A Modular and Configurable Architecture for ROS 2 Hardware Integration with micro-ROS

Jakob Friedl[1] and Markus Bader[1]

***Abstract*— In general, a vehicle cannot follow a given trajectory if the control commands for the motor controllers are not delivered to the hardware in time. This issue arises when a standard computer running ROS 2 is used for control without a real-time extension. This paper presents an architecture that leverages micro-ROS on an ESP32-C6 with a RISC-V CPU running a Real-Time Operating System (RTOS). The goal is to demonstrate that drift compensation, based on odometry and IMU data, can be performed in real-time directly on the microcontroller. As a first step, we show how micro-ROS handles robot kinematics (Ackermann steering) within the firmware, configured via a persistent parameter server. We demonstrate that this design improves integration simplicity, adaptability, separation of concerns and evaluate real-time compliance.**

***Index Terms*— micro-ROS, mobile robotics, embedded systems, ROS 2**

I. INTRODUCTION

This paper proposes a microcontroller-based ROS 2 integration architecture aimed specifically at modular, configurable robotics hardware. Leveraging the micro-ROS framework, it provides a plug-and-play solution that simplifies interfacing embedded hardware with higher-level ROS 2 ecosystems shown in Figure 1. Traditional designs often use onboard computers that interact directly with hardware components (as can be seen in [7]), a strategy that can lead to redundant software development, a mixing of low-level hardware interactions with higher-level control concerns, and challenges in ensuring real-time performance. Micro-ROS addresses these limitations by extending ROS 2 functionalities directly to resource-constrained microcontrollers.

In this approach, the micro-ROS agent on the ROS 2 host translates the lightweight eXtremely Resource-Constrained Environments-Data Distribution Service (XRCE-DDS) middleware used by micro-ROS into standard DDS messages [1], [2]. It supports transports such as UDP and USART by default, and can be extended with custom implementations. However, hardware variations often force firmware rebuilds or manual tweaks; our design instead embeds an NVS-backed parameter server, allowing kinematic and hardware settings to be adjusted live via ROS 2 parameters.

Incorporating an RTOS into the firmware permits local prioritization of time-critical tasks, ensuring reliable operation under strict deadlines. Prior work in underwater vehicles demonstrates a micro-ROS RTOS setup [6], but offers no built-in mechanism for runtime customization or a clear task breakdown. In contrast, we leverage advanced RTOS features alongside our persistent parameter server and detail a modular FreeRTOS task architecture (see II-B). While demonstrated on an Ackermann-steering robot, our modular FreeRTOS task breakdown and NVS parameter server generalize directly to other ROS 2–integrated hardware and form the basis for more advanced microcontroller-based trajectory following.

[1]The authors are with Faculty Informatics at TU Wien, Vienna, Austria. firstname.lastname@tuwien.ac.at

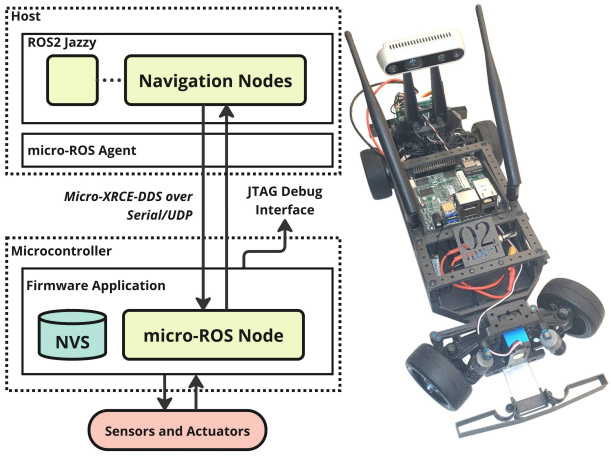

Fig. 1. System overview and test platform on the right

II. PROPOSED ARCHITECTURE

A. Hardware Components and System Overview

The initial question was how best to structure motor control for an Ackermann robot integrated with ROS 2. As highlighted in the introduction, conventional approaches often conflate low-level hardware interfacing with high-level control, impede real-time performance, and lack a common framework for comparing firmware designs. The goal of the specific design in this paper is to accept twist commands via the `/cmd_vel` topic, compute the necessary kinematics for motor speeds and steering in the firmware, while publishing the resulting odometry, thus simplifying the interface between ROS 2 system and hardware. Kinematic and control variables are live-adjustable through the parameter server. This approach is demonstrated on the MX-Car [4], a mobile robot developed at TU Vienna with an Ackermann drivetrain featuring two non-steering rear wheels (driven by BLDC hub motors) and a front steering servo.

Figure 1 shows the overall system with the hardware platform at the right. The top segment depicts the onboard or

https://doi.org/10.34749/3061-0710.2025.22

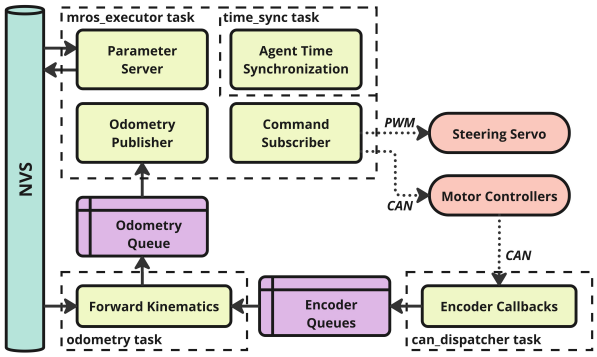

Fig. 2. Firmware application architecture overview

network-connected computer running ROS 2 (with a Dockerized micro-ROS agent), the middle segment shows the ESP32 running FreeRTOS with a dedicated micro-ROS node using Non-Volatile Storage (NVS) for persistent configuration and the respective interaction with the sensors and actuators. A more detailed look at the firmware is provided in the next section.

The single-core ESP32-C6-DevKitM-1 is used as primary controller due to its processing capabilities, peripherals and open RISC-V architecture. Micro-ROS integrates into its build system via an ESP-IDF component [3]. Motor control is provided by two daisy-chained ODrive-Micro controllers, with a Control Area Network (CAN) transceiver interfacing the microcontroller to exchange commands (e.g., position, speed, effort) and feedback (e.g., velocity, current) [5].

B. Firmware and Communication

Figure 2 shows a simplified view of the firmware architecture. We enforce strict timing by splitting work across four FreeRTOS tasks with distinct priorities. The can_dispatcher and mros_executor tasks run at highest priority, ensuring no inbound messages are lost. The odometry task runs at medium priority, and the low-priority time_sync task handles agent clock alignment via micro-ROS mechanisms (obtaining the host timestamp) without interfering with real-time deadlines.

The can_dispatcher manages all CAN traffic to and from the motor drivers, and the mros_executor drives the micro-ROS executor, handling subscribers, timers, and an NVS-backed parameter server. Configuration parameters and kinematic properties (e.g., wheel base, track width) are stored persistently in non-volatile storage and can be updated via standard ROS 2 param calls.

Incoming twist messages on `/cmd_vel` are processed in the mros_executor: the callback computes motor and steering setpoints using the Ackermann kinematic model as described by [8] and forwards them to the drives, so no command queuing is needed. In the odometry task, execution blocks until fresh encoder estimates arrive from both drives; it then performs forward kinematics to update an internal pose. A timer-driven publisher retrieves this pose through a single-element FreeRTOS queue (buffering only the latest state) and publishes on `/odom`, timestamping the message with the arrival time of the latest encoder sample. An onboard or remote ROS 2 computer (depending on the transport) can then integrate higher-level features such as path planning or trajectory control.

III. EVALUATION

To assess the real-time performance of the integration of this architecture, we measured timing at $10\,Hz$ over the 115 200 baud serial transport during full system operation. For odometry, 1 200 messages were received by the host. The reception intervals exhibited a mean of $99.998\,ms$, a standard deviation of $5.63\,ms$, and a peak-to-peak jitter of $52.0\,ms$. We then measured the round-trip delay from publishing a stamped twist command message on the host to applying motor setpoints on the microcontroller by again logging 1200 messages. This delay averaged $28.684\,ms$, with a standard deviation of $4.73\,ms$, and spanned $27.82\,ms$ peak-to-peak. In both cases, no messages were lost.

Compiled with space optimization, the complete firmware occupies $342.9\,kB$ of flash (4.09%) and $118.2\,kB$ DIRAM (26.15%), confirming a compact footprint.

IV. SUMMARY AND OUTLOOK

This paper presents an architecture that integrates resource-constrained microcontrollers with ROS 2 using micro-ROS on an RTOS. It exposes hardware interfaces as standard topics and services and simplifies configuration via a persistent parameter server. We evaluated its real-time performance at $10\,Hz$ over serial transport, observing latency and jitter levels acceptable for typical mobile robotics applications, while still leaving room for optimization.

Future work will build on this architecture to realize complex trajectory control on the microcontroller. The aim is to calculate collision-free trajectories in the ROS 2 framework on the host and then pass them on to the micro-ROS controller for execution.

V. ACKNOWLEDGMENT

This research is supported by the Austrian Science Fund (FWF), under project No. 923138, GreenFDT.

REFERENCES

[1] "Features and architecture of micro-ros," https://micro.ros.org/docs/overview/features/, 2025, [Online; 22 April 2025].
[2] "Micro xrce-dds documentation," https://micro-xrce-dds.docs.eprosima.com/en/latest/, 2025, [Online; 22 April 2025].
[3] "Micro-ros esp-idf component," https://micro.ros.org/docs/tutorials/core/first_application_rtos/esp32/, 2025, [Online; 22 April 2025].
[4] "Mx-car github repository," https://github.com/tuwien/mx-car, 2025, [Online; 22 April 2025].
[5] "Odrive-micro hardware datasheet," https://odriverobotics.com/docs/micro-datasheet, 2025, [Online; 22 April 2025].
[6] P. A. Gutiérrez-Flores and R. Bachmayer, "Concept development of a modular system for marine applications using ros2 and micro-ros," in *2022 IEEE/OES Autonomous Underwater Vehicles Symposium (AUV)*. IEEE, 2022, pp. 1–6.
[7] W. Jo, J. Kim, R. Wang, J. Pan, R. K. Senthilkumaran, and B.-C. Min, "Smartmbot: A ros2-based low-cost and open-source mobile robot platform," *arXiv preprint arXiv:2203.08903*, 2022.
[8] A. Kaltenegger, "Physical and graphical simulation of an ackermann steered vehicle," *Bachelor's Thesis, TU Wien*, 2016.