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 for developing distributed and modular robotic applications. It has native support for real-time communication, cross-platform compatibility, and easy interfacing 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.

1 Introduction

Autonomous driving technologies essentially rely 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 every subsystem 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 that is tailored for the creation of 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 adaptation of the system afterward straightforward. 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 the implementation details. 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

The official NVIDIA Jetson Nano website provides an image of Ubuntu 18.04, which is not recommended for ROS2 functionality. Further research was required to find a suitable Jetson Nano image that is running Ubuntu 20.04, making ROS2 Foxy deployment possible.

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

During the utilization of the RPLIDAR A2M8 sensor, it was observed that the official `rplidar_ros` package from Slamtec was incompatible with the Windows operating system. Due to this compatibility issue, direct use of the RPLIDAR on a Windows system was not possible. The only viable solution was to execute the official package within an Ubuntu environment.

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

Figure 1 shows the modular framework employed in this project. The system architecture 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`.

Then, the Jetson 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, thereby illustrating the flexibility fostered by ROS2 in modular architectures.

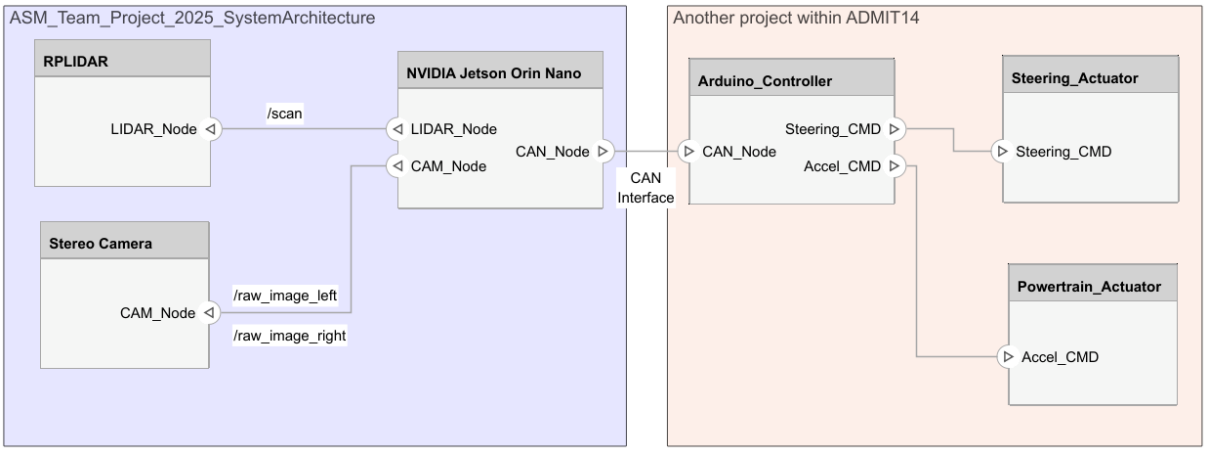

Figure 1: Proposed System Architecture showing sensor input, microcontrollers, and actuation modeled in System Composer.

4 Implementation

This chapter presents an overview of the practical implementation of the system, encompassing the hardware setup, 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.

4.2 ROS2 Environment

The selection of ROS2 Humble was due largely to its Long-Term Support (LTS) status and because it supports both the Jetson Orin Nano platform and MATLAB. The sensor nodes were built using Python and the `rclpy` library. For extra data processing, visualization, and prototyping convenience, MATLAB was utilized to subscribe to related 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 is comprised 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.

This node structure allows modularity and encapsulation. Every node serves one purpose and is interchangeable, allowing for ease of sensor or interface upgrades.

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 visualize and process image streams captured from the stereo camera. A key reason for employing MATLAB here was its simplicity, particularly in comparison to the more complex construction of a combined image subscriber pipeline in Python with `cv_bridge` and OpenCV.

MATLAB has a special ROS Toolbox that fully supports ROS2 Humble and enables the fast creation of subscriber nodes and the automatic handling of messages. The toolbox simplifies many low-level intricacies, thereby enabling developers to concentrate on data processing notions instead of struggling with interfacing and message conversions.

The integration process with ROS2 nodes, which are based on Python, was attained effortlessly. 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.

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 experiments carried out to validate the proper functioning of the modular ROS2 system and the interaction between components, along with the visual results obtained.

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 were visualized using `rviz2`.

Figure 2: 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. The toolbox handled message decoding internally, allowing the developer to focus on image processing rather than low-level data conversion. This approach is particularly convenient during development and debugging phases.

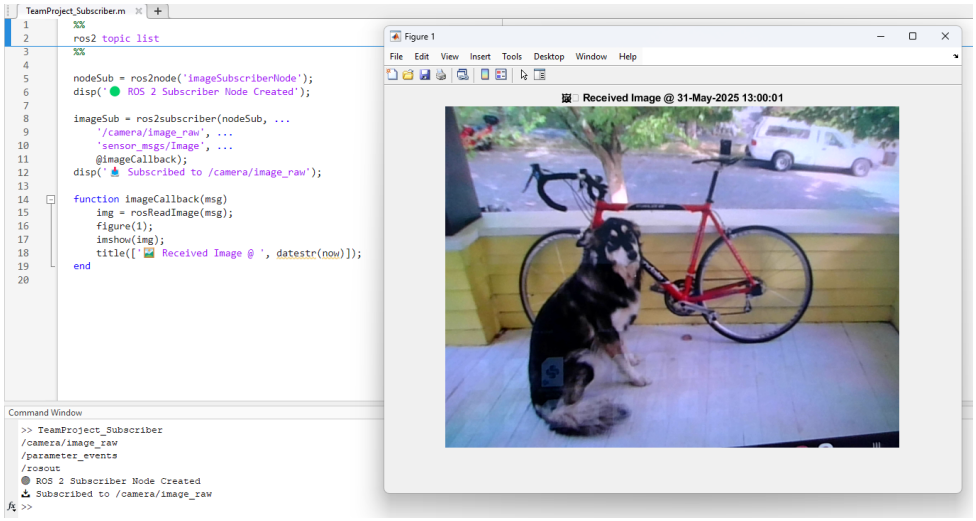

Figure 3: MATLAB subscribing to the ROS2 topic `/image_raw` and displaying the incoming image from the stereo camera.

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.

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.

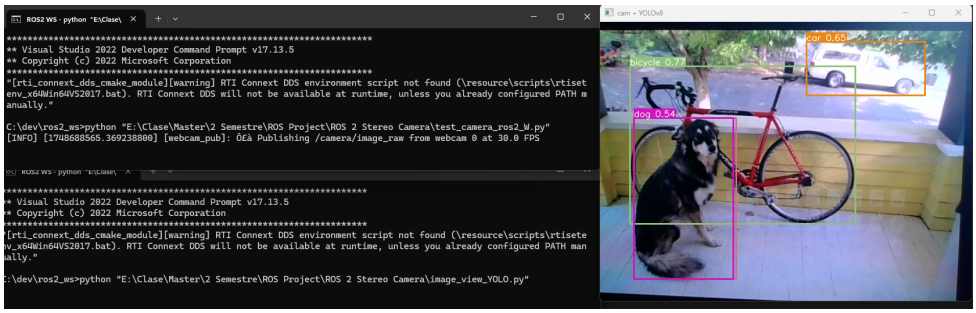

Figure 4: YOLO-based object detection applied in real time to images received from the stereo camera using a custom Python ROS2 node.

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.

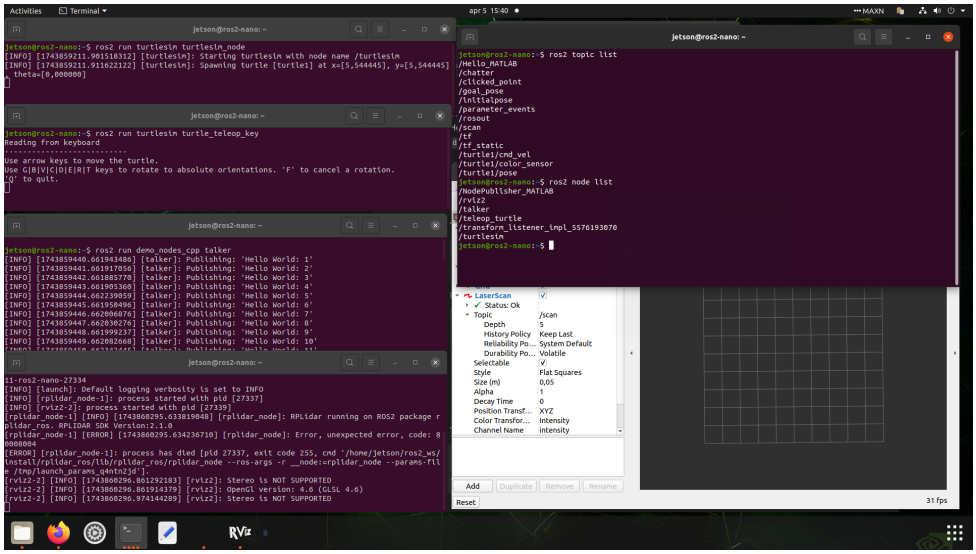

Figure 5: Full system test with all modules running: LiDAR, stereo camera, ROS2 nodes, CAN interface, Arduino control, and MATLAB subscriptions.

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.

ROS2 enabled modularity, which was a significant feature of the technique; each component, from sensors to actuators, was placed within distinct nodes. Future improvements, such as replacing the LiDAR,

updating the image processing method, or altering the communication protocol, can be implemented with this architecture without requiring large architectural changes.

Future developments will concentrate on expanding the system's functionality even more within the framework of the ADMIT14 project. This includes implementing sensor fusion techniques (for example, integrating LiDAR and camera data), incorporating SLAM algorithms for mapping and localization, and increasing the platform's autonomy through real-time perception and decision-making modules. Furthermore, more dependable and scalable development will be facilitated in the future by formalizing the system model in System Composer with traceable requirements and maximizing computational performance on embedded hardware such as the Jetson Nano.

References

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

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

[3] 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`

[4] 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`

[5] 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`

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

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

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

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

[10] 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`