SAE5 F1TENTH PRESENTATION

Development of functionalities for the autonomous F1Tenth car

Céleste KOLOUSSA, Vincent ARIMON

IUT de NANTES

Academic Year 2024-2025

Presentation Outline

1. Introduction
2. Requirements
3. Work Environment
4. Introduction to ROS
   - Definition
   - Getting Started
5. Developing Functional Code for the Car
6. Electronic Board - Ultrasonic Sensors
7. Transition to ROS2
8. Difficulties Encountered
9. Future Objectives for the next third years
10. Sources

Introduction

Project Objective:

- Develop an autonomous car to participate in the **F1Tenth** race.
- Work on trajectory planning, emergency braking, and optimal speed selection.
- Rely on **ROS** (Robot Operating System) for modularity and inter-node communication.
- Future integration with traffic lights.

Autonomous Car Challenge: Obstacle Avoidance

What's the most crucial things in autonomous vehicles?

Requirements

Features to implement:

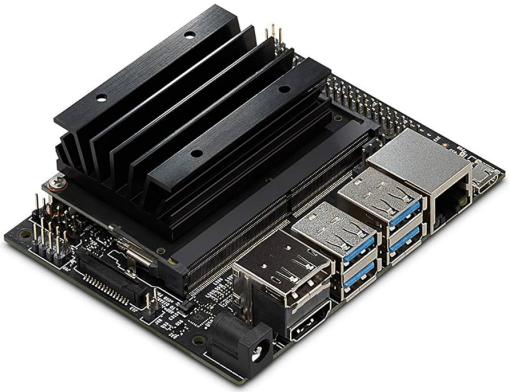
- Trajectory planning.
- Emergency braking management.
- Steering angle control.
- Optimal speed selection.
- Transition to ROS2.

Integration:

- Based on ROS (C/C++).
- Planned extension: communication with traffic lights and 3D camera for traffic light detection.
- Robustness and flexibility for an F1Tenth race environment.

Work Environment

- **Development Platform:** Jetson Nano running Linux.
- **IDE:** Visual Studio Code (accessing Jetson Nano via SSH).
- **Version Control:** GitLab (collaboration, traceability).
- **Goal:** Continuous integration and rapid deployment.



Jetson Nano

GitLab

GitLab

Visual Studio Code

VSCode

ROS: Definition

ROS (Robot Operating System): A set of open source libraries and tools for developing complex robotic systems.

- **Modular architecture**: independent nodes communicating via topics.
- **Tools**: visualization, simulators, mathematical libraries, sensor/actuator drivers.

Before ROS: Client-server architectures.
With ROS: Publisher-subscriber architectures.

Getting Started with ROS

Training through lessons on the **Madoc** platform:

- Creating ROS packages.
- Setting up nodes and managing *topics*.
- Using custom messages.
- *Launch files* and parameter files.

Goal: Understand ROS architecture and prepare for the F1Tenth implementation.

Developing Functional Code for the Car

Study of the existing code: Understanding the overall architecture.

Two projects:

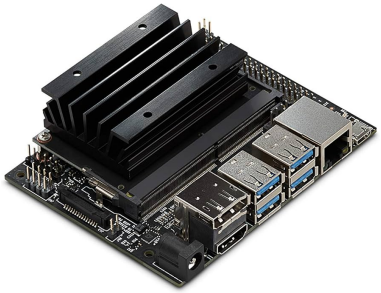
1. Based on legacy code.
2. Starting from scratch.

Objective: Compare robustness and performance, then combine the best elements.

Main Hardware Components

LiDAR (RPLIDAR A2)

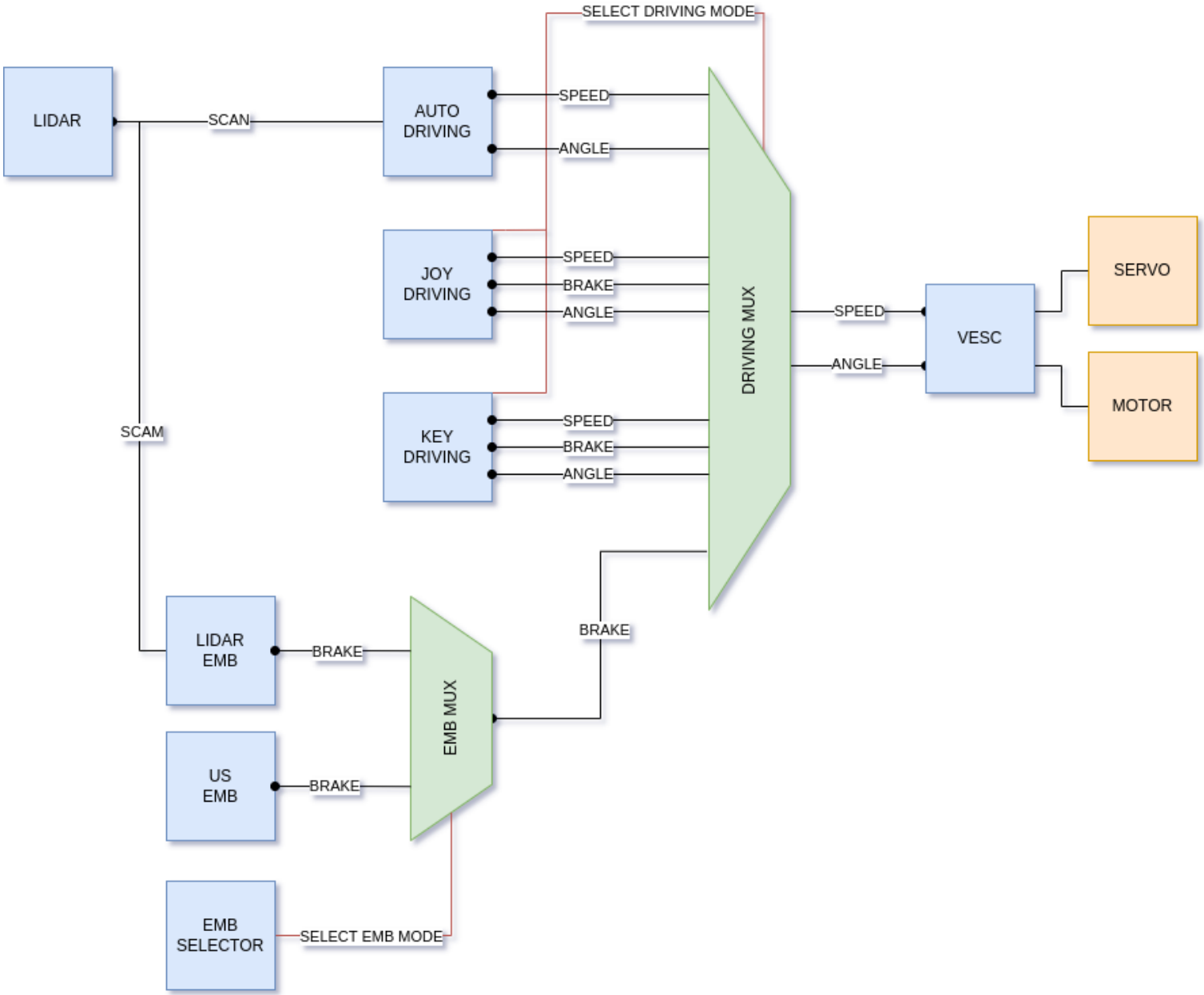
Chassis



Jetson Nano

Power Supply

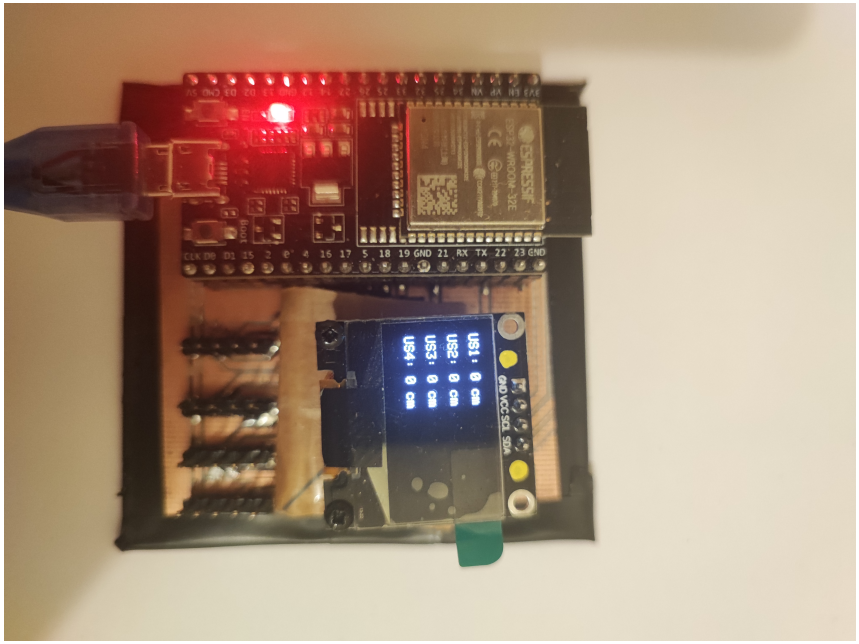
Conceptual ROS System Diagram



Electronic Board - Ultrasonic Sensors

Goal: Acquire data from 4 ultrasonic sensors + ESP32

- Data processing by ESP32 WROOM32E.
- Display on OLED screen (I2C).
- Serial transmission (UART) to ROS.



Ultrasonic Board (ESP32 + sensors + OLED screen)

Data Structure / Threads

Data structure: Stored in the ESP32 before transmission.

3 Main Threads:

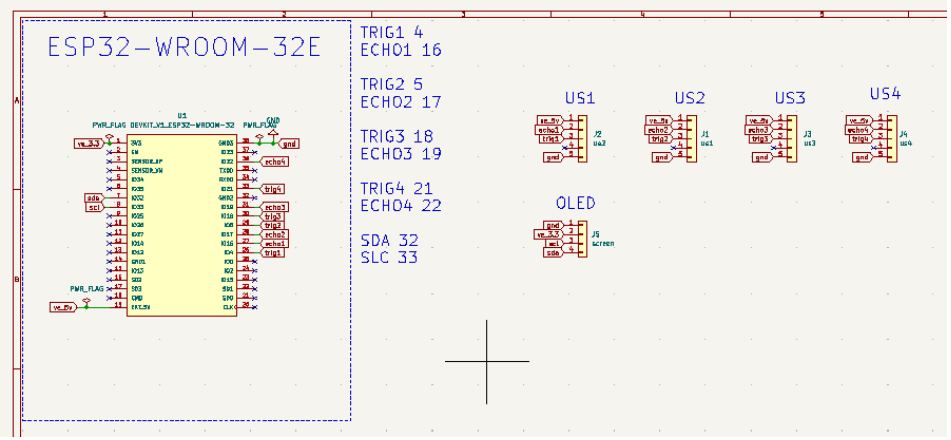
- **Measurement**: Periodically reads the sensors.
- **Display**: Updates the OLED screen.
- **Serial Publishing**: Periodically sends data to the ROS node.

Test Program

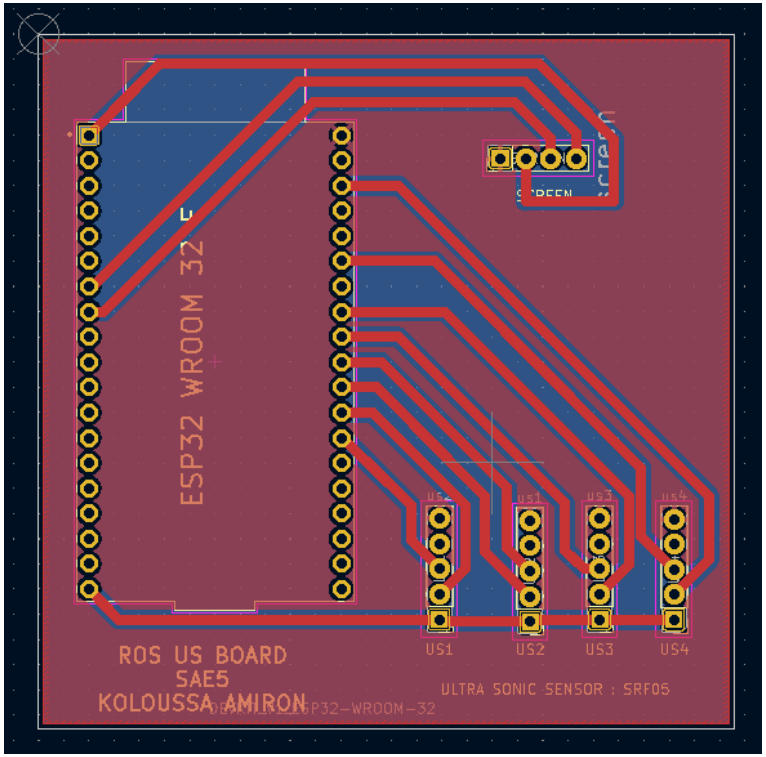
- Validation of the board and serial connection.
- Script to listen on the serial port: Displaying received measurements.
- Results confirmed, ultrasonic distances OK.

ROS Integration: Write a node subscribing to the serial port to publish on a topic.

KiCad Design

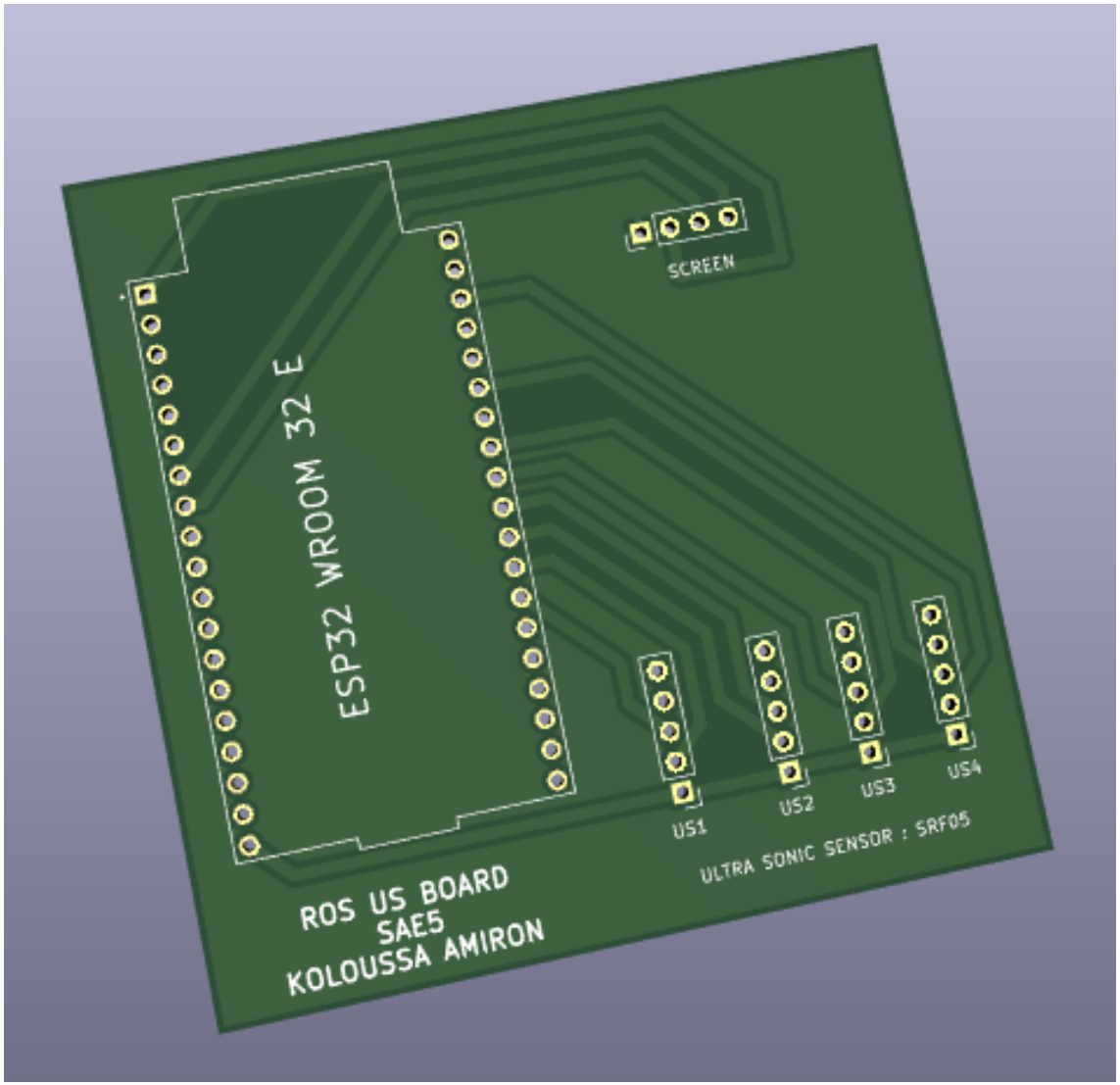


KiCad Schematic



PCB Routing

3D Rendering of the PCB



3D Rendering of the PCB

Why Transition to ROS2?

Motivation: ROS2 is the evolution of ROS1, providing enhanced capabilities for real-time processing, security, and scalability.

Key advantages:

- **Improved performance:** Multi-threading and better communication latency.
- **Real-time capability:** Supports deterministic execution for time-critical tasks.
- **Security features:** Improved authentication and encryption for safe data exchange.
- **DDS-based communication:** More robust inter-node communication model.

Differences Between ROS1 and ROS2

- **Middleware:** ROS1 uses a custom message-passing system, while ROS2 leverages DDS (Data Distribution Service) for efficient and scalable communication.
- **Launch System:** ROS2 uses Python-based launch files, offering more flexibility than ROS1's XML-based launch files.
- **Package Management:** ROS2 integrates with Python 3 and colcon for improved dependency handling.
- **Backward Compatibility:** Some ROS1 packages require migration, but ROS2 supports dual execution (ROS1-ROS2 bridge).

Migration Plan for F1Tenth

Steps for migrating to ROS2:

- Refactoring nodes to use ROS2 API.
- Replacing legacy publishers/subscribers with DDS-based topics.
- Testing and validating performance improvements.
- Gradually integrating ROS2-native features for better efficiency.

Difficulties Encountered

Challenges faced during development:

- **ROS Installation Issues:**
  - Compatibility problems between different versions of ROS and the Jetson Nano.
  - Dependency conflicts requiring manual resolution.
- **Power Management:**
  - Initially, the car was always powered by a battery but it caused the jetson nano to go on sechure mod when the battery was low.
- **Vesc configuration**

Future Objectives for the next third years

- Integrate **SLAM TOOLBOX** for mapping and localization.
- Finish the transition to ROS2.
- Implement overtaking algorithm (tests in simulator and on the F1Tenth track).
- Implement communication with traffic lights (HTTP or video).

Final Goal: A **reliable and high-performance** autonomous vehicle for the F1Tenth competition.

Sources

VI.I ROS

- https://www.generationrobots.com
- https://digitalcorner-wavestone.com

VI.II Figures

- Jetson Nano Image: https://www.amazon.fr
- GitLab Logo: https://www.stickpng.com
- VSCode Logo: https://www.logos-marques.com
- Cover Image: https://roboracer.ai

End of Presentation

Thank you for your attention