Modular Sensor-testing Framework for Autonomous Rail Vehicles

Proposal

David Christian Horn

Bachelor Theses
March 2026

Real-Time/Embedded Systems
Department of Computer Science
Kiel University

Advised by
Prof. Dr. Reinhard von Hanxleden, Dr.-Ing. Alexander Schulz-Rosengarten

Contents

List of Acronyms

ROS2 Robot Operating System. 3

SIL Software in the Loop. 1

Chapter 1

Proposal for Modular Sensor-testing Framework for Autonomous Rail Vehicles

1.1 The Problem

In the DataTrain project, we are developing an autonomous rail vehicle, whose perception, control, and decision logic must be tested under realistic and reproducible virtual conditions. To achieve this, we are creating a Software in the Loop (SIL) environment that allows us to simulate real-world scenarios, including the physical characteristics of the train, the track, and the surrounding environment, within a 3D simulation framework.

The overarching goal is to provide a simulation-based testbed that reduces the need for costly and error-prone real-world testing, while increasing reproducibility, safety, and scalability for future research within the DataTrain and REAKTOR projects.

The focus of this Bachelor project is the implementation of a deployment and testing architecture that enables a model rail vehicle to be equipped with different types and configurations of virtual sensors, and to evaluate the impact of these sensor setups on obstacle detection under various conditions. We are trying to find and implement a modular Solution, which is easily reproducible, maintainable and allows for future extension to be build upon.

The Software-in-the-Loop approach ensures that perception and control components can be tested early in the development cycle, before any hardware prototypes are available.

1.2 Planned Solutions and Goals: Include Visualization, Implementation, Further tasks

The primary goal of this work is to design and implement a pipeline that enables the visualization and simulation of a rail vehicle on tracks in different environments. This setup should be both interactive and debuggable, while also allowing for headless batch processing to systematically verify and evaluate results. The project combines 3D simulation with simplified physical modeling to achieve a balance between realism and computational efficiency. Additionally, the virtual sensor setups must be easily testable and extensible to support future research within the DataTrain project.

1. Proposal for Modular Sensor-testing Framework for Autonomous Rail Vehicles

1.2.1 Hard Goals

1. Design a modular, ROS 2-based architecture that supports plug-and-play integration of at least two virtual sensors
2. Implement a rail vehicle on a simple track with basic dynamic behaviors (acceleration, deceleration, stop).
3. Define standardized ROS 2 message types for sensor topics and simulation feedback to ensure reusability.
4. Develop a simulation management node that enables parameterized testing across both obstacle and environmental scenarios (e.g., light, fog, rain).
5. Introduce a quantitative system for evaluating detection sensor fusion performance, (maybe inspired by prior DataTrain work (e.g., Rail Score / Obstacle Score)).

1.2.2 Evaluation Metrics

To measure the performance of the developed system, the following metrics will be applied:

▻ **Detection accuracy:** precision, recall, and F1-score across simulated scenarios.

▻ **Latency:** average perception-to-decision delay measured in simulation time.

▻ **False detections:** analysis of false positives and negatives under different weather and lighting conditions.

These metrics can be extended in future work to include computational performance or system robustness.

1.2.3 Soft Goals

1. Extend the framework to include additional virtual sensors (e.g., radar, ultrasonic).
2. Implement a minimal user interface for scenario configuration.
3. Enable real-time visualization of perception results in RViz2.
4. Add optional actuation logic (e.g., PID-based braking control).
5. Improve the physical modeling of the train dynamics.
6. Reduce overall usability complexity to simplify SIL testing workflows.
7. Containerize the entire setup for reproducible deployment (Docker-based).

The following goals define the scope of this work and serve as milestones for the project's completion within the thesis timeframe.

1.3 Technical Approach

subsection Preliminary Tooling The implementation relies on the following core technologies:

- **ROS 2:** Robotics middleware providing inter-process communication and modular node orchestration.
- **rclpy:** The canonical Python API for ROS 2, used to implement nodes and manage message passing.
- **Gazebo:** Open-source 3D simulation tool offering realistic physics, sensor, and lighting models.
- **ros_gz_bridge:** Middleware bridge enabling communication between Gazebo's internal transport system and ROS 2 topics.

To achieve the defined goals, the project adopts a Robot Operating System (ROS2)-based architecture as the middleware for communication between simulated components. ROS2 (Robot Operating System 2) is the de facto standard for modern robotics research and development, offering modularity, interoperability, and scalability through standardized communication interfaces such as topics, services, and actions. It is widely used in both academia and industry for autonomous systems, including self-driving vehicles, mobile robots, and industrial automation.

Using ROS2 ensures that the developed system follows state-of-the-art robotics practices and remains compatible with future extensions or integration into larger projects. The implementation will be primarily carried out in Python, using the `rclpy` client library for rapid prototyping and testing, while allowing the integration of custom analysis and evaluation tools.

1.3.1 Simulation Environment

For the simulation environment, the project employs **Gazebo** a 3D physics simulator that integrates seamlessly with ROS2.

Gazebo provides realistic simulation of train dynamics, track geometries, and sensor models, while supporting the inclusion of physical effects such as noise, occlusion, friction, and weather.

This combination enables the controlled evaluation of perception and control layers within a fully virtual, reproducible environment.

1.3.2 System Architecture

The system is structured as a set of modular ROS2 nodes, each responsible for a distinct functional aspect of the simulation. This modular design supports extensibility and independent testing of subsystems.

1. **Simulation Manager Node:** Configures and launches experiments, sets scenario parameters (e.g., fog density, lighting, obstacle placement), and manages batch or headless test runs.
2. **Virtual Sensor Nodes:** Simulate different sensing modalities such as camera, LiDAR, or radar using Gazebo's native sensor plugins and customizable noise or range models. Each sensor publishes standardized ROS 2 topics for easy substitution and comparison.
3. **Perception Node:** Performs obstacle detection and optional sensor fusion using deep-learning models (e.g., YOLO-based detection) or rule-based fusion logic. Outputs are published as structured detection messages.
4. **Scoring and Evaluation Node:** Computes quantitative performance metrics such as precision, recall, and latency, to generate comparable evaluation outputs.
5. **Visualization Node:** Provides live visualization of sensor and perception data in RViz2, and optionally exports logged data to `rosbag2` or CSV files for offline analysis.

1.3.3 Implementation Details

All nodes and dependencies will be encapsulated in Docker containers to ensure reproducibility, platform independence, and consistent version control. This setup allows the complete system to be deployed on different machines or shared across research environments without configuration conflicts.

The system will be validated through scenario-based testing, where obstacle configurations and environmental conditions are varied systematically to analyze detection accuracy and latency.

Future extensions may include additional sensor modalities, or optional control components such as a simple PID-based train actuation module.

1.3.4 Debugging, Logging, and Evaluation Tooling

In order to validate and iteratively improve the SIL framework, comprehensive debugging and logging tools are essential. The proposed workflow combines online visualization, structured data logging, and offline analysis.

To support iterative development and reproducible analysis, the system includes a dedicated observation and metrics layer:

RViz2 Live Debugging. A pre-configured RViz2 layout visualizes raw and processed data in real time, possibly : Camera stream, LiDAR point clouds, rail segmentation masks, TF frames and so on.

Metrics & Logging Node. A dedicated node subscribes to perception outputs and publishes aggregate metrics . After each run, metrics are written to CSV together with the exact scenario configuration and code revision for traceability.

Rosbag2 Recording. All experiments are recordedad as Bags capturing all parameters (e.g., weather, lighting, obstacle type, seed).

Offline Analysis. Two complementary paths are supported: (i) Foxglove Studio dashboards for interactive inspection of recorded data; (ii) Jupyter notebooks for statistical analysis and publication-grade plots. Bags are converted into pandas DataFrames to compute condition-wise performance (e.g., precision/recall under fog/low-light) and latency distributions

Topic and Message Conventions. Standard message types are used where possible (`sensor_msgs`, `vision_msgs`, `diagnostic_msgs`); all messages carry simulation timestamps for synchronized fusion (ApproximateTime).

Reproducible Tooling. All components (simulation, perception, metrics, visualization, notebooks) are containerized via Docker Compose, with persistent volumes for `bags/` and `reports/`. This enables headless batch runs and consistent environments across machines.

Chapter 2

Schedule

2.1 Schedule

The planned work period for the thesis extends from **3 November 2025** to **30 March 2026**. The active implementation and testing phase is expected to conclude by **20 March 2026**, leaving the final week for report completion and revisions. Table 2.1 outlines the proposed timeline with milestones and deliverables.

Table 2.1. Planned schedule for implementation and documentation.

Phase	Timeframe	Objectives and Deliverables
1. Project Setup & Planning	03.11. – 10.11. 2025	Finalize requirements, prepare ROS 2 workspace, configure development environment (ROS 2, Gazebo, Docker). Set up repository structure and initial documentation.
2. Simulation Base Setup	11.11. – 25.11.2025. 2025	Implement a simple Gazebo scene with track and basic train model. Establish ROS–Gazebo bridge and verify message exchange. Define coordinate frames (TF tree) and basic motion behavior (accelerate, stop).
3. Sensor Integration	26.12. – 12.01. 2026	Integrate at least two virtual sensors (e.g., camera, LiDAR). Publish sensor data through standardized ROS 2 topics. Validate sensor parameters, noise models, and visualization in RViz2.
4. Perception & Detection Node	13.01. – 20.01. 2026	Develop a perception node that processes sensor data for obstacle detection (baseline model, rule-based or learned). Output structured detections and verify synchronization with TF frames.
5. Evaluation and Scoring System	21.01. – 27.01. 2026	Implement scoring and evaluation node. Compute quantitative metrics (precision, recall, latency, false detections). Introduce data logging to CSV and initial plotting of results.
6. Simulation Management & Batch Testing	3.02. – 10.02. 2026	Create scenario manager node for parameterized runs (weather, lighting, obstacles). Enable headless batch execution and reproducible configuration storage. Record experiments with rosbag2.
7. Debugging & Visualization Layer	11.02. – 18.02. 2026	Configure RViz2 layouts and Foxglove dashboards. Test offline analysis pipeline (Jupyter notebooks for metrics evaluation). Refine logging and container setup.
8. Final Integration & Documentation	19.02. – 28.02. 2026	Conduct final integration tests, verify reproducibility, and clean up codebase. Prepare user documentation and internal Wiki pages.
9. Thesis Writing & Submission	1.03. – 30.03. 2026	Finalize written report, figures, and references. Prepare presentation slides and submit thesis.