Integration of LiDAR and Stereo Camera in a ROS2-based Environment using an NVIDIA Jetson Orin Nano

Samuel Sanchez

Hochschule Esslingen

May 2025

Abstract

Robot Operating System 2 (ROS2) is an open source middleware framework to develop distributed and modular robotic applications. It has native support for real-time communication, cross-platform compatibility, and easy interface with packages like Simulink and MATLAB. This renders it a perfect middleware for prototyping and deployment of autonomous features in both academic and industrial environments. This research covers the integration and implementation of ROS2 nodes for fusing LiDAR and stereo camera sensors on a Jetson Orin Nano controller. The theoretical foundation underlying this project is part of a broader research project named Automated Driving in MIniaturized Traffic scenarios 1:14 (ADMIT14), which are intended to validate autonomous driving technology in a controlled, small-scale environment. In this study, we concentrate on creating a modular configuration, where Python Publishers transmit raw sensor data, and Python subscribers or MATLAB decode them at higher levels. This isolated configuration allows independent development, testing, and replacement of individual hardware components, showcasing the versatility that ROS2 provides.

Contents

1 Introduction **2**
1.1 Robot Operating System 2 (ROS2) . . . 2

2 Challenges and Issues Encountered **2**
2.1 ROS2 compatibility across OS and Jetson platforms . . . 3
2.2 ROS2 Installation . . . 3
2.2.1 NVIDIA Jetson Nano . . . 3
2.2.2 Windows and iOS . . . 3
2.3 Issues with RPLIDAR ROS2 Package . . . 3
2.3.1 Incompatibility of Official RPLIDAR ROS2 Package on Windows . . . 3
2.3.2 rplidar_ros running . . . 3
2.4 Issues with cv_bridge and Image Data on Windows and iOS . . . 3

3 System Architecture **3**
3.1 Logical Architecture . . . 3
3.2 Physical Architecture . . . 4

4 Implementation **5**
4.1 Hardware Setup . . . 5
4.2 ROS2 Environment . . . 6
4.3 ROS2 Node Architecture . . . 6
4.4 Arduino and CAN Integration . . . 6
4.5 MATLAB Integration . . . 6
4.6 Reusability and Modularity . . . 7

5 Experiments and Results **7**
5.1 LiDAR Data Acquisition 7
5.2 Stereo Camera Image Reception in MATLAB and Python 7
5.3 CAN Message Transmission 9
5.4 System-Level Test 9

6 Conclusion and Future Work **9**

1 Introduction

Autonomous driving technologies are mainly based on the integration of various sensors and control components to perceive, interpret, and react to their environment. In complex systems such as autonomous cars, it is necessary to ensure that all subsystems can be developed, tested, and updated independently, which is crucial for scaling up and for the sake of long-term sustainability.

The ADMIT14 program, Automated Driving in MIniaturized Traffic environments scale 1:14, was started at Hochschule Esslingen with the vision of validating and testing autonomous functionality in a manageable miniature environment. Our work within this paradigm revolves around the realization of a sensor fusion architecture with ROS2, by means of LiDAR technology and stereo cameras on an NVIDIA Jetson platform.

1.1 Robot Operating System 2 (ROS2)

ROS2 is an open source middleware framework designed to create distributed and modular robot applications. ROS2 extends the basic ROS environment with the inclusion of real-time communication, multi-platform runtimes, and enhanced security features. In the context of this project, ROS2 serves as the centerpiece of the communication system among the sensors (stereo camera and LiDAR), the processing module (Jetson Nano), and external modules (Arduino, MATLAB) with standardized topics and interfaces.

The publish-subscribe architecture allows for a loose coupling between data producers and consumers, thereby enabling easy integration and subsequent substitution of components with minimal impacts on the overall system.

One of the primary motivations for using ROS2 is because of its modularity, which promotes breaking up functionalities into independent nodes. Modularity renders both the development process and the system adaptation straightforward afterward. For instance, swapping a LiDAR sensor, updating the stereo camera, or changing the communication interfaces (i.e., from CAN to Ethernet) does not necessitate a redesign of the overall system. Rather, only the respective ROS2 node must be recompiled or adjusted.

Additionally, ROS2 has real-time communication, platform independence, and seamless interfacing with Simulink and MATLAB tools built in. All these make it an ideal middleware candidate for the development and deployment of autonomous capabilities in academic and industrial environments. The remainder of this paper is organized as follows: Section 2 covers the development challenges faced. Section 3 presents the system architecture. Section 4 reports on the details of the implementation. Section 5 gives the experimental results, and Section 6 concludes the paper with future research directions.

2 Challenges and Issues Encountered

Throughout the development of this project, several technical challenges and incompatibilities were encountered, particularly due to the heterogeneous software and hardware platforms involved.

2.1 ROS2 compatibility across OS and Jetson platforms

ROS 2 Distribution	Ubuntu Version	End of Support	Jetson Kits	MATLAB
Iron Irwini	Ubuntu 22.04 (Jammy)	November 2029	Jetson Orin Nano	Planned
Humble Hawksbill	Ubuntu 22.04 (Jammy)	May 2027	Jetson Orin Nano	Yes
Galactic Geochelone	Ubuntu 20.04 (Focal)	November 2022	Jetson Nano	Yes
Foxy Fitzroy	Ubuntu 20.04 (Focal)	May 2022	Jetson Nano	Yes
Dashing Diademata	Ubuntu 18.04 (Bionic)	May 2021	Not Recommended	Legacy
Crystal Ckemmys	Ubuntu 18.04 (Bionic)	January 2020	Not Recommended	No

Table 1: ROS 2 compatibility across OS and Jetson platforms

2.2 ROS2 Installation

2.2.1 NVIDIA Jetson Nano

An image of Ubuntu 18.04, which is not advised for ROS2 functionality, is available on the official NVIDIA Jetson Nano website. To enable ROS2 Foxy deployment, more investigation was needed to locate an appropriate Jetson Nano image running Ubuntu 20.04.

2.2.2 Windows and iOS

Installation of ROS2 on non-Linux systems, like Windows or iOS, requires strict following of the installation guide available on the official ROS website. It is important that all dependencies must be installed according to the guide, and commands executed in respective terminals; failure in these steps could lead to problems and a non-functional environment.

2.3 Issues with RPLIDAR ROS2 Package

2.3.1 Incompatibility of Official RPLIDAR ROS2 Package on Windows

The official `rplidar_ros` package from Slamtec was found to be incompatible with Windows during the use of the RPLIDAR A2M8 sensor. This compatibility problem prevented the RPLIDAR from being used directly on a Windows system.

2.3.2 rplidar_ros running

Additional read and write permissions for the serial device are necessary to initialize the rotation of the LiDAR and then broadcast data.

2.4 Issues with cv_bridge and Image Data on Windows and iOS

The `cv_bridge` package that was required for the conversion of ROS2 image messages into OpenCV-compatible formats was a significant obstacle in the early phases of Windows and iOS platform testing. It thus became requisite to move image processing tasks to Linux-based operating systems, as the issues encountered prevented direct processing of visual data on Windows operating systems.

3 System Architecture

3.1 Logical Architecture

Figure 1 shows the modular framework employed in this project. The architecture of the system is divided into two areas. On the left side, in the blue area, the Jetson ROS2 setup takes care of the management of data from the RPLIDAR and Stereo Camera. The RPLIDAR and Stereo Camera are each interfaced with specific Python-based ROS2 nodes that publish sensor data on the following topics: `/scan`, `/raw_image_left`, and `/raw_image_right`.

The Jetson then executes a series of ROS2 nodes that receive the sensor data published by the sensor nodes and then publish appropriate control commands through a CAN interface. The interface serves to connect the ROS2 system to an external Arduino microcontroller.

In the right side of the architecture, red area, the Arduino processes incoming CAN messages and transmits commands to the respective actuators. This divided arrangement enables each hardware unit to be developed, tested, and replaced independently, thus illustrating the flexibility that ROS2 promotes in modular architectures.

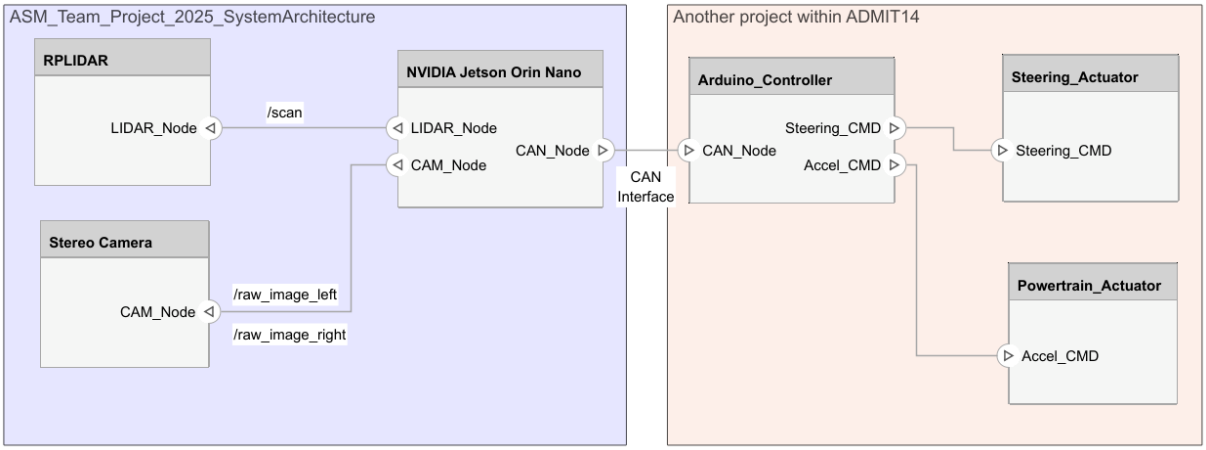

Figure 1: Proposed System Architecture showing sensor input, microcontrollers, and actuation modeled in System Composer.

3.2 Physical Architecture

To verify the architecture in a practical setting, a physical hardware setup was built in addition to the software's implementation of logical modularity. A stereo camera and an RPLIDAR A2M8 sensor are connected to the NVIDIA Jetson Orin Nano development board, which serves as the system's central processing unit. These components are mounted on a 1:14 scale model vehicle, which serves as the base for the ADMIT14 autonomous platform.

The Jetson Nano, which is connected to an Arduino unit via a CAN interface, controls every ROS2 node. The Arduino controls the low-level actuation of the throttle and steering motors. All components are powered by an onboard battery system, and communication between modules is fully integrated.

Figure 2 shows the assembled hardware platform used for testing and validation.

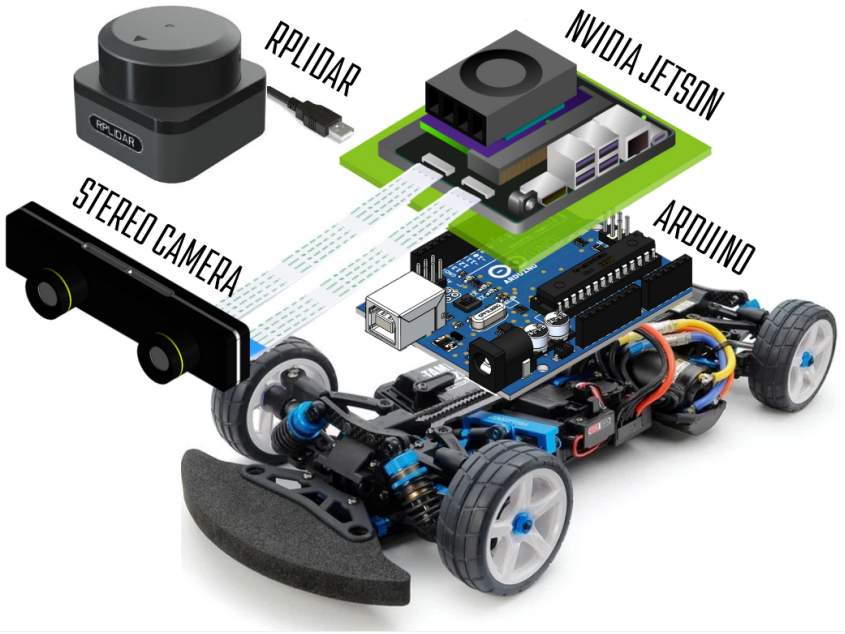

Figure 2: Physical implementation of the logical architecture on the 1:14 scale ADMIT14 platform. Components include the Jetson Nano, RPLIDAR A2M8, stereo camera, Arduino controller and RC Chassis.

4 Implementation

An overview of the system's actual implementation is provided in this chapter, along with details on the hardware configuration, software environment, ROS2 node design, Arduino integration, and MATLAB/Simulink interface for data processing and monitoring.

4.1 Hardware Setup

The system operates on a development board named the NVIDIA Jetson Orin Nano, with Ubuntu 20.04 and ROS2 Humble. Powered via a USB interface, a Slamtec RPLIDAR A2M8 provides distance measurement data. Visual perception is through a stereo camera (IMX219), transmitting image streams through the CSI interface. Control instructions are transmitted to an Arduino board via a CAN interface.

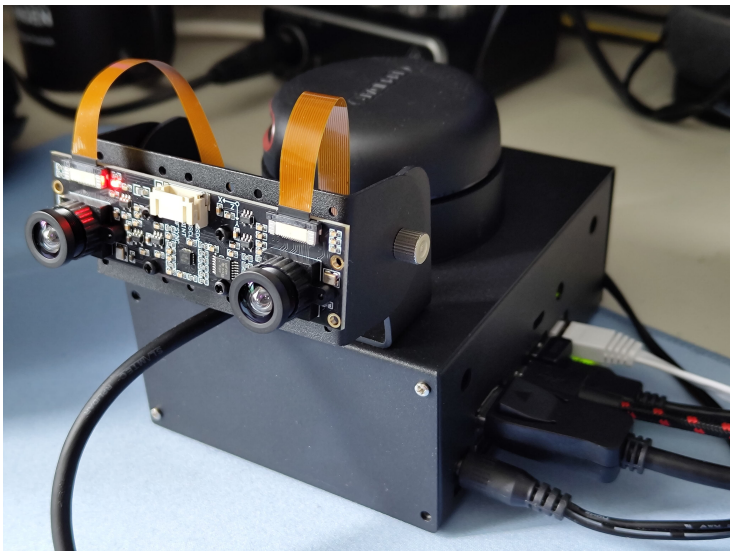

Figure 3: Stereo camera setup with dual IMX219 camera module and RPLiDAR A2 sensor mounted on a Waveshare Jetson Nano Metal Case (C).

4.2 ROS2 Environment

The selection of ROS2 Humble was largely due to its long-term support (LTS) status and because it supports both the Jetson Orin Nano platform and MATLAB. Sensor nodes were built using Python and the `rclpy` library. For added convenience in data processing, visualization, and prototyping, MATLAB was utilized to subscribe to pertinent ROS2 topics. Additionally, a custom CAN bridge node was implemented to properly format and forward control signals to the Arduino microcontroller.

4.3 ROS2 Node Architecture

The architecture consists of the following ROS2 nodes:

- **LIDAR_Node**: Talks to the RPLIDAR A2M8 through Slamtec's SDK. Publishes laser scan data to the /scan topic.
- **CAM_Node**: Takes the left and right visual inputs from the stereo camera and publishes them to `/raw_image_left` and `/raw_image_right`.
- **CAN_Node**: Gathers pertinent processed data and organizes it into CAN messages for transmission.

Encapsulation and modularity are made possible by this node structure. Because each node has a single function and can be swapped out, upgrading sensors or interfaces is simple.

4.4 Arduino and CAN Integration

*

  *

4.5 MATLAB Integration

MATLAB was used as a ROS2 subscriber for the topics `/raw_image_left` and `/raw_image_right` to facilitate visualization and processing of image streams coming from the stereo camera. One of the key motivations for the use of MATLAB in this context was its simplicity, especially when compared to more complicated development of an integrated image subscriber pipeline with Python, involving the use of `cv_bridge` and OpenCV.

MATLAB features a unique ROS Toolbox that fully supports ROS2 Humble and lets fast creation of subscriber nodes and automatic message handling possible. The toolbox streamlines many low-level complexities so that developers may concentrate on data processing ideas rather than interfacing and message conversions.

The integration process with ROS2 nodes, which are based on Python, was easily achieved. On initiation of the publishing of image topics by the ROS2 node on the Jetson Nano, MATLAB could sense and subscribe to such topics using easy commands. The data could then be observed in real time or used as input for additional prototyping as well as algorithmic refinement within the MATLAB environment.

In addition to the main integration with the Jetson Orin Nano, cross-platform interoperability was further evaluated through a series of practical tests. On a macOS system, a dedicated ROS2 publisher node that streamed live video from the built-in webcam was developed using Python and OpenCV. When MATLAB successfully subscribed to the published image topic via the ROS Toolbox while operating on the same computer, it was verified that ROS2 nodes built on non-Linux platforms are fully compatible with MATLAB's subscription interface.

Apart from the primary integration with the Jetson Orin Nano, cross-platform interoperability was also assessed by means of several pragmatic tests. Using Python and OpenCV, a dedicated ROS2 publisher node was built on a macOS system and streams real-time video from the built-in webcam. Running on the same machine, MATLAB effectively subscribed to the published image topic using the ROS Toolbox, so verifying that ROS2 nodes developed on non-Linux platforms are entirely compatible with MATLAB's subscription interface.

4.6 Reusability and Modularity

Every module in the system was built to work independently. For example, swapping out the LiDAR model or reconfiguring the stereo camera needs modification only to the respective node, hence not impacting the rest of the system. In the same way, communication protocols (e.g., changing from CAN to UART) can be modified by modifying only the respective communication node. The modular design represents a huge advantage that comes with the use of ROS2, supporting the system's ability to scale seamlessly or adapt to different hardware setups without requiring an overall redesign.

5 Experiments and Results

This section presents the visual results of the experiments that were carried out to verify that the modular ROS2 system was operating correctly and that its components were interacting with one another.

5.1 LiDAR Data Acquisition

The first test consisted of launching the custom Python ROS2 node that publishes data from the RPLIDAR A2M8. The LiDAR node was run on the Jetson Nano, and data was visualized using `rviz2` 4.

Figure 4: Visualization of `/scan` topic data from the RPLIDAR A2M8 using RViz2. The environment and object contours are correctly mapped in 2D.

5.2 Stereo Camera Image Reception in MATLAB and Python

The stereo camera publishes left and right image streams on the ROS2 topics `/image_raw_left` and `/image_raw_right` using a Python script. These streams were accessed using two different approaches: MATLAB for rapid debugging and Python for object detection.

Image Subscription and Visualization in MATLAB

Using the ROS2 Toolbox in MATLAB, the topics were subscribed and displayed using only a few lines of code 5. The developer was able to concentrate on image processing instead of low-level data conversion because the toolbox took care of message decoding internally. This method is especially useful when developing and debugging.

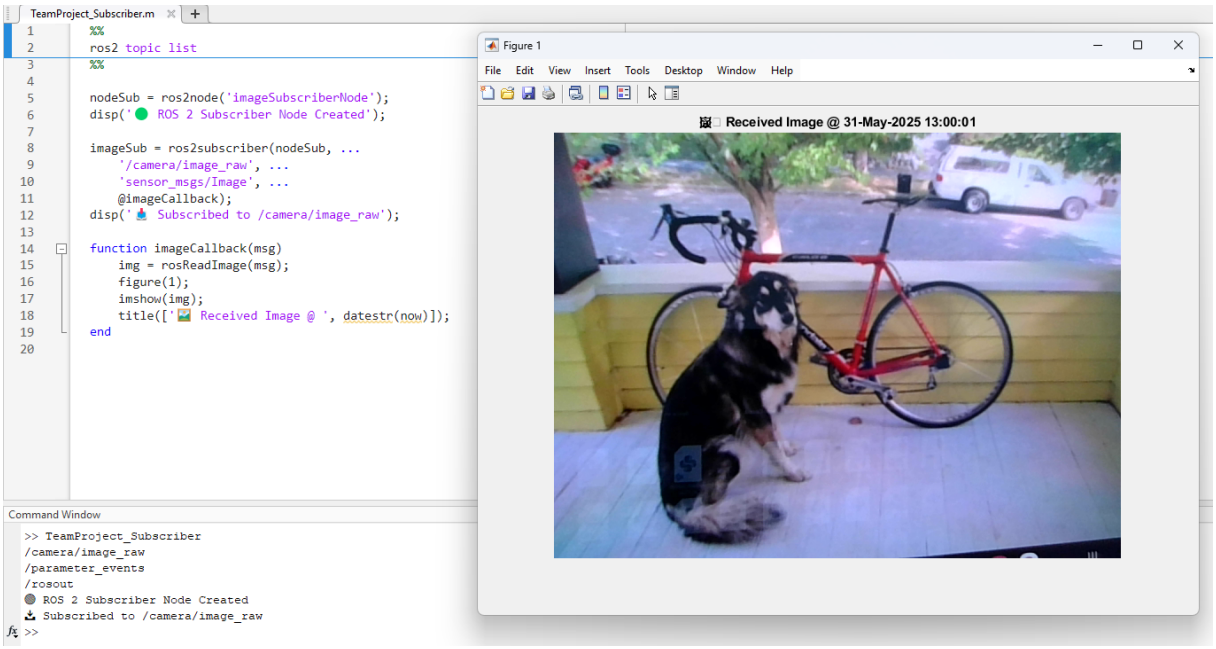

Figure 5: MATLAB subscribing to the ROS2 topic `/image_raw` and displaying the incoming image from the stereo camera.

Image Subscription and Visualization in Python

In parallel, a separate Python script was developed to subscribe to the same image topics and perform real-time object detection using the YOLOv5 algorithm. The image messages were decoded using `cv_bridge`, and detections were rendered over the original images using OpenCV 7.

This script enables a more advanced application layer where perception modules can be integrated directly into the ROS2 ecosystem. Detected objects can later be published to a new topic or influence decision-making logic within the system.

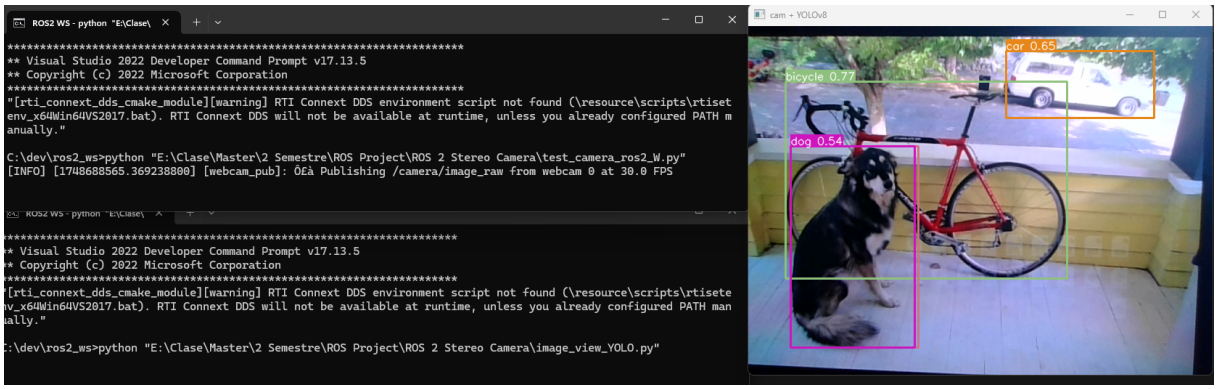

Figure 6: YOLO-based object detection applied in real time to images received from a single sensor of the stereo camera using a custom Python ROS2 node.

Figure 7: YOLO-based object detection applied in real time to images received from both sensors of the stereo camera using a custom Python ROS2 node.

5.3 CAN Message Transmission

*

*

5.4 System-Level Test

Finally, an integrated system test was conducted to verify that the LiDAR and camera nodes could operate simultaneously, data could be transmitted over CAN, and MATLAB could subscribe to all relevant topics 8.

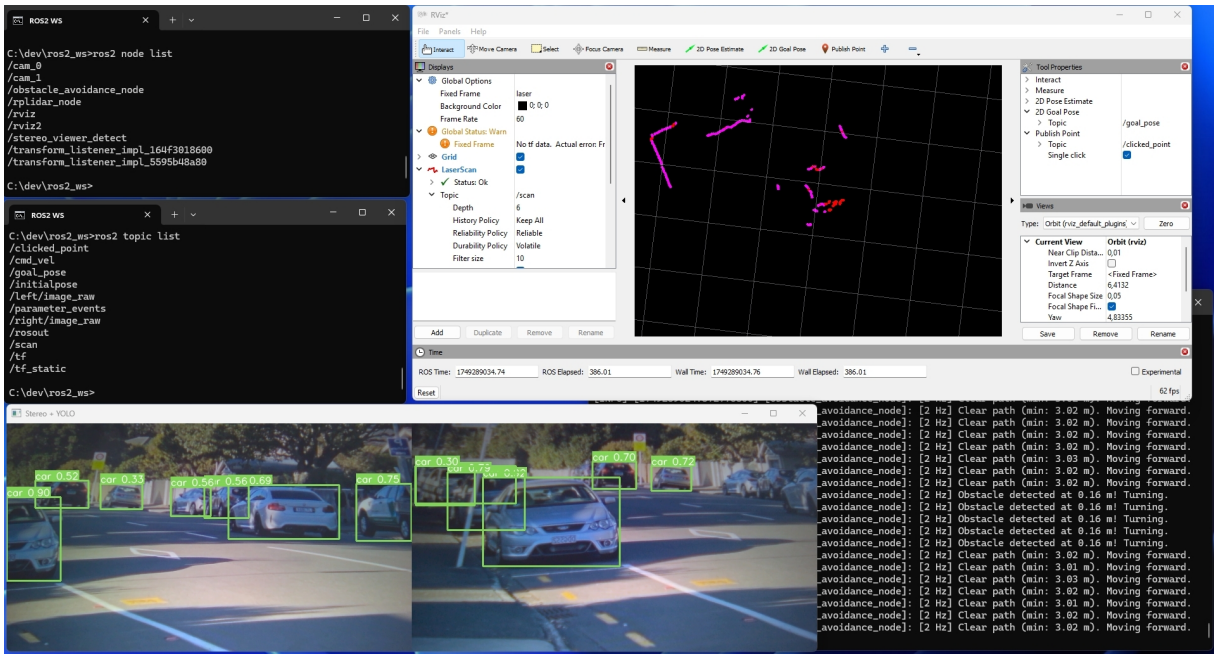

Figure 8: Full system test with all modules running: LiDAR, stereo camera, ROS2 nodes, CAN interface, Arduino control, and MATLAB subscriptions.

6 Conclusion and Future Work

This research successfully combined LiDAR and stereo camera sensors into a small autonomous driving platform based on a modular ROS2 environment. MATLAB provided a rapid and easy environment for

data analysis and prototyping, whilst Python was utilized to create ROS2 nodes and ensure effective interface with hardware devices such as the stereo camera and RPLIDAR.

One important aspect of the strategy was the modularity made possible by ROS2; every part, from sensors to actuators, was positioned inside separate nodes. With this architecture, future enhancements like swapping out the LiDAR, modernizing the image processing technique, or changing the communication protocol can be made without requiring significant architectural modifications.

The ADMIT14 project's future advancements will focus on further enhancing the system's functionality. Modules for real-time perception and decision-making, sensor fusion methods (like merging LiDAR and camera data), and SLAM algorithms for mapping and localization are added to the platform to increase its autonomy. Additionally, future development will be more dependable and scalable if the system model is formalized in System Composer with traceable requirements and computational performance is optimized on embedded hardware such as the Jetson Nano.

Project Repository

The full source code, implementation scripts, and tutorials for this project are available at the following GitHub repository:

`https://github.com/snchz46/ASM_ROS2_Team_Project_2025`

References

[1] Sanchez, S. (2025). *ROS2-ADMIT14 Platform Integration*. GitHub repository. Available at: `https://github.com/snchz46/ASM_ROS2_Team_Project_2025`

[2] Open Robotics. *ROS 2 Documentation*. Available at: `https://docs.ros.org/en`

[3] MathWorks. *MATLAB Help-Center: ROS 2*. Available at: `https://de.mathworks.com/help/ros/ug/get-started-with-ros-2.html`

[4] MathWorks. *Get Started with ROS 2 in Simulink*. Available at: `https://de.mathworks.com/help/ros/ug/get-started-with-ros-2-in-simulink.html`

[5] MathWorks. *Generate a Standalone ROS Node from Simulink*. Available at: `https://de.mathworks.com/help/ros/ug/generate-a-standalone-ros-node-from-simulink.html`

[6] MathWorks. *Generate Code to Manually Deploy a ROS 2 Node from Simulink*. Available at: `https://de.mathworks.com/help/ros/ug/generate-code-to-manually-deploy-ros-2-node.html`

[7] Slamtec. *RPLIDAR A2 Support Page*. Available at: `https://www.slamtec.com/en/Support#rplidar-a-series`

[8] Slamtec. *RPLIDAR A2 ROS 2 Package*. Available at: `https://github.com/Slamtec/rplidar_ros/tree/ros2`

[9] NVIDIA. *Jetson Nano Developer Kit - Getting Started*. Available at: `https://developer.nvidia.com/embedded/learn/get-started-jetson-nano-devkit`

[10] Q-engineering. *Installing Ubuntu 20.04 on Jetson Nano*. Available at: `https://qengineering.eu/install-ubuntu-20.04-on-jetson-nano.html`

[11] Q-engineering. *Jetson Nano Ubuntu 20.04 Image (GitHub)*. Available at: `https://github.com/Qengineering/Jetson-Nano-Ubuntu-20-image?tab=readme-ov-file#installation`