LMR-ONE User Manual

Lenna Robotics Group

Arian Hajizadeh & **Erfan Riazati**
Supervisor: Dr. Esmailzadeh

Iran University of Science & Technology
Tehran, Summer 2025

To all the soldiers fallen in the war against robotics. This page is intentionally not blank!

Contents

List of Figures

List of Acronyms

Computer Aided Design ..... CAD
Commercial Of The Shelf ..... COTS
Direct Current ..... DC
High-level Controller Board ..... HCB
Hardware In the Loop ..... HIL
Human-Machine Interface ..... HMI
Inertial Measurement Unit ..... IMU
Low-level Controller Board ..... LCB
Low Drop Out ..... LDO
Light Detection and Ranging ..... LiDAR
Lenna Mobile Robot ONE ..... LMRO
Micro Controller Unit ..... MCU
Printed Circuit Board ..... PCB
Proportional-Integral-Derivative ..... PID
Robot Operating System ..... ROS
Simultaneous Localization and Mapping ..... SLAM
Simulink Desktop Real-Time ..... SLDRT
Universal Asynchronous Receiver/Transmitter ..... UART
Unified Robot Description Format ..... URDF
Wheeled Mobile Robot ..... WMR

1 | Introduction

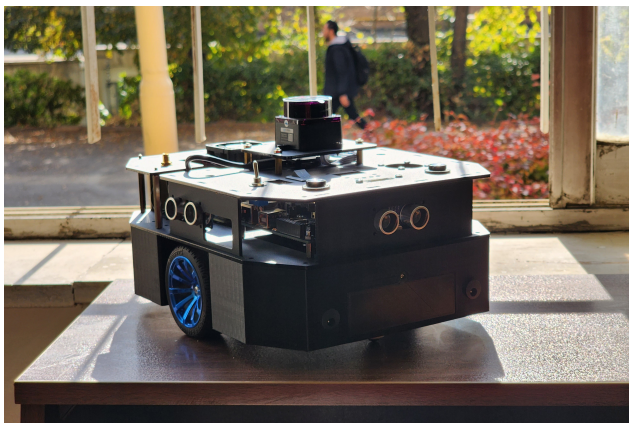
The Lenna Mobile Robot ONE (LMRO) project began in Summer 2023 at the Electrical Engineering Research Laboratories of the Iran University of Science and Technology, with the goal of creating a low-cost, modular robotics platform compatible with the Robot Operating System (ROS). Designed for a wide range of users, from high school students to university researchers, LMRO is fully open-source and available on GitHub.

The primary objective of the LMRO project is to provide a versatile platform for implementing and testing robotics algorithms at various stages of development. This enables users to begin with fundamental tasks in different areas of robotics, e.g., navigation, Simultaneous Localization and Mapping (SLAM), and path planning, or to develop and deploy their own custom applications on the robot for verification and testing.

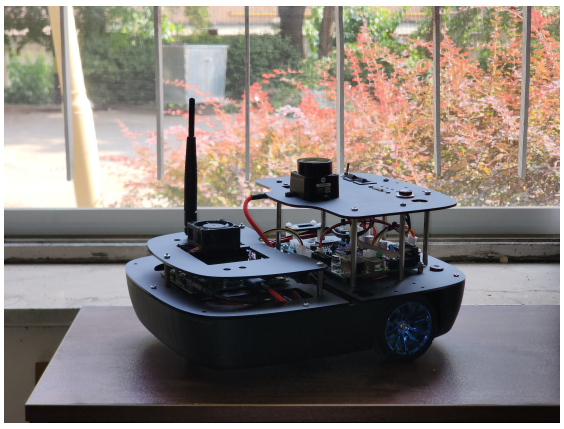
The current design is a differential-drive Wheeled Mobile Robot (WMR) that employs a two-layer hierarchical architecture consisting of a High-level Controller Board (HCB) and a Low-level Controller Board (LCB). The HCB, which is either an Nvidia Jetson Nano B01 or an Nvidia Jetson Orin Nano depending on the target ROS version, handles high-level processing, runs ROS, and interfaces with the Light Detection and Ranging (LiDAR) sensor. The LCB is a custom-designed microcontroller-based embedded board built, responsible for sensor interfacing and motor control. Communication between the two boards is established via Universal Asynchronous Receiver/Transmitter (UART) serial communication and a custom protocol.

This document serves as a manual for LMRO users. Chapter 2 gives an overview of the system; Chapter 3 covers robot assembly and mechanical design; Chapter 4 explains system identification and motor control; Chapter 5 details the LCB design, programming, and power management; Chapter 6 describes the HCB setup and robot software; and Chapter 7 provides procedures to run your Lenna robot. Each subsystem can also be used independently for other applications with proper precautions.

This manual assumes familiarity with ROS, Linux, Python, C/C++, embedded programming, and PCB design. The robot is intended for flat surfaces, as it has not been tested in outdoor or uneven environments.



(a) LMRO V1.0.0



(b) LMRO V2.0.0

Figure 1.1: Lenna Mobile Robot ONE

2 | System Overview

2.1 | Sensory-Motor Configuration

The designed robot is equipped with two Direct Current (DC) motors, which are employed for motion control. The rotational speed and relative position of each wheel are measured using incremental optical encoders mounted on the motor shafts. The data acquired from these sensors serve as inputs to the odometry and dead-reckoning subsystems. Although this approach provides effective short-term estimation of the robot's relative position, its accuracy deteriorates significantly over long trajectories due to error accumulation arising from measurement limitations and the simplified kinematic model. To achieve more precise localization, the robot is also equipped with an Inertial Measurement Unit (IMU) that integrates accelerometers, gyroscopes, and a magnetometer. Finally, to enable environmental perception, obstacle avoidance, and mapping, a LiDAR-based optical rangefinder is utilized.

Table 2.1: Robot Sensory-motor Configuration

Sensor/Actuator Type	Part Number	Description
Geared DC motor with encoder	JGA25-370	Nominal speed 300RPM, Gear ratio 1:34 and Incremental two-phase optical encoder
LiDAR distance sensor	LD19 LiDAR	360° LiDAR, range up to 12 m, accuracy
Inertial Measurement Unit	GY-87	Includes MPU6050 IMU and HMC5883L magnetometer
Ultrasonic Sensor	SRF-04	Maximum range of 3m for close-range obstacles

2.2 | Differential-Drive Robot Modeling

To describe the state of the robot and its equations of motion, two coordinate frames are required: (a) the inertial coordinate frame, which is fixed in the environment where the robot operates and serves as the reference plane. This frame is denoted by $\{\boldsymbol{X}_I, \boldsymbol{Y}_I\}$. (b) the robot coordinate frame, which is a local frame attached to the robot and moves along with it. This frame is denoted by $\{\boldsymbol{X}_R, \boldsymbol{Y}_R\}$.
The two defined frames, together with the robot's kinematic variables, are illustrated in 2.1.

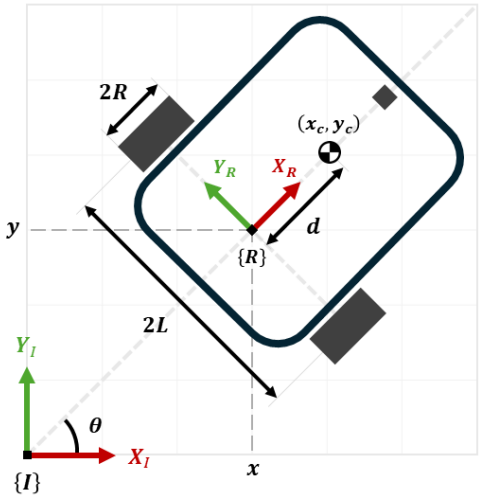


Figure 2.1: Differential-Drive Coordinate System for Modeling

In this model, $\boldsymbol{R}$ denotes the radius of the robot's wheels, and $\boldsymbol{L}$ represents the distance from the center of each wheel to the midpoint of the wheel axis. The origin of the robot coordinate frame is defined at this midpoint on the wheel axis. Moreover, it is assumed that the center of mass of the robot is located at a distance $\boldsymbol{d}$ from this point, along the robot's axis of symmetry. Consequently, the position and orientation of the robot in the reference coordinate frame are defined as follows:

$$\boldsymbol{q}^I = \begin{bmatrix} x^I \\ y^I \\ \theta^I \end{bmatrix} \tag{2.1}$$

The most critical aspect in formulating the kinematic and dynamic equations of the robot is to determine the mapping between the two coordinate frames. If any point in the inertial and robot coordinate frames is denoted respectively as $\boldsymbol{X}^I = \begin{bmatrix} x^I & y^I & \theta^I \end{bmatrix}^\top$ and $\boldsymbol{X}^R = \begin{bmatrix} x^R & y^R & \theta^R \end{bmatrix}^\top$, then these two state vectors are related to each other through a linear transformation.

$$\boldsymbol{X}^I = \boldsymbol{R}(\theta)\, \boldsymbol{X}^R \tag{2.2}$$

where $\boldsymbol{R}(\theta) \in \mathbb{SO}(3)$ is an orthogonal rotation matrix, computed as follows:

$$\boldsymbol{R}(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & \mathbf{0} \\ \sin\theta & \cos\theta & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix} \tag{2.3}$$

Ultimately, the equations of motion between the local and reference frames can be expressed solely through this transformation.

$$\dot{\boldsymbol{X}}^I = \boldsymbol{R}(\theta)\, \dot{\boldsymbol{X}}^R \tag{2.4}$$

The primary goal of kinematic modeling in this robot is to express the linear and angular velocities as functions of the wheel speeds and the robot's geometric parameters. In a differential-drive robot, the linear velocity is computed as the average of the linear velocities of the wheels, while the angular velocity is determined based on the difference between the wheel velocities.

$$v = \frac{v_R + v_L}{2} = \frac{R(\phi_R + \phi_L)}{2}, \tag{2.5}$$

$$\omega = \frac{v_R - v_L}{2L} = \frac{R(\phi_R - \phi_L)}{2L}, \tag{2.6}$$

Finally, the forward kinematic equations for a differential-drive wheeled robot are given by:

$$\dot{\boldsymbol{q}}^I = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix} \tag{2.7}$$

2.3 | Control Architecture

As previously mentioned, this robot employs a distributed architecture. In this architecture, robot control is divided into two layers: the hardware control layer (LCB) and the robot control layer (HCB). The hardware control layer is responsible for reading and filtering sensor data as well as controlling the motor dynamics. These computations are implemented on a real-time embedded system based on an ARM microcontroller from the STM32 family. In contrast, the robot control layer handles more complex computations and high-level decision-making. In this layer, sensors such as the LiDAR are processed and fused with the data from the lower layer. This layer is implemented on a NVIDIA Jetson Nano single-board computer using the ROS framework. It is noteworthy that the stability and synchronization of communication between these two layers are critical, and this is achieved via a serial connection.

A schematic overview of the hierarchical architecture is shown in Figure 2.2.

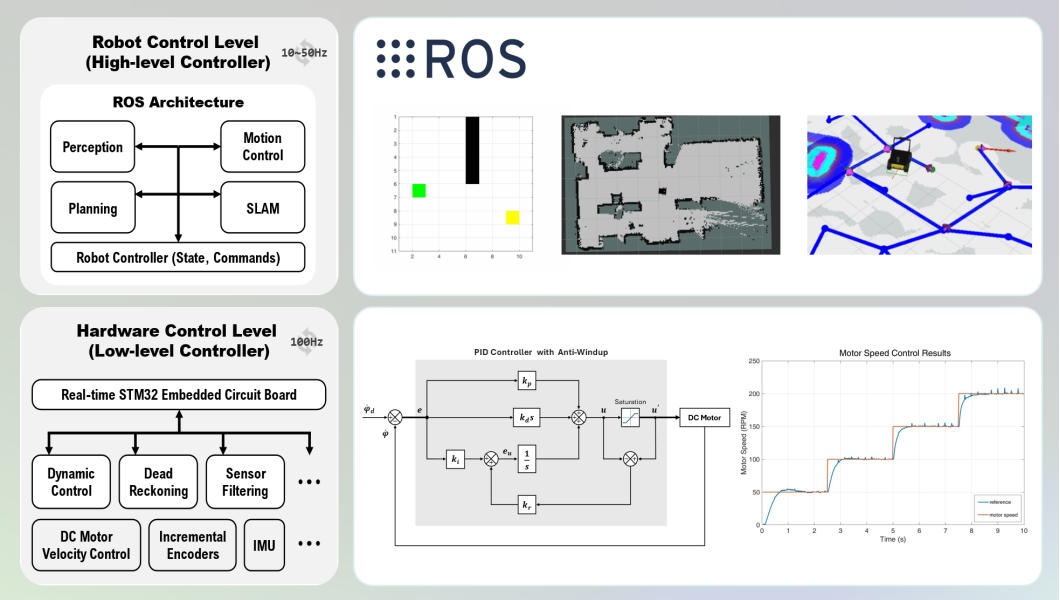


Figure 2.2: Lenna Mobile Robot Hierarchical Control Architecture

The physical architecture of Lenna Mobile Robot ONE, following this hierarchical architecture, is presented in Figure 2.3.

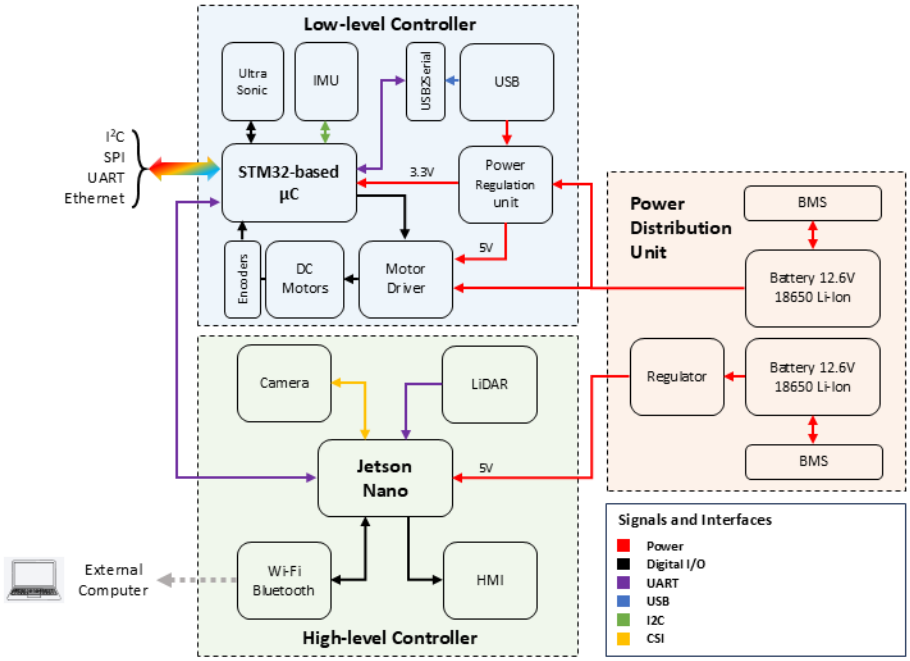


Figure 2.3: Lenna Mobile Robot Physical Architecture

3 | Mechanical Design

The mechanical design of an autonomous mobile robot is a critical stage in its overall development, as it establishes the physical platform upon which sensing, computation, and control systems are integrated. In this work, a differential drive mobile robot was developed following the unicycle kinematic model, which is one of the most widely adopted models for wheeled robotic systems. The design process was carried out in SolidWorks, with an emphasis on modularity, simplicity, and ease of future expansion.

All mechanical designs were developed in SolidWorks, and the outputs were exported in STL and DWG formats. These files are openly available for reproduction, modification, or integration with robotic simulation environments through Unified Robot Description Format (URDF) models. The complete CAD and manufacturing files are hosted on the project's GitHub repository.

The proposed robot architecture follows a three-level stacked structure, where each layer hosts a distinct set of components. This modular design allows rapid prototyping and straightforward replacement or modification of subsystems without interfering with the core mechanical or electrical functionalities.

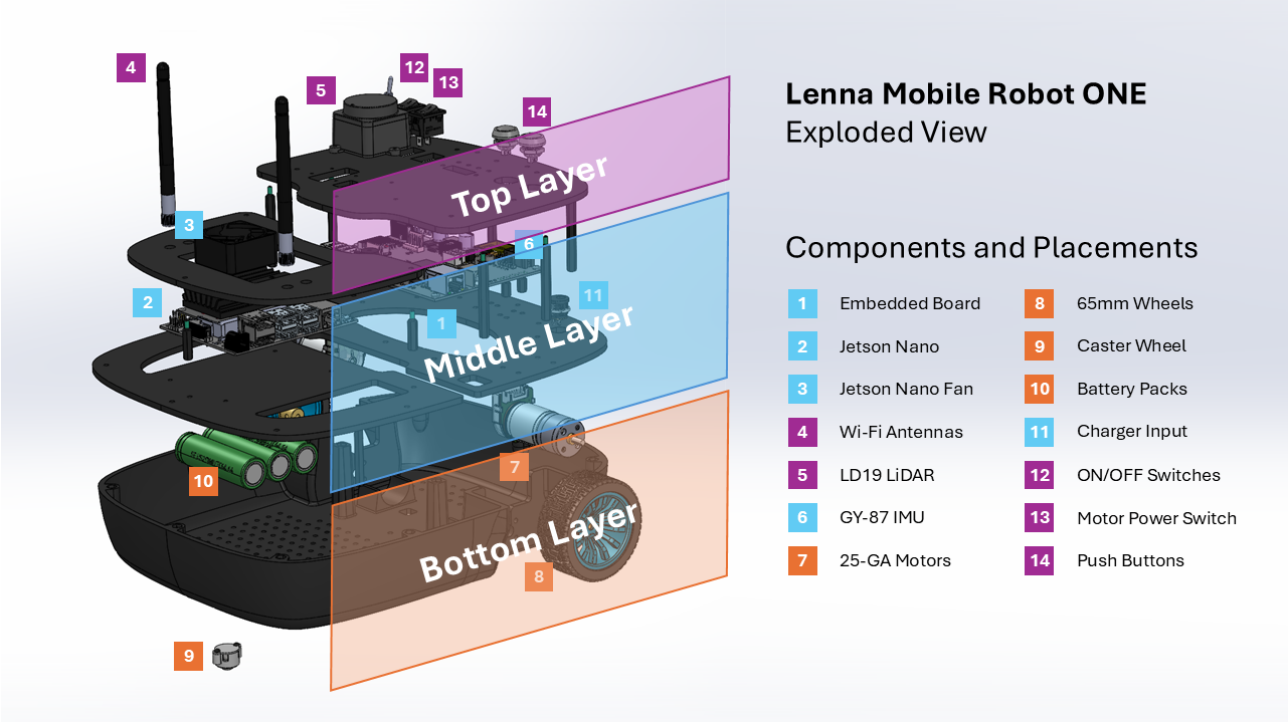


Figure 3.1: Lenna Mobile Robot ONE Revised Model Exploded View

3.1 | Design Concepts and Specifications

The robot structure consists of three mechanically independent yet vertically stacked levels. Each level is connected using M3 screws, spacers, and nuts, providing both rigidity and modularity. The overall dimensions of the robot are 30 cm × 25 cm × 18 cm, yielding a compact footprint suitable for indoor experimentation while leaving sufficient space for sensors, computation boards, and power systems.

The robot's locomotion is realized by a differential drive mechanism. Two independently actuated DC motors, each driving one wheel, generate linear and angular velocities according to the unicycle kinematics. The wheel radius is 65 mm, and the wheel separation is 190 mm, parameters that are critical for accurate motion modeling and control.

3.1.1 | Bottom Layer: Drive and Power Unit

The bottom level is the foundation of the robot, fabricated using PLA+ 3D-printed material to ensure a lightweight yet sufficiently strong base. This layer houses the DC motors, battery packs, and power management circuits. Given its fundamental role in supporting locomotion and power delivery, this level was designed to remain unchanged throughout the robot's lifecycle.

All external connections, such as charging ports and motor encoder cables, are routed to higher layers. This design choice minimizes mechanical wear on the base body and provides user-friendly access during operation and maintenance. The 3D-printed approach allows rapid customization while balancing cost-effectiveness with adequate structural integrity.

3.1.2 | Middle Layer: Electronics and Sensors Platform

The middle level is constructed using laser-cut plexiglass plates with pre-defined screw holes and mounting blocks. This layer accommodates the majority of the robot's electronic control boards, sensor modules, and associated cabling.

The choice of plexiglass for this level was driven by two factors:

1. Modularity – plexiglass plates are inexpensive and easy to fabricate, allowing rapid redesign or reconfiguration whenever new hardware needs to be integrated.
2. Transparency and accessibility – plexiglass provides a clear view of underlying components, facilitating debugging and maintenance.

By sandwiching electronic modules between upper and lower plexiglass plates, a neat and secure layout is achieved, reducing the risk of loose cabling and mechanical damage. The modular nature of this layer ensures that experimental setups can be easily adapted to different research tasks, such as navigation, mapping, or perception.

3.1.3 | Top Layer: Human-Machine Interfaces and Advanced Sensors

The top level integrates the robot's Human-Machine Interface (HMI) and high-level sensors. It is designed to provide the user with direct control and status feedback. Components on this level include:

- OLED display for system monitoring.
- Push buttons for mode selection or system reset.
- ON/OFF switches for power management.
- Status LEDs for visual feedback.
- SWD ST-Link programming connector for firmware flashing and debugging.
- LiDAR sensor for environment perception and navigation.

This layer thus bridges the gap between user interaction and robotic autonomy. Its layout ensures that all control and feedback mechanisms are accessible without interfering with the operation of the middle and bottom levels.

3.2 | Assembly and Modularity

The complete structure is assembled exclusively using M3 screws, spacers, and nuts, which simplifies both manufacturing and maintenance. The modular three-layer design offers distinct advantages:

- Subsystems can be replaced independently.
- Rapid prototyping is facilitated by reconfigurable middle and top layers.
- Maintenance and upgrades do not affect the stable drive and power system.

The modularity also provides a scalable platform for research and education, as new sensors, computation units, or experimental tools can be integrated without significant redesign.

3.3 | Conclusion

The proposed mechanical design establishes a robust, modular, and flexible platform for autonomous mobile robotics research. By separating the robot into three functional layers—drive and power, electronics and sensors, and HMI and advanced sensing—the design achieves a balance between mechanical stability and experimental versatility. Its compact form factor, differential drive mechanism, and openly available Computer Aided Design (CAD) resources make it a practical solution for both educational and research applications.

4 | Motor Controller Design

The LMRO includes a pre-designed PID controller for users who prefer to focus on managing the HCB without engaging in low-level embedded programming. Both the system identification of the DC motors and the controller design were carried out using MATLAB tools. The PID controller is tunable, allowing users to adjust its parameters either at the high-level through the HCB or directly at the low-level within the embedded software. This chapter presents the process of system identification and PID controller design for the robot's DC motors, while the following chapter describes the programming and implementation of the controller on the STM32 platform.

4.1 | System Identification

To design a model-based controller, it is first necessary to obtain an accurate representation of the motor dynamics through system identification. For this purpose, the LCB was connected to a laptop via USB, using the on-board UART interface and a USB-to-Serial driver. Experimental input–output data were collected in MATLAB through Simulink Desktop Real-Time (SLDRT). The applied input was the motor drive voltage command (PWM duty cycle), while the measured output was the actual motor speed, calculated on the LCB from the encoder signals and transmitted via UART. Using MATLAB's System Identification Toolbox and the `ident` app, the motor dynamics were approximated by a discrete-time second-order transfer function with a sample time of 0.01 s, shown in (4.1). Figure 4.1 illustrates the open-loop step response of the identified model, which exhibits a relatively slow response and a steady-state error.

$$G(z) = \frac{0.02517}{1 - 1.56z^{-1} + 0.5847z^{-2}} \tag{4.1}$$

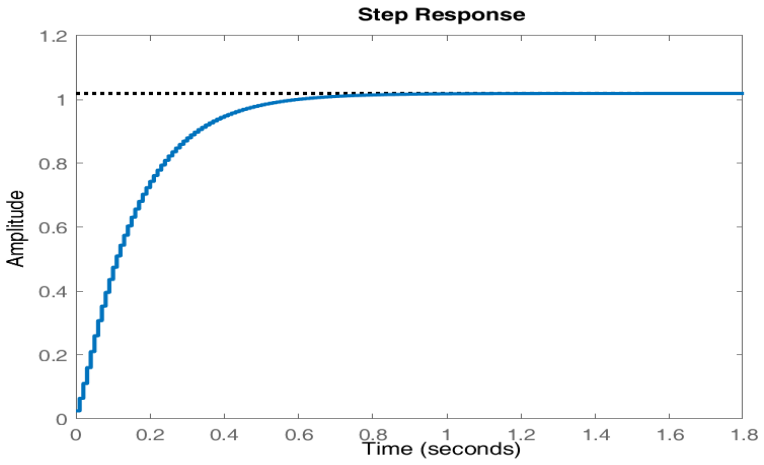


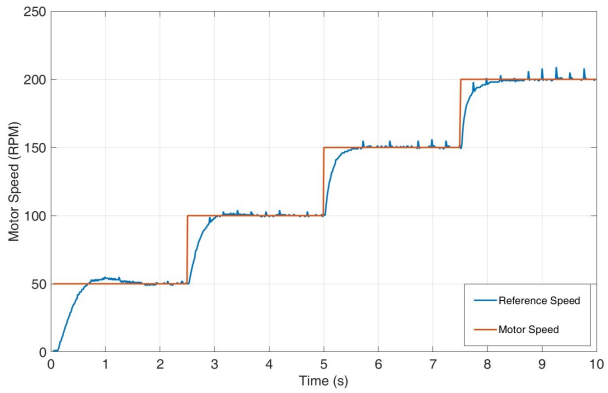
Figure 4.1: Step Response of the Identified Transfer Function

4.2 | Controller Design

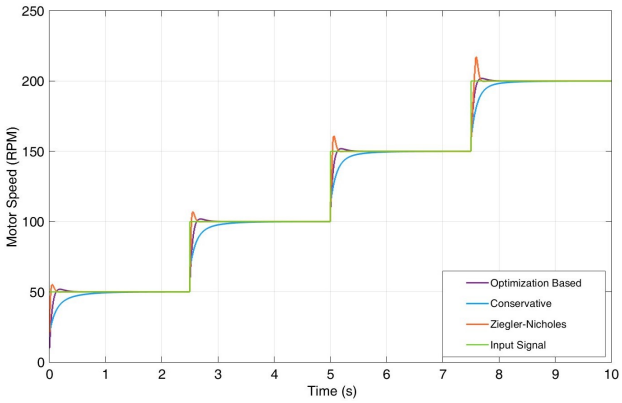
The identified transfer function was used in MATLAB's `sisotool` to design a controller. Both the Ziegler–Nichols tuning method and the optimization-based tuning features of `sisotool` were explored. To address potential issues of integrator windup, an anti-windup mechanism was incorporated into the controller with adjustments for each motor. The performance of the simulated controller is shown in Figure 4.2b. The final controller was then validated through hardware-in-the-loop Hardware In the Loop (HIL) testing, with the experimental results presented in Figure 4.2a. The tuned PID parameters are summarized in Table 4.1.

Table 4.1: Designed PID Parameters for Left and Right Motors

Method	$K_{p,L}$	$K_{i,L}$	$K_{d,L}$	$K_{p,R}$	$K_{i,R}$	$K_{d,R}$
Ziegler–Nichols	7.2	72	0.18	7.2	72	0.16
Optimization-based	1	3	0.05	1.2	4.6	0.01



(a) Real-world



(b) Simulation

Figure 4.2: Real-world and Simulation Results of the Motor Controller

5 | Low-level Controller Board

This section begins with an overview of the hardware of the LCB, including the Printed Circuit Board (PCB) design, the selection of key modules, and the power management and requirements of the LCB. Moreover, the motivation behind certain design choices is discussed to provide context for why specific components were selected. Finally, the section presents the software and programming aspects, along with detailed explanations of the modules and their integration with the rest of the system. This combination of hardware and software perspectives is intended to give the reader a complete picture of the role of the LCB within the overall LMRO architecture. The final board design is illustrated in Figure 5.1.

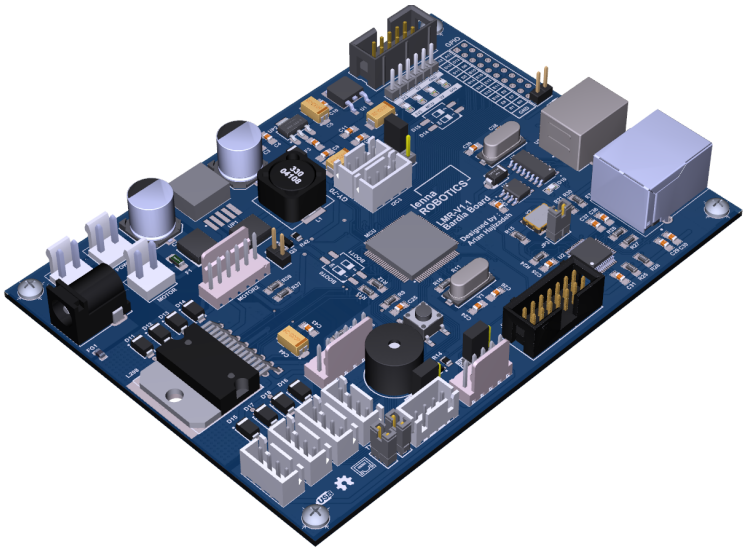


Figure 5.1: Lenna Bardia Board

5.1 | Board Design

As mentioned earlier, the Lenna Robotics team has designed a custom PCB for the LCB, referred to as the Bardia Board. The complete design is available on the Lenna Robotics GitHub page, and the following sections provide details on the design choices, the considerations that shaped them, and the resulting component selection.

A natural question is why a custom board was developed instead of relying on Commercial Of The Shelf (COTS) development boards commonly available from various vendors. While such boards could be used as an alternative, the decision to design a dedicated controller board was motivated by several requirements: the need for specific interfaces, simplified wiring, an integrated motor driver, and careful treatment of electrical noise. These considerations made a custom solution both more practical and more reliable for this platform than a general-purpose COTS board, and they continue to define the robustness of the current architecture.

Overall features of the Bardia Board are as follows:

- STM32F407VGT6 microcontroller as the core processor
- On-board L298 DC motor driver
- USB Type-B interface with CH340G USB-to-Serial driver
- Support for simultaneous USB and battery power
- Dedicated ADC channel for battery-level monitoring
- Dedicated fan connector
- Expansion ports: 1× I^2C bus, 1× UART, 1× SPI, 20× GPIO (with 1× CAN available)
- Four dedicated SRF-04 ultrasonic sensor slots
- Ethernet connectivity via DP83848 physical layer
- Dedicated ST-Link programming/debugging port
- A buzzer provided for alarming purposes
- A dedicated slot for GY-87 IMU

The final design is a 13 cm × 10 cm controller board developed in Altium Designer. At its core, the board uses an STM32F407VGT6 Micro Controller Unit (MCU), selected for its balanced performance, broad availability in the Iranian market, and extensive set of peripherals and I/O interfaces. The board includes an on-board L298 DC motor driver to interface two JGA-25 DC motors. Because motor drivers can inject electrical noise, grounding and power return paths were treated carefully: the motor-driver ground is tied to the logic/system ground at a single point through a ferrite bead to reduce noise coupling. In addition, the L298 is mounted on a stitched thermal pad (without soldermask overlay) to improve heat dissipation.

USB–serial connectivity is provided by a CH340G bridge. As the USB port presents a separate 5 V source, particular care was taken to avoid conflicts with the battery supply. Power-path OR-ing between the USB input and the battery rail is implemented using an op-amp–based control circuit. For the battery, a dedicated BMS provides overcharge, over-discharge, and short-circuit protection. One MCU ADC channel is reserved for battery-voltage monitoring to trigger a low-voltage alarm. A dedicated fan connector is also included for optional active cooling when required.

For communications and expansion, specific interfaces are reserved for system functions while additional ports remain available to the user. A UART link connects the HCB and LCB (used with ROS Serial), the I^2C bus services the onboard IMU, and further interfaces (SPI, GPIO, and a CAN-capable pin set) are exposed for peripherals. Ethernet connectivity is also available via an onboard PHY (DP83848) for future or user-defined networking needs. An ST-Link programming/debug header is provided for low-level firmware

development and updates, and an onboard buzzer offers audible feedback—primarily for fault and low-battery alarms. Figure 2.2 shows the placement of the major components on the board, and pinouts for user-accessible connectors are provided in subsequent sections.

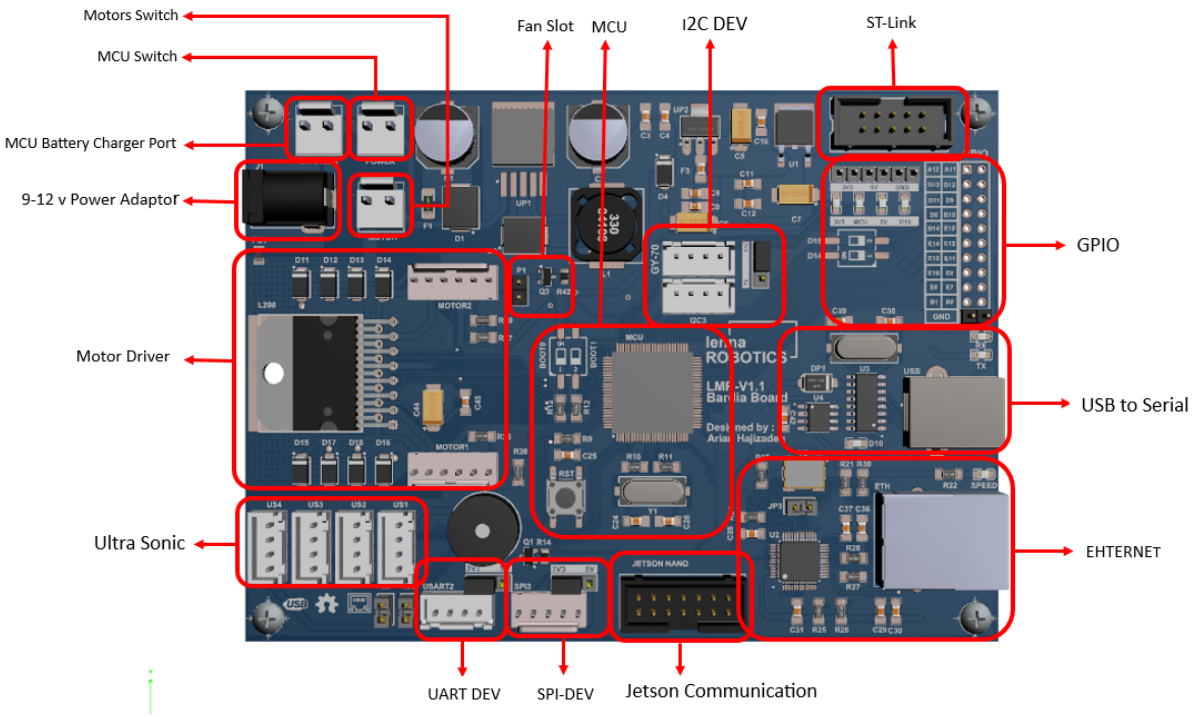


Figure 5.2: Component Placement of the Board

5.2 | Power Management

As noted in Chapter 2, the LCB is powered by a dedicated 12 V battery pack that supplies both the board and the DC motors. The 12 V rail feeds the on-board motor driver, which drives the motors, and also powers an LM2576 step-down switching regulator that generates a stable 5 V output. This 5 V rail supplies the USB and buzzer interfaces, all other 5 V peripherals on the board, and two separate 3.3 V regulators. To manage the 5 V input safely, an op-amp–based OR-ing circuit is used to prioritize the main supply over USB power. When a 12 V supply is connected, the board operates from the regulated 5 V generated from the main supply. If the board is powered only via USB, the 5 V peripherals and logic remain operational from the USB port, while the motors are disabled. For this reason, the inputs of the 3.3 V regulators are derived from the 5 V rail rather than directly from 12 V, ensuring that the logic can still operate under USB power alone. The use of two 3.3 V regulators, instead of a single source, provides redundancy: if the peripheral 3.3 V supply is damaged or overloaded, the MCU remains powered from its dedicated regulator, protecting the core functionality of the system. In this design, the LF33-CDT linear regulator powers the 3.3 V peripheral rail, while the AMS1117 Low Drop Out (LDO) regulator exclusively powers the MCU.

5.3 | Programming Modules

This subsection is organized into several parts. It begins with an explanation of the overall software structure and the programming conventions adopted by Lenna Robotics. Then, the subsection concludes with a brief review of the individual code modules. The complete code modules can be found in Lenna Robotics GitHub page. It is tried that sufficient comments are written in each module to help the user and address possible complexities.

5.3.1 | Standard and Convention

It is intended throughout our code that there is a unified convention on naming and definitions. All of the programming for this section is done in embedded C\C++ in STM32CubeIDE. To make it easier for the user to understand the provided code, our naming convention is demonstrated in this section.

variables

Normal variables are defined in Snake Case

```
#Define THIS_IS_A_MACRO     MACRO   \\ macros are defined in Screaming Snake Case

int this_is_a_variable =    0       \\ variables are defined in Snake Case

void LRL_{Component}_{FunctionalityOfComponent}()
{
\\ functions are defined in a custom form
    int _private = 1 \\ variables are defined
}
```

Packet Handler

This module handles serial communication between the microcontroller and an external device (e.g., a PC). It implements a robust communication protocol with a defined packet structure, a handshake mechanism, and CRC validation for data integrity.

Files: `packet_handler.c`, `packet_handler.h`

- **LRL_Packet_Init**: Initializes the UART receive interrupt.
- **LRL_Packet_Handshake**: Establishes a connection by sending a signature message and waiting for an acknowledgment.
- **LRL_Packet_UpdateCRC**: Calculates the CRC-16 checksum for a data block.
- **LRL_Packet_TX**: Assembles and transmits a data packet containing sensor and odometry data.
- **LRL_Packet_RX**: Receives an incoming packet, validates its CRC, and parses the contained data.

Motion Control

This module contains the low-level functions for controlling the robot's motors. It translates high-level commands into PWM signals for motor speed and GPIO states for direction. It provides functions to control individual motors as well as a complete differential drive robot.

Files: `motion.c`, `motion.h`

- **LRL_Motion_MotorSpeed**: Sets the speed and direction of a single motor using a duty cycle value.
- **LRL_Motion_Control**: A high-level function that controls both the left and right motors for differential steering.
- **LRL_Motion_MotorTest**: Executes a pre-programmed sequence of movements for debugging and testing.

PID Controller

This module provides a Proportional-Integral-Derivative (PID) controller implementation for closed-loop control, primarily for motor speed. It includes functions for initializing the controller, updating the control signal based on a set point and measurement, and includes advanced features like anti-windup and output saturation to ensure stable and safe operation.

Files: `pid.c`, `pid.h`

- **LRL_PID_Init**: Initializes and resets the PID controller's parameters.
- **LRL_PID_Update**: Calculates the error and computes the P, I, and D terms to produce a control signal, applying anti-windup and saturation.

Odometry

This module implements the odometry system for a differential drive robot. It uses encoder tick counts from the motors to estimate the robot's motion. The code correctly handles timer overflows and underflows, calculates wheel speeds in RPM, and computes incremental distances traveled.

Files: `odometry.c`, `odometry.h`

- **LRL_Odometry_Init**: Resets the encoder counters and initializes odometry variables.
- **LRL_Odometry_ReadAngularSpeed**: Reads encoder ticks, calculates the tick difference, accounts for direction and overflows, and converts the values into wheel speed and distance.

IMU

This module handles communication with an MPU-6050 IMU and an HMC5883L magnetometer via I2C. It reads raw data from both sensors, applies calibration offsets, and fuses the accelerometer and gyroscope data using a complementary filter to provide a stable orientation (roll, pitch, and yaw). It also includes magnetic declination correction for the heading.

Files: `imu.c`, `imu.h`

- **LRL_IMU_MagInit**: Initializes the magnetometer and performs a heading calibration.
- **LRL_IMU_MagSetDeclination**: Calculates the magnetic declination angle for a specific location.
- **LRL_IMU_MagReadHeading**: Reads magnetic data, calculates the heading, and applies corrections.
- **LRL_IMU_MPUInit**: Initializes the MPU-6050 and calibrates the accelerometer and gyroscope offsets.
- **LRL_IMU_ReadAccel**: Reads and scales accelerometer data.
- **LRL_IMU_ReadGyro**: Reads and scales gyroscope data.
- **LRL_IMU_MPUReadAll**: Reads both accelerometer and gyroscope data.
- **_LRL_IMU_MPUBypassEn**: Toggles I2C bypass mode to access the magnetometer.
- **LRL_IMU_ComplementaryFilter**: Combines IMU sensor data to estimate orientation.

Ultrasonic Sensor

This module is responsible for interfacing with a single HC-SRF04 ultrasonic sensor to measure distance. It uses a hardware timer in Input Capture mode to precisely measure the width of the echo pulse, which is then converted into distance.

Files: `ultrasonic.c`, `ultrasonic.h`

- **LRL_US_Init**: Initializes the timer and input capture interrupts.
- **LRL_US_Trig**: Generates the 15 µs trigger pulse to the sensor.
- **LRL_US_TMR_OVF_ISR**: Increments an overflow counter when the timer wraps around.
- **LRL_US_TMR_IC_ISR**: The main interrupt handler for input capture. It measures the high pulse duration and calculates the distance.
- **LRL_US_Read**: Returns the last measured distance.

This module acts as a central repository for all hardware-related definitions. It uses preprocessor macros to define GPIO pin assignments, timer counters, and other peripheral registers. This approach simplifies code maintenance and improves readability by giving meaningful names to hardware pins and registers.

MCU Layout

Files: `mcu_layout.h`

- **No Functions**: This is a header-only module that contains only definitions and macros, not executable functions.

6 | High-level Controller Board

As previously introduced, the Lenna Mobile Robot ONE adopts a hierarchical control architecture. All high-level perception, decision-making, and planning are handled at the HCB, which is powered by an NVIDIA Jetson Nano. To ensure modularity and reusability, the development effort focused on implementing all drivers and hardware abstractions within the ROS framework.
This chapter presents the software architecture designed around ROS Melodic, which provides a robust communication layer between the HCB and the LCB.

6.1 | ROS Packages

6.1.1 | lenna_bringup

The lenna_bringup package is responsible for handling all hardware-specific drivers for the Lenna Mobile Robot. It opens a serial port to establish communication with Lenna's low-level controller boards. This package isolates hardware interaction nodes from the high-level control nodes that are responsible for robot control algorithms (e.g., SLAM, navigation, path planning, etc.). It must be launched before any other program, since it initializes all the low-level nodes required for reliable communication with the robot hardware. We will shortly describe the ROS nodes provided by lenna_bringup in the following sections.

serial_handshake_node.py

A ROS node responsible for establishing communication over a serial connection. It listens for a predefined handshake keyword from the robot, sends back a confirmation byte if successful, and publishes the handshake status as a `std_msgs/Bool` message on the `/handshake` topic. Other nodes (e.g., odometry publishers and `cmd_vel` subscribers) rely on this handshake to ensure the robot is ready before operation.

serial_odom_node.py

A ROS node that reads wheel encoder data via a serial connection and publishes the robot's odometry as a `nav_msgs/Odometry` message. It performs dead reckoning to estimate the robot's pose (x, y, theta) and velocity (v, w), and broadcasts this information to other ROS components. A handshake mechanism is used to ensure proper initialization before odometry data is published.

serial_cmd_node.py

A ROS node that listens to velocity commands (`geometry_msgs/Twist`) on the `/cmd_vel` topic and converts them into motor commands for the Lenna mobile robot. Using the robot's kinematic model, it calculates left and right wheel speeds, converts them into RPM, clamps them to allowed motor limits, and sends the values over a serial connection. The node requires a successful handshake before executing commands.

odom_tf_node.py

A ROS node that listens to odometry data (`/odom`) and broadcasts the corresponding TF transform between two coordinate frames (default: `odom` → `base_link`). This allows other ROS nodes (e.g., navigation or visualization tools like RViz) to use the robot's odometry as a transform tree in the ROS TF system.

6.1.2 | lenna_description

The lenna_description package provides the core resources and configurations required to simulate and operate the LMRO within the ROS ecosystem. It contains the robot's URDF model, which defines its physical structure, sensors, and actuators for visualization and simulation. The package also includes Gazebo simulation worlds that allow testing in different environments, along with essential nodes for tele-operation, SLAM, and navigation. These components enable users to manually control the robot, perform autonomous mapping, and execute path planning tasks, making the package a comprehensive foundation for experimenting with and extending the Lenna robot's capabilities.

7 | Plug and Play

This section provides step-by-step procedures to make your own ROS workspace and run the nodes required to work with Lenna Mobile Robot.

7.1 | ROS Workspace

Always source your ROS distribution:

```
$ source /opt/ros/<distro>/setup.bash
```

Create your ROS workspace and add the mentioned lenna packages to the `/src` directory.

```
$ mkdir -p ~/lenna_ws/src
$ cd ~/lenna_ws
$ catkin build
```

Source the compiled workspace:

```
$ source ~/lenna_ws/devel/setup.bash
```

7.2 | ROS Launch

7.2.1 | Tele-operation

you can use ROS package `teleop_twist_keyboard` to control LMRO. using the `lenna_teleop_keyboard` lauch file inside `lenna_bringup` package:

```
$ roslaunch lenna_bringup lenna_teleop_keyboard.launch
```

7.2.2 | SLAM

Both `slam_toolbox` and `hector_slam` can be used for LMRO:

```
$ roslaunch lenna_bringup lenna_slam_toolbox.launch
```

or

```
$ roslaunch lenna_bringup lenna_slam_hector.launch
```

7.2.3 | Autonomous Navigation

Using the `lenna_navigation_stack` launch file, you can use the ROS NavStack `move_base` and amcl packages to implement autonomous navigation.

```
$ roslaunch lenna_bringup lenna_navigation_stack.launch
```


GitHub