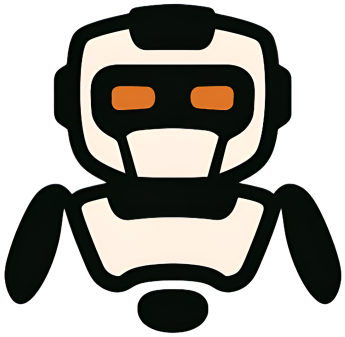

LF - ROBOTICS

Electronic Upgrade of a FIRA Differential Robot v1.0

Developed for:
Ph.D. Mario Fernando De La Rosa Rosero

Submitted by:
Ing. Leffer Trochez

July 28, 2025

LF-ROBOTICS	Electronic Upgrade of a FIRA Differential Robot	Version 1.0
July 28, 2025		Ing. Leffer Trochez

Table of Contents

1 Introduction

This report presents the electronic upgrade of a differential robot originally designed for participation in the FIRA football league. The original system followed a remote-brainless architecture, where all sensing, decision-making, and control logic were executed by a host computer. The robot received movement commands via radio frequency (RF) communication and responded with basic motion, without any onboard intelligence or sensors [1]. Vision processing was handled externally using a top-mounted CCD camera connected to a frame grabber card, and software developed in Visual C++ [2].

With advancements in robotics hardware and software, the robot's electronic system has become outdated and incompatible with modern development tools. To address this, the current project replaces the legacy RF-based architecture with a modular ROS 2-compatible design. The new system integrates onboard sensors (MPU6050, VL53L0X, INA219), a camera (ESP32-CAM), dual communication modes (Bluetooth and Wi-Fi) into a perfboard-based prototype circuit. Although the LiDAR sensor (LD19) is pending integration and the PCB design is not yet finalized, the new architecture enables real-time sensing, onboard processing, and future autonomous behavior.

Figure 1 shows the robot's physical appearance before and after the upgrade. The left side corresponds to the original RF-based version used in competition, and the right side shows the current state of the upgraded robot, where components are mounted on a perfboard. The final PCB and LiDAR integration will be addressed in future work.

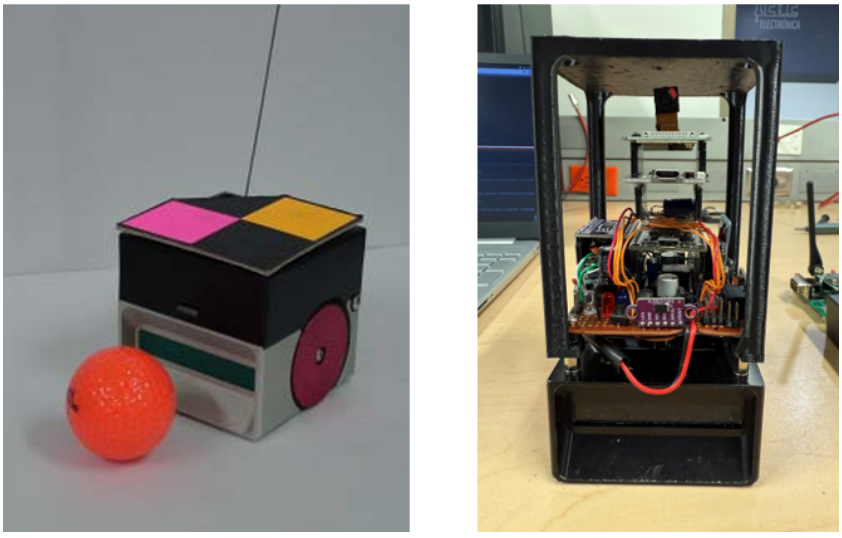

Figure 1: Comparison of the robot before (left) [1] and after (right) the electronic upgrade.

This upgrade provides a flexible foundation for ongoing development, including autonomous navigation, advanced control strategies, and open-source collaboration.

1.1 Project Background

The robot presented in this report was originally designed for participation in the FIRA (Federation of International Robot-soccer Association) football league. It followed a remote-brainless architecture, in which all vision processing and decision-making were performed on a host computer. The robot received commands wirelessly through an RF module and executed basic movements accordingly. Its intelligence was external, relying entirely on a top-mounted CCD camera, a Matrox Meteor frame grabber, and a Visual C++ control program.

Although the robot was functional for its intended purpose, it lacked onboard autonomy, modern communication interfaces, and real-time data acquisition. The original hardware included an ATmega128 microcontroller, a custom PCB, encoders, and a simple power system. Over time, this architecture became obsolete, especially with the rise of more intelligent, decentralized systems using ROS 2 and modern microcontrollers.

1.2 Scope and Limitations

This project focuses exclusively on upgrading the electronic and control architecture of the robot. The mechanical structure, motors, and drivetrain remain unchanged. The new design introduces modular electronics, ROS 2 integration, wireless communication via Bluetooth and Wi-Fi, and onboard sensors, including a camera (ESP32-CAM), IMU (MPU6050), voltage/current sensor (INA219), and a distance sensing system consisting of VL53L0X sensors with I^2C multiplexing. A LiDAR unit (LD19) is planned but not yet integrated.

At this stage, the system is implemented on a perfboard prototype. Full PCB development and final performance characterization will be conducted once all components, including the LiDAR, are fully integrated. The software is provided as a ROS 2-based reference implementation to support future development. Therefore, this project represents a transitional step between the legacy design and a fully autonomous, modern robotic platform.

2 Original Robot

This section presents an overview of the original differential robot developed for participation in the FIRA competition. It details the robot's mechanical and electronic architecture, highlighting its remote-brainless design philosophy, where all processing was performed externally. The limitations of this legacy system are also identified as the main motivation for the electronic upgrade.

2.1 FIRA Robot Description

The original robot was designed for the FIRA competition, where small differential drive robots play soccer autonomously. This specific robot followed a remote-brainless design paradigm, relying entirely on external computation for decision-making and sensing. The robot's mechanical platform consisted of a simple differential drive base with two DC motors and wheels. The frame was lightweight and compact, designed to fit within the dimensional constraints of the competition. The robot was equipped with a top-mounted marker for visual tracking, but it lacked any onboard sensors or processing capabilities.

Instead of local intelligence, the robot received high-level commands over RF communication from a host PC, which observed the playing field through an overhead CCD camera and processed video data to detect the robot and the ball.

2.2 Legacy Electronic Features

The robot's electronics were minimal and centered around signal reception and motor control. As illustrated in Figure 2, the system was composed of:

- A communication module (e.g., ZigBee) to receive commands from the external PC.
- A microcontroller (PIC16F877) programmed in assembly, responsible for interpreting these commands.
- A dual H-bridge motor driver for controlling the two DC motors.
- A voltage regulation circuit and power supply (lithium-ion battery).

The entire electronics were integrated onto a compact PCB, developed specifically for this platform, as documented in the original user manual [1].

Communication from the host PC was one-way: it sent direction and speed instructions derived from video analysis. Figure 3 shows the Matrox Meteor II frame grabber card, which interfaced the CCD camera with the host PC for vision processing.

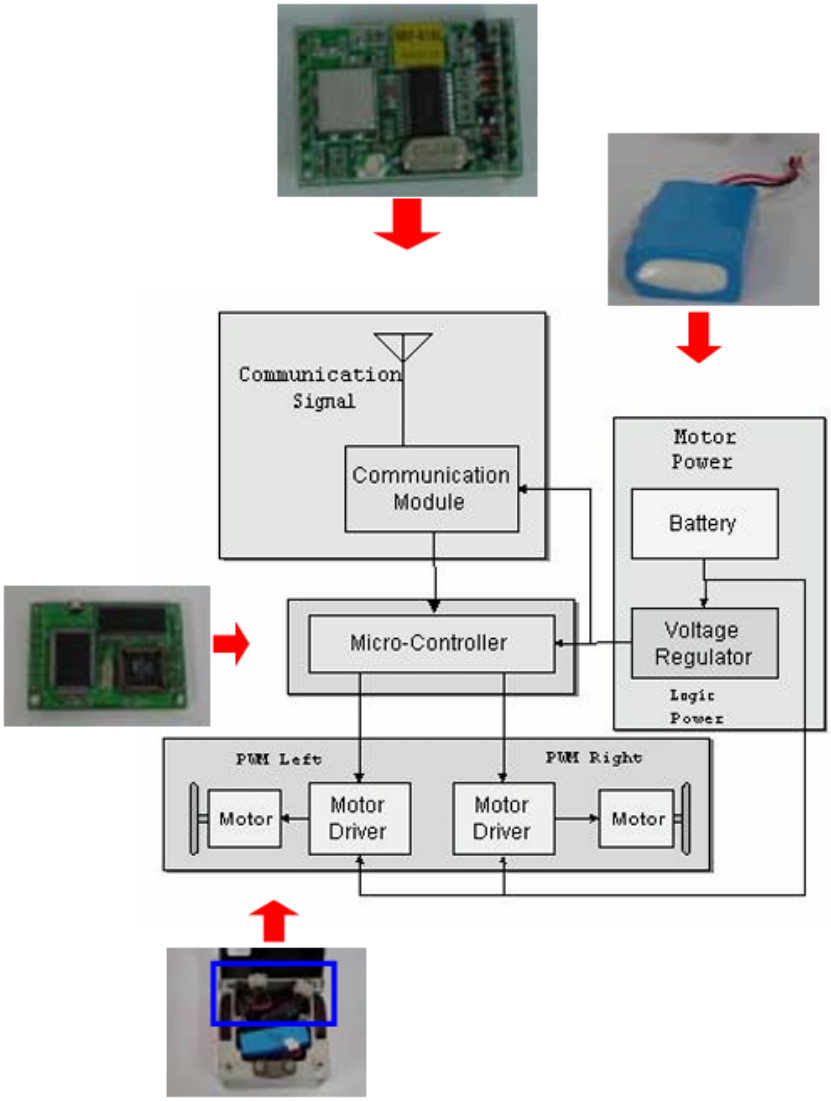

Figure 2: Main parts of robot system from the original architecture [1].

Figure 3: Matrox Meteor 2 card used for video capture (frame grabber) [1].

2.3 Identified Limitations

While the original system fulfilled its role in early competitions, it had significant limitations that restricted further development:

- **No onboard intelligence:** All processing occurred externally. The robot could not react to local events or adjust its behavior autonomously.
- **No sensors:** It lacked IMUs, distance sensors, encoders, or any internal feedback mechanisms.
- **RF dependency:** Movement was completely dependent on wireless communication. Loss or delay in the signal rendered the robot unresponsive.
- **Outdated tools:** The firmware was written in assembly, and the host software was developed in Visual C++ for Windows XP. The system was not compatible with modern platforms or open-source tools.

These limitations motivated a full electronic redesign to support modularity, real-time sensing, and ROS 2-based development workflows. Figure 4 displays the main mechanical and electronic components that made up the original robot. The kit includes two geared DC motors, aluminum wheels, bearings, motor drivers, a communication module, a battery pack, chassis parts, and various mounting accessories. These components were designed for quick assembly and interfacing with the remote control system. The control board and communication units illustrate the modularity and educational intent behind the robot's design.

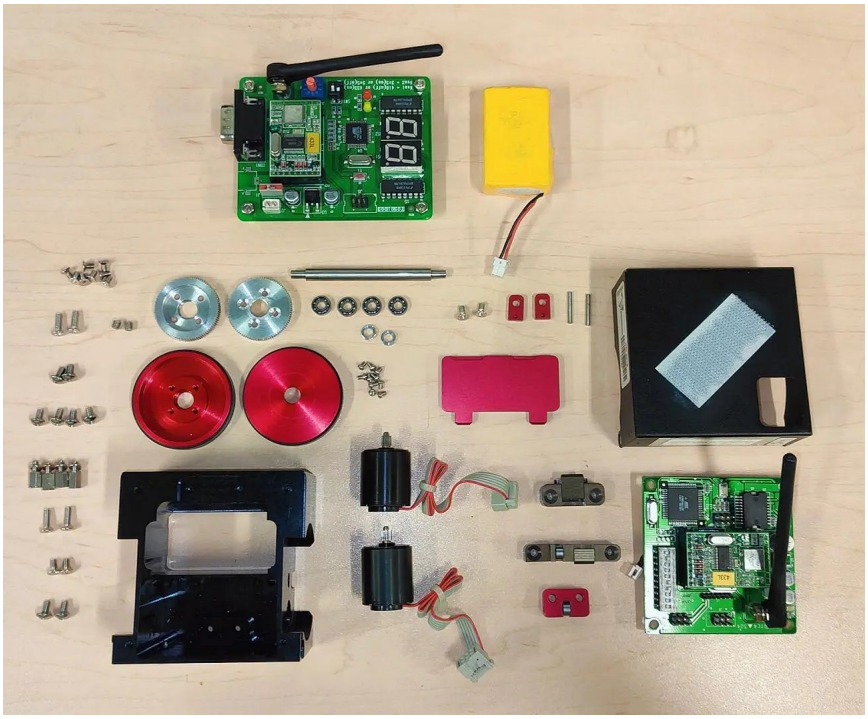

Figure 4: Main components of the original differential robot. Source: Author.

3 Justification of the Upgrade

The original differential robot, while effective for its time, was built around a remote-brainless architecture that lacks the autonomy, flexibility, and scalability demanded by modern robotics research and education. This motivated a comprehensive electronic upgrade to align the system with modern standards and enable future enhancements.

3.1 Needs and Design Requirements

The main needs identified for the upgrade included:

- **Onboard Intelligence:** Eliminate dependency on external computing by incorporating microcontrollers and edge processing capabilities.
- **Sensor Integration:** Add real-time sensing for motion estimation, obstacle detection, and battery monitoring (e.g., MPU6050, VL53L0X, INA219).
- **Modular Architecture:** Support flexible hardware configuration and scalable software integration.
- **Communication Versatility:** Enable Bluetooth for local control and Wi-Fi for ROS 2 communication with a remote agent.
- **Future-Proofing:** Design the electronics to support advanced features such as LiDAR integration and autonomous behaviors.

3.2 Advantages of ROS 2 and Modern Hardware

The adoption of ROS 2 and modern embedded platforms like the ESP32-CAM and ESP32 microcontrollers introduces significant advantages:

- **Real-Time Communication:** ROS 2 ensures reliable and low-latency data exchange between components using DDS (Data Distribution Service).
- **Decentralized Architecture:** Tasks such as sensing, actuation, and decision-making can be distributed among nodes, improving robustness and scalability.
- **Ecosystem Compatibility:** The robot becomes compatible with widely used tools in academia and industry, facilitating research, collaboration, and future upgrades.
- **Open-Source Tools:** Access to a large collection of ROS packages reduces development time and allows integration with simulators, visualization tools, and SLAM algorithms.
- **Energy Efficiency:** Modern microcontrollers are optimized for low-power operation, increasing the system's autonomy.

4 Electronic System Architecture

This section describes the new electronic architecture developed for the upgraded robot. The system is designed around modular components including dual ESP32 microcontrollers, sensor interfaces, communication modules, and motor drivers. The architecture supports ROS 2 integration, enabling real-time data exchange and decentralized control. A perfboard prototype was used for initial testing, with plans for a custom PCB once all components are finalized.

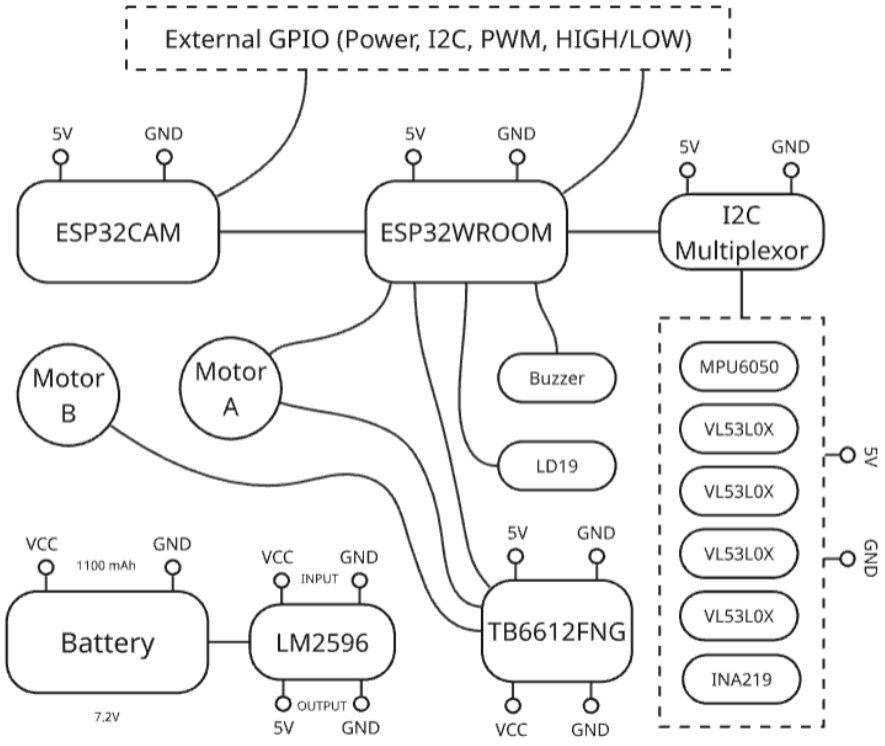

Figure 5: Electronic schematic showing the connection between microcontrollers, sensors, actuators, and power system.

Figure 5 shows the electronic schematic of the upgraded robot. A custom battery pack (7.2V, 1100mAh) built from AAA cells powers the system, with a diode and capacitor for reverse current protection. A step-down regulator converts this voltage to 5V, supplying the ESP32, ESP32-CAM, TB6612FNG motor driver, and all sensors. The motors are powered directly from the battery's unregulated voltage. The I^2C multiplexer connects to the ESP32 and distributes the 5V supply to 4 VL53L0X sensors, an MPU6050 IMU, and an INA219 power sensor. A buzzer and LED indicators provide audio and visual feedback, while the mode selector button is connected to a GPIO with pull-down resistance. The LD19 LiDAR connects to the ESP32 via PWM and UART. Additional GPIO pins from the ESP32, ESP32-CAM, and multiplexer are exposed for future sensor or actuator expansion. Capacitors and filters stabilize the power lines throughout the system.

4.1 Power Distribution

The robot is powered by a custom-built battery pack composed of six AAA rechargeable Ni-MH batteries connected in series, each rated at 1.2V and 1100mAh. This configuration provides a nominal voltage of 7.2V, with an operating range between 6.0V and 8.0V, accounting for the voltage drop introduced by a protection diode (1N4004) placed in series to prevent reverse current. A 1000 µF capacitor is also integrated at the battery output to stabilize transient voltages during load variations. The battery feeds a step-down regulator that supplies a constant 5V rail. This regulated 5V line powers the ESP32, ESP32-CAM, TB6612FNG motor driver (logic section), motors (via motor power input), sensors (VL53L0X, MPU6050, INA219), the LD19 LiDAR, and other peripherals. A 3.3V line derived from the ESP32 is used to drive indicator LEDs. Capacitors are placed at key power entry points (e.g., motors, sensors) to reduce noise and maintain voltage stability, and inductive filtering (RC filter) is used before the driver motor and voltage regulator inputs for additional protection and decoupling. The custom battery housing was designed and 3D printed to fit securely within the robot's chassis (see Figure 6).

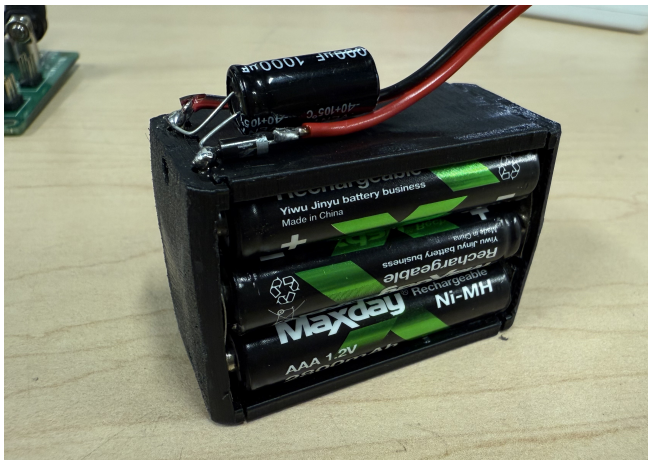

Figure 6: Custom battery pack using six AAA Ni-MH cells with diode and capacitor protection.

Preliminary tests indicate that the custom battery pack provides approximately 20 minutes of continuous operation under standard usage. This estimation considers the simultaneous use of sensors, communication modules, and motors. When the full system is active, it draws approximately 750 mA at 5 V, which corresponds to a power consumption of around 3.75 W. A complete characterization of power consumption will be performed once all components are fully integrated into the perfboard.

4.2 Mode Switching (Bluetooth/Wi-Fi)

The robot includes a mode switching mechanism that allows the user to alternate between Bluetooth control and Wi-Fi communication for ROS 2 integration. This functionality is implemented using a single push-button connected to a digital input pin on the ESP32, configured with an internal pull-down resistor to ensure stable readings.

The system behavior in response to user interaction is as follows:

- **Single press:** The system enters **Bluetooth mode**, enabling control via the **Dabble** mobile application. A red LED turns on to indicate this mode. Additionally, the buzzer emits a single short beep as acoustic feedback.
- **Double press:** The system switches to **Wi-Fi mode**, allowing the robot to communicate with ROS 2 nodes over a local wireless network. A blue LED indicates that Wi-Fi mode is active, and the buzzer emits two short beeps.
- **Long press (more than 2 seconds):** The ESP32 performs a system reset. This is indicated by a continuous 1-second tone from the buzzer, signaling that the system is reinitializing.

4.3 Prototype Built on Perfboard

The complete electronic system was manually assembled on a double-layer perfboard, as illustrated in Fig. 7. This prototyping method enabled full customization of the layout to fit the mechanical structure of the robot while maintaining a compact footprint. The top view shows the main modules, including the ESP32, ESP32-CAM, voltage regulator, push-button, I2C multiplexer, and capacitors. Each component is securely soldered and connected with jumper wires for reliable operation.

The bottom view of the perfboard reveals the dense and carefully routed wiring that interconnects all system components. Color-coded wires help distinguish between power, ground, and signal lines, ensuring easier debugging and maintenance. Despite the complexity, the design remains robust and serviceable.

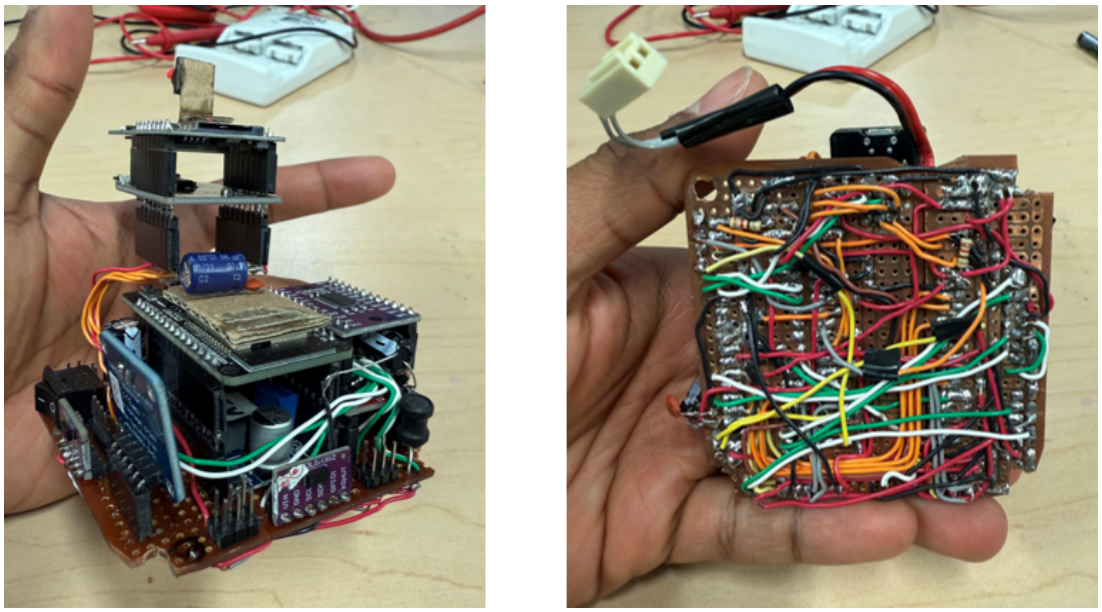

Figure 7: Prototype assembled on a perfboard: front (left) and back (right) views.

5 Hardware Components

This section describes the key hardware components used in the upgraded differential robot. The integration of these elements was carefully designed to ensure real-time performance, modularity, and efficient energy usage. The system achieves a balance between powerful onboard capabilities and economic design.

5.1 List of Components

The upgraded robot includes the following main hardware components:

- **ESP32 DevKit V1:** Main microcontroller.
- **ESP32-CAM:** Camera module for video streaming.
- **Faulhaber MOTOR 2224U006SR (x2):** Compact DC motors.
- **TB6612FNG:** Dual motor driver.
- **VL53L0X (x4):** ToF distance sensors.
- **TCA9548A:** I2C multiplexer.
- **MPU6050:** IMU sensor.
- **INA219:** Voltage/current sensor.
- **LD19 LiDAR:** (Pending) 360° distance scanning.
- **Battery Pack:** 7.2V made of six AAA batteries.
- **5V Regulator:** Step-down converter.
- **Diode, Capacitors, Inductors:** For protection and filtering.
- **Buzzer & LEDs:** Mode indicators.
- **Push Button:** Mode selector and reset.
- **Perfboard:** Modular layout with expansion pins.

5.2 Sensor Integration (VL53L0X, MPU6050, INA219, etc.)

The ESP32 serves as the central processor for multiple onboard sensors. It interfaces with four VL53L0X time-of-flight distance sensors using a TCA9548A I^2C multiplexer, which enables individual addressing and simultaneous data acquisition. The MPU6050 provides real-time acceleration and gyroscopic measurements, allowing motion and orientation estimation. The INA219 sensor is responsible for monitoring voltage and current consumption of the system.

The logic implemented in the microcontroller reads each sensor sequentially and outputs the measurements as a single comma-separated string via serial. This string is structured for easy parsing by an external system, such as a ROS 2 node. The serial format helps unify sensor data into a common interface for robotic control and monitoring. The source code for this integration is available in the project's GitHub repository under the filename `ESP32code.ino`.

5.3 Camera Integration (ESP32-CAM)

The ESP32-CAM module is responsible for streaming live video over Wi-Fi and offering basic control via Bluetooth through the Dabble application. The camera is configured with the OV2640 sensor to output VGA-resolution JPEG frames. On startup, the ESP32-CAM connects to a predefined Wi-Fi network and launches an HTTP server on port 80, serving MJPEG video streams. Mode selection is handled via a digital input pin (`MODE_PIN`). When the pin reads HIGH, the device enters Bluetooth mode, activating the Dabble Gamepad module for mobile control. When LOW, it runs in Wi-Fi mode, making the video stream accessible on any browser within the local network.

In Bluetooth mode, joystick and button inputs from Dabble are mapped to PWM signals using `ledcWrite()` on GPIO14, enabling control of actuators. The complete source code is available in the GitHub repository under the filename `ESP32CAMcode.ino`.

5.4 Expansion Interface for External Devices

To ensure future scalability, the prototype includes a set of accessible pins routed from the ESP32, ESP32-CAM, and I2C multiplexer. These allow easy connection of additional actuators or sensors without modifying the main circuit. The available expansion interface includes:

- **Power Pins:**
  - **5V (x3):** Regulated power output for 5V devices.
  - **3.3V (x2):** Direct supply from ESP32 for 3.3V modules.
  - **GND (x5):** Ground connections.
  - **VP:** Analog pin from ESP32 (input only).

- **ESP32 GPIOs:**
  - **D13:** General-purpose digital I/O.
- **ESP32-CAM GPIOs:**
  - **IO4, IO2, IO15, IO16, IO13:** Usable digital I/O for external control or sensing.
- **I2C Multiplexer (TCA9548A):**
  - **SC4, SD4, SC3, SD3:** Extra I2C channels for connecting more I2C-compatible devices.

These expansion ports make the system flexible and adaptable for future robotic experiments or sensor integration.

Through these external pins, users can connect and control actuators using different signal types: PWM signals in the range [0–254], digital signals such as HIGH/LOW (0 or 1), or angle values between 0 and 180 for servo motors. These pins allow users to expand the robot's capabilities by integrating additional components. With ROS 2 integration, these actuators can be managed from a central interface, enabling the development of custom control algorithms, which will be described in the following section.

6 Software Architecture and ROS 2

This section outlines the software architecture implemented in the upgraded robot, emphasizing the integration with ROS 2. The system is designed to ensure modularity, scalability, and real-time interaction between hardware and software components, enabling advanced behaviors and remote control via ROS 2 nodes.

6.1 Node-Based System Overview

The diagram in Figure 8 illustrates the node-based software architecture developed for the upgraded robot. At the core is the `/robot_fira_4_node`, running on the ESP32 with micro-ROS. To interface with the ROS 2 ecosystem, a micro-ROS agent is required to expose this node to the ROS 2 network.

Several nodes were implemented to demonstrate real applications, including camera streaming (`/camera_node`), sensor reading (`/sensors_node`), encoders (`/encoders_node`), and actuator control (`/actuators_node`). All these nodes communicate with the main node, which is (`/teleop_node`) through ROS 2 topics. The `/teleop_node` provides a terminal interface to control motors, actuators, and monitor sensors in real-time, including battery percentage and system feedback. Each of these nodes and their respective topics will be explained in detail in the following subsections.

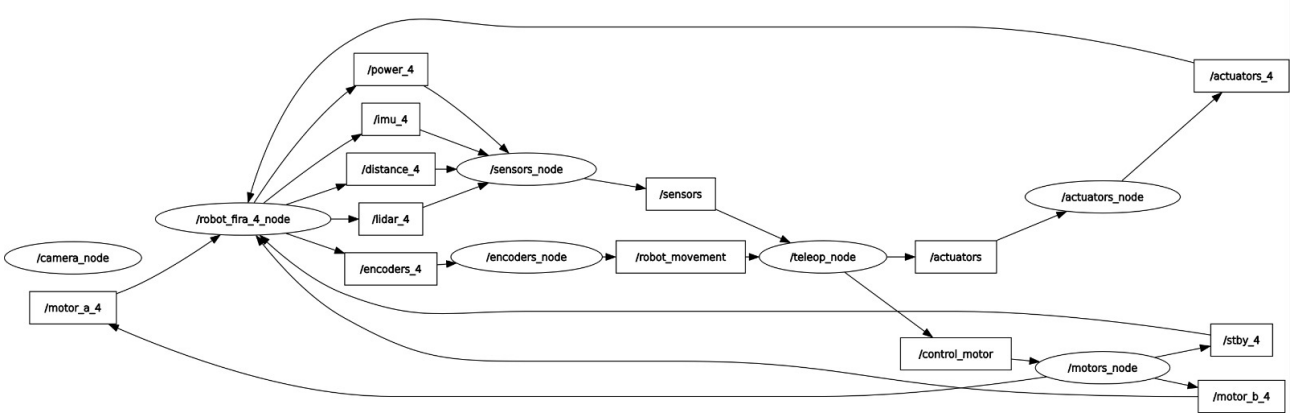

Figure 8: ROS 2 node-based system architecture of the upgraded differential robot. The central node `/robot_fira_4_node` interfaces with multiple sensor, actuator, and control nodes using micro-ROS. The `/teleop_node` acts as the main controller, receiving sensor data and user inputs to manage robot behavior.

6.2 Description of Each Node

This section provides a concise explanation of each ROS 2 node used in the upgraded robot system. The full source code for all nodes is available in the project repository. For each node, the logic, role in the system, and the relevant topics it publishes or subscribes to are described to facilitate understanding and customization.

6.2.1 encoders_node.py

This ROS 2 node processes encoder pulse counts received from the ESP32 via the topic `/encoders_4`, which sends a `Float32MultiArray` containing four integer values: `[aA, bA, aB, bB]`, corresponding to the quadrature encoder counts for both left (A) and right (B) wheels (channels A and B respectively).

Upon receiving the data, the node computes:

- **Left wheel linear velocity** (`vel_izq`) in centimeters per second (cm/s),
- **Right wheel linear velocity** (`vel_der`) in centimeters per second (cm/s),
- **Robot total linear velocity** (`vel_total`) in cm/s, as the average of both wheels,
- **Angular velocity** (ω) in radians per second (rad/s), derived from the velocity difference divided by the robot's track width (7.0 cm),
- **Movement state** as an integer:
  - 0: Stopped,
  - 1: Moving forward,
  - 2: Moving backward,
  - 3: Turning.

These processed values are published to the topic `/robot_movement` as a `Float32MultiArray` with the structure:

```
[vel_total, vel_izq, vel_der, movement_type, omega]
```

This node enables the real-time tracking of robot mobility status for further use in teleoperation, motion planning, or logging. The full implementation is available in the repository under `encoders_node.py`.

6.2.2 actuators_node.py

This ROS 2 node handles the control signals for external actuators. It listens to the topic `/actuators`, which sends a `Float32MultiArray` containing the PWM values or control parameters for connected actuators (such as servos or digital outputs).

Upon receiving a message, the node immediately republishes the same data to the topic `/actuators_4`, which is subscribed by the ESP32. This design allows modular separation between high-level decision-making (e.g., from teleoperation or AI logic) and low-level hardware actuation. The expected message structure is:

```
[pwm1 (0-255), pwm2 (0-255), ...]
```

Where each value corresponds to the PWM duty cycle for a specific actuator. The values can be used to:

- Drive servo motors with angular position encoding (0–180 degrees mapped from PWM),
- Control digital actuators with high (1) / low (0) logic,
- Modulate power to devices using PWM intensity (0–255).

This node facilitates direct actuator control via ROS 2 while maintaining the communication protocol compatible with the ESP32 firmware. The code is available in the repository under the file name `actuators_node.py`.

6.2.3 camera_node.py

This node manages the reception and visualization of real-time video from the ESP32-CAM module. It connects to the MJPEG HTTP video stream exposed by the ESP32-CAM at the specified URL (`http://192.168.0.103/` in this case). Upon initialization, the node attempts to open the video stream using OpenCV's `cv2.VideoCapture()`. If the connection fails, it retries every 3 seconds until successful. Once connected, a ROS 2 timer triggers the `capture_frame` method every 0.1 seconds (i.e., 10 Hz), allowing continuous video retrieval and display. The key functions of the node include:

- Maintaining a persistent connection to the camera stream.
- Displaying each frame using `cv2.imshow()`.
- Reconnecting automatically if the stream is interrupted.
- Gracefully shutting down the video feed upon user interruption (e.g., `Esc` key).

No ROS topics are published or subscribed in this implementation, as the primary purpose is to visualize the video feed. This allows users to monitor the robot's environment visually during development or testing. The code is available in the repository under the file name `camera_node.py`.

6.2.4 motor_node.py

This node receives high-level control commands and translates them into specific PWM and direction signals for the robot's left and right motors (Motor A and Motor B). It also manages the standby ('STBY') state of the TB6612FNG motor driver. The node subscribes to the topic:

- `control_motor` (`Float32MultiArray`): Receives an array with four values: PWM_A, PWM_B, STBY (boolean), and movement code (1: forward, 2: backward, 3: turn left, 4: turn right, 0 or others: stop).

It publishes to:

- `motor_a_4` (`Float32MultiArray`): Contains the PWM and direction values [PWM, DIR1, DIR2] for Motor A.
- `motor_b_4` (`Float32MultiArray`): Contains the same structure for Motor B.
- `stby_4` (`Bool`): Publishes the current standby state to indicate if the driver is active.

The PWM values range from 0 to 254, while DIR1 and DIR2 control the rotation direction. For example, [PWM, 1, 0] results in clockwise rotation. Depending on the value of the movement code, the node calls predefined functions that encapsulate motion logic (e.g., `forward()`, `backward()`, etc.). If the standby flag is set to false, the motors are stopped regardless of the movement instruction. This code is available in the repository under the name `motor_node.py`.

6.2.5 sensors_node.py

This node is responsible for aggregating sensor data from multiple sources and publishing it as a single message on the `/sensors` topic. It subscribes to the following topics:

- `imu_4` (`Float32MultiArray`): Receives six values from the IMU sensor — acceleration $[a_x, a_y, a_z]$ in $\mathrm{m/s^2}$ and angular velocity $[g_x, g_y, g_z]$ in °/s.
- `distance_4` (`Float32MultiArray`): Receives four values from VL53L0X distance sensors in mm.
- `power_4` (`Float32MultiArray`): Receives two values from the INA219 sensor — voltage (V) and current (mA).
- `lidar_4` (`Float32MultiArray`): Receives LiDAR data (up to 360 values) representing distances in mm or cm.

It publishes aggregated data on:

- `sensors` (`Float32MultiArray`): Contains a vector of 12 values (6 from IMU, 4 from distance sensors, and 2 from power measurement).

The node validates incoming messages to ensure data integrity. If unexpected values are received, especially from the power topic (e.g., voltage $> 30\,\text{V}$ or current outside valid range), correction logic is applied to maintain robustness. This node facilitates centralized access to the robot's environmental and internal state. The code is available in the repository under the name `sensors_node.py`.

6.3 Launch and Configuration Files

6.3.1 launch_nodes.launch.py

This launch file initializes the complete set of ROS 2 nodes required for the robot system using a single unified process. It is written using the `launch` and `launch_ros` Python API. The launch description includes:

- **Micro-ROS Agent:** A process executed with `ExecuteProcess` that launches the Micro-ROS Agent in UDP mode, listening on port `8888`. This agent is essential for enabling communication between the ROS 2 environment and microcontroller-based clients (e.g., ESP32).
- **ROS 2 Nodes:** Each of the following nodes is launched from the `robot_fira` package and is responsible for a specific subsystem:
  - `actuators_node`: For forwarding actuator control commands.
  - `camera_node`: For streaming video from the ESP32-CAM.
  - `encoders_node`: For processing encoder data and calculating robot motion.
  - `motors_node`: For interpreting motor commands and generating control signals.
  - `sensors_node`: For aggregating data from IMU, distance sensors, power monitoring, and LiDAR.

All nodes and processes are set to output their logs to the screen, aiding real-time debugging. This centralized launch script simplifies deployment and ensures all interdependent nodes start in the correct sequence.

6.3.2 setup.py

The `setup.py` script defines the ROS 2 package structure for `robot_fira`. It installs necessary files (e.g., launch files and `package.xml`) and registers executable nodes so they can be launched with:

```
ros2 run robot_fira <node_name>
```

The main nodes included are: `motors_node`, `teleop_node`, `actuators_node`, `sensors_node`, `encoders_node`, and `camera_node`.

6.4 Code as a Base Proposal for Developers

The file `teleop_node.py` serves as the core interface for controlling and monitoring the differential robot. It is proposed as a base implementation that developers can extend to suit their use cases. This node publishes control signals to `/control_motor` and `/actuators`, and subscribes to `/sensors` and `/robot_movement` to visualize real-time feedback.

The interface is built using the `curses` library to enable:

- Setting PWM values for motors (`pwm_a`, `pwm_b`) and actuators (0–255).
- Switching standby state (`stby`).
- Real-time robot movement using keyboard arrows ($\uparrow \downarrow \leftarrow \rightarrow$).
- Visualization of IMU data (acceleration in m/s^2, angular velocity in rad/s), distance sensors (mm), voltage (V), current (mA), power (W), and battery percentage.
- Monitoring of speed data from encoders (left, right, total velocity in cm/s, and angular velocity in rad/s).

This node runs alongside all other nodes via a unified ROS 2 launch file. Additionally, the launch file executes the `micro_ros_agent` over UDP on port 8888, allowing the ESP32 to connect and publish data as a native ROS 2 node. Figure 9 shows the terminal output of the micro-ROS agent confirming the ESP32 connection.

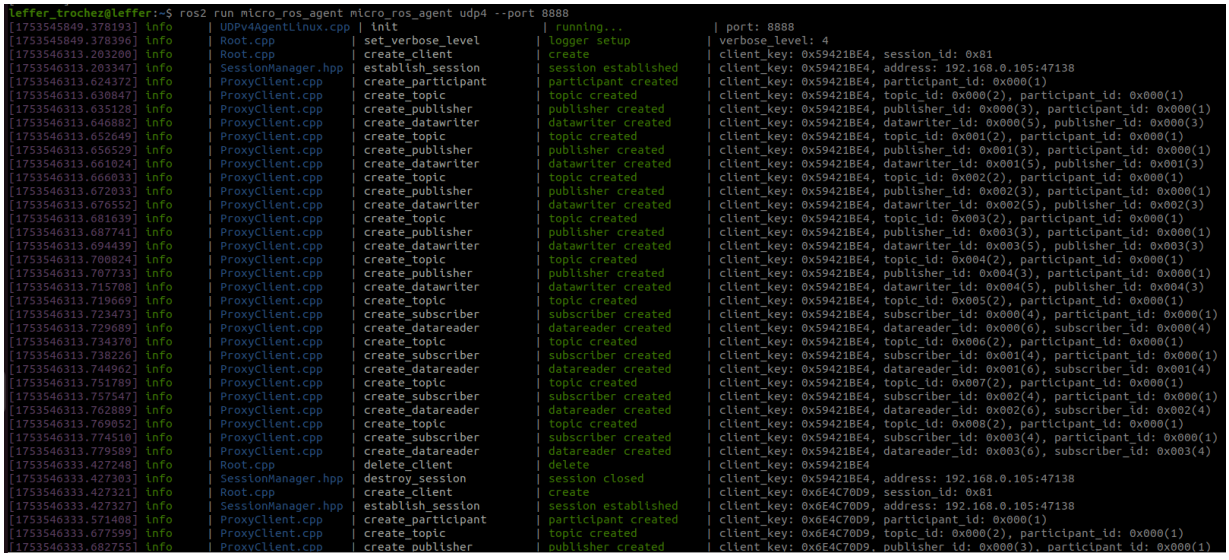

Figure 9: Execution of `micro_ros_agent` over UDP (port 8888) showing ESP32 connection.

Figure 10 shows the ESP32-CAM video stream running in `camera_node.py`, while Figure 11 displays the terminal interface of the teleoperation node.

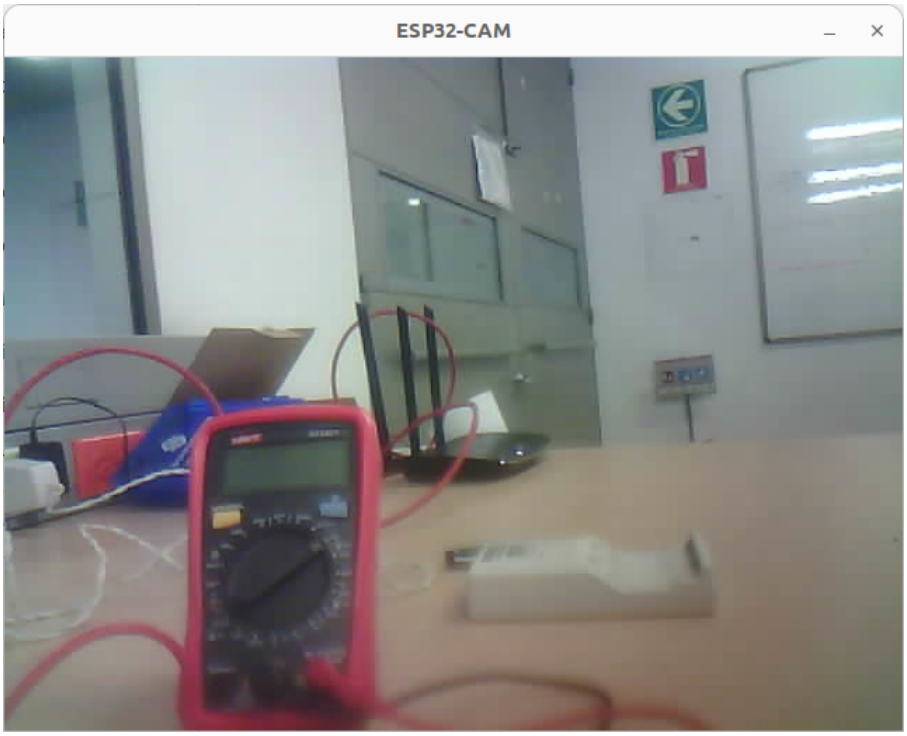

Figure 10: Live video stream from `ESP32-CAM` displayed via `camera_node.py`.

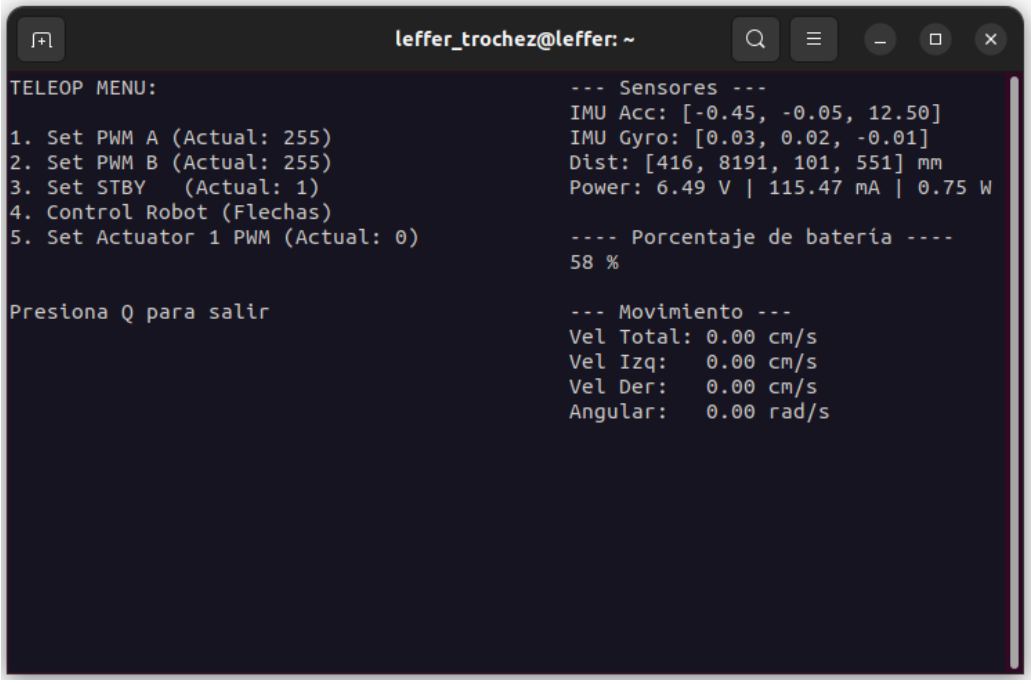

Figure 11: Terminal interface of `teleop_node.py`, used to control the robot and visualize real-time telemetry.

7 System Features and Behavior

The robot operates through a modular ROS 2 system composed of nodes for motors, encoders, sensors, actuators, and camera control. These nodes are launched together with the `micro_ROS_agent`, enabling integration with the ESP32 microcontroller. A terminal-based teleoperation interface allows real-time control of the robot's movement and actuators, while also displaying IMU data, distance measurements, and power status. In addition, the system supports Bluetooth control through the Dabble mobile app, offering an alternative manual interface when ROS 2 is not active.

7.1 Bluetooth Mode

The robot includes a Bluetooth-based manual control mode, which is activated by pressing the onboard push button once. When this mode is active, micro-ROS is disabled to prevent communication conflicts, ensuring exclusive control via Bluetooth. The Dabble app (available on Google Play and App Store) is used in this mode, specifically through its `Gamepad` module. The user interface resembles a standard game controller. The control mapping is as follows:

- **Up Arrow**: Move robot forward
- **Down Arrow**: Move robot backward
- **Square Button**: Turn left
- **Circle Button**: Turn right
- **Triangle Button**: Increase PWM speed
- **X Button**: Decrease PWM speed

This mode is designed for quick manual testing and allows full directional movement and speed control without requiring a ROS 2 interface.

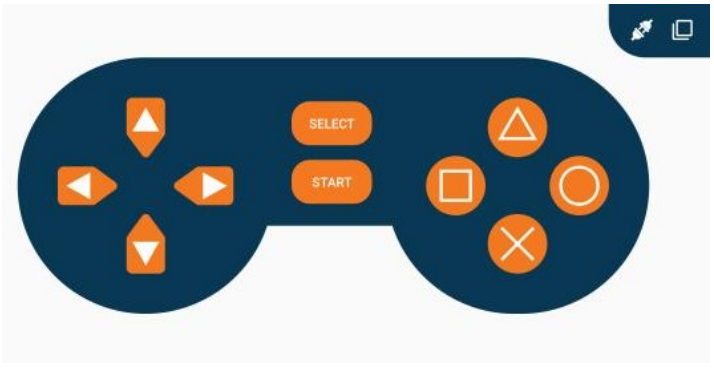

Figure 12: Control interface in the Dabble app's Gamepad module

7.2 Wi-Fi Mode with ROS 2

The Wi-Fi mode is activated by pressing the onboard push button twice. This mode enables full integration with the ROS 2 ecosystem via micro-ROS over UDP, allowing advanced control, data visualization, and node communication. When this mode is active, the Bluetooth interface is automatically disabled to ensure communication integrity.

To illustrate the capabilities of ROS 2, the `teleop_node.py` script is proposed as a user interface. It allows real-time control of the robot, displays sensor readings, actuator states, and movement feedback directly in the terminal using a graphical curses-based menu. Several demonstration videos are included in the repository to confirm and showcase the system's development, behavior, and performance under this mode.

7.3 LED Indicators and Buzzer Alerts

The robot is equipped with three LED indicators that provide clear visual feedback regarding its operational state:

- **Reset LED:** Briefly turns on during system initialization or manual reset.
- **Bluetooth Mode LED:** Lights up when the robot is operating in Bluetooth mode, allowing the user to confirm the selected interface.
- **Wi-Fi Mode LED:** Illuminates when the system enters Wi-Fi mode and establishes connection with the ROS 2 micro-ROS agent.

In addition to the LEDs, a buzzer is used to provide audible alerts when switching between modes. Each mode change is confirmed with a distinct beep pattern, ensuring both visual and auditory confirmation for the user.

7.4 Sensor Readings and Processing

The robot integrates multiple sensors to perceive its environment and enhance navigation and control. These include:

- An **IMU** (Inertial Measurement Unit) for measuring linear acceleration (cm/s^2) and angular velocity (rad/s).
- Four **VL53L0X time-of-flight distance sensors**, providing proximity data in millimeters for obstacle detection.
- A **voltage and current sensor** (INA219), measuring the battery's voltage (V) and current (mA), allowing real-time power monitoring.
- A **LiDAR** (LD19), which provides dense spatial data used for mapping and navigation.

7.5 Modular and Expandable Design

The robot has been designed with modularity and expandability in mind, allowing future developers and researchers to integrate additional electronic components with ease. The architecture supports further extensions such as new sensors, actuators, or communication interfaces. This flexibility enables the platform to adapt to diverse research applications and educational purposes, promoting continuous development and innovation.

7.6 Legacy vs. Upgraded Electronic System

Feature	Legacy Robot	Upgraded Robot
Microcontroller	STM32F103 (ARM Cortex-M3, 32-bit MCU)	ESP32 + ESP32-CAM (dual-core, Wi-Fi + Bluetooth)
Camera	CMOS camera module with FIFO buffer	ESP32-CAM (OV2640)
Motor Drivers	Integrated DC motor drivers	TB6612FNG dual H-bridge drivers
Motors	2 × DC motors with encoders (RS-360SH)	2 × Faulhaber 2224U006SR
Battery	7.4V LiPo (2S)	7.4V Li-ion 2S (AAA batteries)
Sensors	Encoders	Encoders, IMU (MPU6050), TOF (VL53L0X), LiDAR, INA219
Communication	Serial USB and XBee	Wi-Fi (ROS 2) + Bluetooth (Dabble App)
Indicators & Feedback	LED indicators	3 LEDs (Reset, BT, Wi-Fi) + Buzzer alerts
Expandable Ports	Minimal	Multiple I2C, UART, PWM ports available

Table 1: Comparison between the legacy and upgraded robot electronic systems.

8 Future Work

- **Integration and Testing of LD19 LiDAR**
  The LD19 LiDAR sensor will be fully integrated with the ROS 2 ecosystem to enhance environmental perception. Tests will be conducted to validate its mapping accuracy and performance under various lighting and surface conditions.

- **Final Performance Evaluation**
  Comprehensive testing of the robot will be carried out in realistic scenarios to evaluate its locomotion stability, sensor accuracy, wireless communication robustness, and overall system behavior in both control modes.

- **PCB Design and Manufacturing**
  A custom printed circuit board (PCB) will be designed to optimize the wiring, reduce interference, and improve reliability. The PCB will integrate all essential electronic components with proper connectors for modularity and maintainability.

- **Advanced Software Development (SLAM, Navigation)**
  The system will be enhanced with ROS 2 packages for Simultaneous Localization and Mapping (SLAM) and autonomous navigation using the fused sensor data (IMU, LiDAR, and TOF), enabling it to operate independently in unknown environments.

- **Documentation and Open-Source Release**
  All hardware schematics, codebases, and tutorials will be documented and published in an open-source repository to promote collaboration and further research by the community.

9 Appendices

The complete repository for this project is available at:

`https://github.com/LefferTrochez/Robot-FIRA-v1.0.git`

This repository contains all the necessary resources to replicate the project, including:

- Source code for the ESP32 microcontroller.
- Source code for the ESP32-CAM.
- ROS 2 workspace (`ros2_ws`) with the Python nodes used.
- Support videos demonstrating the system in action.
- Instructions for cloning and running the example code.

9 References

[1] Yujin Robot Co., Ltd. *YSR-A (5vs5 Ver128) User Manual*, 2012.

[2] Adept Technology Inc. *SDC-310/240 CCD Camera Module Manual*, 2003.

[3] micro-ROS. *micro-ROS: Embedded ROS 2 for Microcontrollers*. Available at: `https://micro.ros.org/`

[4] STEMpedia. *Dabble App for Bluetooth-based Control*. Available on Google Play and App Store. `https://thestempedia.com/docs/dabble/`

[5] Python Software Foundation. *Python 3 Programming Language*. Available at: `https://www.python.org/`

[6] Arduino. *Arduino IDE*. Available at: `https://www.arduino.cc/en/software`

[7] Espressif Systems. *ESP32 Series Datasheets and Documentation*. Available at: `https://www.espressif.com/en/products/socs/esp32/resources`

[8] Open Robotics. *ROS 2 - Robot Operating System*. Available at: `https://docs.ros.org/en/humble/index.html`

[9] Python Software Foundation. *curses — Terminal Handling for Character-Cell Displays*. Available at: `https://docs.python.org/3/library/curses.html`