Real-Time Remote Bike Ride Monitoring Using a Sustainable IoT System and Integrated Android App

Giuseppe Pasquini, *239978*

***Abstract*—The Internet of Things (IoT) has revolutionized real-time data monitoring across various fields, including transportation and sports. In cycling, monitoring data during rides provides critical insights for performance analysis and route tracking. This project builds upon a sustainable IoT system for real-time bike ride monitoring, with a focus on optimizing energy consumption and simplifying the system design. Specifically, the solar panel has been removed, and energy is harvested exclusively using thermoelectric generators (TEGs). Additionally, the GPS module has been eliminated to reduce power consumption, and a custom Android app has been developed from scratch to collect the IoT system's data via Bluetooth Low Energy (BLE). The smartphone app integrates its GPS data with the sensor data from the IoT system and pushes this information to an MQTT server for remote storage in InfluxDB. The data is then visualized through Grafana, maintaining the ability to provide real-time performance monitoring while significantly improving energy efficiency.**

I. Introduction

Recent innovations in network and communication technologies have catalyzed the rapid expansion of the Internet of Things (IoT), within the telecommunications sector. Currently, over 40 billion devices are interconnected globally, enabling a new era of connectivity that encompasses a wide range of physical objects equipped with sensors, actuators, and other smart technologies. These devices collect, share, and analyze data, allowing for improved interaction and control through dedicated IoT platforms [1].

At the heart of IoT ecosystems are these interconnected devices, which communicate seamlessly over various networks, often leveraging wireless technologies. Centralized IoT platforms aggregate data from multiple sources, facilitating real-time monitoring, automation, and decision-making processes across diverse applications. This integration empowers industries to optimize operations, enhance user experiences, and develop innovative solutions for pressing challenges [2].

The applicability of IoT spans numerous sectors, including smart cities, healthcare, industrial automation, and environmental monitoring. In urban environments, IoT technologies can optimize traffic management, enhance energy efficiency, and improve public safety through real-time data collection and analysis. In healthcare, remote monitoring devices can track patient health metrics, reducing the need for in-person visits and enabling timely interventions [3].

Building on the transformative potential of the IoT, its applications in sports, particularly cycling, have gained significant attention. The integration of IoT devices in cycling enables the collection and analysis of critical performance metrics, such as speed, distance, and cadence, as well as environmental factors like temperature and humidity. Downhill biking presents unique challenges and opportunities for data tracking, as riders often face varying terrain and rapid changes in speed. By equipping bikes with IoT sensors, cyclists can monitor their performance in real-time, optimizing their technique and enhancing safety. However, the deployment of these IoT solutions in cycling faces significant challenges, particularly concerning energy management. The energy consumed by IoT systems is influenced by various factors, including frequency bandwidth, duration of data transmission, and data sensing, processing, and storage needs [4]. Many devices rely on batteries with limited capacity, making them susceptible to power shortages during long rides. This challenge underscores the urgent need for energy-efficient designs and innovative energy harvesting solutions, such as thermoelectric generators, to sustain continuous monitoring and data transmission without the burden of frequent recharging.

One potential solution involves utilizing cloud-based energy management systems, which leverage the processing power and storage capabilities of the cloud to optimize energy consumption. This approach reduces energy demands on individual devices and enhances the overall efficiency of the network [5]. Additionally, network-level optimization techniques can further improve energy efficiency by minimizing data transmission requirements and developing energy-efficient communication protocols. Energy harvesting from natural sources offers a promising self-sustaining solution that extends battery life, reduces maintenance costs, and supports long-term deployment. Techniques such as thermal energy and kinetic energy harvesting are suitable for IoT devices, with selection depending on the application, energy availability, and required power output [6].

In this project, we implemented a self-powered IoT system aimed at collecting and visualizing real-time bike ride data. The primary innovation lies in eliminating the GPS module and solar panel, present in previous designs, to reduce power consumption and enhance simplicity. Instead of a dedicated GPS module, the project integrates GPS data directly from a custom-built Android app, which collects data from the smartphone, merges it with the bike's sensor data via Bluetooth Low Energy (BLE), and transmits the combined dataset to an MQTT server for real-time monitoring. This streamlines the system and shifts computational tasks to the smartphone, thereby reducing the energy burden on the bike-mounted IoT

device.

The bike-mounted system utilizes thermoelectric generators (TEGs) as the sole energy harvesting component, capturing the heat generated by the bike's friction during braking to power the sensors and microcontroller. Current and voltage measurements are recorded to compare the power consumption of the system with the energy harvested, ensuring the system's energy autonomy.

This project's contributions include:

- Developing a custom Android app from scratch that collects bike ride data via BLE, integrates the phone's GPS data, and pushes the information to an MQTT server.
- Streamlining the energy harvesting system by retaining the TEG component and removing the solar panel, leveraging smartphone GPS to reduce overall power consumption.
- Integrating various sensors (IMU, environmental, energy production, and consumption metrics) with a microcontroller for data collection and BLE transmission.
- Establishing a comprehensive data pipeline from data sensing to visualization, utilizing InfluxDB for storage and Grafana for real-time monitoring of the bike ride data.

In the following sections, a detailed overview of the developed system is provided. The review of related works is presented in II, followed by a discussion on sensor selection and data collection by the microcontroller in III. Details of the Android app's development are covered in IV, along with the transmission, storage, and visualization of collected data in VII. The configuration of the Raspberry Pi using Telegraf, InfluxDB, and Grafana for efficient MQTT data management is outlined in V. Additionally, the straightforward bike setup and the *RideSyncIoT* app testing process are discussed in VI, focusing on the stability of BLE and MQTT connections, as well as evaluations of core features. Finally, VIII concludes the paper with remarks and suggestions for future work.

II. Related Work

The rapid expansion of IoT technologies across various sectors has significantly improved the quality of many services and applications, particularly in sports and transportation. In the context of cycling, IoT systems enhance athletic performance by enabling continuous real-time monitoring of ride metrics such as speed, distance, and environmental factors. Furthermore, IoT applications in smart transportation systems can streamline bike-sharing services, optimize routes, and improve overall urban mobility [2]. For instance, Yang et al. developed a demand-aware mobile bike-sharing service that utilizes collaborative computing and information fusion in a 5G IoT environment [7]. Their work analyzed historical bike-sharing data, predicting rental and return demands through a dynamic clustering algorithm. This 5G-enabled approach improved the availability of bike-sharing services by enhancing communication between bikes and stations. Similarly, IoT systems in cycling can interconnect bikes and share collected data with a network for real-time analysis and feedback, thereby improving both user experience and system efficiency.

A critical challenge in deploying IoT systems is energy management, especially for battery-operated devices reliant on limited energy reserves. To address this issue, research has increasingly focused on energy harvesting techniques. Among these, thermoelectric generators (TEGs) have garnered attention for their ability to convert waste heat into usable electrical power. Studies have demonstrated the potential of TEGs in motor vehicles for recovering energy lost through heat dissipation. For example, Yadav et al. investigated energy recovery in heavy-duty vehicles (HDVs), where approximately 40% of fuel energy is lost as heat, particularly through exhaust gases and braking systems [8]. TEGs positioned along the exhaust pipe showed promise in recapturing this energy for onboard systems. Additionally, Coulibaly et al. explored the use of TEGs in electric vehicles, focusing on brake discs as heat sources for energy recovery [9]. Their research demonstrated that TEGs could generate sufficient power to drive various vehicle systems, achieving up to 4.25 W at a brake disc temperature of 200°C. This energy, while modest, was enough to power essential instrumentation. Such findings suggest that TEGs could be effectively applied in bicycle systems, especially during downhill rides or in urban settings where frequent braking occurs.

While research on TEG-based energy harvesting for bicycles is still limited, Ahmed et al. made significant progress by developing an energy-efficient cooling and energy recovery system for electric bikes [10]. Their work highlighted the potential for TEGs to harness heat generated by batteries and braking mechanisms, extending battery life and improving overall system efficiency. These approaches underscore the importance of optimizing the temperature gradient across TEG surfaces to maximize energy harvesting, particularly in environments with high temperature fluctuations.

In this project, we focus on eliminating the need for a dedicated GPS module in an IoT system designed for bicycles, utilizing an Android application to integrate GPS data with ride metrics. This approach significantly minimizes energy consumption by leveraging the smartphone's built-in GPS capabilities, thereby enhancing overall system efficiency. The IoT system collects and transmits real-time data, including speed, IMU readings, and environmental factors, through BLE.

Energy is harvested solely from the braking system using thermoelectric generators (TEGs), enabling the system to achieve energy autonomy and eliminate the need for frequent battery recharges. These advancements in energy harvesting are crucial for developing sustainable, self-powered IoT devices, particularly in resource-constrained environments such as sports and remote cycling applications [5].

III. Sensors and Data Acquisition

The design of an efficient IoT system for real-time bike ride data monitoring hinges on selecting suitable sensors and microcontrollers to ensure accurate data collection with minimal power consumption. Due to the system's bike-mounted nature, it is crucial to prioritize compact and lightweight components. In contrast to some of the more elaborate sensor, our system focuses on a streamlined approach that excludes GPS tracking

and real-time power consumption monitoring, instead relying on simulated data and smartphone integration.

For this project, we selected the Arduino Nano BLE 33 Sense Rev 2 as the primary microcontroller due to its small form factor and low power consumption, making it well-suited for IoT applications in cycling. The onboard BLE capability allows seamless data transmission to a paired smartphone, which will be used to handle GPS tracking rather than an additional GPS sensor module. This decision eliminates the need for extra components, simplifying both the design and energy management.

Key data collected during the bike ride focus on environmental and motion metrics:

- **IMU:** The 9-axis IMU on the Arduino Nano BLE 33 Sense Rev 2 includes the BMI270 accelerometer/gyroscope and the BMM150 magnetometer. This sensor provides real-time data on the bike's motion and orientation, capturing important parameters as acceleration, angular velocity, and magnetic heading. These measurements are essential for tracking ride dynamics and monitoring rider performance.
- **Temperature and Humidity Sensor:** The onboard HS3003 sensor measures environmental temperature and humidity, providing insight into the conditions in which the ride occurs. These metrics can influence energy harvesting efficiency, making them crucial for accurate data analysis.
- **Barometric Pressure Sensor:** The LPS22HB embedded barometric sensor records atmospheric pressure, which can help in estimating altitude changes during the ride. This feature is particularly useful for rides in hilly or mountainous terrain, where elevation changes are significant.

A. Simulated Power Consumption and Harvesting Data

In contrast to more advanced setups that monitor power consumption and energy harvesting in real-time, our project simulates these data using Arduino's random value generation feature. While the INA226 modules can measure voltage and current from power sources in real-world applications, they are not implemented here due to the absence of physical energy harvesters, such as thermoelectric generators. Instead, we simulate power consumption and harvesting values to facilitate testing of the system's data visualization and processing capabilities.

The simulation of power data is handled by generating random values for the Arduino, allowing us to mimic the system's power management without relying on physical sensors. This approach ensures that the system logic for monitoring and adjusting energy usage can still be developed and tested, while the actual hardware implementation remains a future goal.

B. Data Collection and Transmission

For data acquisition, we use the libraries specific to the onboard sensors of the Arduino Nano BLE 33 Sense Rev 2. These include:

- **Arduino BMI270 BMM150.h** for accelerometer, gyroscope, and magnetometer data from the IMU.
- **Arduino HS300x.h** for environmental data from the temperature and humidity sensor.
- **ReefwingLPS22HB.h** for atmospheric pressure data from the barometric sensor.

The data collected by these sensors is transmitted to the smartphone over BLE, where it is processed and visualized using an Android-based application. Unlike setups that monitor power harvesting or consumption data in real-time, our system prioritizes motion and environmental metrics. This allows the rider to monitor essential ride statistics without the additional overhead of energy management data.

C. Data Acquisition Timing

To optimize power usage, sensor data is collected and transmitted at defined intervals. Table I outlines the timing for each sensor type.

Data Type	Acquisition Interval
IMU Data	Every 2 seconds
Environmental Data	Every 20 seconds
Power Data (Simulated)	Every 1 second
GPS Data	Every 1 second and when position changes

TABLE I: Data acquisition intervals

Environmental data, such as temperature, humidity, and pressure, change slowly, allowing for acquisition intervals of 20 seconds. This minimizes the system's power consumption without compromising the accuracy of the data. In contrast, IMU data are collected more frequently, every 2 seconds, as they provide real-time motion and orientation information, which is crucial for monitoring ride dynamics.

Once the data has been collected from the various sensors, it is transmitted via BLE using the `ArduinoBLE.h` library. This library facilitates the creation of both standard and custom BLE services and characteristics, allowing efficient data transmission with minimal power consumption. For this project, three services were defined:

- **Environmental data service (UUID 0x181A)**: This standard BLE service is used for transmitting environmental data, which includes three standard characteristics for pressure (UUID 2A6D), temperature (UUID 2A6E), and humidity (UUID 2A6F).
- **IMU data service (custom 128-bit UUID)**: A custom service defined with three unique characteristics, each transmitting data from different IMU components accelerometer, gyroscope, and magnetometer allowing real-time motion tracking.
- **Power data service (custom 128-bit UUID)**: This service is used to transmit power-related data collected from the INA226 sensors. Two custom string characteristics are used to send data on voltage, current, and power generated from different sources, such as a battery and TEG (Thermoelectric Generator).

Each of these services ensures reliable and continuous data transmission while optimizing energy efficiency during both

data collection and communication over BLE. By structuring the data into distinct services and characteristics, the system minimizes the risk of data loss and maintains seamless BLE communication.

IV. RideSyncIoT Android App

The *RideSyncIoT* app was developed to serve as an interface for collecting and visualizing data from an Arduino-based sensor system via Bluetooth, along with real-time GPS data from the smartphone itself. In the background, the app handles three essential functions: maintaining a Bluetooth connection to receive data from the Arduino, collecting GPS data from the smartphone, and publishing both sensor and GPS data via MQTT to a cloud server. While these processes run seamlessly in the background, the app's main interface presents the user with an interactive map and the latest GPS and sensor data in real time.

A. Development Tool: MIT App Inventor

For the development of the *RideSyncIoT* Android app, the **MIT App Inventor** was chosen as the primary tool. MIT App Inventor is a cloud-based platform that enables users to build Android applications through a visual, drag-and-drop interface. The platform's intuitive design allows developers to focus on the app's functionality by arranging components rather than writing extensive code, making it especially useful for projects requiring rapid prototyping.

Key features of MIT App Inventor include:

- A **visual block-based interface**, where code is represented by blocks that can be snapped together, minimizing syntax errors and simplifying development.
- **Component-based architecture**, offering pre-built elements like buttons, text boxes, sensors, and GPS modules, which can be added to the app with minimal effort.
- A **cloud-based environment**, allowing developers to work from anywhere using a web browser and to save their projects online without the need for local installations.
- **Real-time testing** with the MIT AI2 Companion app, which allows users to instantly see how their app performs on a connected Android device, speeding up the debugging process.

In comparison to **Android Studio**, which is the official integrated development environment (IDE) for Android, MIT App Inventor offers significant advantages in terms of simplicity and development speed. While Android Studio is designed for professional developers and supports more complex app development through coding in Java, Kotlin, and XML, it has a steeper learning curve and requires more advanced knowledge of programming and Android APIs. Android Studio provides more flexibility, high-performance optimization, and advanced debugging tools, making it suitable for larger, resource-intensive projects.

For the *RideSyncIoT* app, MIT App Inventor was chosen due to its simplicity and rapid development process. The app required basic functionalities such as GPS integration and Bluetooth communication. The block-based approach of MIT App Inventor is an ideal choice for the project focused on speed and ease of use, without the need for the complexity of Android Studio.

B. GPS, Map, and Navigation Features

The core of the application's interface is a dynamic map, powered by the OpenStreetMap service. OpenStreetMap is a collaborative platform that provides open-source geographic data to display detailed maps, making it a popular choice for mobile applications due to its flexibility and free access. The *RideSyncIoT* app leverages this service to visualize real-time location and navigation data.

Once the smartphone's GPS is enabled, the app begins acquiring the user's position in latitude and longitude coordinates. As the location changes, the map updates to reflect the user's movements by drawing the path traveled. In addition to displaying the route, the app also retrieves altitude and speed data using the `LocationSensor.LocationChanged` event, which triggers with each GPS update.

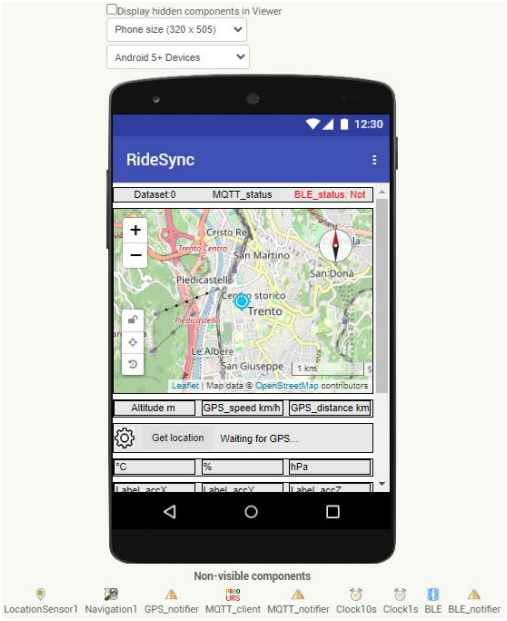

Fig. 1: App Inventor designer tool showcasing GPS functionality and map.

The total distance traveled is calculated by summing the distance between consecutive GPS points. The distance between two points on the Earth's surface is calculated using the spherical law of cosines formula:

$$d = R \times \arccos\left(\sin(\phi_1)\sin(\phi_2) + \cos(\phi_1)\cos(\phi_2)\cos(\lambda_2 - \lambda_1)\right)$$

Where:

- d is the distance between the two points.
- R is the radius of the Earth (approximately 6371 km).
- ϕ_1 and ϕ_2 are the latitudes of the two points in radians.
- λ_1 and λ_2 are the longitudes of the two points in radians.

Each latitude ϕ and longitude λ must be converted from degrees to radians before applying the formula. The resulting

distance d is in kilometers. This allows the app to update the total distance traveled by the user in real-time as they move along the mapped route.

C. Bluetooth Connection

The Bluetooth connection utilizes the BLE standard. This feature employs built-in functions to scan for devices, connecting to the one that matches the name "Nano33BLErev2" associated with the Arduino. While the Arduino is connected via Bluetooth, the *RideSyncIoT* app registers for each service UUID and characteristic UUID, as outlined below:

- **Services:**
  - Environmental Data Service, UUID: `181A`
  - IMU Data Service, Custom UUID: `12345678-1234-1234-1234-1234567890AB`
  - INA Data Service, Custom UUID: `12345678-1234-1234-1234-1234512345AB`
- **Characteristics:**
  - Pressure, UUID: `2A6D`
  - Temperature, UUID: `2A6E`
  - Humidity, UUID: `2A6F`
  - Accelerometer Data, UUID: `12345678-1234-1234-1234-1234567890CD`
  - Magnetometer Data, UUID: `12345678-1234-1234-1234-1234567890FF`
  - Gyroscope Data, UUID: `12345678-1234-1234-1234-1234567890EF`
  - INA Measurement, UUID: `12345678-1234-1234-1234-1234512345AC`
  - INA Measurement 2, UUID: `12345678-1234-1234-1234-1234512345AD`

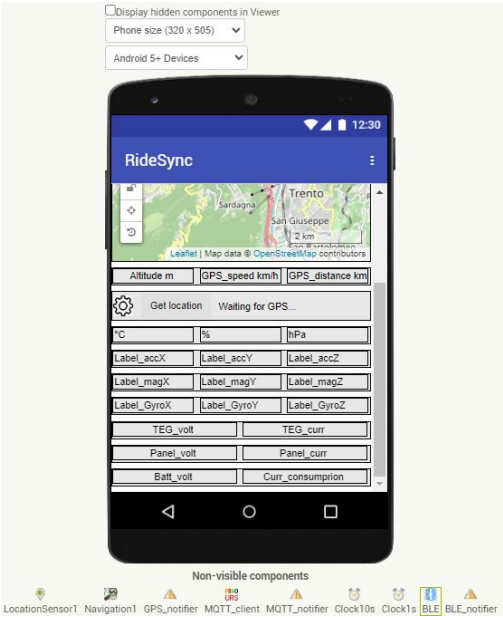

Fig. 2: App Inventor designer tool illustrating BLE data integration.

Whenever the `BLE.receive` event is triggered, the app updates the data labels and stores the values in a list for subsequent publication via MQTT. Additionally, the app handles connection errors by providing alert notifications and incorporates advanced connection parameters, which are detailed in the configuration section IV-E.

D. MQTT Connection

The MQTT connection is managed by the UrsAI2PahoMqtt extension. After configuring the connection parameters, the app connects to the UniTN MQTT server and subscribes to the appropriate topics for data handling. The configuration parameters are as follows:

- **Broker:** 193.205.194.147
- **Port:** 10883
- **Protocol:** TCP
- **Username:** RideSync
- **Password:** RideSync

Once connected, the app subscribes to the topic `SHIELD4US/CL-202-E-001/#` to listen for incoming data, allowing it to continuously monitor the MQTT connection. At the same time, it publishes data to the `RideSync/#` topic. There are two subtopics: one for GPS data, which updates whenever there is a location change, and another for Arduino data, which is published every second. Due to the varying update frequencies of IMU and environmental data, MQTT messages are sent only when new data is available, resulting in a non-fixed payload size for the messages.

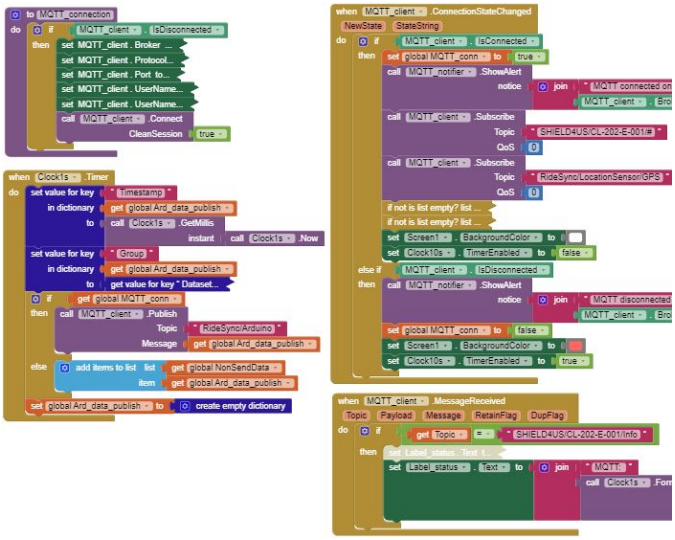

Fig. 3: App Inventor Blocks Tool, MQTT functionality.

Each MQTT message is sent in JSON format and includes two additional parameters. The first parameter is the timestamp, which is utilized in InfluxDB to accurately reconstruct the original time of the data instead of relying on the reception time. The second parameter is the Group, which indicates the dataset value. Both of these features are explained in more detail in Section IV-E.

Example of an MQTT message formatted in JSON, including various sensor readings and metadata:

```
{
  "TEG_volt": 2695,
  "TEG_curr": 0,
  "Battery_volt": 3737,
  "Curr_consumption": 10,
  "Timestamp": 1729764607352,
  "Group": "Dataset:A4F6797"
}
```

E. Background Features and Data Management

The app is designed to handle data management effectively, particularly in scenarios where the MQTT connection is lost. During such interruptions, both GPS and Arduino data are recorded and queued for transmission once the connection is restored. However, it is important to note that any new data generated during this downtime will not be captured and will be lost. This data recovery process ensures that previously collected data is sent without loss once the connection is reestablished.

Each dataset value is uniquely generated based on the timestamp of the first recorded data point; however, users have the option to modify this in the configuration screen (see Fig. 4). This feature is particularly beneficial for organizing and sorting different datasets in InfluxDB.

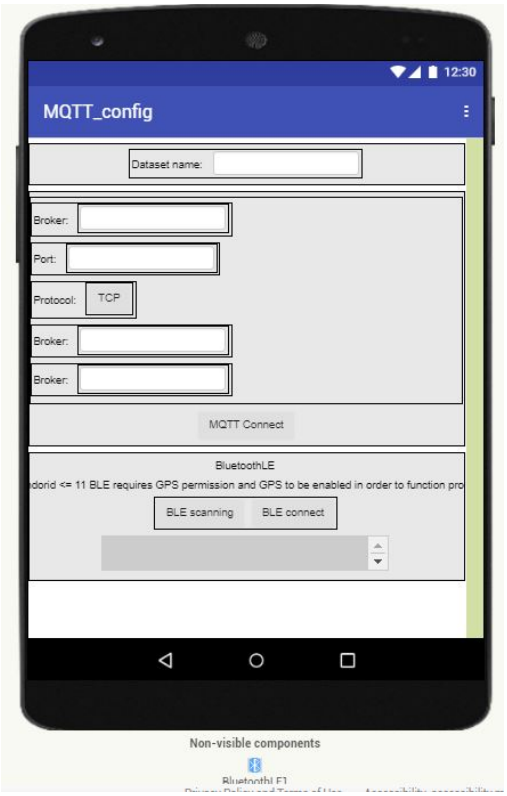

Fig. 4: Configuration screen of the App Inventor blocks tool.

In the configuration screen, accessible via the gear button next to the ”Get Location” button on the main screen, users can adjust the following settings:

- MQTT broker parameters, including broker address, port, username, and password.
- Bluetooth device scanning options, useful for reconnecting to devices if the initial connection attempt fails during the app’s initialization phase.

The main screen (Fig. 1) of the app displays the connection statuses for both Bluetooth and MQTT. Alerts are implemented to notify users when connection issues occur. Specifically, if the MQTT connection is lost, the app will attempt to reconnect every 10 seconds, utilizing a timer to manage this process efficiently.

F. Navigation Directions Using OpenRouteService

An additional feature implemented in the *RideSyncIoT* app is navigation and route guidance. By long-pressing on a point of interest (POI) within the map interface, users can set a destination for navigation. Upon setting a POI, the app calculates and displays the optimal route (Fig. 5 in blue) to reach the selected location from the user’s current position.

This navigation functionality is powered by the OpenRouteService (ORS) API, a comprehensive route planning service that leverages OpenStreetMap (OSM) data. The ORS API processes the starting and ending points based on the user’s current GPS location and the selected POI, and returns detailed routing information. This includes:

- **Turn-by-turn directions**: ORS provides a sequence of directional instructions to guide users along the path.
- **Estimated distance and travel time**: This feature helps users gauge how long it will take to reach their destination.
- **Alternative route suggestions**: In cases where multiple routes are viable, ORS can offer alternatives with different travel times and paths.

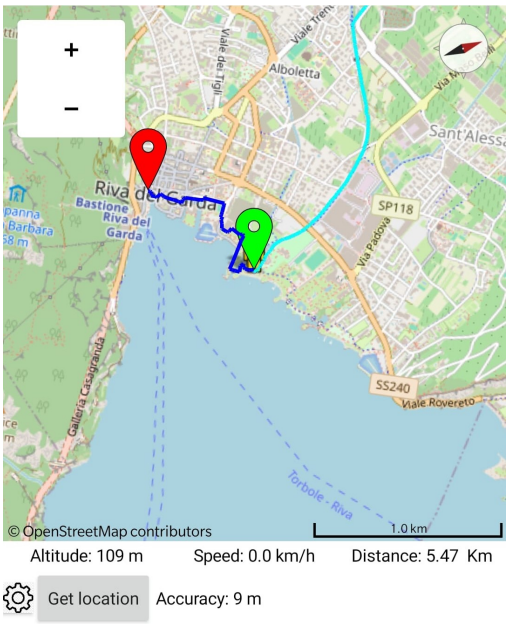

Fig. 5: App Inventor Navigation feature.

Due to the limited availability of API tokens, the app does not support real-time route updates as the user progresses along the path. However, the initial route remains displayed, providing a reliable guide toward the selected destination. This integration with ORS greatly enhances the navigation experience by providing reliable and accurate pathfinding functionality within the app, making it versatile for both urban and rural exploration.

V. Raspberry Pi Configuration

The storage and management of MQTT data are handled by an InfluxDB database [Fig. 7], which is installed and configured on a Raspberry Pi 4 Model B. InfluxDB is a time-series database, well-suited for handling IoT data due to its high write performance and efficient data compression. In this setup, we use InfluxDB to store the continuous stream of MQTT messages from the *RideSyncIoT* system, capturing data such as speed, location, temperature, and other ride metrics.

To facilitate the data ingestion from MQTT, we employ Telegraf, an open-source server agent designed to collect and send metrics and event data. Telegraf has a built-in MQTT consumer plugin that listens to the specified MQTT broker topics, parsing the incoming messages and structuring them for storage. This module enables us to configure Telegraf to subscribe to the MQTT topics used by the *RideSyncIoT* system, ensuring that each metric is captured and sent directly to the InfluxDB database with minimal latency.

The following is an example configuration for the Telegraf agent, designed to collect MQTT data and store it in InfluxDB.

```
[agent]
  interval = "0.1s"
  round_interval = true
  metric_batch_size = 1000
  metric_buffer_limit = 10000
  flush_interval = "10s"
  precision = "1ns"
  hostname = ""
  omit_hostname = false

[[outputs.influxdb_v2]]
  urls = ["http://localhost:8086"]
  token = "$INFLUX_TOKEN"
  organization = "RideSync"
  bucket = "GPS_MQTT_connection"

[[inputs.mqtt_consumer]]
  servers = ["mqtt://193.205.194.147:10883"]
  username = "Telegraf_client"
  password = "Telegraf_client"
  topics = ["RideSync/LocationSensor/GPS"]

  name_override = "gps_location"
  data_format = "json"
  json_time_key = "Timestamp"
  json_time_format = "unix_ms"
  json_timezone = "Local"
  tag_keys = ["Group"]
  json_string_fields = []
```

Once the data is stored in InfluxDB, it becomes available for real-time monitoring and analysis. For visualization, we use Grafana [Fig. 6], a robust data visualization platform that integrates seamlessly with InfluxDB. Grafana connects to the InfluxDB instance on the Raspberry Pi through Flux queries, a powerful query language optimized for time-series data. The Flux queries allow us to customize data retrieval, aggregating and filtering the metrics as needed to generate detailed graphs and dashboards. These visualizations provide real-time insights into metrics such as speed trends, distance over time, and environmental conditions, enhancing both usability and data-driven decision-making for users.

In summary, this setup consisting of Telegraf, InfluxDB, and Grafana on a Raspberry Pi provides a streamlined solution for data acquisition, storage, and visualization of the *RideSyncIoT* data. By combining these tools, we create a robust, scalable system capable of efficient data handling and insightful analysis, tailored for real-time performance monitoring in a compact IoT environment.

VI. Bike Setup and App Testing

To facilitate testing of the *RideSyncIoT* app, the bike setup is designed to be as straightforward as possible. A smartphone is mounted on the bike using a smartphone holder, and the Arduino Nano 33 BLE Sense Rev2 is positioned on the bike's handlebar, powered by a portable power bank for ease of use. The spatial orientation for the sensor is configured as follows: positive x-axis is oriented to the left, positive y-axis points toward the back of the bike, and positive z-axis points upward.

This configuration supports comprehensive testing of the app's functionalities, including standard operations, connection handling, and error scenarios. The primary focus of testing is to evaluate the stability of the IoT system and assess the reliability of both BLE and MQTT connections.

A. Testing Criteria

The app testing process covers a range of core features to verify functionality, usability, and performance:

- **Data Visualization Accuracy:** Verifying that BLE data from Arduino sensors is correctly visualized on the app, with real-time updates of environmental and IMU sensor data.
- **GPS Data Accuracy:** Ensuring the correct display and update of GPS parameters such as position, altitude, speed, distance, and location accuracy.
- **Configuration Functionality:** Confirming that all configuration settings can be adjusted in the app's settings menu, including MQTT broker parameters and Bluetooth device selection.
- **Navigation Directions:** Testing the navigation directions feature through the OpenRouteService API, verifying route path display and user navigation prompts.
- **Connection Stability and Recovery:** Assessing BLE and MQTT connection stability, and observing app behavior during disconnections and reconnections.

B. Connection Stability and Error Handling

The app's connection stability mechanisms were rigorously tested to ensure reliable data transmission and recovery:

- **BLE Disconnection and Timeout:** BLE services are registered during initial connection, with characteristics for each data type (e.g., environmental data, IMU data) enabling streamlined data transfer. In cases of BLE disconnection, the app automatically attempts reconnection, typically retrying at set intervals to re-establish communication with the Arduino device.
- **MQTT Disconnection and Data Caching:** The app subscribes to relevant topics upon connection to the MQTT

Fig. 6: Grafana dashboard.

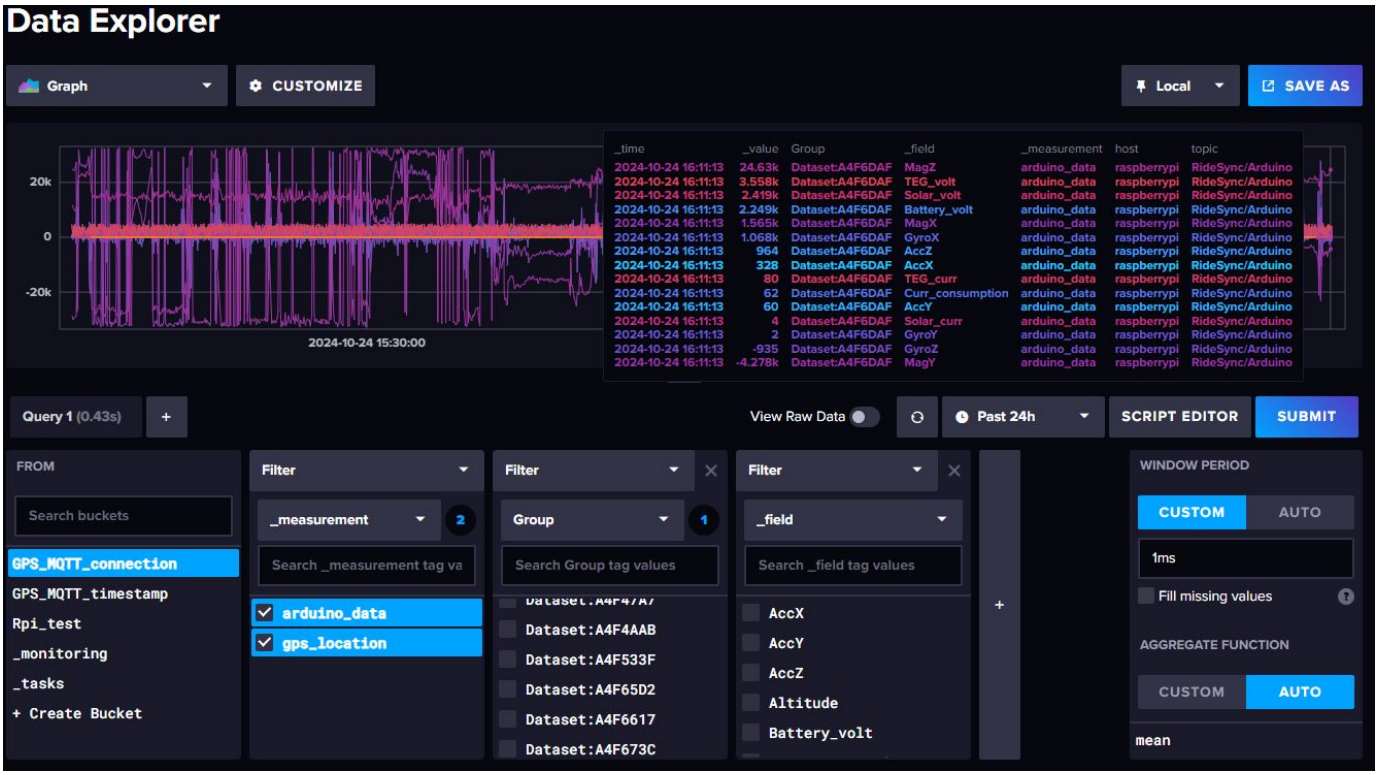

Fig. 7: InfluxDB data explorer.

broker, including `SHIELD4US/CL-202-E-001/#` for monitoring incoming messages and `RideSync/#` for publishing data. If an MQTT connection is lost, the app temporarily caches both GPS and Arduino data locally. This data is published once the connection is restored, ensuring continuity and avoiding data loss. A timer initiates reconnection attempts every 10 seconds until successful.

- **Service and Characteristic Registration:** During initial setup, the app registers all necessary BLE services and characteristics using custom UUIDs to facilitate the reading and notification of Arduino sensor data. This registration process enables reliable data reception during each `BLE.receive` event and prompts data updates in

the app's display and storage.

Through these features, the app maintains robust functionality under various scenarios, providing accurate data display, reliable navigation, and consistent MQTT and BLE connectivity. These tests have allowed for fine-tuning of the *RideSyncIoT* app to ensure it meets real-world demands for IoT-based bike tracking.

VII. Experimental Results and Discussion

The system was tested during multiple bike rides to evaluate the quality and accuracy of the data collected and transmitted by the IoT system, as well as to assess the stability and performance of the *RideSyncIoT* Android app. Figure 6 shows the Grafana dashboard, displaying data collected over a typical bike ride.

As illustrated in the dashboard, the system successfully collects, transmits, stores, and displays data, starting from the Arduino Nano and continuing through the smartphone app. Due to BLE message constraints, the Arduino transmits integer data values, as shown in Fig. 7. These values are adjusted in the Grafana Flux query to restore accuracy before visualization. This process ensures that data is correctly represented both on the smartphone's main screen and in the Grafana dashboard.

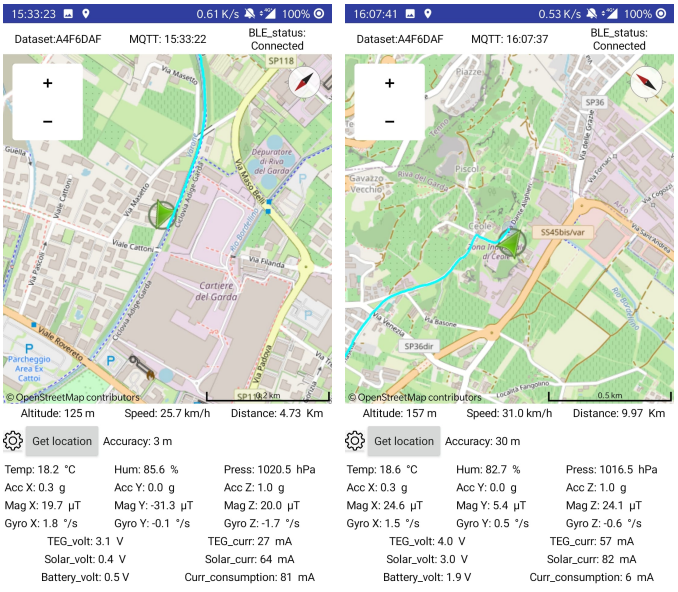

Fig. 8: *RideSyncIoT* bike ride data.

The testing of the *RideSyncIoT* app yielded positive results, demonstrating effective functionality and robust performance across various scenarios. As shown in Figures 8, the app successfully visualizes bike ride data, including real-time updates from the Arduino sensors. The first set of screenshots illustrates the accuracy of data visualization, with environmental parameters and IMU sensor data displayed clearly on the app interface.

Additionally, the continuous reception of MQTT messages is accurately reflected by the *MQTT Timestamp* displayed at the top center of the app's main screen, providing users with real-time confirmation of data transmission status. The total distance value, displayed on the app, incrementally updates as expected, accurately reflecting the cumulative distance covered during the ride.

Furthermore, the app's error handling capabilities were evaluated, as depicted in Figure 9. The initialization process is shown, confirming that the app successfully sets up the necessary connections before establishing the MQTT connection, while the BLE connection has not yet been established. During testing, the app effectively managed MQTT disconnections, as evidenced by the screenshot showing the handling of lost connections. Notably, the data caching mechanism was verified; the app temporarily stored GPS and Arduino data, ensuring no information was lost during disconnections. Once connectivity was reestablished, the app seamlessly pushed the cached data, demonstrating its reliability in maintaining data integrity.

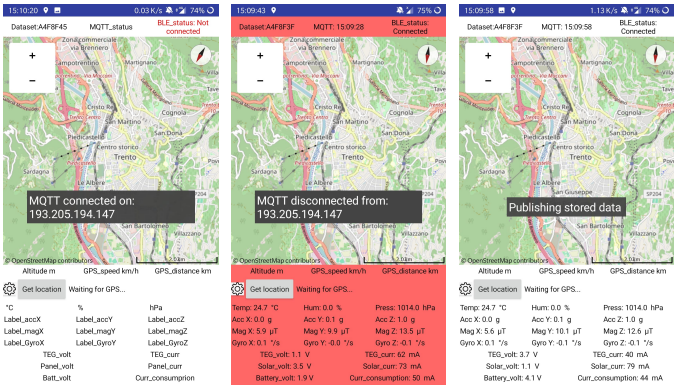

Fig. 9: *RideSyncIoT* error handling.

The *RideSyncIoT* configuration screen, shown in Figure 10, provides essential settings for customizing the app's connection and device parameters. Key configuration options include:

- **MQTT Broker Parameters:** Users can specify the MQTT broker's IP address, port number, and protocol settings, allowing the app to connect to a designated server for data transmission.
- **Bluetooth Device Selection:** The screen enables scanning and selection of available Bluetooth devices, ensuring connectivity with the Arduino Nano 33 BLE Sense Rev2 or other compatible sensors.
- **Dataset Configuration:** Users can create and manage dataset identifiers, enabling organized data tracking in InfluxDB by assigning a unique label to each data recording session.

These settings are accessible via the gear icon next to the location button on the app's main screen, enabling users to easily adjust connection parameters and device preferences for optimal performance.

Overall, the successful execution of these tests highlights the app's stability, accurate data representation, and effective error handling, confirming that the *RideSyncIoT* app is well-equipped for real-world bike tracking scenarios.

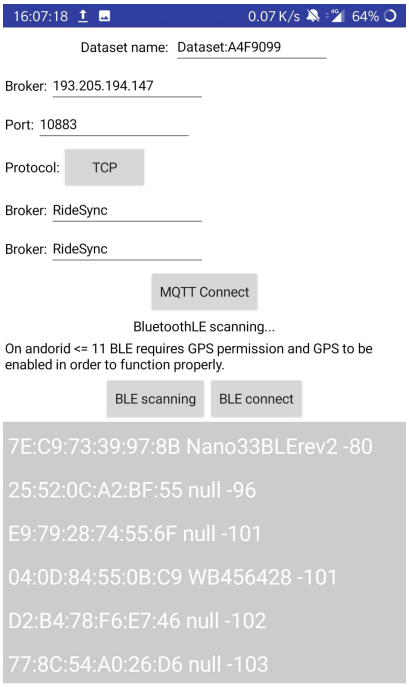

Fig. 10: *RideSyncIoT* configuration screen

VIII. Conclusion and Future Work

The *RideSyncIoT* system has demonstrated notable success in autonomously collecting, transferring, storing, and visualizing bike ride data. A significant enhancement in this version of the project was the integration of the smartphone's GPS, reducing energy consumption by approximately 80% compared to previous models, thus supporting long-term operational efficiency.

The *RideSyncIoT* system holds potential applications in sports analytics by providing real-time performance data, as well as in downhill or urban contexts where bike-to-bike IoT connectivity could benefit monitoring events. Such applications include rider localization and sports monitoring, where connectivity between bicycles may improve safety and performance tracking.

Future work will prioritize advancements in energy harvesting and consumption, specifically focusing on optimizing the thermoelectric generator module Additionally, an in-depth evaluation of energy consumption metrics and continuous monitoring of energy harvesting efficiency will offer insights for optimizing the system's autonomous performance.

In conclusion, the IoT-based bike monitoring system, featuring the Arduino Nano BLE 33 Sense Rev 2 and the *RideSyncIoT* Android app, has successfully met its design objectives of efficiency, compactness, and power optimization. By leveraging the smartphone for GPS functionality and prioritizing low-power sensor integration, the system minimizes hardware requirements while maintaining robust data acquisition and real-time monitoring capabilities. The *RideSyncIoT* app, developed through MIT App Inventor, offers an intuitive interface for data visualization and seamless BLE and MQTT communication with the Arduino. It further enhances the user experience with real-time GPS mapping, distance tracking, and integrated navigation.

The app's MQTT and BLE connectivity was rigorously tested to confirm stability, including reconnection protocols and data caching mechanisms to ensure data continuity during connectivity lapses. Testing demonstrated the system's accuracy in environmental, motion, and navigation data visualization, as well as effective error handling under various real-world conditions. The successful integration of Grafana for data analytics further underscores the system's capability to collect, process, and display bike ride data efficiently.

This work exemplifies a well-rounded approach to IoT-driven bike monitoring, providing a foundation for future expansions. Overall, the *RideSyncIoT* system proves to be a practical, user-friendly solution for real-time bike tracking and data analysis, well-suited for both urban commuting and recreational use.

References

[1] D. Li, Y. Zhao, Y. Wang, D. An, and Q. Yang, "The privacy preserving auction mechanisms in iot-based trading market: A survey," *Internet of Things*, vol. 26, p. 101178, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2542660524001197

[2] D. Miorandi, S. Sicari, F. De Pellegrini, and I. Chlamtac, "Internet of things: Vision, applications and research challenges," *Ad Hoc Networks*, vol. 10, no. 7, pp. 1497–1516, 2012. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1570870512000674

[3] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi, "Internet of things for smart cities," *IEEE Internet of Things Journal*, vol. 1, no. 1, pp. 22–32, 2014.

[4] T. Sanislav, G. Mois, S. Zeadally, and S. Folea, "Energy harvesting techniques for internet of things (iot)," *IEEE Access*, vol. PP, pp. 1–1, 03 2021.

[5] S. A. Hashmi, C. F. Ali, and S. Zafar, "Internet of things and cloud computing-based energy management system for demand side management in smart grid," *International Journal of Energy Research*, vol. 45, no. 1, pp. 1007–1022, 2021. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/er.6141

[6] H. N. S. Aldin, M. R. Ghods, F. Nayebipour, and M. N. Torshiz, "A comprehensive review of energy harvesting and routing strategies for iot sensors sustainability and communication technology," *Sensors International*, vol. 5, p. 100258, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2666351123000323

[7] X. Yang, Y. Xu, Y. Zhou, S. Song, and Y. Wu, "Demand-aware mobile bike-sharing service using collaborative computing and information fusion in 5g iot environment," *Digital Communications and Networks*, vol. 8, pp. 984–994, 2022. [Online]. Available: https://doi.org/10.1016/j.dcan.2022.06.004

[8] D. Yadav, R. Kumar, U. Kulshrestha, A. Jain, and S. Rani, "Enhancement of fuel efficiency in heavy duty vehicles through integrated module of teg, piezoelectric and regenerative braking solutions," *Materials Today: Proceedings*, vol. 63, pp. 1–5, 2022. [Online]. Available: https://doi.org/10.1016/j.matpr.2021.10.372

[9] A. Coulibaly, N. Zioui, S. Bentouba, S. Kelouwani, and M. Bourouis, "Use of thermoelectric generators to harvest energy from motor vehicle brake discs," *Case Studies in Thermal Engineering*, vol. 28, 2021.

[10] A. Ahmed, R. Zhao, and Z. Zhang, "Energy recovery in electric bicycles through thermoelectric generators," *Journal of Clean Energy Technologies*, vol. 10, no. 1, pp. 50–57, 2022.

The author has the following address:
giuseppe.pasquini@studenti.unitn.it
This report is the final document for the course of "Laboratory of Internet of Things".