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

Jakob Friedl[1] and Markus Bader[1]
Faculty of Informatics, Automation Systems Group, TU Wien

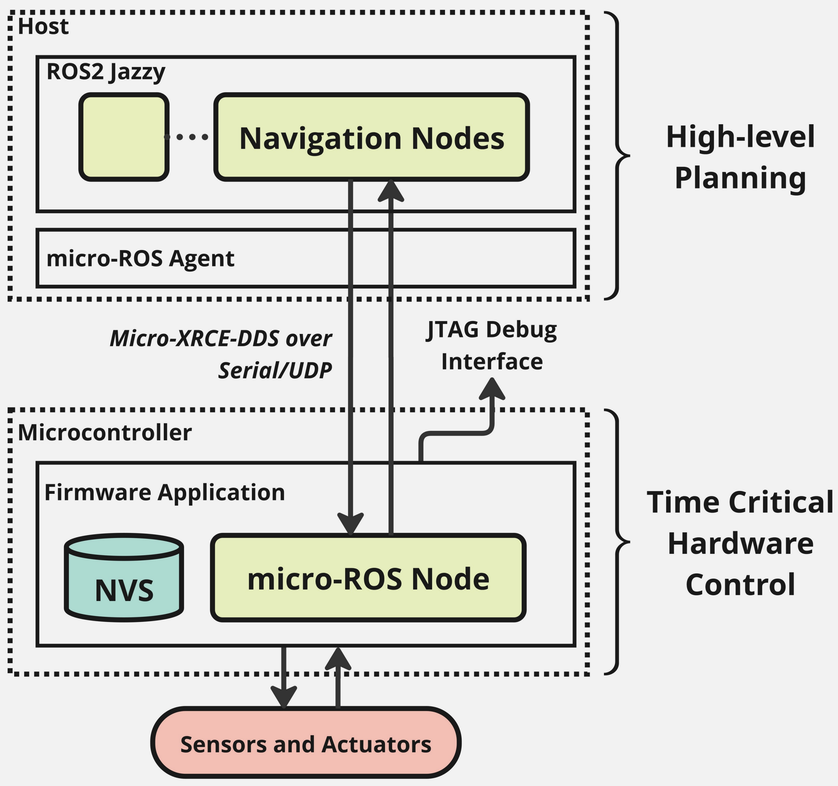
Abstract:
In general, a vehicle cannot follow a given trajectory if the control commands for the motor controllers are not delivered to the hardware in time. This issue arises when a standard computer running ROS 2 is used for control without a real-time extension. This paper presents an architecture 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.

Motivation

Without a clear, standardized architecture and unpredictable ROS 2 communication performance, benchmarking and optimization of in-firmware trajectory control becomes difficult.

Approach

We propose a microcontroller-based ROS 2 integration architecture aimed specifically at modular, configurable robotics hardware. With a clear task breakdown and evaluation, we provide the foundation for further additions, analyses and optimization.



System Overview

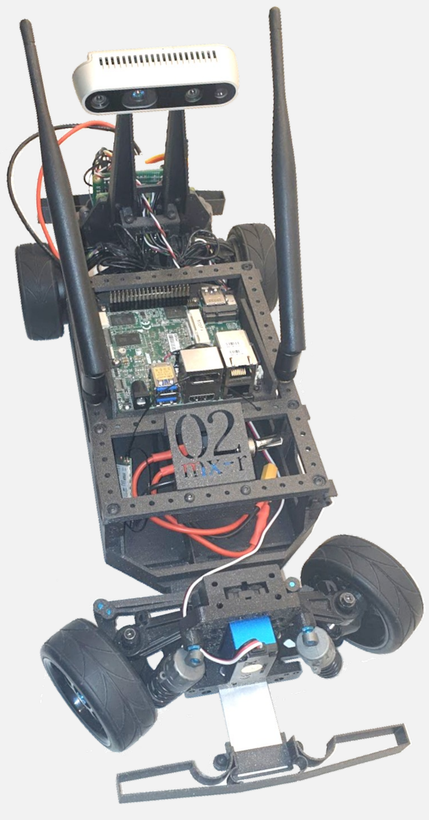
- Embedded NVS-backed parameter server, allowing live kinematic and hardware adjustments via ROS 2 parameters
- RTOS incorperation permits local prioritization of time-critical tasks

Architecture generalizes directly to other ROS 2–integrated hardware and form the basis for more advanced microcontroller-based trajectory following.

Target Platform

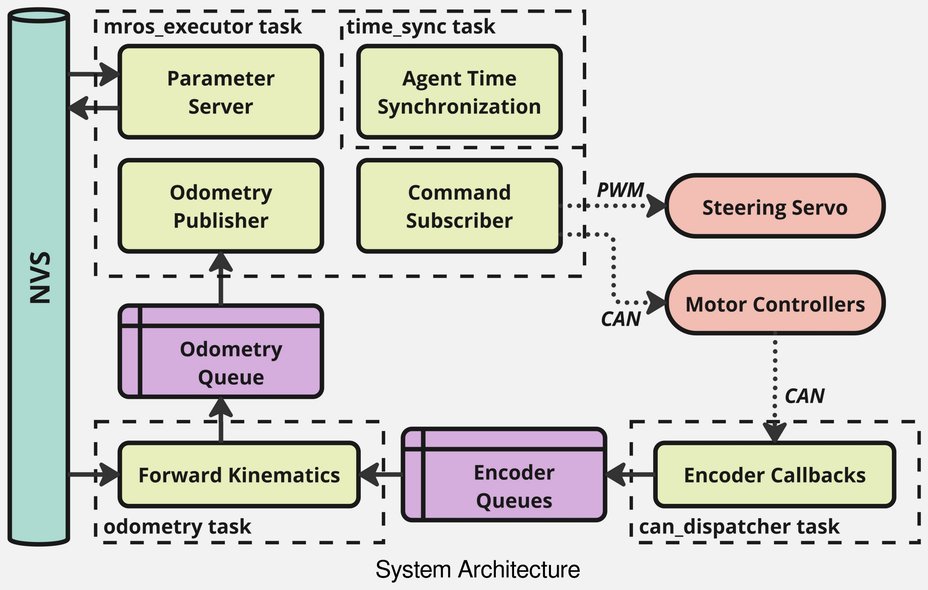
The approach is demonstrated on the MX-Car, a mobile robot developed at TU Vienna with an Ackermann drivetrain featuring two non-steering rear wheels (driven by BLDC hub motors) and a front steering servo.

- ESP32-C6-DevKitM-1 as hardware controller
- Odrive-Micro motor controllers (CAN daisy chain)
- Single front steering servo
- BLDC hub motors for rear drive



Architecture

We splitt work across four FreeRTOS tasks with distinct priorities



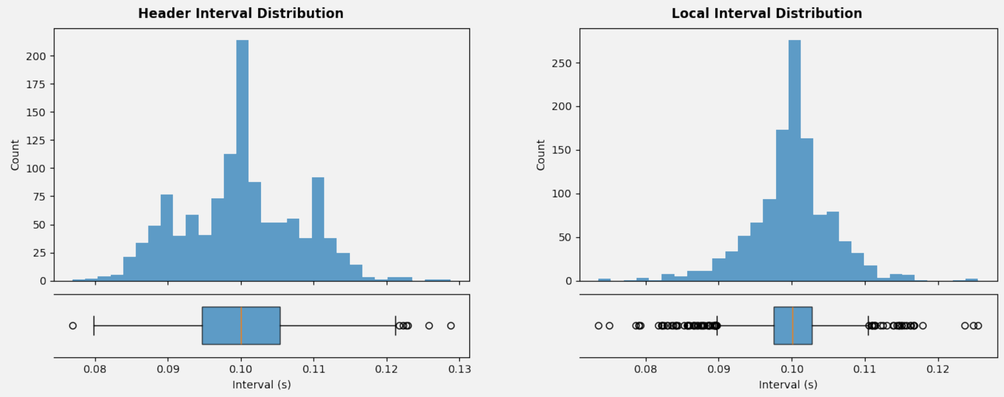
System Architecture

- **can_dispatcher task**
  manages all CAN traffic to and from the motor drivers
- **mros_executor task**
  drives the micro-ROS executor, handling subscribers, timers, and an NVS-backed parameter server
- **time_sync task**
  handles agent clock alignment
- **odometry task**
  blocks execution until new encoder estimates arrive from both drives and performs forward kinematics to update an internal pose. Timer-driven publisher retrieves and publishes it, timestamping the message with the arrival time of the latest encoder sample

Evaluation

Odometry Publishing Jitter

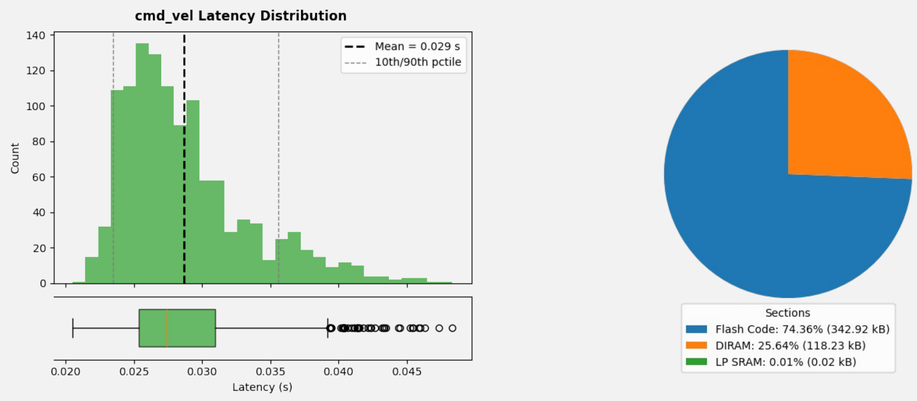
- 1200 messages, reliable QoS, 10Hz publishing frequency



Distinction between time sources:

- Header: latest encoder measurement
  - → ***51,82 ms*** peak-to-peak jitter
  - → ***100 ms*** median, ***7,96 ms*** standard deviation
- Local: arrival on host system
  - → ***51,97 ms*** peak-to-peak jitter
  - → ***100 ms*** median, ***5,63 ms*** standard deviation

Twist publish to action latency



- → ***27,82 ms*** peak-to-peak jitter
- → ***27,42 ms*** median activation time, ***4,73 ms*** standard deviation

TECHNISCHE UNIVERSITÄT WIEN

[1]first.lastname@tuwien.ac.at,

Austrian Robotics Workshop 2025, FH Salzburg

This work was funded by the Austrian Science Foundation under grant agreement No.~923138~(GreenFDT)

automation systems