elektron ámbar

Original Article

Revista elektron, Vol. 7, No. 2, pp. 53-60 (2023)
https://doi.org/10.37537/rev.elektron.7.2.184.2023

Differential-drive Mobile Robot Controller with ROS 2 Support

Controlador para Robot Móvil de Tracción Diferencial Compatible con ROS 2

Gustavo Albarrán[†1], Juan Nicolodi[†], Dante Ruiz[†2], Diego González-Dondo[†3] and Gonzalo Perez-Paina[†4]

[†] *Centro de Investigación en Informática para la Ingeniería (CIII),*
Facultad Regional Córdoba de la Universidad Tecnológica Nacional (UTN-FRC)
Maestro López esq. Cruz Roja Argentina, X5016ZAA Córdoba, Argentina
[1]galbarran@frc.utn.edu.ar
[2]druiz@frc.utn.edu.ar
[3]dgonzalezdondo@frc.utn.edu.ar
[4]gperez@frc.utn.edu.ar

Recibido: 02/10/23; Aceptado: 06/12/23

***Abstract*—Autonomous Mobile Robots, known as AMRs, are used in the internal logistics of many types of industries and production sectors. This type of robots replaces the traditional Automated Guided Vehicles (AGVs). In the case of AGVs, the path to follow is previously defined, and these robots do not have the ability to choose a different path. On the other hand, AMRs are more flexible, safe, and precise, due to the incorporation of technologies reserved until recently for research, such as autonomous navigation, computer vision systems, and Simultaneous Localization and Mapping (SLAM) technology, among others. Many of these technologies are implemented using the Robot Operating System (ROS). ROS is a set of free and open-source software libraries and tools for building robot applications. Its new version, ROS 2, was developed to be applied to production environments. This paper describes the development of a controller for a differential-drive AMR with support for ROS 2 using its implementation for embedded systems, micro-ROS. This controller is the evolution of a previous version that was used in different mobile robots for over 10 years at CIII (UTN). It is worth clarifying that this work is mainly focused on hardware development. However, some preliminary software tests have been carried out, mainly to evaluate the correct functioning of the differential-drive robot controller. Firstly, the design requirements are defined, and a microcontroller with native support for micro-ROS is selected. Then, the development of each controller stage is described, such as the power supply, the USB communication, the battery voltage sensing, the debugging port, and the final PCB design. Finally, the initial software tests that allow verifying the correct operation of the controller and the improvements compared to the previous version are mentioned.**

Keywords: autonomous mobile robot; differential drive; embedded controller; ROS 2; micro-ROS

***Resumen*—Los robots conocidos con el nombre de AMR (Autonomous Mobile Robots) se utilizan en la logística interna en muchos tipos de industrias y sectores de la producción. Este tipo de robots sustituyen a los tradicionales AGVs (Automated Guided Vehicles) en los cuales el camino a seguir está definido previamente y no tienen la capacidad de elegir un camino diferente. Por otro lado, los AMRs resultan más flexibles, seguros y precisos, debido a la incorporación de tecnologías que hasta hace poco estaban reservadas al ámbito de la investigación, tales como: navegación autónoma, sistemas de visión por computadoras, tecnología de SLAM (Simultaneous Localization and Mapping), entre otras. Muchas de estas tecnologías se implementan utilizando ROS (Robot Operating System). ROS es un conjunto de bibliotecas de software y herramientas de código abierto y libre para el desarrollo de aplicaciones de robots, cuya nueva versión ROS 2 tiene como uno de sus objetivos ser aplicable a entornos de producción. El presente trabajo describe el desarrollo de un controlador para robots de tracción diferencial tipo AMR con soporte para ROS 2 utilizando la implementación para sistemas embebidos micro-ROS. Este controlador es la evolución de una versión anterior utilizada en diferentes robots por más de 10 años en el CIII (UTN). Vale aclarar que este trabajo está enfocado principalmente en el desarrollo de hardware. Sin embargo, se han realizado algunas pruebas preliminares de software, principalmente, para evaluar el correcto funcionamiento del controlador de tracción diferencial. En primer lugar, se definen los requerimientos de diseño y se selecciona un microcontrolador con soporte nativo para micro-ROS. Luego se describe el desarrollo de cada etapa del controlador, tales como: la alimentación, la comunicación USB, el sensado de tensión de batería, el puerto de depuración y el diseño final del PCB. Por último, se hace mención a las pruebas iniciales de software que permiten verificar el correcto funcionamiento del controlador y las mejoras respecto a la versión anterior.**

Palabras clave: robot móvil autónomo; tracción diferencial; controlador embebido; ROS 2; micro-ROS

I. Introduction

Automated Guided Vehicles (AGVs) [1] are a key component in the internal logistics in many types of industries and production sectors in their implementation of Flexible Manufacturing Systems (FMSs) [2]. AGVs provide the ability to move products and parts efficiently in relation to handling time. The first AGVs were guided by electrical conductors that were mounted on the ground, known as inductive guidance [3]. Currently, the navigation [4] of AGVs is based on magnetic and laser scanning sensors for safety reasons [5]. However, the routes or paths to follow are previously defined and the AGV does not have the ability to choose a different path.

Recently, there have been important advances in autonomous vehicles and their application as a service robotics platform [6] known as Autonomous Mobile Robot (AMR) or Autonomous Intelligent Vehicle (AIV), mainly focused on promoting the flexibility of relocation of tasks within factories and boosting the implementation of Industry 4.0

technologies. AMRs are more flexible than AGVs in respect of programming and configuring the tasks they have to carry out and the ability to work precisely and collaboratively with human workers or other automatic platforms. Their ability to adapt to different navigation scenarios can be applied to reduce the kilometers that plant personnel cover every day pushing carts to distribute products [7]. AMRs are capable of moving from one place to another safely and without human intervention, applying different technologies that until a few years ago were reserved only for the academic and research field [7], [8], such as: a) autonomous navigation [9], map creation [10] c) 2D laser scanning sensors, d) computer vision systems [11], e) Simultaneous Localization and Mapping (SLAM) technology [12], among others. Many of these technologies are implemented using ROS [13].

ROS is a set of free and open-source software libraries and tools (process monitoring, communication introspection, visualization, etc.) to build robot applications [14], [15]. The ROS project began in 2007 at Stanford University under the name Switchyard, and from 2008 onwards it was run by a robotics research company called Willow Garage, which developed most of the libraries and tools. In 2013, Willow Garage researchers formed the Open Source Robotics Foundation (OSRF), which is currently in charge of maintaining ROS.

The core of ROS is the message-passing middleware for the communication between processes that allows data exchange even when it is run from different computers. ROS also provides a hardware abstraction layer on which developers can build robotics applications without worrying about the underlying hardware. Using the hardware abstraction layer and this message-passing middleware, different robotic capabilities can be created, such as mapping, localization, navigation, etc.; which are generally agnostic to the robot.

While ROS solves many of the complex problems inherent in robotics, it also has some shortcomings. One of its main limitations is that it was not developed as production software. In addition, it has problems when working with data networks with connection loss (e.g. in WiFi networks); it has a main point of failure which is the ROS Master; it does not incorporate any network security mechanism; it is not natively supported to embedded systems; etc. These limitations gave rise to the ROS 2 [16] [17] project. ROS 2 was built as a parallel set of packages that can be installed alongside and interoperate with ROS 1 (for example, via message bridges) [18].

The following use cases were taken into consideration in the design of ROS 2: *i)* multi-robot teams, *ii)* application in embedded systems, *iii)* real-time systems, *iv)* non-ideal networks, *v)* production environment, and *vi)* prescribed patterns for building and structuring systems.

It is important to note here that the architecture of a robotic system generally includes a network of one or more microprocessors with high computing power and multiple microcontrollers. The latter are processing units with low performance and computing capabilities, generally used to access sensors and actuators, for low latency control functions, to save energy (particularly in consumer applications), and for security functions. This is why the second use case considered is of great relevance, which allows microcontrollers to be included as "first-class participants" in the ROS environment, instead of being segregated to interact through a device driver. This implies that robotics software must be developed on microcontrollers using the same ROS concepts as on powerful microprocessors (CPUs). This demands that the software on the microcontroller (μC) must be accessible with the same development tools used in CPUs to carry out introspection, monitoring, and configuration tasks at runtime.

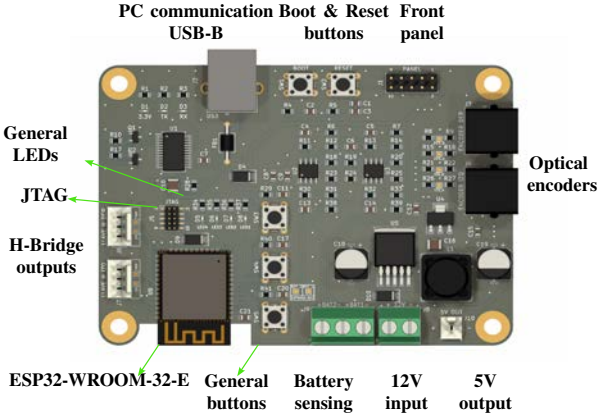


Fig. 1: New design of the differential-drive robot controller (DDRC-ESP32) board with ROS 2 support.

ROS 1 is built on communication middleware created specifically by ROS developers. Its functionality involves node discovery, definition, serialization, and communication of messages, etc. On the other hand, to cover this functionality, ROS 2 adopts the DDS (Data Distribution Service) standard [19], which is an open standard for communications used in critical infrastructures, such as military, space, and financial systems. DDS allows ROS 2 to obtain: *i)* an improvement in the security and integrity of information, *ii)* support for embedded and real-time systems, *iii)* communication between multiple robots, and *iv)* operations in non-ideal network environments.

In order to integrate ROS 2 into embedded devices with low hardware resources, the micro-ROS project was developed [20]. This considers the following objectives: *i)* seamless integration of microcontrollers with ROS 2, *ii)* easy portability of ROS 2 code to microcontrollers, and *iii)* ensure long-term maintenance of the micro-ROS stack. To achieve these goals, the founding partners of micro-ROS designed a software stack that uses the layered architecture of the standard ROS 2 stack and integrates seamlessly with DDS. The micro-ROS stack reuses as many packages as possible of the standard ROS 2 stack. On the middleware layer, it uses an open-source implementation of the eProsima's eXtremely Resource Constrained Environments DDS (XRCE-DDS) standard, named Micro XRCE-DDS [21]. On the client library layer, micro-ROS extends the `rcl` (ROS 2 Client Library) using the `rclc` (ROS Client Library for C language) packages to form a feature-complete client library in C.

The present work describes the design of a control board for a differential-drive mobile robot compatible with ROS 2 through micro-ROS (see Fig. 1). The project is license

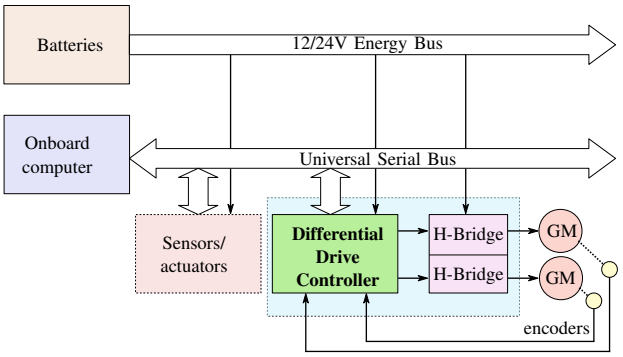


Fig. 2: Block diagram of a differential-drive robot.

free and is available in a public repository [22]. This development is the evolution of a previous design that did not have native support for ROS [23]. The purpose of having an embedded controller with support for micro-ROS is to serve as a basis for the development of AMR-type robots where their autonomous navigation and control algorithms are developed entirely using ROS 2. It is worth mentioning that the development was focused mainly on the hardware design, although some programs were carried out to verify the correct operation of the board.

The article is organized as follows: Section II describes the architecture of a differential-drive robot and its main characteristics, together with the embedded controller for differential-drive robots used until now at the CIII research center, which was replaced by the controller presented in the following section. Section III describes the development of the new controller for differential-drive robots, including the design requirements, the selection of components, a general description of the controller and its main characteristics, and the tests carried out to verify the correct functioning. Finally, section IV mentions the conclusions and future work.

II. Differential-drive Mobile Robot

A differential-drive mobile robot, also known as a unicycle, has two independently-controlled drive wheels along with one or more non-drive wheels that serve as support. This architecture is appropriate for indoor environments since it can rotate around its odometric center, which means a change of orientation without displacement.

A. Robot architecture

Fig. 2 shows the block diagram of a differential-drive robot. The gearmotors and incremental optical encoders that serve to measure the rotation speeds of each wheel can be observed. The differential-drive controller (indicated in the light blue box) operates in conjunction with two power boards in H-bridge configuration for the excitation of each of the gearmotors (left and right). The figure also shows the energy system composed of batteries and the onboard computer to process the information from the sensors and to implement the robot's autonomous navigation algorithms.

This type of robot is controlled by linear speed v and angular speed ω commands, expressed in a local coordinate system. From these commands, the robot's internal controller generates the angular velocity of each of the wheels. On the other hand, the information obtained from the incremental optical encoders coupled to the drive wheels is used to perform the odometry calculation. Odometry is a method for estimating the robot's pose given by the rotation of the wheels, being the robot's pose its position and orientation in the operating plane; that is, (x, y, θ).

B. Embedded controller

Fig. 3 shows a previous development of a controller for a differential-drive robot.

The main functions of this controller are:

1) Communicating with the robot's onboard PC to receive speed commands and communicate the robot's states.
2) Reading the information from the optical encoders coupled to the drive wheels.
3) Adjusting the speeds of the drive motors using PID controllers to meet the desired setpoints for the linear and angular speeds of the robot.
4) Performing the odometry calculation with the information from the incremental optical encoders coupled to the wheels.

This version of the controller is currently being used by the RoMAA-II robot and the AMR Aimu robot (see Fig. 4), both with differential drive, developed at the Research Center on Informatics for Engineering (CIII) from National Technological University (UTN).

The Open Architecture Mobile Robot, named RoMAA, was developed as a platform to carry out experiments in the research areas of mobile robotics and computer vision. Its development began as an internal CIII project and concluded with the manufacturing of the first prototype presented in [24], on which the constructive, functional and manufacturing cost characteristics were evaluated. With the experience acquired, the new and current version, named RoMAA-II, was designed [25] [26] [27]. On the other hand, the Aimu robot is an AMR-type robot developed through a specific agreement between the UTN-FRC and the company Caima-Segal S.R.L. [28]. Aimu has been operational for three years in the Denso Manufacturing company in the city of Córdoba and has been recently replaced by a new version.

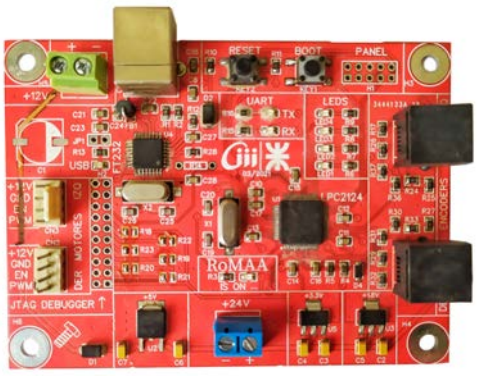


Fig. 3: Old version of the differential-drive robot controller based on a μC without ROS 2 support.

(a) RoMAA-II research robot.

(b) Aimu industrial AMR.

Fig. 4: Robots that use the controller.

C. Using ROS with the current driver

RoMAA-II and Aimu AMRs (Fig. 4) use ROS through a node [29] which acts as a device driver for the embedded controller used (Fig. 3). This driver allows the autonomous navigation programs of these robots to be developed using ROS and thus to take advantage of the large number of software packages available in the ROS ecosystem. In addition, this environment can be used as a development and evaluation tool for the generation of new robotics algorithms. The driver node is responsible for translating the high-level information managed by ROS (messages) into low-level commands of the robot's embedded control system [30].

It is worth mentioning that the current version of the controller is based on the NXP *LPC2114* microcontroller, which has a 32-bit ARM7TDMI-S processor. This microcontroller is not recommended for new developments and is not supported by micro-ROS. To overcome these drawbacks it is necessary to have a new controller based on a more modern microcontroller and supported by micro-ROS to be able to run applications based on ROS 2.

III. Development of the Mobile Robot Controller

This section details the design of the new controller for differential-drive robots.

A. Design requirements

One of the main requirements of the new design was to maintain the same Printed Circuit Board (PCB) size and the same type and layout of connectors for direct replacement of the previous controller. Moreover, some additional improvements were required, such as incorporating battery status sensing and reducing overall energy consumption, which are described later.

In relation to the processing capacity and number of available peripherals, the μC to be selected must meet the following characteristics:

- Support micro-ROS.
- Ability to drive two Pulse Width Modulation (PWM) outputs.
- Pulse reading for incremental optical encoders.
- Analog to Digital Converter (ADC).
- Universal Serial Bus (USB) or Universal Asynchronous Receiver-Transmitter (UART) communication module.

The micro-ROS is supported by some mid-range 32-bit microcontroller families and the official website [31], until August 2023, provides a list of different development boards supported. From this list, the Espressif family was chosen because it is suitable for this application and is available in the local market.

The selected model that meets the mentioned characteristics is the ESP32-WROOM-32E module (see Fig. 5), which has a System on Chip (SoC) model ESP32-D0WD-V3.

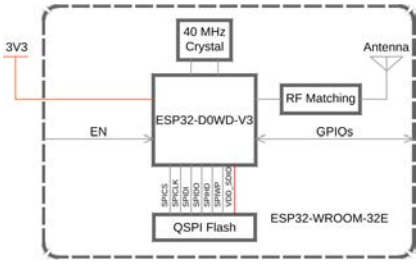
The main features of the module are:

- μP Xtensa dual-core 32-bit LX6 with a working frequency up to 240 Hz.
- 448 KB ROM memory and 520 MB SRAM.
- 16 MB SPI Flash Memory.
- 40 MHz oscillator.
- Peripherals such as UART, SPI, I^{2}C, PWM Motor, Pulse Counter, GPIO, ADC, etc.
- Communication via WiFi and Bluetooth.
- Antenna integrated into the PCB.

B. Controller description

With the ESP32 SoC chosen and taking into account the requirements, the schematic and the PCB of the new controller were designed. The final differential-drive controller

(a) PCB.



(b) Block diagram.

Fig. 5: Espressif module model ESP32-WROOM-32E.

includes the following improvements over the previous version:

- Switching power supply to increase energy efficiency.
- Robot battery status sensing circuit.
- General purpose inputs and outputs.
- Wireless communication via WiFi or Bluetooth.

The most relevant design characteristics of the blocks that make up the controller are detailed below.

1) Voltage supplied and current consumption: The controller was designed so that it can be powered by the robot's battery pack and can support up to $40\,\mathrm{V}$ of continuous voltage. According to the selection of the different integrated circuits that make up the controller, it is necessary to have continuous voltages of $5\,\mathrm{V}$ and $3.3\,\mathrm{V}$. The maximum current consumption is determined considering that each block is operating in the worst-case scenario. The Table I presents the consumption of each block that makes up the controller and the total consumption.

TABLE I: Controller current consumption.

Block	Components	Consumption (in mA)	
		per unit	per block
ESP32	ESP32-WROOM-32E	240	260.89
	General outputs (LEDs)	19.24	
	Inputs (pushbuttons)	1.65	
USB	FT232RL	15	24.62
	Activity LEDs	9.62	
I/Os	Operational Amplifiers	4	8.81
	Power LED	4.81	
Total			294.32

One of the requirements of the new design was to reduce the power consumption of the board. For this reason, it was decided to use an integrated switching power supply in combination with a linear regulator. Such power supply must provide an output current greater than $300\,\mathrm{mA}$.

For the input stage, the *LM2596-5* integrated circuit (IC) was selected. It provides the function of a $5\,\mathrm{V}$ switching regulator of the *Step-Down* type. This IC allows up to $40\,\mathrm{V}$ of input voltage and can deliver up to $3\,\mathrm{A}$ to the load allowing a maximum power of $15\,\mathrm{W}$. Requiring a minimum number of external components, these regulators are easy to use and include internal frequency compensation and a fixed frequency oscillator. This series operates at a switching frequency of $150\,\mathrm{KHz}$, allowing for smaller filter components than those required with lower frequency switching regulators. It has a surface-mount *TO-263* package, which reduces the size and cost of the PCB.

An *AMS1117-3.3* linear regulator with a *SOT-223* SMD package was selected to obtain the voltage of $3.3\,\mathrm{V}$ necessary for the different blocks of the controller. Its output provides a current of up to $1\,\mathrm{A}$ and operates with at least a drop-out voltage of $1\,\mathrm{V}$.

2) USB Communication: The board has a type-B USB connector to connect an external PC. This controller block is responsible for establishing the communication between a computer and the ESP32 module. The ESP32 SoC does not have a native USB module, but it does have a UART serial communication module. Thus, it is necessary to use a USB-UART interface circuit. The integrated circuit used for this purpose is the *FT232RL*, which is compatible with USB 2.0, does not require an external oscillator, and allows transfer rates from 300 baud to 3 Mbaud at TTL levels. A pair of data transmission and reception indicator LEDs were added.

One of the fundamental requirements is that the microcontroller can be programmed via USB, for which it is necessary to be able to enter in *Boot* mode of the ESP32. This is achieved by placing $0\,\mathrm{V}$ on the *GPIO0* pin at the *Reset* time of the microcontroller. To solve it, the *DTR* and *RTS* outputs of the *FT232RL* were connected through a circuit to the *EN* and *Boot* pins of the ESP32 module.

3) Battery voltage sensing: The board has two independent inputs for sensing the charge level of up to two $12\,\mathrm{V}$ batteries. For this measurement, a battery voltage sensing circuit was set up using one of the two ADC converters available in the ESP32 module. The ADC is a 12-bit successive approximation converter multiplexed on up to 16 channels, with a $V_{Ref} \approx 1,100\,\mathrm{mV}$. It was configured to allow an input voltage in the range of $150\,\mathrm{mV}$ to $2,450\,\mathrm{mV}$. The circuit for conditioning the battery charge level signal was designed to measure voltages within the range of $9\,\mathrm{V}$ to $15\,\mathrm{V}$ for each input and adapt those levels to the values of the input range of the ADC.

For this purpose, a circuit with two operational amplifiers that adjust the input voltage values to those required for the ADC input was designed. Both amplifiers were connected in cascade and in differential configuration, with the addition of an anti-alias filter. The gain was configured to achieve the voltage ranges as shown in Table II. The table also shows the ADC counts equivalent to the input voltage range. The *MPC6002* dual operational amplifier was used, which has very low current consumption ($I_Q = 100\,\mu\mathrm{A}$ typical) and a supply voltage in the range of $1.8\,\mathrm{V}$ to $6\,\mathrm{V}$.

TABLE II: Voltage range and ADC counts.

Battery voltage	ADC input voltage	ADC counts
9 V	680 mV	1,137
15 V	2,250 mV	3,761

4) Debug Port: The board has a Joint Test Action Group (JTAG) port for debugging and execution control of the firmware that runs on the ESP32 module. More details regarding the debug port will be given later.

5) PCB Design: For the design of the schematic and the PCB, the free CAD design tool, KiCAD, was used. The PCB was designed based on the technical specifications required for compatibility with the previous version and those imposed by the use of the ESP32 module, which requires being located on one of the edges of the PCB. The following list shows the main characteristics of the PCB:

- Dimensions: length $= 105\,\mathrm{mm}$ and height $= 75\,\mathrm{mm}$.
- Material: FR4, thickness $= 1.6\,\mathrm{mm}$.
- Designed in two layers with metalized holes and anti-soldering mask.

Fig. 6 shows the arrangement of the blocks that make up the controller and the corresponding connectors.

C. Main features

The main characteristics of the developed controller are detailed below.

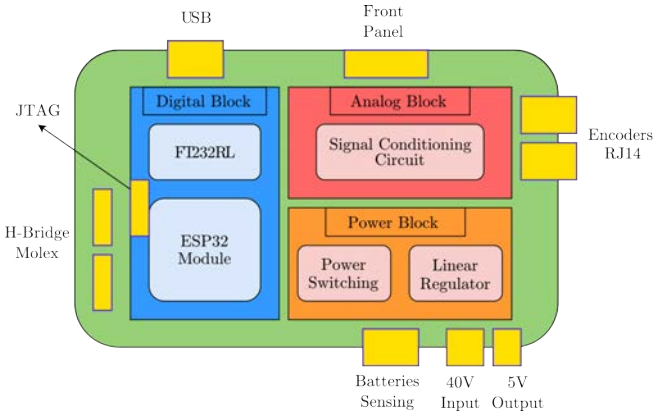


Fig. 6: Layout of the controller blocks on the PCB.

Inputs:

- Power supply: 40 V DC max. (Terminal block).
- 2 incremental optical encoders (*RJ14*).
- 2 analogues for measuring battery voltage (Terminal block).

Outputs:

- 2 PWM signals of 5 V for H-type bridges (Molex).
- Regulated voltage output of 5 V for general purposes (Molex).

Pushbuttons and LED indicators:

- Reset and boot pushbuttons.
- LED indicator for power supply.
- 2 LEDs indicating the status of USB transmission and reception.
- 3 pushbuttons and 4 general-purpose LEDs.
- Front panel connector that extends the reset and boot buttons and signaling LEDs (Header).

Communication ports:

- USB to flash the μC and communication (Type-B).
- JTAG for program debugging (Header).

Fig. 7 shows the PCB of the new controller with all its components mounted, and Fig. 8 shows the controller mounted on the RoMAA-II robot.

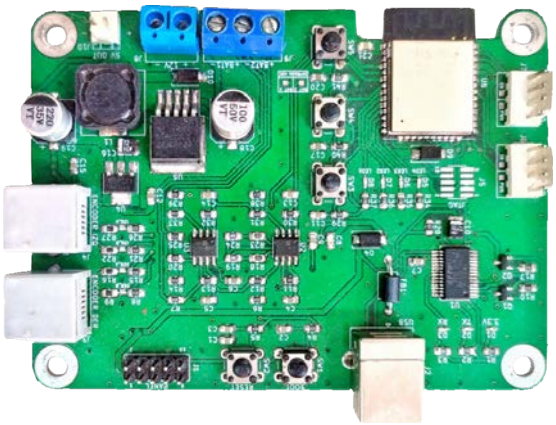


Fig. 7: New differential-drive robot controller board based on ESP32 SoC (DDRC-ESP32).

Fig. 8: New differential-drive robot controller mounted on RoMAA-II robot.

D. Software

Software development is mainly focused on checking the correct functioning of the different parts that make up the differential-drive robot controller.

The repository of the differential-drive robot controller includes some example codes and documentation that enable the user to start up the tools for software development. The current content of the repository is:

1) An application based on the ESP-IDF framework to evaluate the correct functioning of all controller components. This application generates an HTTP server that allows the user to interact with all the controller components to verify their correct functioning. The documentation to build and flash the application in the controller using a Docker container is also included.
2) A document to install and start up a Docker container with the software development tools (framework) for the ESP32 based on ESP-IDF (IoT Development Framework), along with the building and flashing of the blink example on the board developed. From this installation, other examples included in the ESP-IDF framework can be evaluated to study the different microcontroller peripherals that will be used in the final robotics application.
3) A document to install the development environment for ROS 2 with micro-ROS support for the ESP32 architecture using Docker. It contains the building, running, and verification of the correct functioning of an example node. This environment will allow the user to develop the final micro-ROS node that will run on the controller of the final robotics application.

1) JTAG Debug Port: The Espressif software development framework, ESP-IDF (IoT Development Framework), is based on the FreeRTOS real-time operating system which facilitates the programming of applications for multiple cores, as is the case of the SoC used in the controller. This type of application entails the difficulty of discovering errors in programming, due mainly to the running subprocesses, which can be corrected by using a debugging port such as JTAG [32].

On the other hand, Espressif has ported OpenOCD (Open On-Chip Debugger) [33] to support ESP32 processors

that run applications developed with FreeRTOS, and has also developed additional tools that provide functions that OpenOCD does not support natively. One of the applications developed by Espressif is useful to write the program in the Flash memory of the μC and monitor its functioning through diagnostic messages (logging) using a serial terminal.

The developed controller includes the necessary connections for debugging using JTAG, which must work in conjunction with a JTAG-to-USB adapter that has OpenOCD support. On the Espressif webpage, the users can find the design of an adapter board (ESP-Prog) that meets this objective. ESP-Prog [34] is a development and debugging hardware from Espressif which connects to a PC via a USB cable.

E. Experimental evaluation

In a first stage, the design of each of the circuits that make up the controller was validated independently. After the design and final manufacturing of the controller PCB, the corresponding integration tests were carried out.

The controller was mounted on the RoMAA robotic research platform for the functional verification. In the uploaded video[1], the operation of the robot controlled from a mobile phone through web access can be observed. Jointly, an application that runs on the microcontroller to test the functioning of each component of the board was developed. This program was uploaded to the project repository. As mentioned, the test program generates an HTTP server and configures a WiFi network using the communication module included in the SoC, allowing it to be accessed from a web browser running on an external PC connected to said network.

This program generates a web user interface whose main functionalities are:

1) Actuating on each of the robot motors, adjusting their speeds.
2) Actuating on the controller indicator LEDs.
3) Reading the pulse counters of the incremental optical encoders.
4) Reading the battery voltages.
5) Reading the state of the controller pushbuttons.
6) Communicating via WiFi.

Furthermore, as mentioned, the correct functioning of the micro-ROS environment with support for ESP32 was evaluated by building and flashing in the controller an example of a ROS node. This node is built using the ESP-IDF framework and makes use of FreeRTOS. The node publishes a standard message over a ROS topic and communicates through a WiFi network. In a remote PC with connection to the same network, the correct reception of the messages is verified by running a ROS 2 agent that executes in a Docker container.

IV. Conclusion and Future Work

As a final result of the presented development, the hardware of a controller for differential-drive robots was obtained. It replaces a previous design used for more than 10 years at CIII. This new controller is based on a modern SoC that has native support for ROS 2 through its version for embedded systems, micro-ROS; which was one of the main design requirements. The new controller also presents some improvements compared to the previous version: it uses a switching power supply to optimize energy consumption, it has a conditioning circuit for measuring the voltage of the robot's battery, it has general-purpose inputs and outputs, and has the possibility of wireless communication via WiFi. Along with the hardware development, some test applications of the Expressif ESP32 SoC that the controller has were carried out.

To validate the correct functioning of the controller, a test application–based on an HTTP server–that allows actuating and monitoring the correct functioning of all the components of the controller was developed. This test application was uploaded to the project repository. In addition, the necessary documentation to setup up the software development framework for the ESP32 SoC, provided by Espressif, as well as the software for the implementation of micro-ROS nodes in said SoC, are included. In both cases, Docker containers are used, which makes installation and startup easier. This will allow to carry out the software development stage that will implement the final robotics application of the controller as a micro-ROS node.

Future work will include the development of applications for the evaluation and characterization of the operation of each component of the controller in an independent way, such as wheel speed measurement using incremental optical encoders, battery voltage measurement, and other subsystems. The aim is to finally develop an integrated application for controlling a robot based on micro-ROS, in order that those robots that use the controller have direct support for ROS 2.

[1] https://www.youtube.com/watch?v=9FsznQ60jsQ

Acknowledgment

This work is funded by the National University of Technology under the grant UTN-PID 8477, *"Navigation and control of an AMR-type industrial mobile robot based on ROS 2"*. We thank María Laura Guerrini for the English language editing of the manuscript.

References

[1] A. Moshayedi, J. Li, and L. Liao, "AGV (automated guided vehicle) robot: Mission and obstacles in design and performance," *Journal of Simulation & Analysis of Novel Technologies in Mechanical Engineering*, vol. 12, pp. 5–0018, 11 2019.

[2] G. Ullrich, *Automated Guided Vehicle Systems: A Primer with Practical Applications*. Springer Publishing Company, Incorporated, 2014.

[3] L. Lynch, T. Newe, J. Clifford, J. Coleman, J. Walsh, and D. Toal, "Automated Ground Vehicle (AGV) and sensor technologies- a review," in *2018 12th International Conference on Sensing Technology (ICST)*, 2018, pp. 347–352.

[4] F. Gul, S. S. N. Alhady, and W. Rahiman, "A review of controller approach for autonomous guided vehicle system," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 20, no. 1, pp. 552–562, oct 2020. [Online]. Available: https://doi.org/10.11591/ijeecs.v20.i1.pp552-562

[5] C. Ilas, "Electronic sensing technologies for autonomous ground vehicles: A review," in *2013 8TH International Symposium on Advanced Topics in Electrical Engineering (ATEE)*, 2013, pp. 1–6.

[6] IFR, "Service robots," https://ifr.org/service-robots.

[7] C. Cronin, A. Conway, and J. Walsh, "State-of-the-art review of autonomous intelligent vehicles (AIV) technologies for the automotive and manufacturing industry," in *2019 30th Irish Signals and Systems Conference (ISSC)*, 2019, pp. 1–6.

[8] L. Lynch, F. McGuinness, J. Clifford, M. Rao, J. Walsh, D. Toal, and T. Newe, "Integration of autonomous intelligent vehicles into manufacturing environments: Challenges," *Proc. Manufacturing*, vol. 38, pp. 1683–1690, 2019.

[9] S. G. Tzafestas, "Mobile robot control and navigation: A global overview," *Journal of Intelligent & Robotic Systems*, vol. 91, pp. 35–58, 2018.

[10] Y. Abdelrasoul, A. B. S. H. Saman, and P. Sebastian, "A quantitative study of tuning ROS gmapping parameters and their effect on performing indoor 2D SLAM," in *2016 2nd IEEE International Symposium on Robotics and Manufacturing Automation (ROMA)*, 2016, pp. 1–6.

[11] C.-W. Chen, C.-L. Lin, J.-J. Hsu, S.-P. Tseng, and J.-F. Wang, "Design and implementation of AMR robot based on RGBD, VSLAM and SLAM," in *2021 9th International Conference on Orange Technology (ICOT)*, 2021, pp. 1–5.

[12] D. V. Nam and K. Gon-Woo, "Solid-State LiDAR based-SLAM: A concise review and application," in *2021 IEEE International Conference on Big Data and Smart Computing (BigComp)*, 2021, pp. 302–305.

[13] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: An open-source Robot Operating System," in *Workshops at the IEEE International Conference on Robotics and Automation*, 2009.

[14] L. Zhang, R. D. Merrifield, A. Deguet, and G. Yang, "Powering the world's robots - 10 years of ROS," *Sci. Robotics*, vol. 2, no. 11, 2017. [Online]. Available: https://doi.org/10.1126/scirobotics.aar1868

[15] L. Joseph, *ROS Robotics Projects*. Packt Publishing, March 2017.

[16] B. Gerkey, "Why ROS 2?" https://design.ros2.org/articles/why_ros2.html, 2022.

[17] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot Operating System 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, no. 66, may 2022.

[18] OSRF, "Programming multiple robots with ROS 2," https://osrf.github.io/ros2multirobotbook/, 2023.

[19] D. Thomas, W. Woodall, and E. Fernandez, "Next-generation ROS: Building on DDS," in *ROSCon Chicago 2014*. Mountain View, CA: Open Robotics, sep 2014.

[20] K. Belsare, A. C. Rodriguez, P. G. Sánchez, J. Hierro, T. Kołcon, R. Lange, I. Lütkebohle, A. Malki, J. M. Losa, F. Melendez, M. M. Rodriguez, A. Nordmann, J. Staschulat, and J. von Mendel, *Micro-ROS*, ser. Studies in Computational Intelligence, Vol. 1051. Cham: Springer International Publishing, 2023, pp. 3–55.

[21] eProsima, "eProsima Micro XRCE-DDS," https://micro-xrce-dds.docs.eprosima.com/en/latest/, Accessed 2023.

[22] CIII-UTN-FRC. (2023) Differential drive robot controller based on ESP32. [Online]. Available: https://github.com/ciiiutnfrc/ddrc_esp32

[23] M. Baudino and S. Pérez, "Hardware de Control de Plataforma Robótica Móvil con Arquitectura ARM y RTOS. Caracterización." Universidad Tecnológica Nacional - Facultad Regional Córdoba, Tech. Rep., 2010.

[24] D. A. Gaydou, G. F. Pérez Paina, G. M. Steiner, and J. Salomone, "Plataforma móvil de arquitectura abierta," in *Proceedings of the V Jornadas Argentinas de Robótica (JAR)*. Ediuns, November 2008.

[25] G. Perez Paina, G. Araguas, D. Gaydou, G. Steiner, and L. Rafael Canali, "RoMAA-II, an open architecture mobile robot," *Latin America Transactions, IEEE (Revista IEEE America Latina)*, vol. 12, no. 5, pp. 915–921, Aug 2014.

[26] G. F. Perez Paina, F. E. Elizondo, D. A. Suares, and L. R. Canali, "Design and implementation of a multi-sensor module for mobile robotics applications," in *Proceedings of the Argentine Conference on Embedded Systems (CASE)*, 2012, pp. 269–274.

[27] G. F. Perez Paina, D. A. Gaydou, N. L. Palomeque, and L. A. Martini, "Librerías embebidas para microcontroladores LPC2000 de aplicación en robótica," in *Proceedings of the Argentine Conference on Embedded Systems (CASE)*, 2011.

[28] Convenio de transferencia tecnológica: CIII-UTN-FRC y Caima Segall S.R.L., "Robot Móvil Autónomo para Transporte de Partes en Logística Interna de Planta DENSO Manufacturing Argentina," 2020.

[29] CIII-UTN-FRC. (2021) romaa_ros: ROS packages for the RoMAA robot. [Online]. Available: https://github.com/ciiiutnfrc/romaa_ros

[30] G. Perez-Paina, D. Gaydou, and G. Araguás, "Driver de ROS para el robot móvil RoMAA," in *Proceedings of the X Jornadas Argentinas de Robótica (JAR)*, 2019.

[31] micro ROS. (2020) Supported hardware. [Online]. Available: https://micro.ros.org/docs/overview/hardware/

[32] Espressif, "JTAG Debugging," https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/jtag-debugging/index.html, Accessed 2023.

[33] OpenOCD, "Open On-Chip Debugger," https://openocd.org/, Accessed 2023.

[34] Espressif, "Introduction to the ESP-Prog Board," https://docs.espressif.com/projects/espressif-esp-iot-solution/en/latest/hw-reference/ESP-Prog_guide.html, Accessed 2023.