim
Avenida Iguaçu
Rua Desembargador Westphalen
Avenida Marechal Floriano Peixoto
Canaleta Exclusiva BRT
Map © OpenStreetMap.org | Data © OpenStreetMap contributors

Figure 6: Handlebar display.

7.4 Multithreaded software architecture

To effectively manage the simultaneous demands of sensor I/O (from GPS and BLE), data processing (like FFT calculations), and UI updates, the software is built on a multithreaded Python [10] architecture. This concurrent design is essential to prevent blocking operations and ensure a responsive, real-time system.

7.4.1 Thread structure and communication

Communication between the threads is mediated by queues and signals/events to ensure synchronization and prevent race conditions.

1. **GpsGatherThread** and **GpsProcessorThread**: The first thread reads the raw data from the serial port; the second performs the parsing of the GNGGA/GNRMC messages, calculates the distance traveled (m), and the speed (m/s).

2. **CrankProcessorThread**: Receives the raw IMU data (from the Pedal) and applies the FFT algorithm to extract the rotation frequency, which is then converted into cadence (rpm) and used, together with the force (N), to calculate power (W) and calories (kcal).

3. **MsgCreatorThread**: Acts as an integrator, consolidating the processed GPS and crank data, packaging them into telemetry messages, and sending them to the main window for UI updates and to the FileManagerThread.

4. **CentralPeripheralBLEThread** and **PeripheralBLEThread**: Manage the BLE connection. The first is the client that reads data from the pedal (server); the second is the server that publishes telemetry data to the mobile application (client).

5. **FileManagerThread**: Responsible for data persistence. It receives messages from the MsgCreatorThread and manages the local SQLite [11] for logging the ride's data points, ensuring the integrity and atomicity of write operations.

7.5 Connection

It simultaneously manages a fixed Nano device (identified by his fixed MAC address) and a mobile device (identified by self.company_id and self.secret_key in the advertising data's Manufacturer Specific Data). Both connection attempts use BleakClient with a persistent retry logic.

For data reception from the Nano, the characteristic 0000ffe1-0000-1000-8000-00805f9b34fb is used to receive notifications. The _notification_handler collects incoming chunks, decodes them, and parses the final JSON data.

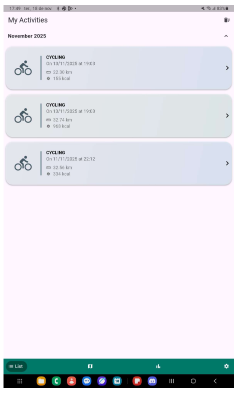
For data transmission to the mobile device, the characteristic 12345678-1234-5678-1234-56789abcdef0 is used with Write with response. The outgoing telemetry data is Jsonl encoded, Gzip compressed, and sent in batches of four. The system uses a priority queue to recover unsent packets following disconnection or critical GATTBlueZ errors (e.g., timeout, 0x12), and signals the FileManagerQueue to delete the ride file upon complete and successful transfer.

8 Mobile app

The BeForBike mobile application[10] serves as a secondary interface for the cyclist to see after ride statistics, providing a persistent, local history of data sent by the Bicycle Smart Computer.

8.1 Application interfaces

The mobile application has four main screens, each dedicated to a specific function:


17:49 ter., 18 de nov. 83%

My Activities

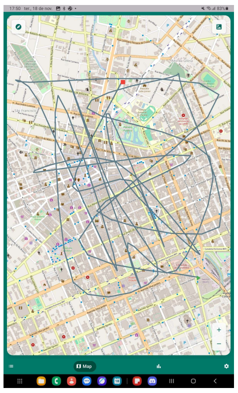
November 2025

CYCLING
On 13/11/2025 at 19:03
22.30 km
155 kcal

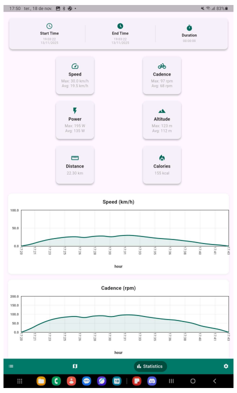
CYCLING
On 13/11/2025 at 19:03
32.74 km
968 kcal

CYCLING
On 11/11/2025 at 22:12
32.56 km
334 kcal

(a) First screen: list of rides from database



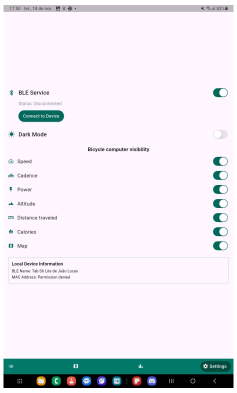
(b) Second screen: track of the path


17:50 ter., 18 de nov. 83%

Start Time 19:03:22 13/11/2025
End Time 19:03:22 13/11/2025
Duration 00:00:00

Speed Max: 30.0 km/h Avg: 19.5 km/h
Cadence Max: 97 rpm Avg: 68 rpm
Power Max: 195 W Avg: 135 W
Altitude Max: 123 m Avg: 112 m
Distance 22.30 km
Calories 155 kcal

(c) Third screen: historical data of the ride


17:50 ter., 18 de nov. 83%

BLE Service [✓]
Status: Disconnected
Connect to Device

Dark Mode [ ]

Bicycle computer visibility

Speed [✓]
Cadence [✓]
Power [✓]
Altitude [✓]
Distance traveled [✓]
Calories [✓]
Map [✓]

Local Device Information
BLE Name: Tab S6 Lite de João Lucas
MAC Address: Permission denied

(d) Fourth screen: settings

Figure 7: Screens of the BeForBike app

1. **Ride list screen (history):** Displays all ride entries saved in SQLite (data from the Ride table). It allows the user to select a session for detailed inspection.

2. **Map screen (navigation):** Upon selecting a ride, it loads the latitude and longitude points from the Ride_data table and plots them on a map, showing the path traveled.

[10]Cycling app code: `https://github.com/ViDaProjects/BeForBike_App`

3. **Charts screen (analytics):** Presents time-series charts for the main metrics (power, cadence, speed and altitude) using data from Ride_data. It also displays aggregated metrics (averages and maximums), total distance, calories and duration and start/end time from the Ride table.

4. **Settings screen (options):** Allows the user to start/stop BLE receiver, showing Bluetooth status, switch UI light/dark mode of colors and switch on/off data features of the Bicycle Smart Computer screen.

8.2 Architecture and technology

The app is built using a hybrid architecture: Dart/Flutter [12] was chosen for the customizable UI, ensuring a consistent user experience. Kotlin is used to handle system-level tasks on the Android platform, specifically for managing device permissions, SQLite database queries and the Bluetooth Low Energy (BLE) communication layer.

8.3 Data persistence (SQLite)

The local SQLite database stores all ride information, organized into a simple two-table relational model. This model consists of the Ride table, which stores general data for a ride, and the Ride_data table, which stores the historical, time-series information for that same ride, shown on the tables 1 and 2.

Table 1: Schema of the *Ride* and *Ride_data* Tables in SQLite.

Table	Attribute	Type	Description
Ride	*ride_id*	INTEGER	Primary Key (PK)
	date	DATETIME	Ride date
	time_init	TIMESTAMP	Start time
	average velocity	REAL	Average velocity (km/h)
	average cadence	REAL	Average cadence (rpm)
	duration	TIMESTAMP	Total duration (dd:hh:mm:ss)
	average power	REAL	Average power (W)
	calories	REAL	Total calories (kcal)
	distance	REAL	Total distance (km)
	max velocity	REAL	Maximum velocity (km/h)

Table 2: *Ride_data* Tables in SQLite.

Table	Attribute	Type	Description
Ride_data	*ride_id*	INTEGER	Foreign Key (FK)
	date	DATETIME	Collection date
	time	TIMESTAMP	Collection time
	latitude	REAL	GPS Position
	longitude	REAL	GPS Position
	altitude	REAL	Altitude (m)
	speed	REAL	Instantaneous speed (km/h)
	direction	REAL	Direction of movement
	fix_satellites	INTEGER	Number of satellites
	joules	REAL	Instantaneous energy (J)
	calories	REAL	Accumulated calories (kcal)
	speed_ms	REAL	Speed (m/s)
	distance	REAL	Accumulated distance at the point (m)
	power	REAL	Instantaneous power (W)
	cadence	REAL	Instantaneous cadence (rpm)

8.4 Connection

The BleServerService operates as a BLE GATT Server, including Gzip data compression, Jsonl file receiving and parsing, and database queries.

The server advertises using Manufacturer Data containing the companyId and secretKey for client discovery. The GATT profile uses the data characteristic UUID 12345678-1234-5678-1234-56789abcdef0. Advertising is stopped upon connection and restarted with a delay upon disconnection to avoid concurrency issues.

Received data is first Gzip decompressed and the payload, which is expected to be a JSONArray of telemetry data, is then inserted into the database in the background. The GATT response is only sent back to the client after the background processing completes. Service stability is maintained by a bluetoothStateReceiver which monitors the Android Bluetooth adapter status and shuts down the service if the adapter is turned off.

9 Results

9.1 Quantifying Power

For quantifying the results of the data measurements of the cycling computer system it is necessary to know the time and distance the bike has travelled, as well as the elevation increase obtained during the measurement time.

The power P (W) is calculated by

$$P = \frac{m \cdot g \cdot h}{t} \tag{5}$$

where m (kg) is the mass, g (9.81 m/s^2) is the gravity constant and h (m) is the height. From this formula we made 5 climbs up 7 de Setembro in front of UTFPR, going up 8.4 meters and ranging different powers from 75 to 140 Watts, depending on how fast we were going.

From the 5 measurements taken, 3 of them showed our fourrier transform is not as ideal as previously thought, obtaining distances travelled of over 140 meters for our 104 meters of road (as estimated by google maps). For 2 measurements, however, the travelled distance was 104 and 108 meters, good enough to provide a power estimation that can be evaluated.

The 2 measurements show a factor of around 3 needs to be multiplied to the result to obtain the correct result, we believe this is due to the force not being applied directly downwards on the pedals, meaning our results showed 1/3 the value we expected and displaying limitations of the Risk Response plan activated, switching from strain gauge to load cell. When this factor was accounted for both results were within 6.4% of their expected measurement, showing that when the cadence calculation works we can obtain quality results.

9.2 Budget

While far cheaper than commercial products, the budget for the project is rather extensive. Firstly, an estimation was made falling within the R$ 1200 limit. More specifically, the initial spenditure, meant to cover all costas was calculated (and done) at a total cost of R$ 1186. While ideally this budget would be all that was spent, this was not the case. Upon finding difficulties with the installation of the strain gauge system an additional R$ 200 was spent in extra gauges due to installation issues and the piece not being reusable. Additionally, when the strain gauge was abandoned, an additional 90 reais was spent in buying load cells, as well as a HX-711 module. With this in mind, the total project spenditure added up to R$ 1476, above the spending limit, however, only by R$ 276, which happened due to installation difficulties related to a high danger risk-response plan as well as it's activation.

9.3 Hours worked

Due to the complex nature of the software's project, as well as the difficulties with the strain gauge and the painstakingly long PCB manufacturing process, our total hours worked exceed what most teams did for Integration Workshop 3. The first 6 weeks, which should be the project's development time, amounted a total of **652** hours worked, which was consistent with the estimated time of **634.4** hours for that development period.

However, these six weeks did not total all hours worked, due to the project not being completed we had to spend an additional **375.5** hours, meaning our total time spent was **1027.5** hours. These additional hours after the 6 weeks period included tasks such as debugging the wireless connections to the Raspberry Pi computer, calibrating the power measurements for more precise readings, solving electromagnetic interference with the GPS module as well as the presentation related tasks such as: writing the document, producing the video, designing the slides and more.

10 Conclusion

The development of the BeForBike system successfully demonstrated the feasibility of creating an integrated, low-cost cycling performance and safety monitoring platform. By combining accessible hardware components, such as the Raspberry Pi and Arduino Nano, with a sophisticated multithreaded software architecture, the project achieved its primary objective of democratizing access to high-end cycling metrics.

Future iterations of this work could focus on the industrialization of the hardware through the design of industrial-grade surface-mount PCBs, replacing the current prototyping shields to reduce weight and volume. Additionally, further refinement of the load cell calibration process aggregating more sensors like environmental, magnetic spin and better IMU module and power management optimization could extend the device's autonomy and precision, bringing the prototype closer to a marketable consumer product. Machine learning and Kalman Filter algorithms could be used to better fuse sensor values and estimate the power applied to pedals. In summary, BeForBike stands as a functional proof of concept that advanced sports engineering can be achieved with limited resources through optimized software integration and efficient electronic design.

Acknowledgments

We are grateful for the support of the professors, their feedback throughout the development of the project, as well as for financial and emotional support from our families. Finally, we would like to thank ”A Bicicletaria” for selling us a bicycle.

Links

This section presents the supplementary resources, including the project blog and video documentation, referenced throughout this report.

Blog:https://verbose-pigeon-06a.notion.site/Be-for-Bike-25603987007e8078ab14c2dd9d0d7a4b

Video: https://youtu.be/Na0ywmCL6qE

References

[1] Raspberry Pi Foundation. Official raspberry pi os and gpio documentation. `https://www.raspberrypi.org/`. Accessed: 2025.

[2] National Marine Electronics Association. Nmea 0183 standard for interfacing marine electronic devices, n.d.

[3] Bluetooth Special Interest Group (SIG). Bluetooth low energy gatt profile specification, n.d.

[4] H. K. P. Neubert. *Instrument Transducers: An Introduction to their Performance and Design.* Oxford University Press, 1975.

[5] J. C. D. Smith. *The Aerodynamics of Human-Powered Land Vehicles.* Springer, 1984.

[6] D. E. Newland. *An Introduction to Random Vibrations, Spectral and Wavelet Analysis.* Longman Scientific & Technical, 1993.

[7] Arduino. Official arduino and esp32 platform documentation. `https://www.arduino.cc/`. Accessed: 2025.

[8] J. E. Hughes. *Electric Motors and Drives: Fundamentals, Types and Applications.* Elsevier, 2008.

[9] Qt Project. Qt for python (pyside6) documentation. `https://doc.qt.io/qtforpython/`. Accessed: 2025.

[10] Guido Van Rossum. *The Python Language Reference.* Python Software Foundation, 2025. Available at `https://docs.python.org/3/reference/`.

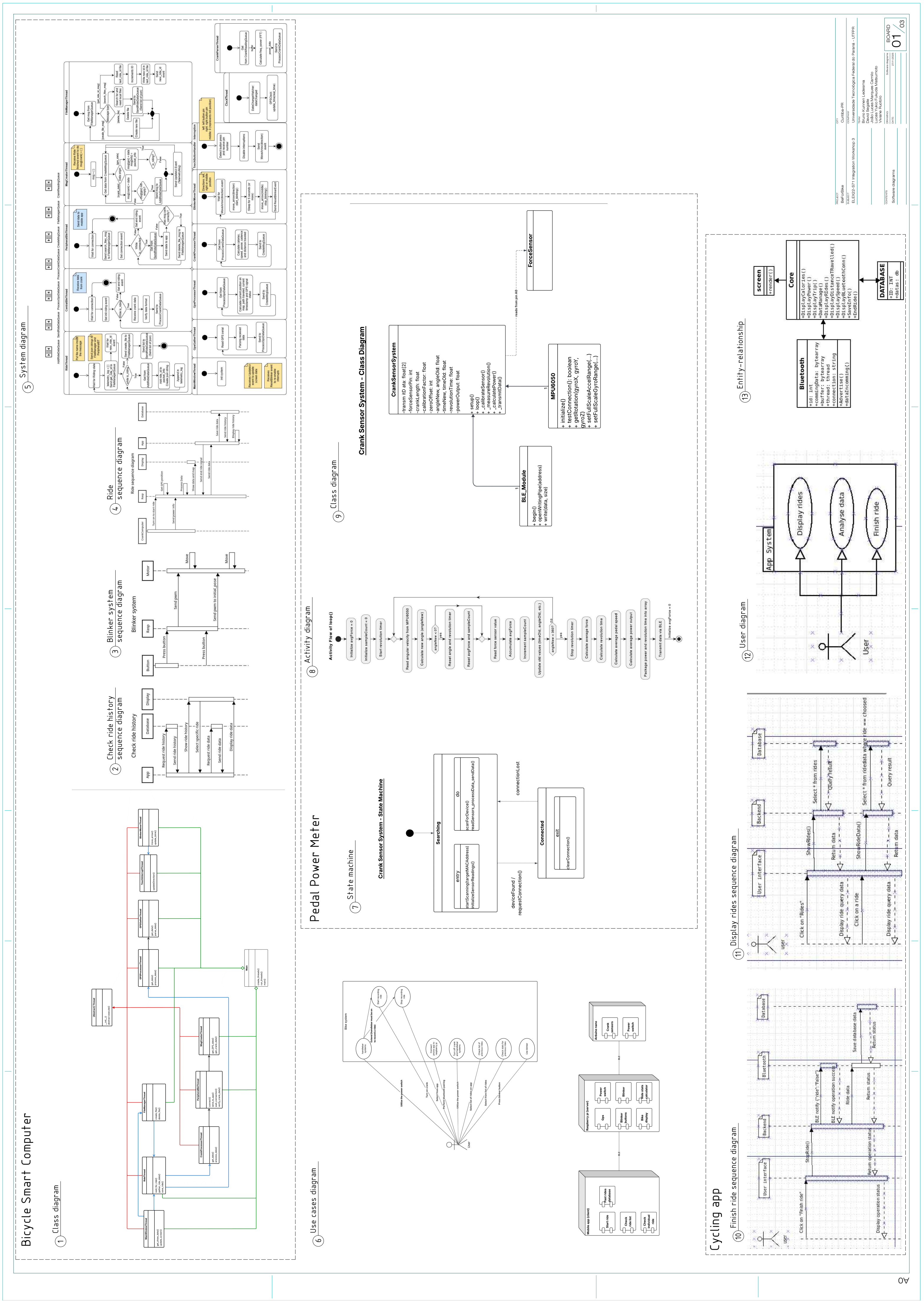
[11] SQLite. Official sqlite documentation. `https://www.sqlite.org/`. Accessed: 2025.

[12] Google. Flutter documentation. `https://flutter.dev/`. Accessed: 2025.

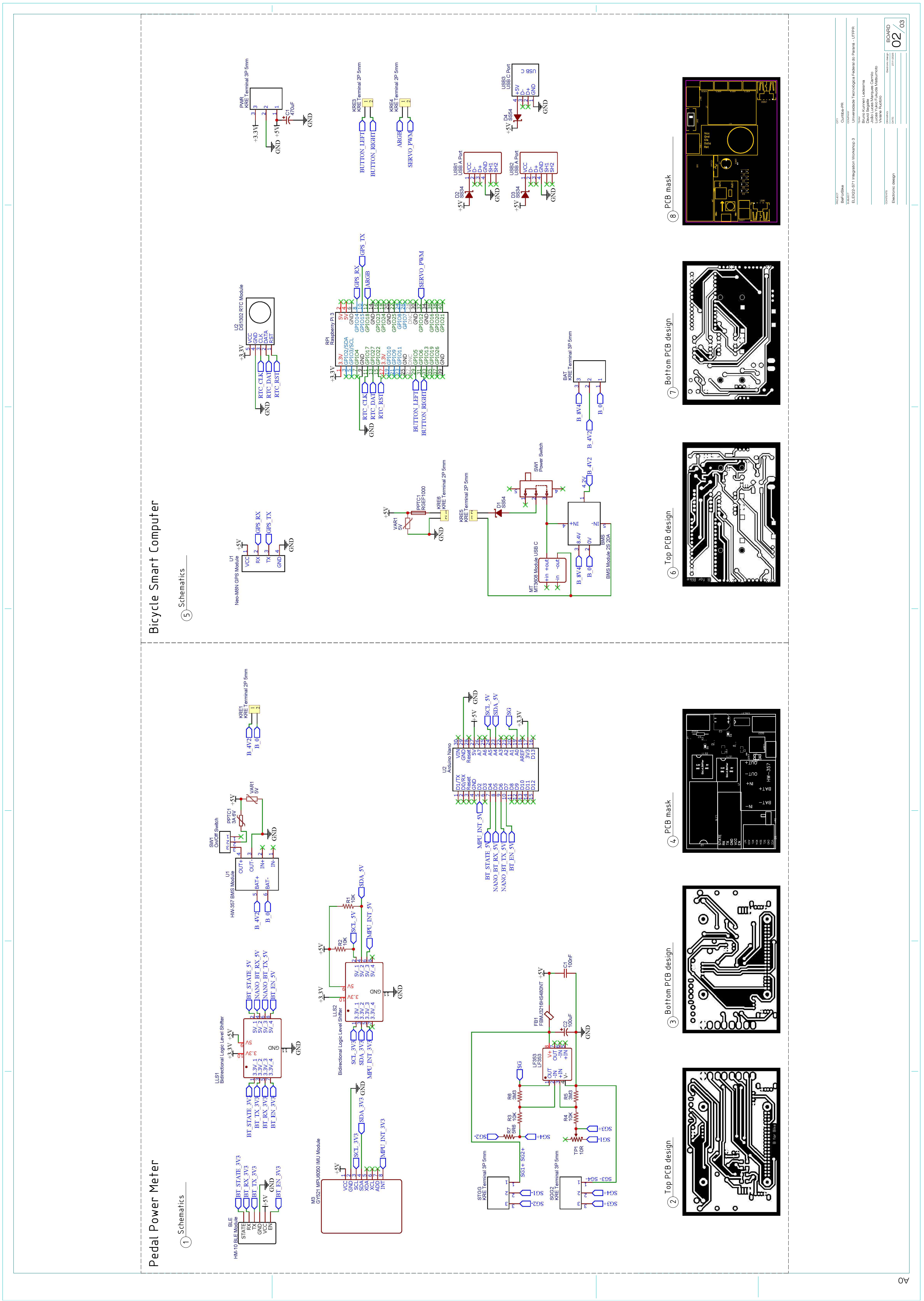
Appendix

A CrankProcessorThread source code details

```
def run(self):
    while self.running:
        # 1. Obtain IMU data (Z accelerometer)
        imu_data = self.imu_queue.get()

        # 2. Filtering and windowing (Hanning)
        filtered_data = apply_filter(imu_data)
        windowed_data = apply_hanning_window(filtered_data)

        # 3. Calculate FFT
        fft_result = np.fft.fft(windowed_data)
        fft_magnitude = np.abs(fft_result)

        # 4. Find dominant frequency (peak)
        sampling_rate = 10 # Hz
        fundamental_freq = find_dominant_frequency(fft_magnitude, sampling_rate)

        # 5. Convert frequency to cadence (RPM)
        cadence_rps = fundamental_freq
        cadence_rpm = cadence_rps * 60

        # 6. Obtain force and calculate power
        force_n = self.loadcell_queue.get()
        torque_nm = force_n * CRANK_LENGTH # Lever arm
        power_w = torque_nm * (cadence_rps * 2 * np.pi)

        # 7. Send processed data
        self.output_queue.put({'cadence': cadence_rpm, 'power': power_w})
```


Bicycle Smart Computer
1 Class diagram
2 Check ride history sequence diagram
3 Blinker system sequence diagram
4 Ride sequence diagram
5 System diagram
Pedal Power Meter
6 Use cases diagram
7 State machine
Crank Sensor System - State Machine
8 Activity diagram
9 Class diagram
Crank Sensor System - Class Diagram
Cycling app
10 Finish ride sequence diagram
11 Display rides sequence diagram
12 User diagram
13 Entity-relationship
PROJECT BeForBike
SUBJECT ELEX22-S71 Integration Workshop 3
CITY Curitiba-PR
COMPANY Universidade Tecnológica Federal do Paraná - UTFPR
TEAM Bruno Kunnen Ledesma
David Segalle
João Lucas Marques Camilo
Lucas Yukio Fukuda Matsumoto
Viviane Ruotolo
CONTENTS Software diagrams
BOARD 01/03
A0

C Electronics design



Pedal Power Meter
1 Schematics
2 Top PCB design
3 Bottom PCB design
4 PCB mask
Bicycle Smart Computer
5 Schematics
6 Top PCB design
7 Bottom PCB design
8 PCB mask
HM-10 BLE Module
GY521 MPU6050 IMU Module
Bidirectional Logic Level Shifter
HW-357 BMS Module
Arduino Nano
LF353
Neo-M8N GPS Module
DS1302 RTC Module
Raspberry Pi 3
MT3608 Module USB C
BMS Module 2S 20A
PROJECT
BeForBike
SUBJECT
ELEX22-S71 Integration Workshop 3
CITY
Curitiba-PR
COMPANY
Universidade Tecnológica Federal do Paraná - UTFPR
TEAM
Bruno Kunnen Ledesma
David Segalle
João Lucas Marques Camilo
Lucas Yukio Fukuda Matsumoto
Viviane Ruotolo
CONTENTS
Electronic design
DRAWING: Electronic design
DATE: 27/11/2025
BOARD 02/03
A0