UNIVERSIDAD DE MÁLAGA

ESCUELA DE INGENIERÍAS INDUSTRIALES

TRABAJO FIN DE GRADO

DESARROLLO DE UN ENTORNO DE SIMULACIÓN UNITY-ROS2 PARA LA EVALUACIÓN SLAM DE VEHÍCULOS AUTÓNOMOS.

GRADO EN INGENIERÍA
ELECTRÓNICA, ROBÓTICA Y MECATRÓNICA

SUPERVISOR:	JUAN MANUEL GANDARIAS PALACIOS
CO-SUPERVISOR:	ALONSO LLORENTE
KNOWLEDGE AREA:	INGENIERÍA DE SISTEMAS Y AUTOMÁTICA

LUIS ARCE ARANDA
MÁLAGA, JANUARY 15, 2025

ESCUELA DE INGENIERÍAS INDUSTRIALES, UNIVERSIDAD DE MÁLAGA

DESARROLLO DE UN ENTORNO DE SIMULACIÓN UNITY-ROS2 PARA LA EVALUACIÓN SLAM DE VEHÍCULOS AUTÓNOMOS.

Autor: Luis Arce Aranda

Tutor: Juan Manuel Gandarias Palacios

Cotutor: Alonso LLorente

Departamento: Ingeniería de Sistemas y Automática

Titulación: Grado en Ingeniería Electrónica, Robótica y Mecatrónica

Palabras clave: SLAM, marco de simulación, Unity, ROS2, sensores LiDAR, simulación de lluvia, sistemas de percepción, condiciones ambientales, robótica autónoma.

Resumen

Este proyecto tiene como objetivo desarrollar un entorno de simulación realista utilizando Unity y ROS2 para la evaluación de algoritmos de SLAM en vehículos autónomos. La simulación incorpora sensores como LiDAR e IMU, además de condiciones ambientales adversas, como lluvia, para evaluar la robustez de los sistemas de percepción y navegación. Se diseñaron escenarios específicos que emulan entornos complejos y dinámicos, como canales urbanos, permitiendo pruebas exhaustivas de los sistemas autónomos en condiciones controladas.

Los resultados incluyen un modelo de simulación de lluvia que introduce ruido y atenuación en las mediciones de LiDAR, mejorando la validación de algoritmos de percepción. Este marco de simulación no solo permite evaluar la precisión y fiabilidad de los sistemas SLAM, sino que también proporciona datos sintéticos para la mejora de estos algoritmos, contribuyendo al desarrollo de sistemas autónomos más robustos y preparados para entornos reales.

SCHOOL OF INDUSTRIAL ENGINEERING, UNIVERSITY OF MÁLAGA

DEVELOPMENT OF A UNITY-ROS2 SIMULATION ENVIRONMENT FOR SLAM EVALUATION FOR AUTONOMOUS VEHICLES

Author: Luis Arce Aranda

Supervisor: Juan Manuel Gandarias Palacios

Co-supervisor: Alonso LLorente

Department: Systems and Automation Engineering

Degree: Bachelor's Degree in Electronics, Robotics, and Mechatronics Engineering

Keywords: SLAM, simulation framework, Unity, ROS2, LiDAR sensors, rain simulation, perception systems, environmental conditions, autonomous robotics.

Abstract

This project aims to develop a realistic simulation environment using Unity and ROS2 for evaluating SLAM algorithms in autonomous vehicles. The simulation incorporates sensors such as LiDAR and IMU, as well as adverse environmental conditions, like rain, to assess the robustness of perception and navigation systems. Specific scenarios emulating complex and dynamic environments, such as urban canals, were designed to enable comprehensive testing of autonomous systems in controlled conditions.

The results include a rain simulation model that introduces noise and attenuation in LiDAR measurements, enhancing the validation of perception algorithms. This simulation framework not only enables the evaluation of the accuracy and reliability of SLAM systems but also provides synthetic data to improve these algorithms, contributing to the development of more robust autonomous systems prepared for real-world environments.

Acronyms

ADAS	Autonomous Driving Assisted Systems
AMCW	Amplitude Modulated Continuous Wave
API	Application Programming Interface
APDs	Avalanche Photodiodes
AR	Augmented Reality
ASCII	American Standard Code for Information Interchange
AURORA	Aquatic Unique Research and Operating platform for Radio Applications
BGN	Bayesian Graph Network
CAD	Computer-Aided Design
CP	Closest Point
DDS	Data Distribution Service
DLR	German Aerospace Center
DR	Detection Rate
DT	Detection Threshold
EKF	Extended Kalman Filter
EU	European Union
FFT	Fast Fourier Transform
FDR	False Detection Rate
FMCW	Frequency Modulated Continuous Wave
FOV	Field of View
FPA	First Peak Averaging

GIS	Geographic Information System
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
HDRP	High Definition Render Pipeline
IMU	Inertial Measurement Unit
IWT	Inland Waterway Transport
KITTI	Karlsruhe Institute for Technology and Toyota Dataset
KPIs	Key Performance Indicators
LAS	LASer File Format
LAZ	Compressed LAS File Format
LiDAR	Light Detection and Ranging
MAPE	Mean Absolute Percentage Error
MB	Megabyte
ML	Machine Learning
MSS	Multi-Sensor Systems
NIR	Near-Infrared
OOI	Object of Interest
PCD	Point Cloud Data
PCL	Point Cloud Library
PLY	Polygon File Format
PNT	Position, Navigation and Timing
PPP	Precise Point Positioning
QGIS	Quantum Geographic Information System
RADAR	Radio Detection and Ranging
RAM	Random Access Memory
RE	Returned Energy
RGB	Red Green Blue

RIO	Returned Intensity of Object
RIRD	Returned Intensity of RainDrop
RMSE	Root Mean Square Error
ROS	Robot Operating System
ROS 2	Robot Operating System 2
ROSSharp	ROS for Unity Framework
RTK	Real-Time Kinematics
RViz	ROS Visualization
SLAM	Simultaneous Localization and Mapping
SNR	Signal-to-Noise Ratio
SONAR	Sound Navigation and Ranging
SSIM	Structural Similarity Index Measure
TCP	Transmission Control Protocol
TFG	Final Degree Work
TFM	Master's Final Project
UMA	University of Malaga
URDF	Unified Robot Description Format
URP	Universal Render Pipeline
VLP	Velodyne LiDAR Puck
VR	Virtual Reality
VRS	Virtual Rain Simulator
XR	Extended Reality

Contents

List of Figures

List of Tables

Part I

Introduction

Chapter 1

Introduction and Overview

Contents

1.1 Introduction

This bachelor thesis focuses on developing a rain simulation pipeline specifically designed for LiDAR scans, aiming to create a greater variety of situational samples that can be used to evaluate intelligent transportation systems. The pipeline integrates a Unity-based simulator for generating realistic LiDAR data with a C++ framework that introduces rain-induced noise into the scans. By enhancing the Unity simulator and incorporating rain as an additional source of environmental variability, the system enables more comprehensive testing of SLAM and perception algorithms. The pipeline's functionality is validated in both simulated and real-world scenarios, contributing to the advancement of robust autonomous technologies.

Beyond improving the robustness of perception and navigation algorithms, advancements in autonomous systems bring broader societal benefits. Automation enhances traffic efficiency, reduces emissions, and improves accessibility for individuals with mobility challenges. However, as autonomous systems continue to evolve, ensuring their reliability in diverse and dynamic environments remains a critical research focus. This work contributes to this overarching goal by addressing specific challenges related to perception and navigation under adverse environmental conditions.

An essential element in enabling autonomous systems to operate effectively in such environments is their capacity to perceive and navigate with precision. SLAM is a key technology in this regard. SLAM enables autonomous systems

to construct and update maps of their surroundings while simultaneously determining their position within them. This capability is particularly vital in scenarios where GPS signals are unavailable or unreliable, such as indoor environments or areas with dense foliage.

For SLAM algorithms to be effective, they must demonstrate robustness against diverse and challenging real-world conditions, including dynamic obstacles, changing lighting, and adverse weather. Among these environmental challenges, weather conditions such as rain and fog pose significant difficulties for autonomous systems, necessitating the development of algorithms that can perform reliably under such circumstances. Advances in simulation environments are essential to systematically study and address these challenges.

In parallel with algorithmic developments, recent decades have witnessed significant progress in sensor technology. Innovations such as mass-market LiDAR, MIMO RADAR, and event cameras have revolutionized the capabilities of autonomous systems [8, 9, 10].

LiDAR, in particular, has emerged as a cornerstone technology for autonomous systems. Its ability to generate high-resolution 3D maps of the environment surpasses that of visual or radar technologies, especially under conditions of low visibility, such as fog or rain [11]. However, the sensitivity of LiDAR to environmental factors underscores the necessity for robust evaluation frameworks.

Integrating these sensors into perception systems also presents challenges, including data fusion, computational costs, and environmental sensitivity. Addressing these challenges is critical to fully leveraging the potential of advanced sensors in autonomous systems.

Despite these advancements, the evaluation of autonomous systems, particularly in the context of SLAM, continues to face significant hurdles. While SLAM methods have achieved remarkable advancements, their evaluation often lacks the rigor needed for widespread deployment in real-world scenarios. Real-world testing, though invaluable, is limited by its inability to cover the full range of environmental conditions, particularly those involving adverse weather. For example, rain can introduce significant noise in LiDAR data, leading to errors in mapping and localization [12]. Additionally, relying solely on real-world testing is both time-consuming and resource-intensive, making it impractical for exhaustive evaluations. This limitation poses a significant challenge for researchers and industry stakeholders seeking to validate the robustness of SLAM algorithms under varying and extreme conditions. The lack of proper evaluation frameworks not only slows down innovation but also hinders the trust required for mass-market adoption of autonomous transport technologies.

To address this gap, simulation has emerged as a critical tool. By enabling controlled and repeatable testing environments, simulations provide the flexibility to evaluate SLAM methods under diverse scenarios, including those involving weather-related challenges.

A robust evaluation framework for SLAM systems must account for the inherent variability of real-world conditions. Monte Carlo simulation [13] provides an effective solution by generating a wide range of scenarios to test SLAM algorithms exhaustively. This statistical approach ensures that the system's performance is evaluated across a diverse set of conditions, offering insights into its reliability and robustness [14].

In response to these needs, this thesis aims at developping a rain model simulator for LiDAR scans. Rain introduces unique challenges for LiDAR sensors, such as signal attenuation, noise, and false detections [15]. Accurately modeling these effects in a controlled environment is essential for improving SLAM systems and ensuring their reliability in real-world scenarios.

The rain model simulator integrated with the simulation framework, enables exhaustive testing of SLAM methods under varying rainy conditions. By emulating the interactions between LiDAR signals and rain particles, this simulator provides valuable insights into the limitations and capabilities of current SLAM technologies. Furthermore, the generated datasets serve as a benchmark for developing and validating new SLAM algorithms tailored to adverse weather conditions.

This work contributes to the broader goal of creating robust evaluation frameworks for autonomous systems, bridging the gap between theoretical research and practical deployment.

1.2 Summary of Contributions

This thesis contributes to the advancement of LiDAR-based autonomous systems through the development of a realistic simulation environment and a rain simulation model. The work was structured around the following objectives:

1. **Create a simulation environment in Unity:** Design and implement a high-fidelity virtual framework to simulate realistic navigation scenarios.
2. **Integrate models and sensors, and configure scripts:** Ensure the correct setup of sensor models (LiDAR, IMU) and implement the necessary scripts for seamless operation.
3. **Validate Unity-ROS 2 communication for sensor data transmission:** Confirm the accurate transmission of data from Unity to ROS 2 for sensors like IMU and LiDAR.
4. **Develop a rain simulation model for LiDAR:** Simulate the effects of rain on LiDAR measurements, including noise, signal attenuation, and distortion, in a controlled virtual environment.
5. **Validate the environment and rain model:** Confirm the correct functionality of the simulation environment and assess the reliability of the rain model

through testing.

1.2.1 Organization of the Thesis

The work presented in this thesis was conducted over a period of 6 months, divided into three main parts.

Part I: Introduction
The chapters included in this part are:

- **Chapter 1: Introduction and Objectives** – Introduces the problem, explains the importance of the study, and defines the main objectives and structure of the thesis.
- **Chapter 2: Theoretical Background** – Reviews the theoretical foundations, including LiDAR technology, SLAM algorithms, and weather simulation techniques.

Part II: Project Development
This part describes the work carried out during the project. It is divided into three main chapters:

- **Chapter 3: Simulation Framework Development** – Details the creation of a high-fidelity simulation environment, the integration of sensors, and the implementation of a rain simulation model.
- **Chapter 4: Results and Analysis** – Presents the experimental results, evaluating the performance of the rain model and its impact on LiDAR data.

Part III: Conclusions and Future Work
The final part summarizes the key contributions and findings of the thesis, reflects on its limitations, and proposes directions for future research. The chapters included in this part are:

- **Chapter 5: Conclusions and Future Work** – Consolidates the outcomes of the study and suggests possible improvements and extensions for future research.

Appendices
The appendices provide additional information and supplementary materials relevant to the thesis. Among these, *Appendix B* is of particular importance, as it discusses the validation methodology for the rain model. While this methodology is still in the process of being completed, it lays the groundwork for evaluating the robustness and accuracy of the proposed system. The appendix includes preliminary insights and outlines the steps required to finalize the validation process.

Chapter 2

Theoretical Background

Contents

This chapter provides the necessary theoretical foundation to understand the concepts and technologies underpinning this thesis. It begins with an exploration of LiDAR technology, covering its operational principles, data representation, and common error models. The chapter then delves into LiDAR simulation techniques, presenting current methods for simulating sensor measurements, such as ray casting and machine learning-based approaches.

Furthermore, this chapter addresses rain simulation techniques, including physical models of rain, raindrop distributions, and their impact on LiDAR signals, particularly in terms of attenuation and noise.

By laying out this theoretical background, the chapter establishes the context and key principles that support the design, implementation, and validation of the simulation environment developed in this thesis.

2.1 LiDAR Technology

LiDAR technology consists in sending a laser beam to the target and measuring the reflected light with a photodetector to determine the distance to the target and in this way generating a precise map of the surrounding environment. The main advantages of LiDAR are that it can provide a precise position over large areas and that it is fast, making it possible to collect information with a speed and a degree of detail that would not otherwise be possible.

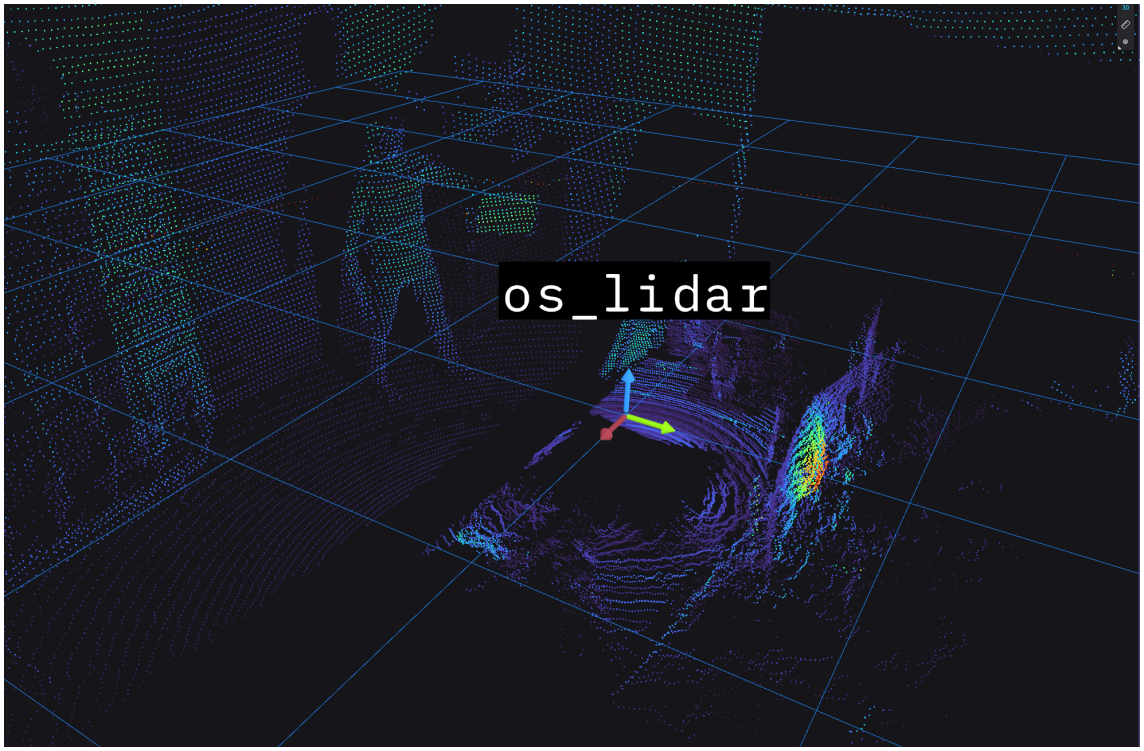


Figure 2.1: Example of LiDAR output illustrating intensity-based grading.

2.1.1 Distance measurement methods

LiDAR operates by emitting laser pulses and measuring the time it takes for them to return after reflecting off an object or surface. This process calculates distances, forming the basis for point cloud generation. While position and orientation of the LiDAR sensor can be useful for aligning scans in broader applications, the core operation of the sensor itself focuses on the distance to an object or surface, which can be determined using various measurement techniques.

One common method for distance measurement with LiDAR involves the use of a pulsed laser to determine the *time of flight* (ToF) [16] (see Figure 2.2). This refers to the time it takes for the emitted light to travel to a surface, reflect back, and return to the sensor. The distance can be calculated using the following

equation:

$$D = \frac{c \cdot \Delta T}{2} \tag{2.1}$$

where D is the distance to the target, c is the speed of light, and ΔT represents the measured time of flight.

This method is highly effective but relies on receiving a detectable return signal. For long-range measurements, high-powered lasers are required to ensure sufficient signal strength upon reflection.

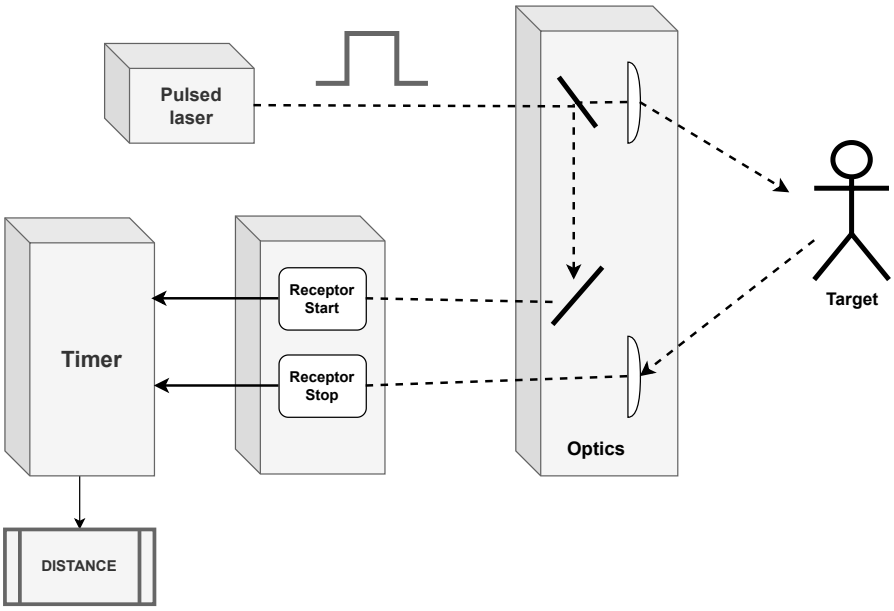


Figure 2.2: Time of Flight Working Principle. This diagram illustrates the operating mechanism of a ToF system, where a pulsed laser emits light towards a target. The reflected light is captured by optics, and the time taken for the light to travel to the target and back is measured using a timer. The system includes a receptor with start and stop signals that enable precise timing calculations, which are then used to compute the distance to the target.

Another method for calculating distances relies on measuring the phase shift. In this case, an amplitude-modulated continuous waveform (AMCW) laser is used [17] (see Figure 2.3). An AMCW laser emits light modulated at a specific frequency, often with a sinusoidal variation in intensity, though other modulation patterns, such as square or triangular waves, may also be used. The distance is determined by analyzing the phase difference between the emitted light and the light reflected back from the target. This phase difference corresponds to the time it takes for the light to travel to the target and back, which can be directly related to the distance.

$$D = \frac{c}{2} \cdot \frac{\Delta\Phi}{2\pi f_M} \tag{2.2}$$

where D and c are the distance to the object and the speed of light, respectively; $\Delta\Phi$ is the phase shift, and f_M is the modulation frequency of the signal amplitude.

The main drawback of this method is that the maximum measurable range without ambiguity is relatively short, typically limited to around 100 meters. This limitation occurs because the maximum measurable distance is tied to the wavelength of the modulation frequency ($\lambda_m = c/f_M$). When the phase shift exceeds 2π, the waves align again and become indistinguishable from their earlier positions, making it impossible for the system to detect how many full cycles have occurred. This results in ambiguity for longer distances.

Increasing the modulation frequency improves measurement precision but shortens the range at which distances can be measured without confusion. As a result, this method is best suited for short- to medium-range applications requiring high accuracy.

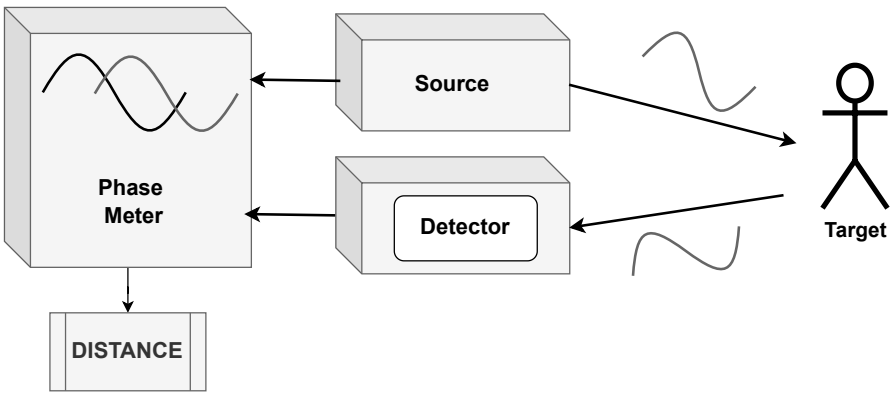


Figure 2.3: AMCW Working Principle. A source emits a modulated wave towards a target, and the reflected wave is captured by a detector. The phase shift between the emitted and received signals is measured using a phase meter. This phase difference is proportional to the distance traveled by the wave, allowing the system to calculate the target's distance with high precision.

A third method for calculating distances is the Frequency Modulated Continuous Wave (FMCW) (Frequency Modulated Continuous Wave) technique [18] (see Figure 2.4). In this method, the instantaneous optical frequency of the emitted signal is periodically varied over time, typically by modulating the power supplied to the source. This frequency modulation causes a frequency shift between the emitted signal and the signal reflected back from the target. By analyzing the frequency difference (beat frequency) between these two signals, it is possible to determine the time delay and thus calculate the distance to the target.

Unlike amplitude-based methods, such as AMCW, the FMCW technique offers higher precision due to its ability to resolve small frequency differences with high accuracy. While the typical depth resolution for AMCW and phase-shift methods is around 1 cm, FMCW can achieve a much finer resolution of approximately 0.1 cm, making it particularly suitable for applications requiring high accuracy, such as autonomous vehicles and industrial metrology.

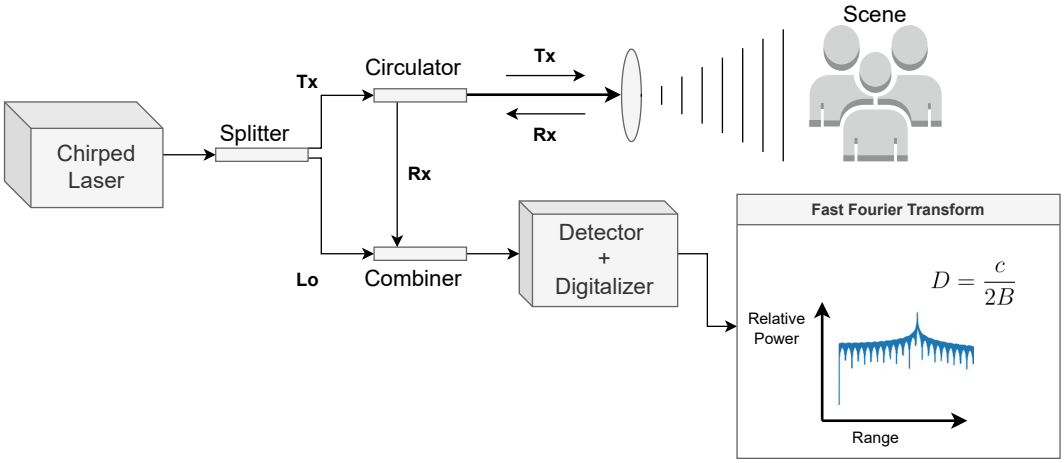


Figure 2.4: FMCW Working Principle. A chirped laser generates a modulated signal that is split and directed toward the scene using a circulator. The reflected signal (Rx) from the target is combined with the reference signal (Lo) and processed by a detector and digitalizer. The frequency difference between the transmitted (Tx) and received (Rx) signals, calculated through Fast Fourier Transform (FFT), is proportional to the distance (D) of the target. The equation highlights the relationship between distance, the speed of light (c), and the bandwidth of the chirped signal (B). FMCW systems are particularly advantageous in automotive LiDAR and radar applications for their ability to simultaneously measure range and velocity with high resolution.

2.1.2 Intensity in LiDAR systems

LiDAR systems provide not only spatial coordinates (x, y, z) but also record additional attributes such as the *intensity* of the reflected laser pulses. This parameter plays a critical role in interpreting the environment captured by the sensor.

Intensity refers to the strength of the return signal and is directly measured by the LiDAR's receiver, typically using photodetectors or avalanche photodiodes (APDs). Photodetectors are devices designed to convert light into electrical signals and form the core of LiDAR's intensity measurement system. They detect the return pulse from the target and translate its energy into an electrical response, which is then processed by the LiDAR's electronics.

Among photodetectors, APDs are a highly sensitive type of photodiode widely used in LiDAR systems due to their ability to amplify weak return signals. Photodiodes, in general, are semiconductor devices that generate a current proportional to the intensity of incident light. APDs take this a step further by incorporating an internal gain mechanism.

This high sensitivity makes APDs ideal for LiDAR applications, as they can detect faint signals even from distant or low-reflectivity surfaces. However, their performance can be affected by noise, temperature variations, and the quality of the optical components in the system. To address this, modern LiDAR systems combine APDs with advanced signal processing algorithms and noise reduction techniques to ensure accurate intensity measurements.

The recorded intensity is influenced by several factors, including:

- **Material properties**: Reflective surfaces produce stronger signals, while absorptive materials result in weaker returns.
- **Angle of incidence**: The signal is reduced when the laser beam strikes the surface at oblique angles.
- **Distance to the target**: Intensity decreases with distance due to signal dispersion and atmospheric attenuation.
- **Environmental conditions**: Factors such as fog, rain, or dust attenuate the signal strength.

In practice, LiDAR sensors measure the return signal's strength in real time without explicitly calculating these factors. Instead, the intensity values reflect the combined effect of these influences. This makes intensity a valuable attribute for interpreting material properties and assessing environmental conditions indirectly.

Having outlined how real LiDAR systems measure the received intensity, we will later delve into the detailed computation of LiDAR performance in simulation environments.

2.1.3 Data representation

While previous sections discussed the principles of LiDAR operation and its applications in mapping, this section focuses on the structuring and representation of the raw data generated during LiDAR scans, critical for enabling further applications.

2.1.3.1 Key components of LiDAR data

LiDAR systems produce rich datasets that capture detailed spatial and environmental information about the surroundings. These datasets, commonly referred

to as point clouds, consist of multiple attributes that form the foundation for further analysis and processing. The key components of LiDAR data include:

- **Spatial Coordinates (x, y, z):** Each point in the LiDAR point cloud is represented by three spatial coordinates x, y, and z that define its 3D position in space. These coordinates describe the physical surfaces detected by the LiDAR sensor and form the geometric foundation of the dataset.
- **Intensity:** The intensity value represents the strength of the returned laser signal. This parameter is influenced by several factors, including the reflectivity of the target material, the angle at which the laser strikes the surface, and environmental conditions such as rain or fog.
- **Timestamps:** Timestamps provide a precise temporal context for each measurement in the dataset, serving as another key piece of information provided by LiDAR systems. These timestamps are generated using either the LiDAR's internal clock or by synchronizing with an external time reference, such as a GNSS (Global Navigation Satellite System) receiver. Accurate time-stamping is critical for ensuring synchronization with other sensor inputs, such as cameras or IMUs, particularly in dynamic environments where both the sensor and objects in the scene may be in motion.

These components form the raw output of LiDAR systems and are typically stored in formats that facilitate analysis and processing, such as LAS, PCD, or custom binary formats. These formats, along with their roles in data storage and processing, will be discussed in the following sections. Understanding these key attributes is essential for interpreting and utilizing LiDAR data effectively.

2.1.3.2 Data structuring formats

- **PCD (Point Cloud Data)** The `.pcd` format is a widely used file type for storing point clouds, commonly utilized in the `PCL` framework. It supports both ASCII and binary encoding, offering flexibility for human-readable and efficient data storage. Each point includes at least the fields `x`, `y`, `z` (spatial coordinates), and optionally additional attributes such as `intensity` or `RGB` values for color representation. For further details on the `.pcd` format, see [19].
  - **Data Type:** LiDAR point cloud data is typically stored using 32-bit float values (4 bytes per value) for each x, y, z coordinate. For a dataset containing 1 million points with just x, y, z values, the storage requirement is approximately 12 MB. When additional attributes, such as intensity or color, are included, the size increases proportionally (e.g., 16 MB for x, y, z, i). Binary encoding is commonly used, as it reduces storage overhead while maintaining precision, making it ideal for large-scale datasets in real-time applications.

  - **Use Case:** The PCD format is frequently used in autonomous driving systems to represent 3D environments in real time. For example, a LiDAR sensor mounted on a vehicle captures point cloud data that is processed to identify pedestrians, vehicles, and static objects like buildings or traffic signs. This format plays a crucial role in collision avoidance systems and SLAM, where precise and efficient data processing is required for safe navigation.
  - **Note:** To minimize numerical rounding errors and optimize storage, point cloud data is often stored with offset positions relative to the vehicle's initial frame. This approach compresses the range of values, reduces file size, and facilitates efficient data transmission. Such practices are particularly valuable in applications requiring frequent data exchanges, such as vehicle-to-vehicle (V2V) or vehicle-to-cloud (V2C) communication, enabling real-time updates for fleet coordination and map sharing.

- **Binary XYZI/XYZIR** These binary formats, commonly used in datasets like KITTI and nuScenes [20, 21], lack headers and use fixed fields: `x`, `y`, `z`, `intensity`, and optionally `ring index`. They are optimized for compact storage and fast sequential read operations, making them ideal for real-time applications.

  - **Data Type:** 32-bit float (4 bytes per field). Each point consumes a fixed amount of memory, e.g., 16 bytes for `XYZI` or 20 bytes for `XYZIR`. For 1 million points, this results in 16 MB or 20 MB of storage.
  - **Use Case:** Standard format for autonomous driving datasets [22]. Commonly used in real-time perception pipelines where minimal overhead and high read/write speeds are critical. For instance, it is used in collision avoidance systems where the rapid ingestion of large point clouds is required.
  - **Note:** Due to its compact binary structure, it avoids the overhead of human-readable formats, making it more efficient for storage and transmission in bandwidth-constrained systems like V2V or V2X communication. As a headerless format, portability depends on well-documented metadata describing the structure (e.g., number of points, fields). This makes it less self-contained but highly interoperable with established libraries such as Open3D or PCL.

- **PLY (Stanford Triangle Format)** The `.ply` format encodes points as vertices and supports both ASCII and binary representations. Optional fields, such as `color` and `intensity`, add flexibility. This makes it ideal for visualization and data interchange between applications [23].

  - **Required Properties:** `x`, `y`, `z`. Optionally `red, green, blue` [0–255], `intensity`.

  - **Use Case:** Suitable for visualization in tools like and Blender [6]. Also used in cases where human-readability or data annotation is necessary.
  - **Note:** ASCII encoding is verbose, leading to larger file sizes, but is useful for manual inspection and debugging. Binary encoding significantly reduces size and read/write times, making it suitable for larger datasets. Widely supported by many visualization and processing tools, ensuring high portability. However, ASCII files are less efficient for storage and transmission in high-volume datasets.

- **LAS (LASer File Format)** The `.las` format is designed for large-scale point clouds and supports RGB color fields and intensity values. It is a widely adopted standard in geospatial applications due to its ability to handle large-scale mapping data [24].
  - **Recommended for:** Large-scale mapping datasets, particularly in forestry, urban planning, and topographic surveys.
  - **Version:** LAS 1.4 (uncompressed only). Extensions like LAZ allow for compressed versions, reducing file size by up to 70% while maintaining compatibility.
  - **Scale/Resolution:** Supports customizable scales (e.g., 10^{-6}) to minimize discretization errors, making it highly precise for geographic datasets.
  - **Use Case:** Commonly used in applications requiring integration with GIS platforms such as ArcGIS and QGIS. For example, LiDAR point clouds collected for terrain modeling are processed in `.las` format and tiled for efficient analysis.
  - **Note:** Although LAS files can be large, tiling is a common strategy to partition large datasets into manageable chunks for processing and storage. Its widespread adoption as a standard ensures high portability across GIS tools and LiDAR-specific software.

- **Gaussian Splat (.splat)** The Gaussian Splat format allows encoding of points as Gaussian distributions, enabling more visually appealing renderings and smoother surface approximations compared to discrete points [25, 26].
  - **Use Case:** Primarily used for visualization in applications where surface rendering is important, such as in augmented reality or high-fidelity visualizations of 3D reconstructions.
  - **Tools:** Conversion from standard formats (e.g., `.ply` to `.splat`) can be done using tools like SuperSplat, which enable advanced rendering techniques such as per-point normals and variable radius splats.

  - **Note:** Compared to discrete point formats, splat-based representations can reduce the number of points needed for smooth visual approximations, thereby saving storage and improving rendering performance. While less common than formats like `.ply` or `.las`, it is gaining traction in specific domains. However, it may require conversion tools to interoperate with standard point cloud libraries.

2.1.4 Error sources

Although LiDAR systems provide highly accurate and dense spatial data, various error sources can degrade the quality and reliability of the measurements. These errors are typically caused by environmental factors, sensor limitations, and physical phenomena. Identifying and understanding these causes is essential to improve the robustness of LiDAR-based systems, particularly in challenging conditions.

Temperature sensitivity Temperature fluctuations can impact the performance of LiDAR sensors, particularly their internal electronics and optics [27].

- **Cause:** Variations in ambient temperature affecting laser diodes, photodetectors, and calibration.
- **Impact:** Causes drift in measurements, reduced accuracy, and changes in range estimation.
- **Mitigation:** Temperature compensation mechanisms and periodic recalibration.

Multiple returns Multiple returns occur when a single laser pulse encounters multiple surfaces along its path before returning to the sensor [28, 29].

- **Cause:** Highly reflective materials, such as mirrors, polished metals, or transparent surfaces like glass, can significantly distort LiDAR measurements. When a laser pulse interacts with these materials, it may reflect at unexpected angles, pass through without returning, or partially reflect at varying depths [30]. This behavior introduces challenges such as the loss of the laser signal, incorrect distance measurements, or the generation of false points in the point cloud. For example, polished metals can cause specular reflections that deflect the beam away from the sensor, while glass may produce multiple weak returns due to partial internal reflections.

  Reflective surfaces can also saturate the photodetectors within the LiDAR sensor, reducing the system's dynamic range and introducing nonlinearities that degrade measurement accuracy. Furthermore, refractive materials, such as transparent glass, may alter the trajectory of the laser beam

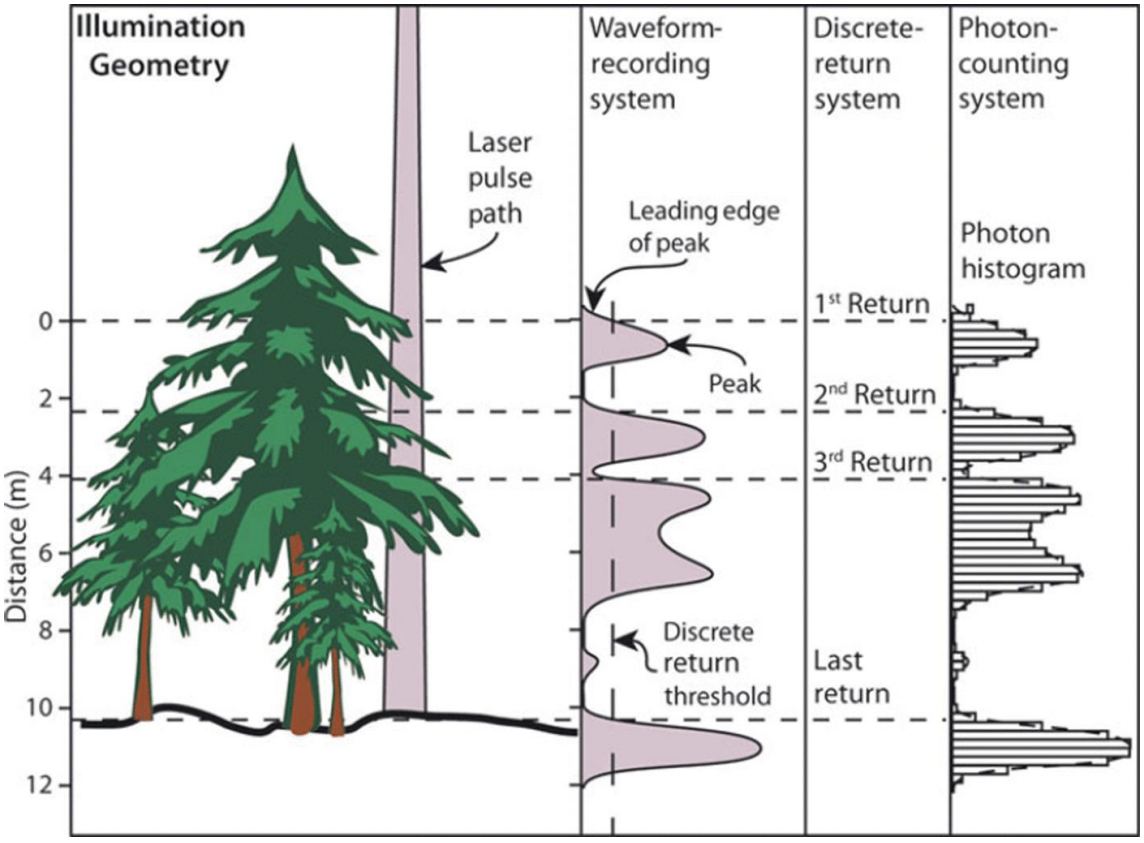


Figure 2.5: Laser illumination and return signal recording. Portions of the emitted laser pulse are reflected by different targets resulting in multiple return signals for each pulse. Different lidar systems have different return signal recording capabilities [1, 2].

.

through refraction, further complicating the interpretation of the return signals. These effects are particularly problematic in environments with complex geometries, such as urban areas with glass façades or metallic structures.

- **Effect:** The LiDAR sensor registers multiple returns for a single emitted pulse, resulting in redundant or ambiguous data points in the point cloud. This can lead to inaccuracies in the spatial representation of the environment, particularly in complex or cluttered scenes.
- **Mitigation:** Addressing multiple returns involves both hardware and software solutions:
  - **Hardware:** Advanced LiDAR systems use high-resolution timing circuits and multiple return detection capabilities to distinguish and record separate signals more accurately. They may also incorporate adjustable laser power to reduce over-reflections.

  - **Software:** Algorithms can filter and classify multiple returns to prioritize the most relevant points (e.g., the first return for surface detection or the last return for ground detection). Additionally, data processing techniques can model and account for multipath effects.

Occlusions

- **Cause:** Physical obstructions, such as buildings, vehicles, dense vegetation, or other objects that block the laser beam from reaching certain areas in the environment.
- **Effect:** Results in voids or blind spots in the point cloud where no data is recorded, reducing the completeness and accuracy of the 3D spatial representation.
- **Mitigation:** Addressing occlusions requires a combination of sensor placement, data fusion, and advanced processing techniques:
  - **Sensor placement and redundancy:** Deploying multiple LiDAR sensors at different vantage points can reduce occlusions by capturing areas hidden from a single sensor's perspective. For example, mounting LiDAR sensors at elevated positions or on moving platforms can help minimize blind spots.
  - **Data fusion:** Combining LiDAR data with information from other sensors, such as cameras or radar, can fill in gaps caused by occlusions. This multimodal approach enhances the completeness of the environmental representation.
  - **Advanced algorithms:** Post-processing techniques, such as interpolation, extrapolation, or machine learning models, can estimate and reconstruct missing data in occluded regions. These methods leverage patterns in the existing point cloud or incorporate prior knowledge of the scene.

Fig: 2.6: When an object is crossing the LiDAR's laser rays, the current points (blue points) will occlude the previous points (orange points) collected at time T_{k-1}. When an object is moving along the laser rays and away from the sensor, the current points (blue points) will be occluded by all previous points (i.e., orange points at T_{k-1} and green points at T_{k-i}) that are further occluded by themselves (i.e., orange points at T_{k-1} are occluded by green points at T_{k-i}). Conversely, when an object is moving along the laser rays and towards the sensor, the current points (blue points) will occlude all previous points (i.e., orange points at T_{k-1} and green points at T_{k-i}) that further occlude themselves (i.e., orange points at T_{k-1} occlude green points at T_{k-i}).

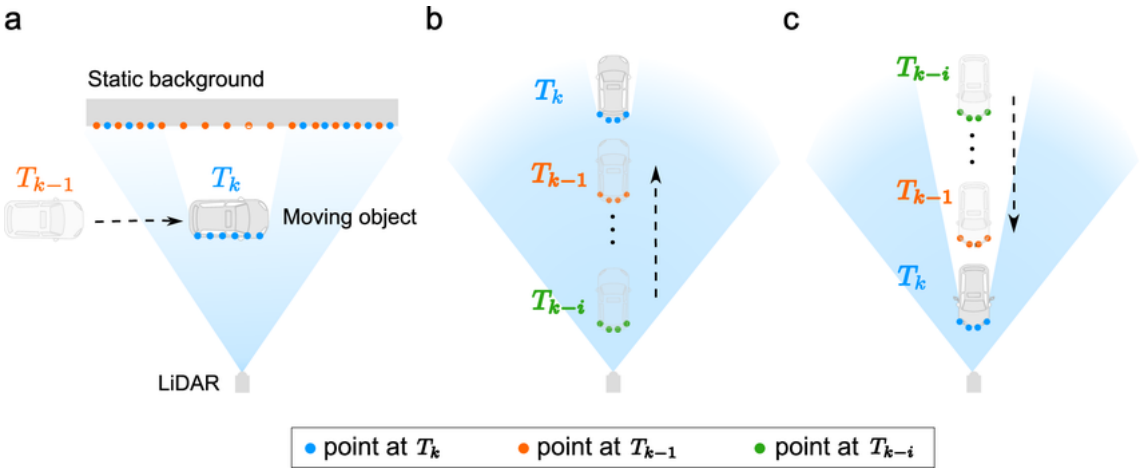


Figure 2.6: Principle of occlusion [2].

.

Adverse weather conditions Environmental conditions such as rain, fog, dust, and snow can significantly degrade LiDAR performance by scattering, absorbing, or attenuating laser beams.

- **Cause:** Interaction of laser pulses with atmospheric particles:
  - *Rain:* Refraction and partial reflection by water droplets, causing false returns and reduced signal strength.
  - *Fog:* Dense microscopic water particles scatter and attenuate the beam, limiting effective range.
  - *Dust/Smoke:* Scattering and absorption introduce noise and reduce accuracy.
  - *Snow:* Large reflective particles create multiple strong returns, confusing the sensor.
- **Impact:**
  - Reduces signal-to-noise ratio (SNR), limiting range and accuracy.
  - Introduces false points or voids in the point cloud.
  - Impairs detection of distant or small objects, critical for applications like autonomous driving.

Sensor noise Intrinsic noise in LiDAR systems originates from internal processes such as signal amplification and data conversion. Variability in components like photodetectors, avalanche photodiodes (APDs), or analog-to-digital converters introduces fluctuations in the measured signal. These inconsistencies can lead to imprecise distance or intensity readings, affecting the overall accuracy of the point cloud and potentially misrepresenting the environment in critical applications.

2.1.5 LiDAR simulations techniques

Simulating LiDAR measurements is a fundamental approach for evaluating sensor performance, testing perception algorithms, and validating autonomous systems in controlled, repeatable environments. These simulations allow researchers to reproduce real-world challenges without the costs and risks associated with physical experiments. Current methods for simulating LiDAR can be broadly divided into two main categories: geometric approaches, such as ray casting, and data-driven methods based on machine learning.

LiDAR simulators can be categorized into two main types based on their modeling approach [31]:

Sensor measurement model This type of simulator replicates the measurement technique used by a real LiDAR sensor and considers the sensor's orientation within the virtual scene. The measurement model uses ray tracing or ray casting algorithms to emulate the process of emitting laser beams, detecting intersections with objects in the scene, and calculating distances. This approach accurately mimics the operational principles of a LiDAR sensor, such as time-of-flight (ToF) or phase shift techniques, commented in section 2.1.

Sensor error model This type focuses on modeling the statistical errors observed in real-world LiDAR measurements. It incorporates both systematic and random errors caused by factors such as environmental conditions, surface reflectivity, and hardware limitations. The error model is designed to approximate the deviations in measurement accuracy introduced by the physical properties of the sensor and its operating environment.

Hybrid approach: Some simulators combine both approaches to achieve greater accuracy and realism. These hybrid models integrate the operational characteristics of the measurement model with the statistical approximations of the error model, ensuring comprehensive simulation capabilities [32].

While the Measurement Model focuses on replicating the physical behavior of LiDAR sensors and the Error Model accounts for imperfections, the implementation of these models relies on computational methods. These methods provide the technical foundation to simulate the interaction of LiDAR rays with a virtual environment and generate synthetic data. Two widely used approaches in this context are *Ray Casting* and *Machine Learning based* LiDAR Simulation.

2.1.5.1 Ray casting for LiDAR simulation

Ray casting is a computational technique used to simulate LiDAR point clouds by projecting rays into a reconstructed 3D scene and determining their intersec-

tions with the scene geometry [33]. In this approach, the LiDAR's configuration is modeled, and rays are emitted from the simulated sensor's origin to interact with objects in the virtual environment.

The 3D scene is typically represented as a dense point cloud or a mesh structure. Ray casting involves dividing the sensor's field of view into discrete frustums, each corresponding to a single laser pulse. Within each frustum, intersections between the emitted ray and the scene geometry are calculated to identify the closest surface point along the ray's path. This process effectively mimics the time-of-flight measurement of a physical LiDAR sensor.

Conventional methods, such as Closest Point (CP) ray casting, select the nearest intersection point for each ray to form the simulated point cloud. However, CP methods may encounter challenges in complex scenarios, such as overlapping rays, reconstruction errors in the virtual environment, or irregular scanning patterns. These issues can lead to redundant points or inconsistencies in the generated data.

Advanced techniques, such as First Peak Averaging (FPA), extend the ray casting process by considering multiple intersections within each frustum. FPA averages the points within the first peak of the depth distribution to estimate the surface representation more accurately. The method uses a fixed peak width and employs weighted averaging to refine the coordinates of the simulated points. In addition to spatial coordinates (x,y,zx,y,z), ray casting can simulate additional attributes, such as return intensity, providing a richer dataset for LiDAR simulation.

Mathematical foundations A ray is defined by its *origin* $\mathbf{O}$ and a *direction vector* $\mathbf{D}$. Here, $\mathbf{O}$ represents the starting point of the ray in 3D space, and $\mathbf{D}$ is a unit vector that indicates the direction in which the ray extends. The position $\mathbf{P}(t)$ along the ray can be described mathematically as:

$$\mathbf{P}(t) = \mathbf{O} + t \cdot \mathbf{D}, \quad t \geq 0$$

where t is a scalar parameter that represents the distance from the origin along the ray's path.

The intersection of a ray with a geometric object is determined by solving equations specific to the object's geometry:

- **Plane:** A plane is defined by a point $\mathbf{P}_0$ on the plane and a normal vector $\mathbf{N}$. The intersection parameter t is computed as:

$$t = \frac{(\mathbf{P}_0 - \mathbf{O}) \cdot \mathbf{N}}{\mathbf{D} \cdot \mathbf{N}}$$

  If $t > 0$, the intersection point $\mathbf{P}(t)$ lies in the direction of the ray.

- **Sphere:** A sphere is defined by its center $\mathbf{C}$ and radius r, where $\mathbf{C}$ represents the position of the sphere's center in 3D space, and r is the distance

from the center to any point on the sphere's surface. The intersection is determined by solving:

$$\|\mathbf{P}(t) - \mathbf{C}\|^2 = r^2$$

Expanding and solving this quadratic equation yields the potential intersection points.

In LiDAR simulation, raycasting generates a synthetic point cloud by tracing rays from the sensor's origin into the virtual environment. The intersection points represent detected surfaces, and additional properties such as reflectivity or angle of incidence can be calculated to simulate return intensities. This process enables the emulation of real-world LiDAR systems for testing and validation in virtual environments.

2.1.5.2 Machine learning-based simulations

Recent advancements in machine learning (ML) have led to the development of data-driven techniques for simulating LiDAR measurements [34, 35].

Overview of DyNFL DyNFL leverages neural fields to create volumetric representations of the environment. The method separates the reconstruction into two components:

- **Static Background:** Modeled as an independent neural field representing the stationary elements of the environment.
- **Dynamic Objects:** Each moving object is modeled as a separate neural field, enabling per-object editing and repositioning.

The final scene is composed by integrating these two components using a ray elimination strategy that handles occlusions and transparency.

Mathematical model for ray composition The compositional neural fields in DyNFL combine static and dynamic elements using the following equation:

$$C(x) = \begin{cases} C_{\text{static}}(x), & \text{if no dynamic intersection.} \\ C_{\text{dynamic}}(x), & \text{if a dynamic object occludes the background.} \end{cases} \tag{2.3}$$

where:

- $C(x)$ is the final composed representation at point x.
- $C_{\text{static}}(x)$ represents the static background field.
- $C_{\text{dynamic}}(x)$ represents the field of dynamic objects.

DyNFL employs ray tracing to simulate LiDAR sensor behavior by casting rays into the compositional neural fields and detecting intersections with surfaces.

The system determines whether the ray intersects a dynamic object or the static background by calculating the nearest intersection point:

$$I(x) = \text{argmin}_{i \in \{\text{static, dynamic}\}}(t_i) \tag{2.4}$$

where t_i is the distance to the closest intersection for the static or dynamic neural fields.

Training and reconstruction The DyNFL framework is trained on real-world and synthetic LiDAR data, incorporating object bounding boxes for dynamic elements. The training process involves minimizing a reconstruction loss:

$$\mathcal{L}_{\text{reconstruction}} = \|P_{\text{real}} - P_{\text{pred}}\|^2 \tag{2.5}$$

where:

- P_{real} is the ground truth point cloud measured by a real LiDAR sensor.
- P_{pred} is the point cloud predicted by the compositional neural field.

Conclusion Machine learning (ML)-based LiDAR simulations, such as DyNFL, provide an advanced approach for re-simulating LiDAR scans in dynamic environments. This method leverages the adaptability of neural fields in conjunction with ray tracing techniques. By independently reconstructing static and dynamic components, it achieves a higher degree of realism and allows for greater flexibility in editing LiDAR data.

2.2 Rain Simulation Techniques

As previously discussed, rain significantly impacts the performance of LiDAR systems by introducing noise and attenuating the accuracy of measurements. Since one of the primary objectives of this work is to simulate the effects of rain on LiDAR data, this section provides a detailed review of state-of-the-art approaches in this area.

The discussion is divided into two main aspects:

- **Physical modeling of rain:** This examines how rain interacts with LiDAR signals and the surrounding environment, focusing on factors such as raindrop distributions, particle scattering, and absorption effects.
- **Integration into LiDAR attenuation mechanisms:** This explores how these physical models are incorporated into LiDAR simulations to replicate the degradation of sensor performance under rainy conditions.

This review lays the groundwork for accurately replicating rain-induced noise in LiDAR simulations, providing a comprehensive framework to understand and simulate the effects of adverse weather on LiDAR measurements.

2.2.1 Phyisical models of rain

The physical modeling of rain focuses on understanding the interaction between rain droplets and laser signals, which involves phenomena such as attenuation, scattering, and absorption.

2.2.1.1 Rain drop size distribution

Rain droplets vary in size, and their distribution significantly impacts signal attenuation and scattering. Understanding these distributions is critical for accurately modeling rain effects in LiDAR systems. Below, we discuss three key models used to describe rain drop size distributions [36].

Marshall-Palmer distribution The *Marshall-Palmer distribution* [37] is one of the most widely used models for describing rain drop size distribution. It defines the relationship between rainfall intensity (measured in mm/h) and the number of droplets per unit volume, assuming an exponential decay:

$$N(D) = N_0 e^{-\Lambda D} \tag{2.6}$$

where:

- $N(D)$: Number of drops per unit volume for a drop diameter D.
- N_0: Concentration parameter related to rainfall intensity.
- Λ: Decay constant, which depends on the intensity of rainfall.

This model is particularly suitable for light and moderate rain but lacks precision for capturing the variability of larger drops in heavy rain.

Rainfall Intensity (mm/h)	Drop Diameter (mm)	Number of Drops ($N(D)$)	Comment
Light Rain (0 - 2.5)	0.5 - 1.0	$N(D) = N_0 e^{-\Lambda D}$	Small, uniformly distributed
Moderate Rain (2.5 - 10)	1.0 - 2.5	$N(D) = 8000 e^{-4.1D}$	Increasing drop size
Heavy Rain (10 - 50)	2.5 - 5.0	$N(D) = 16000 e^{-5.2D}$	Larger, sparse drops
Torrential Rain (>50)	5.0 - 8.0	$N(D) = 32000 e^{-6.3D}$	Dominated by large drops

Table 2.1: Rain drop size distribution and parameters according to Marshall-Palmer

Feingold-Levin distribution The *Feingold-Levin distribution* [38] provides a more detailed representation of rain drop size, especially for heavy rainfall, where coalescence and fragmentation processes dominate. The distribution formula is given by:

$$N(D) = N_0 D^m e^{-\lambda D} \tag{2.7}$$

where:

- $N(D)$: Number of drops per unit volume for a drop diameter D.
- N_0: Initial concentration parameter.
- D^m: A scaling factor that accounts for variability in drop sizes.
- λ: Decay parameter dependent on rainfall intensity.

This distribution captures the behavior of both small and large droplets more effectively than the Marshall-Palmer model. It is particularly useful for simulating scattering and attenuation effects in LiDAR systems under heavy rainfall.

Deirmendjian distribution The *Deirmendjian distribution* [39] is designed for more complex atmospheric conditions and includes a broader range of particle sizes, making it suitable for modeling not just rain, but also fog and clouds. It is often expressed as:

$$N(D) = N_0 D^a e^{-bD} \tag{2.8}$$

where:

- $N(D)$: Number of droplets for a given diameter D.
- N_0: Scaling factor related to the concentration of droplets.
- a, b: Parameters that describe the shape of the distribution, allowing flexibility in modeling.

This distribution excels in scenarios with mixed precipitation, where both rain and smaller particles like mist or fog coexist. Its flexibility makes it a powerful tool for advanced atmospheric modeling.

Comparison of distributions The table below summarizes the key characteristics of the three distributions:

Distribution	Formula	Best Use Case	Limitations
Marshall-Palmer	$N(D) = N_0 e^{-\Lambda D}$	Light to moderate rain	Poor fit for large drops
Feingold-Levin	$N(D) = N_0 D^m e^{-\lambda D}$	Heavy rain	Requires more parameters
Deirmendjian	$N(D) = N_0 D^a e^{-bD}$	Mixed precipitation	Complex parameterization

Table 2.2: Comparison of different rain drop size distributions

Implementation of rain models in simulations In simulation environments, physical rain models are implemented by generating rain droplets in a virtual 3D space using probabilistic distributions. The key steps include:

- Generating rain droplets based on predefined size distributions.
- Simulating the interaction of the LiDAR beam with the droplets, including attenuation, scattering, and noise effects.
- Applying random noise (e.g., Gaussian or Poisson) to replicate signal degradation caused by rain-induced scattering.

2.2.2 Rain-induced attenuation on LiDAR signals

Rain-induced attenuation significantly impacts the performance of LiDAR systems by reducing the strength of laser returns and introducing noise due to scattering and backscattering. This section explores the effects of adverse weather conditions on the optical properties of LiDAR signals, focusing on rain, which is critical for understanding how environmental factors degrade the quality of point cloud data.

The performance of laser scanners in adverse weather conditions is heavily influenced by optical phenomena such as scattering, extinction, and backscattering. These effects are caused by the interaction of the laser beam with atmospheric particles (e.g., water droplets), and they impact both the transmitted signal and the received power. The cumulative result is a limitation in the effective range, resolution, and accuracy of the sensor under such conditions.

Scattering, extinction, and backscattering The scattering (Q_s), extinction (Q_e), and backscattering (Q_b) coefficients describe the attenuation and deviation of laser energy caused by atmospheric particles.

- **Scattering**: In the context of rain, scattering occurs when laser energy is deflected in multiple directions upon interacting with raindrops. The size and distribution of raindrops significantly influence the degree of scattering, with larger droplets causing more pronounced deviations.

- **Extinction**: Extinction refers to the combined effect of scattering and absorption, resulting in a reduction of the laser's transmitted energy. In rain, extinction

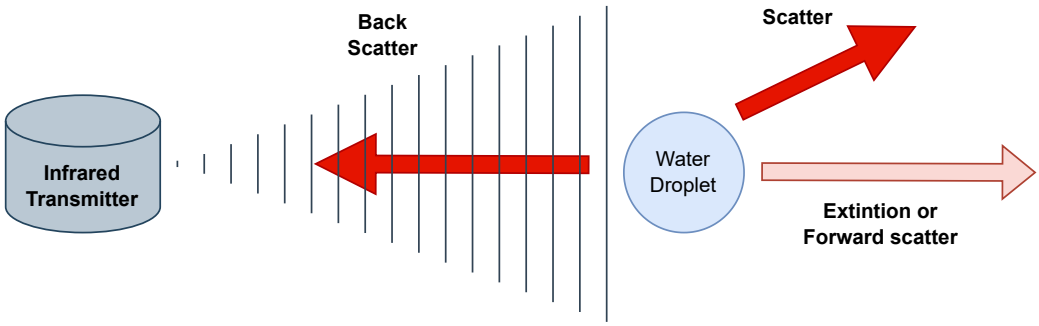


Figure 2.7: Illustration of scattering, backscattering, and extinction.

limits the effective range of the LiDAR by diminishing the power of the laser signal as it propagates through the raindrop-filled medium.

- **Backscattering**: Backscattering is a specific type of scattering where the laser energy is reflected back toward the LiDAR sensor. In rain, backscattering introduces noise into the signal, as raindrops themselves can appear as false detections in the point cloud data.

2.2.2.1 Mie theory and Rayleigh scattering

The interaction of LiDAR signals with atmospheric particles is governed by two main scattering mechanisms: *Rayleigh scattering* and *Mie theory*. These mechanisms describe how light interacts with particles depending on the size of the particle relative to the wavelength of the light.

Rayleigh scattering Rayleigh scattering [40] occurs when the size of the particle (r) is much smaller than the wavelength of the light (λ) ($r \ll \lambda$). This phenomenon is typical for small particles such as air molecules or very fine mist.

Rayleigh scattering plays a role in LiDAR signal attenuation by redirecting a portion of the laser's energy away from the original propagation path. Although this effect is less significant compared to Mie scattering for larger particles, it contributes to the overall reduction in signal intensity, particularly in clear atmospheric conditions with fine particulate matter. The key characteristics of Rayleigh scattering are:

- **Size dependency:** Rayleigh scattering is applicable for particles smaller than $0.1\,\mu m$. In this size range, the interaction with the laser beam primarily depends on the relative size of the particle to the wavelength.
- **Wavelength dependency:** The intensity of Rayleigh scattering is inversely proportional to the fourth power of the wavelength ($I_s \propto \lambda^{-4}$). This means shorter wavelengths (e.g., blue light) are scattered more strongly than longer

wavelengths (e.g., red light). This principle explains why the sky appears blue and also affects the performance of LiDAR systems operating in different wavelength bands.

- **Distribution:** Scattering occurs uniformly in all directions, but with a slightly higher intensity in forward and backward directions relative to the original path of propagation.

Impact on LiDAR Signals In the context of LiDAR systems, Rayleigh scattering contributes to signal attenuation by scattering a portion of the laser's energy out of the forward-propagating beam. While Rayleigh scattering generally has a minor impact compared to Mie scattering, it can still affect the signal-to-noise ratio (SNR) in environments dominated by very fine particles or air molecules. This is especially relevant for LiDAR systems operating at shorter wavelengths, as these are more susceptible to scattering losses due to the λ^{-4} dependency.

Mie theory Mie theory [41] describes scattering when the size of the particle (r) is comparable to the wavelength of the light ($r \sim \lambda$). This is particularly relevant for larger particles such as water droplets.

Mie theory describes scattering by particles whose radius is comparable to the wavelength of the laser light. This applies to a wide range of atmospheric particles commonly encountered in LiDAR applications.

Unlike Rayleigh scattering, the dependence of scattering intensity on wavelength in Mie theory is more complex and less pronounced. Mie scattering does not strongly favor shorter wavelengths, and its intensity can vary non-monotonically based on the particle size and refractive index.

A key characteristic of Mie scattering is its highly directional nature, with a significant portion of the scattered intensity concentrated in the forward direction. This forward scattering plays a critical role in LiDAR systems, as it directly influences the signal-to-noise ratio and the accuracy of distance measurements.

Comparison of Rayleigh and Mie scattering The key differences between Rayleigh scattering and Mie theory are summarized in Table 2.3.

Property	**Rayleigh Scattering**	**Mie Theory**
Particle size relative to wavelength (r)	$r \ll \lambda$	$r \sim \lambda$
Wavelength dependency (I_s)	$\propto \lambda^{-4}$	Weak or negligible
Scattering distribution	Uniform (all directions)	More intense in forward direction
Relevant conditions	Air molecules, fine mist	Rain, dust, large droplets

Table 2.3: Comparison of Rayleigh Scattering and Mie Theory

2.2.2.2 Attenuation

To better understand the role of particle properties in this process, we consider the concept of extinction efficiency (Q_e). Extinction efficiency describes the overall attenuation of the light beam as it propagates through a medium containing particles. This efficiency depends on several factors, including the particle radius (r), the wavelength of the light (λ), and the refractive index of the particle material (m):

$$Q_e = f(r, \lambda, m) \tag{2.9}$$

Extinction efficiency encompasses two primary contributions: scattering efficiency (Q_s) and absorption efficiency (Q_a). Mathematically, this relationship is expressed as:

$$Q_e = Q_s + Q_a \tag{2.10}$$

- Q_s (**Scattering efficiency**) quantifies the proportion of light redirected in various directions due to interaction with the particle.
- Q_a (**Absorption efficiency**) measures the fraction of light energy absorbed by the particle, which is then converted into other forms of energy, such as heat.

The balance between Q_s and Q_a depends on the material properties of the particle (refractive index m) and the wavelength of the incident light. This relationship is critical in understanding how different types of particles attenuate light in LiDAR applications, as both scattering and absorption directly influence the behavior of the laser signal. The forward-directed scattering typical of Mie theory has a significant impact on the signal-to-noise ratio, while the absorption component contributes to energy losses that reduce the detectable signal, leading to extinction. Understanding Q_e is therefore essential for accurately modeling the attenuation of the laser signal.

Relevance to LiDAR systems Understanding these scattering and attenuation mechanisms is essential for accurately modeling the performance of LiDAR systems under adverse weather conditions.

The interaction between the laser and atmospheric particles is influenced by two key factors: the *laser wavelength* and the *particle size and distribution*. LiDAR systems typically operate in one of three wavelength ranges, each with distinct scattering and attenuation characteristics:

- **Visible Light (500 nm):** wavelength.

- **Near-Infrared (NIR, 900 nm):**
- **Short-Wave Infrared (SWIR, 1550 nm):**

Particle Size and Distribution: The size and spatial distribution of droplets strongly influence how light interacts with atmospheric particles. Larger particles (> 1 mm) are less effective at backscattering and primarily affect extinction in the SWIR range due to Fresnel effects. In contrast, smaller particles (< 1 mm) are more efficient at scattering light in the visible and NIR ranges, causing greater signal degradation.

Mathematical models of rain attenuation Building on the principles of Mie and Rayleigh scattering, as well as the impact of raindrop size distribution on scattering and extinction properties, the attenuation of laser signals in rain can be quantitatively modeled. One of the most commonly used approaches is the *Beer-Lambert law* [42], which provides a practical framework for calculating the signal power loss due to rain:

$$P_r = P_t e^{-\alpha R} \tag{2.11}$$

where:

- P_r: Received signal power.
- P_t: Transmitted signal power.
- α: Attenuation coefficient, which depends on rain density, drop size distribution, and the laser wavelength.
- R: Distance traveled by the laser beam.

The attenuation coefficient α encapsulates the effects of scattering and absorption, which were previously discussed in the context of Mie and Rayleigh theories. This coefficient can be derived from empirical or theoretical models that account for the rainfall intensity and the particle size distribution.

The Beer-Lambert law serves as a macroscopic representation of the cumulative effects of scattering and absorption over a given path length. It provides a practical tool for modeling rain attenuation in LiDAR systems while remaining consistent with the underlying physical principles.

Part II

Project development

Chapter 3

Experimental Setup

Contents

This chapter provides a comprehensive overview of the simulation framework developed to support the evaluation and testing of autonomous navigation systems. The framework is built around the integration of key components such as the Unity Engine, ROS 2, and high-fidelity environmental models, as illustrated in the project's flowchart (see Figure 3.1). The chapter is structured to guide the reader through the motivations behind the framework, its design, and the methodologies employed to achieve a modular, scalable, and realistic simulation environment.

The integration of ROS 2 and Unity is a core aspect of the project, enabling seamless communication between sensors and the simulated environment. This is further expanded upon through the detailed design of the simulation framework, which includes high-definition render pipelines, maps, and 3D models, as well as the implementation of the AURORA vessel.

A key contribution of this chapter is the introduction of the Rain Simulation Model, a novel addition to the framework that simulates the impact of rainfall on LiDAR sensor data. This model, along with its structure, assumptions, and validation, plays a critical role in enhancing the realism of the simulation. The chapter concludes with an evaluation of the framework through initial tests and validation metrics, offering insights into its performance under various environmental conditions.

The overall goal of this chapter is to describe the experimental setup in detail, providing the foundation for the subsequent evaluation of the framework in the experiments and results section.

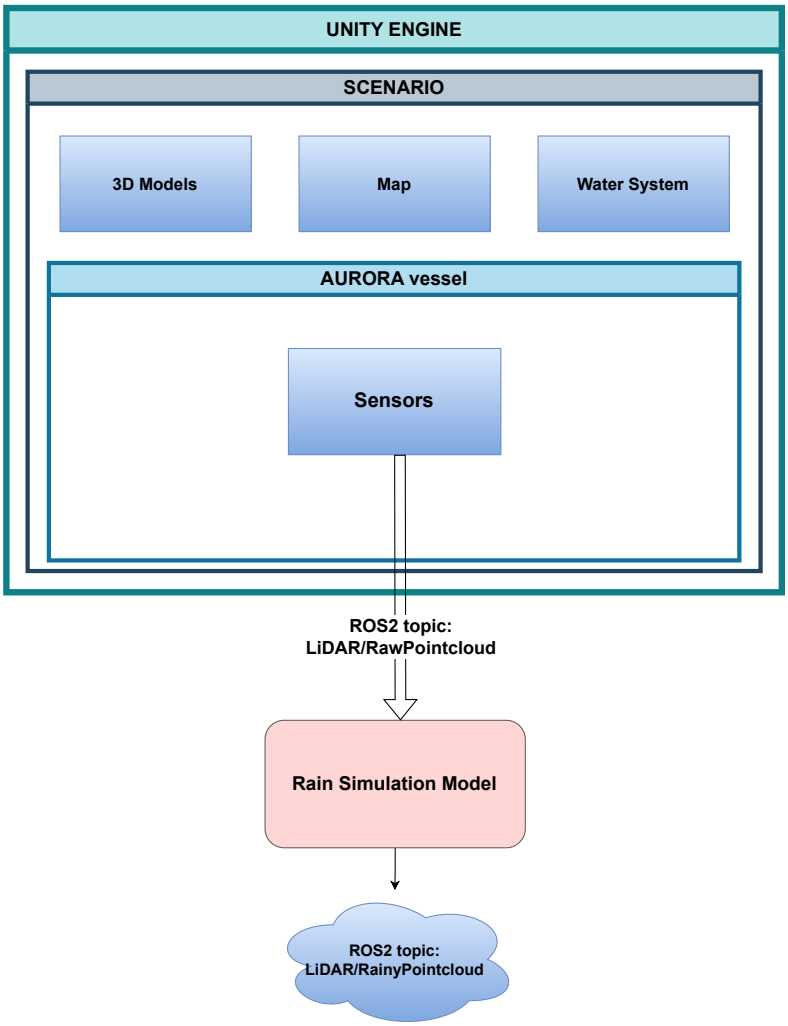


Figure 3.1: Flow chart representing the architecture of the project and its data flow. At the top, the Unity Engine acts as the central simulation environment. Within it, the scenario is defined, comprising 3D models, a map, and a water system, which collectively simulate realistic environmental conditions. The AURORA vessel operates within this environment, equipped with sensors that generate raw LiDAR data. This data is transmitted as an ROS 2 topic to the Rain Simulation Model. The Rain Simulation Model processes the raw LiDAR data, introducing rain-induced noise and effects. The modified data is then transmitted as another ROS 2 topic, enabling further analysis of LiDAR performance under simulated rain conditions. This chart visually encapsulates the interactions and components within the project framework.

3.1 ROS 2 Overview

ROS is a set of open-source software libraries and tools that stands as a cornerstone in the development of robotic applications, offering a powerful framework that facilitates both the creation and the management of complex robotic systems. ROS provides common set of interfaces for accessing sensors and actuators, and it includes libraries for things like message passing, navigation, and control. It also has a large and active community of developers who contribute to its development and provide support to users.

3.1.1 Communication protocol

At the heart of ROS 2 is an enhanced communication system designed for flexibility, scalability, and real-time performance. Unlike ROS, which relies heavily on a centralized *ROS Master*, ROS 2 employs a decentralized architecture based on the *Data Distribution Service (DDS)* middleware. This allows nodes to discover and communicate with one another without requiring a central hub, improving fault tolerance and enabling ROS 2 to operate effectively in distributed systems.

In ROS 2, nodes interact through a dynamic peer-to-peer network, exchanging data and collaborating on tasks such as sensing, actuation, and computation. Instead of relying on a master node for registration and coordination, *Discovery Servers* or DDS mechanisms are used to manage node discovery. This transition allows for greater flexibility and real-time guarantees, particularly in scenarios like robotics fleets or industrial applications.

As with ROS, communication between nodes is facilitated by *topics*, enabling nodes to either *publish* or *subscribe* to specific streams of data. In a typical configuration, a single publisher sends data to multiple subscribers. As depicted in Figure 3.2, nodes in ROS 2 can act both as publishers and subscribers, ensuring efficient data exchange. Topics in ROS 2 also serve as a filtering mechanism, ensuring that nodes only receive messages relevant to their subscribed topics. The underlying DDS middleware handles the delivery of messages with *Quality of Service (QoS)* policies, giving developers fine-grained control over communication parameters such as reliability, durability, and latency.

A key feature of ROS 2 is its improved support for real-time systems. QoS settings allow messages to meet specific timing and reliability requirements. For example, you can configure topics for reliable transmission (guaranteeing message delivery) or best-effort transmission (minimizing overhead for less critical data). This enables ROS 2 to be used in scenarios with strict performance requirements, such as autonomous vehicles or high-frequency sensor networks.

The data exchanged through topics in ROS 2 is encapsulated in *ROS messages*, which remain fundamental to the communication system. ROS 2 messages are used to represent data such as sensor readings, robot poses, or im-

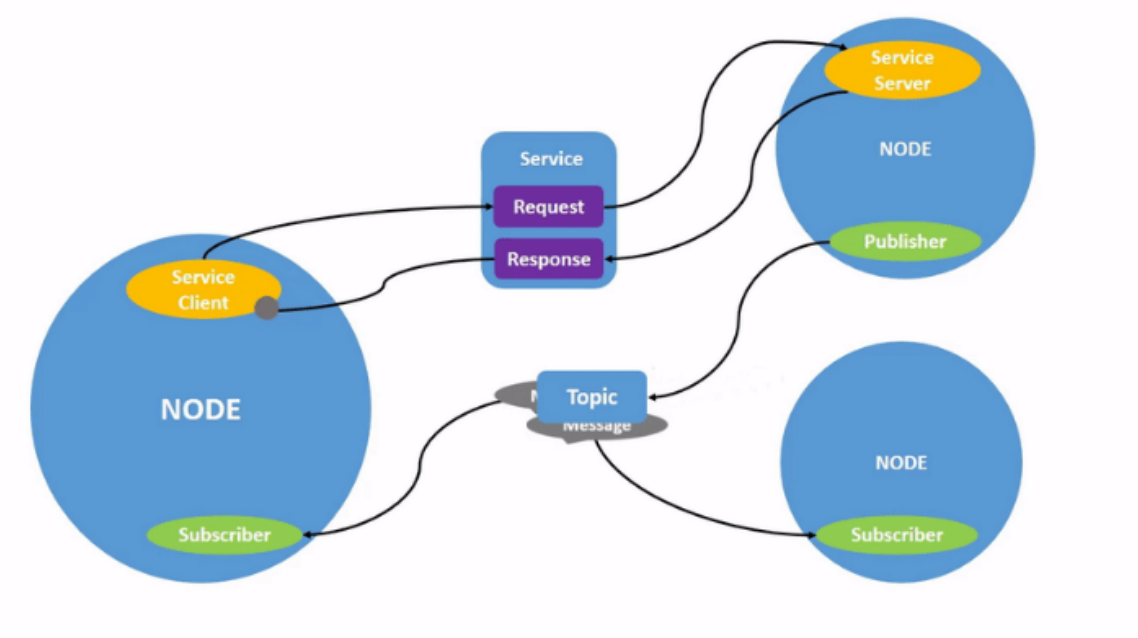


Figure 3.2: ROS 2 communication flow [3].

ages. They follow a structured format, often including integers, floats, booleans, strings, and arrays, and are serialized efficiently for transmission.

Another crucial feature of ROS 2 is its support for *ROS 2 Bags*, similar to ROS, but with enhanced capabilities. These *.bag* files record messages from specified topics for offline analysis, visualization, or debugging. Tools like *ros2 bag* enable recording and playback, making it easier to analyze complex system behaviors or use recorded data for development and testing.

With its decentralized architecture, real-time capabilities, and fine-tuned QoS control, ROS 2 provides a more robust and flexible platform for building and deploying robotics systems, ensuring scalability across distributed and dynamic environments.

3.2 Unity Engine Overview

At the core of Unity Engine lies a highly versatile and powerful real-time development platform designed for creating interactive 2D, 3D, and XR (AR/VR) experiences. Unity's architecture is built to provide a seamless workflow for developers, allowing them to design, simulate, and deploy applications across a wide range of platforms, including mobile devices, desktop computers, consoles, and head-mounted displays.

Unity Engine operates on a component-based architecture. The foundation of any Unity project is the *GameObject* (see Fig: 3.3), which acts as a container for various components. Components define the behavior and properties of a

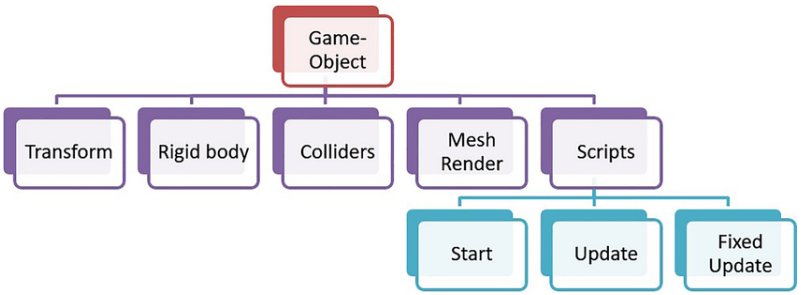


Figure 3.3: Hierarchy of a Unity game object and its components, including transform, rigid body, colliders, mesh render, and scripts. (Source [4]).

GameObject, enabling developers to create complex interactive systems by attaching predefined or custom scripts. This modular design simplifies development, allowing for scalability and reusability of code.

One of Unity's standout features is its *Scene System*, which allows developers to structure their projects as collections of *GameObjects* and assets. Scenes serve as containers for all the elements in a particular environment, including lighting, physics, and scripts. Developers can load and manage multiple scenes dynamically during runtime, enabling smooth transitions and modularity in application design.

Unity supports C# scripting as its primary programming language. Developers use C# to define custom behaviors, implement game logic, and interface with Unity's comprehensive *Application Programming Interface (API)*. The scripting environment integrates tightly with Unity's Editor, enabling real-time updates and debugging through the powerful Visual Studio development tools.

Communication within Unity is facilitated through *events* and *messaging systems*. Unity's built-in event system allows for efficient interaction between GameObjects, while the messaging system supports dynamic communication by invoking methods across different components. These systems are fundamental for complex interactions in applications.

Another key aspect of Unity Engine is its *Physics System*, powered by NVIDIA PhysX. This system provides robust tools for simulating realistic physical interactions, including rigid body dynamics, collisions, and joints. It also includes features for soft body physics and cloth simulations.

Unity's *Rendering Pipeline* is another critical component of its architecture. Unity offers multiple rendering pipelines, including the *Built-in Pipeline*, the *Universal Render Pipeline (URP)* for optimized performance, and the *High Definition Render Pipeline (HDRP)* for cutting-edge visuals. These options allow developers to tailor their projects to the specific needs of their target platforms.

Unity Engine also provides extensive support for *Asset Management*. Developers can leverage Unity's Asset Store to access a vast library of prebuilt models, textures, animations, and scripts. The engine's import pipeline allows seamless

integration of assets from external tools like Blender, ensuring that creative workflows remain uninterrupted.

Unity's *Play Mode* and real-time simulation capabilities empower developers to prototype, test, and debug their projects interactively. This iterative workflow shortens development cycles and facilitates rapid experimentation.

With its versatility, powerful tools, and extensive ecosystem, Unity Engine serves as a cornerstone for real-time 3D application development, enabling creators to bring their ideas to life across a wide variety of platforms and industries [43, 5].

Raycasting in Unity Raycasting is a computational technique widely used in computer graphics and physics simulations to trace rays into 3D scenes. Leveraging Unity's physics engine, raycasting benefits from optimized tools like `Physics.Raycast`, which efficiently handles ray-object intersection calculations using the engine's built-in colliders and spatial partitioning systems.

In the context of LiDAR simulation, raycasting is employed to replicate the behavior of laser sensors, tracing rays from a virtual sensor and detecting their intersection points to generate synthetic point clouds. Unity's integration simplifies the implementation by providing high-performance raycasting capabilities and seamless interaction with 3D scene geometry.

Implementation in Unity Unity provides the `Physics.Raycast` function to perform raycasting in 3D scenes. This function uses the equations above to detect intersections between rays and object colliders in the virtual environment. A basic implementation in Unity C# is shown below:

List 3.1: Implementation of a LiDAR simulation script using Unity's `Physics.Raycast` method [44].

```
using UnityEngine;

public class LidarSimulation : MonoBehaviour
{
    void Update()
    {
        RaycastHit hit;
        Vector3 origin = transform.position;
        Vector3 direction = transform.forward;

        // Perform the raycast
        if (Physics.Raycast(origin, direction, out hit))
        {
            // Process the hit information
            Debug.Log("Object detected at a distance of: "
                + hit.distance);
        }
    }
}
```

Optimizations and configurations Several features enhance the performance and configurability of raycasting in Unity:

- **Layer Masks:** Specify which objects are considered during raycasting to improve performance.
- **Max Distance:** Limit the ray's range to simulate the maximum detection distance of a real LiDAR sensor.
- **Query Trigger Interaction:** Configure whether the ray should interact with colliders marked as triggers.

3.2.1 Key features and advantages of using Unity in this project

The decision to create a custom simulation environment emerged from extensive research into existing simulators and the scientific work in this field. Initially, the project considered improving the simulation model developed by the MSS group at DLR, which utilized Gazebo as its engine. While Gazebo is well-integrated with ROS and widely recognized for its compatibility, a critical limitation was identified: the canal water lacked implemented physics and did not interact realistically with the vessel. Consequently, the vessel's movement resembled that of a car rather than a boat, significantly reducing the realism and applicability for inland waterway scenarios.

This limitation prompted an investigation into alternatives for improving water physics. Most existing simulators focused primarily on underwater robotics,

while the few that implemented realistic surface vessel physics faced issues such as limited availability, lack of maintenance, or incompatibility with the versions of ROS1 or Ubuntu in use. After thorough evaluation, Unity was chosen as the foundation for a new simulation framework, leveraging ROS 2 and targeting Ubuntu 22.04.

Unity was selected for this project due to its ability to handle complex 3D environments and its extensive customization options, addressing the gaps identified in existing solutions. Several key features of Unity directly support the objectives of this work:

Unity's *high-fidelity rendering capabilities* enable the creation of visually accurate and detailed environments, essential for replicating real-world conditions in simulations. The *physics engine* provides tools for modeling realistic interactions, including collisions and environmental effects like rain particles or water dynamics. This is particularly important for creating realistic water interactions and vessel dynamics, essential for high-fidelity inland waterway simulations. Additionally, the use of *customizable shaders* allows for advanced effects, such as light scattering and surface wetness, which are critical for accurately simulating sensor performance under adverse conditions.

The integration with ROS 2 enables efficient real-time communication between the simulation environment and robotic frameworks, facilitating smooth data exchange and control. Unity's *cross-platform support* ensures flexibility for testing across different systems, while its *extensive asset library* accelerates development by providing ready-made models and tools. Together, these features make Unity an ideal platform for creating realistic, high-fidelity simulations tailored to the needs of this project.

In addition to these technical capabilities, Unity was selected over alternatives such as Unreal Engine or custom-built simulation tools due to its numerous strategic advantages (see Table 3.1). Unity offers ease of integration with ROS 2 through established methods and plugins, which simplify the development process and enable seamless communication between components. Its intuitive interface and extensive documentation further accelerate the creation of custom simulations.

Unity's *scalability* allows for the addition of features such as varying weather conditions or sensor models without substantial rework, while its *performance optimization tools* ensure efficient rendering and simulation even in complex scenarios.

By leveraging Unity's advanced capabilities, this project bridges the gap between theoretical models and practical applications, providing a robust platform for evaluating sensor behavior in realistic settings. Furthermore, both Unity and ROS 2 benefit from active development and support, ensuring long-term compatibility and scalability, making them an ideal choice for advancing maritime simulation [45, 46, 47, 48].

Feature	Unity	Unreal Engine	Custom
Ease of ROS 2 Integration	**High**	Medium	Low
Scalability	**High**	High	Medium
Performance Tools	**Comprehensive**	Limited	Custom
Intuitive Interface	**Yes**	No	No
Documentation	**Extensive**	Extensive	None
Asset Library	**Extensive**	Medium	None
Rendering Quality	High	**Very High**	Limited
Development Time	**Fast**	Moderate	Slow

Table 3.1: Comparison of Simulation Platforms for Robotics

3.3 Simulation Setup

This section describes the process of creating the simulation project in Unity, aimed at realistically replicating the navigation of a small vessel through canals. The development of the simulated environment involved various stages, from the initial project setup to the integration of virtual sensors and the implementation of scripts that define the system's behavior.

Figure 3.4: High-fidelity simulation environment

3.3.1 Integrating ROS 2 and Unity for simulation

The integration between *Unity* and *ROS 2* is a critical area of focus in robotics and autonomous systems and serves as a cornerstone of the contributions presented in this work. However, Unity does not natively support ROS 2, requiring

external plug-ins or tools to establish communication. This section explores the available integration options, distinguishes between bridging solutions and embedded approaches, and justifies the selection of `ros2-for-unity` for this project [49] .

Since Unity lacks out-of-the-box support for ROS 2, external solutions are essential to bridge these two ecosystems. These tools enable functionalities such as sensor data simulation, robotic behavior emulation, and control algorithm evaluation. However, existing solutions vary significantly in terms of performance, complexity, and applicability, necessitating a careful selection to achieve high-fidelity simulations.

Three primary categories of solutions address the Unity-ROS 2 integration challenge:

Bridge-based solutions Bridge-based tools act as intermediaries between Unity and ROS 2. Examples include:

- **ROS-TCP-Connector:** A widely-used bridge that uses a TCP-based protocol to connect Unity with ROS 2. While functional, this approach introduces communication latency, making it less suitable for real-time applications [50].
- **ROSSharp:** A lightweight open-source tool that provides basic connectivity between Unity and ROS 2. However, it lacks robustness and native ROS 2 features, making it unsuitable for demanding use cases. Additionally, it requires extensive manual configuration [51].

Custom socket solutions Some developers design bespoke socket-based systems for Unity-ROS 2 communication. These solutions offer flexibility but demand significant development effort. Moreover, they do not fully leverage the advanced features of ROS 2, such as Quality of Service (QoS) policies or native tools.

Embedded solutions Unlike bridge-based approaches, embedded solutions integrate Unity entities directly as native ROS 2 components. This eliminates intermediary communication layers, enabling superior performance and tighter compatibility with ROS 2's middleware stack.

Given the limitations of traditional bridge-based approaches, this project selected `ros2-for-unity`, an embedded solution, for its superior performance and integration capabilities [49]. By embedding Unity entities as native ROS 2 nodes, `ros2-for-unity` ensures seamless communication and efficient simulations.

`ros2-for-unity` provides the following key benefits:

- **High Performance:** By eliminating intermediary layers, it achieves lower latency and higher throughput.
- **Native ROS 2 Compatibility:** Unity entities function as native ROS 2 nodes, fully supporting ROS 2 tools, Quality of Service (QoS) settings, and time synchronization.
- **Rich Toolset:** Includes essential simulation tools such as transformations, sensor interfaces, clock synchronization, and a `MonoBehavior`-encapsulated spinning loop for efficient node execution.
- **Customizable Messaging:** Supports all standard ROS 2 messages and automatically generates custom messages during the build process, simplifying development workflows.
- **Ease of Integration:** Packaged as a Unity asset, it streamlines the setup process for simulation projects.

By directly integrating with the ROS 2 middleware stack, including the `rcl` layer, `ros2-for-unity` overcomes the typical bottlenecks associated with bridging solutions. This approach optimizes performance, ensures seamless compatibility, and simplifies the development of advanced robotic simulations.

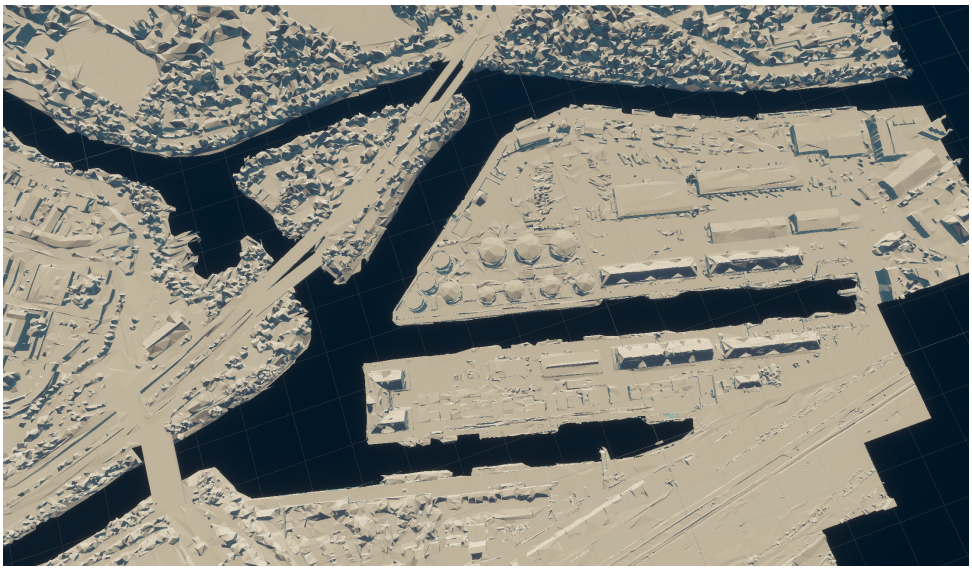
In conclusion, `ros2-for-unity` was selected for its ability to embed Unity entities as native ROS 2 nodes, ensuring real-time performance and scalability. This solution aligns with the project's requirements for high-fidelity simulations and tight integration with the ROS 2 ecosystem.

3.3.2 Components of the simulation framework

3.3.2.1 Maps and 3D models

The first step in building the simulation environment was to design the virtual representation of the Westhafen canals in Berlin. Using real-world measurements and mapping data, a base map was imported into Unity to serve as the foundation of the scene (Fig: 3.5). This map was complemented by 3D models and objects used to test the performance. The models were carefully chosen or created to strike a balance between visual fidelity and computational efficiency, ensuring the simulation could run smoothly while maintaining realism.

(a)



(b)

Figure 3.5: Westhafen Satellite Image (a), Westhafen Map on Unity (b)

The use of graphical tools like Blender [6] allows for precise extraction and refinement of individual objects from the imported map data. By isolating key structures, such as buildings or other relevant features, these objects can be enhanced or modified to improve their realism and suitability for the simulation. For example, specific buildings were extracted from the Westhafen area and integrated as standalone 3D models into the Unity scene (Fig: 3.6). This process not only ensures visual consistency but also enables targeted testing of LiDAR performance on distinct objects within the simulation environment.

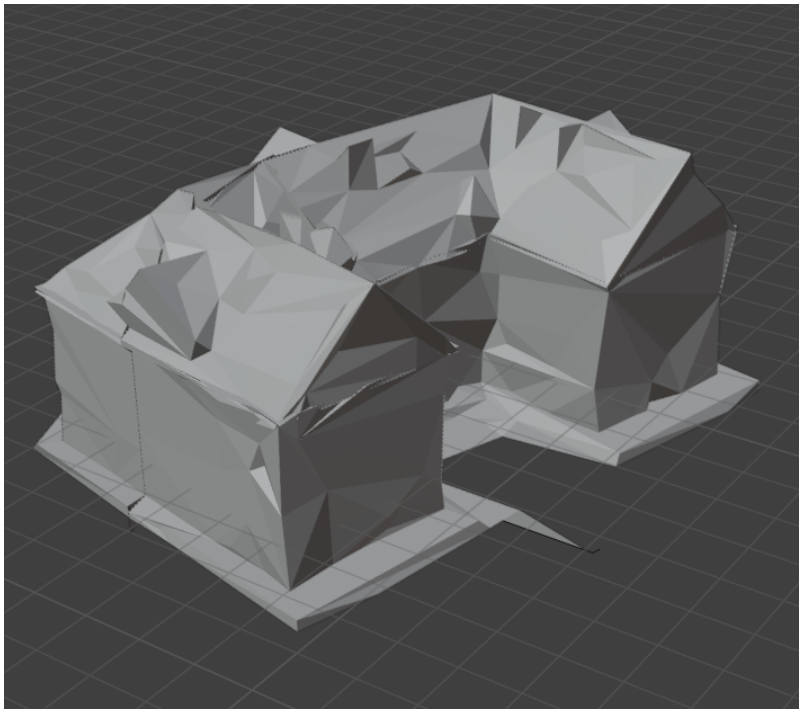
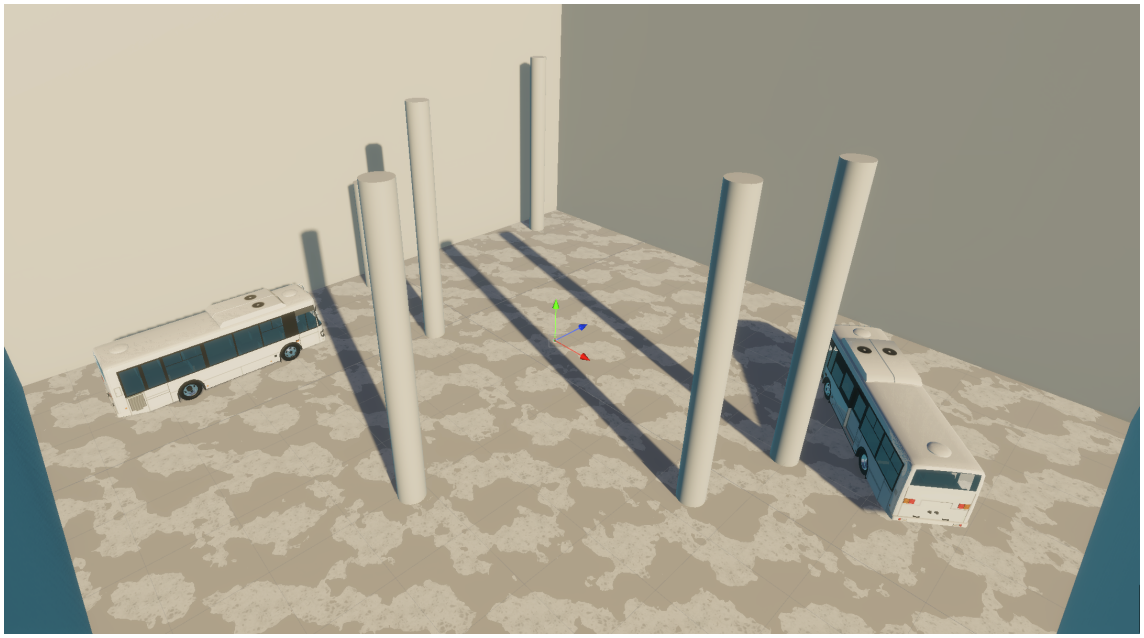
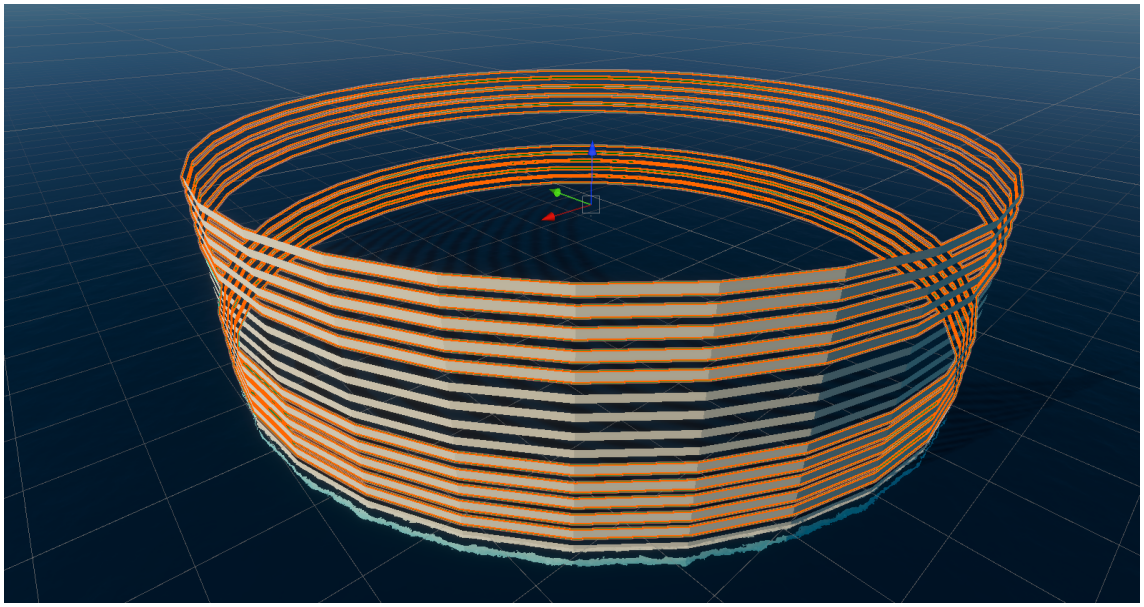


Figure 3.6: An extracted building model from the Westhafen map.

To ensure the proper functioning of the simulation framework and to conduct specific tests on the sensors, additional scenes were developed. These test scenarios were designed to isolate and evaluate individual components, such as LiDAR accuracy and rain simulation effects. Each scene was tailored to address a particular aspect of the project, allowing for a systematic assessment of sensor performance under controlled conditions. These environments provided valuable insights into the robustness and reliability of the simulation framework.



(a)



(b)

Figure 3.7: Additional test scenes developed for sensor validation and framework evaluation. (a) LiDAR accuracy test scene with structured objects and distinct characteristics to assess sensor performance. (b) Sixteen-cylinder scene designed to isolate and evaluate the performance of the Velodyne VLP-16 LiDAR sensor.

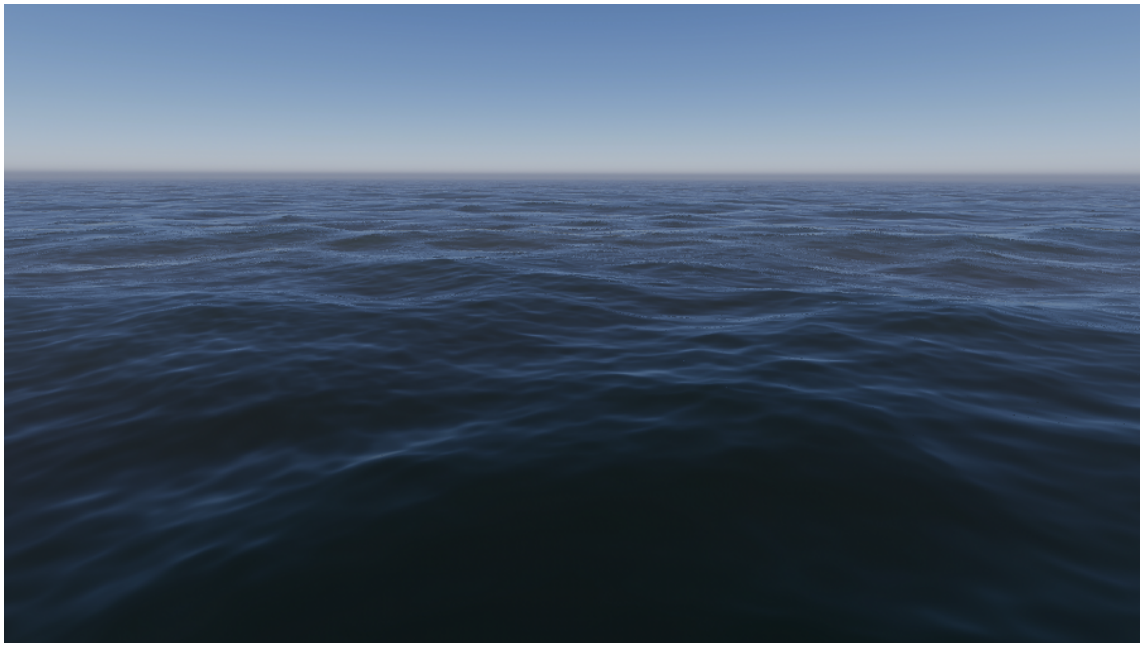
As shown in Figure 3.7(a), the LiDAR accuracy test scene was designed with various objects, each featuring different characteristics, to evaluate how the Li-

DAR sensor performs in diverse scenarios and to test the rain model introduced later. The scene depicted in Figure 3.7(b) was created for a more precise evaluation of the LiDAR sensor's performance. Specifically, Velodyne VLP-16, with its 360° field of view and 16 layers, benefits from this setup as it isolates individual rings, enabling a detailed analysis of how each layer of the LiDAR operates.

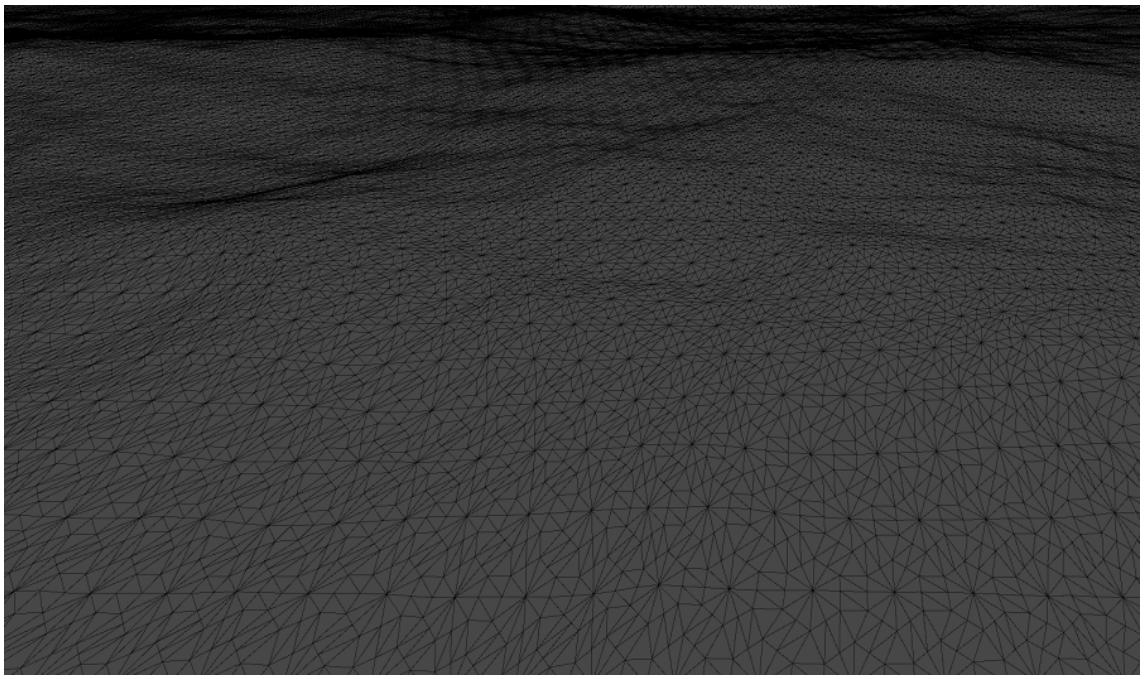
3.3.2.2 High Definition Render Pipeline for ocean simulation

One of the critical components of the scene was the simulation of water. Unlike traditional inland waterway simulations that often neglect water dynamics, this environment incorporates a dynamic water body using Unity's physics engine. The water was configured to respond to interactions with objects, such as the AURORA vessel, creating ripples and drag forces to simulate realistic conditions. This setup added a layer of complexity to the environment, enabling a more authentic testing platform for sensor and system evaluations.
Technical parameters and configuration of the game objects is detailed in Appendix A.

Water physics The High Definition Render Pipeline (HDRP) in Unity provides a robust framework for creating visually realistic and physically accurate simulations of large water bodies, such as oceans, lakes, and rivers. HDRP is designed to handle the complexity of water dynamics, leveraging advanced rendering techniques and integrated physics systems. This makes it particularly suitable for projects requiring high-fidelity environmental simulations.



(a) Rendered HDRP water in Unity with realistic lighting and reflections.



(b) Underlying water mesh structure showing the geometry used for wave simulation.

Figure 3.8: Visualization of the HDRP water system in Unity [5]: (a) rendered ocean surface and (b) water mesh geometry.

Advanced water system Starting with Unity 2023, HDRP includes an advanced water simulation system tailored for oceanic and maritime environments. This system models water dynamics using Gerstner waves and Fast Fourier Trans-

form (FFT) algorithms to simulate realistic wave patterns. Is it possible to fine-tune parameters such as wave height, speed, and direction to replicate specific conditions, such as calm waters or stormy seas.

Environmental interaction HDRP seamlessly integrates atmospheric and lighting effects with water simulation. Features such as real-time reflections, volumetric fog, and global illumination enhance the visual consistency of the scene.

Object interaction and physics One of HDRP's strengths is its ability to simulate the interaction between water and objects. Ships, buoys, and other entities can interact with the water surface, generating realistic waves, wakes, and splashes. This interaction is achieved through Unity's integrated physics engine, which ensures that objects respond dynamically to the water's motion.

Applications in this project While Unity's HDRP is renowned for delivering unparalleled realism and advanced rendering capabilities, its default configurations can be resource-intensive. To address this, the HDRP settings were carefully optimized in this project to ensure computational efficiency without compromising the quality required for realistic maritime simulations. Adjustments were made to parameters such as rendering resolution, lighting effects, and wave complexity to strike a balance between performance and visual fidelity.

By tailoring HDRP to the specific needs of this project, the simulation framework achieves the efficiency necessary for real-time operation on high-performance hardware, enabling robust testing and validation of the vessel's sensors and navigation systems. The ability to model dynamic water behaviors and environmental effects with reduced resource consumption makes HDRP an invaluable tool in this context. Through these optimizations, HDRP supports the creation of a realistic and immersive simulation environment while meeting the computational constraints critical for this project.

3.3.2.3 The AURORA vessel

The centerpiece of the simulation is the AURORA vessel, a pleasure craft utilized by the Multi Sensors Systems (MSS) group at DLR. In real measurement campaigns, this vessel is equipped with a full sensor set mounted on a frame on its exterior and traveled typically along the urban inland waterway from Berlin. This section describes the 3D model of AURORA, its integration into the simulation framework, and its role as the primary testbed for sensor and algorithm evaluation.

Figure 3.9: The AURORA vessel traveling along the Berlin urban canals

The 3D model of the AURORA vessel was designed to reflect its real-world counterpart in terms of dimensions and structure. Using CAD files and reference images, the model was developed and imported into Unity. To balance realism with computational efficiency, unnecessary details were simplified without compromising the essential features of the vessel. Figure 3.10 illustrates the final 3D model integrated into the simulation.

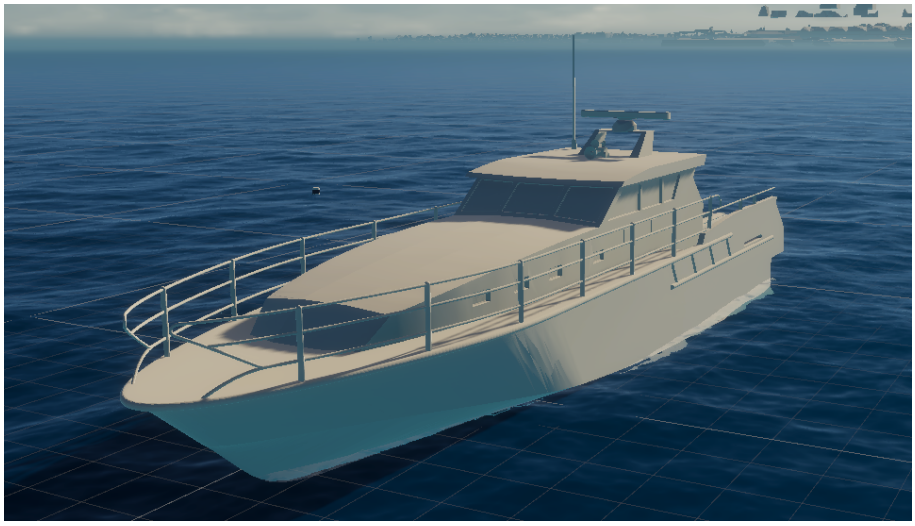


Figure 3.10: 3D model of the AURORA vessel integrated into Unity.

3.3.2.4 Dynamic characteristics and physics configuration

To accurately simulate the behavior of the AURORA vessel, its physical properties and dynamic characteristics were carefully configured in Unity. Key parameters included:

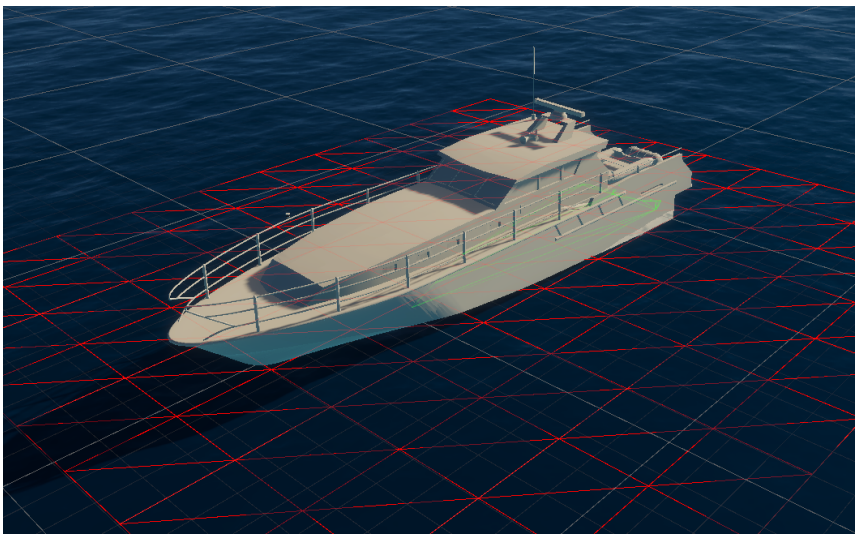
- **Weight and Buoyancy:** The vessel's weight was set according to its real-world specifications, and buoyancy was calculated to ensure proper interaction with the simulated water surface.
- **Propulsion System:** A propulsion model was implemented to simulate the vessel's movement, allowing precise control of speed and direction.
- **Hydrodynamic Effects:** Simplified hydrodynamic forces, such as drag and resistance, were incorporated to mimic the vessel's behavior in water.

These configurations ensured that the AURORA vessel responded realistically to external forces, such as waves and currents generated by the HDRP water system.
More details of parameters used can be found in Appendix A.

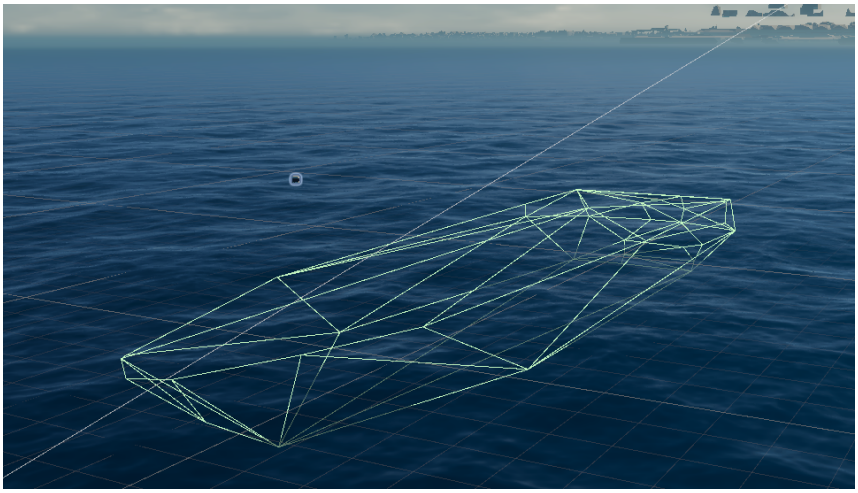
Grid and mesh representation of the AURORA vessel To achieve an accurate simulation of the AURORA vessel's behavior in the virtual environment, both its grid layout and mesh structure were carefully designed and implemented.

Figure 3.11a illustrates the grid representation of the vessel. The grid is used for aligning the vessel within the simulation environment and ensuring proper interaction with the water surface. This grid defines the boundaries and spatial positioning of the vessel, helping to manage its placement relative to environmental elements such as waves, currents, and other objects.

Figure 3.11b shows the vessel's mesh structure. The mesh represents the 3D geometry of the vessel and is essential for calculating physical interactions with the water. By defining the hull's shape, the mesh enables the simulation of hydrodynamic effects, including drag, buoyancy, and wave resistance. These effects are critical for replicating realistic navigation behavior in the simulation.



(a) Grid representation of the AURORA vessel for alignment within the simulation environment.



(b) Mesh structure of the AURORA vessel, used for hydrodynamic calculations.

Figure 3.11: Grid and mesh representations of the AURORA vessel used in the simulation framework.

3.3.2.5 Role in the simulation framework

The AURORA vessel serves as the primary platform for evaluating sensor performance and navigation algorithms under realistic conditions. Sensors such as LiDARs, and IMUs were virtually mounted on the vessel to simulate real-world scenarios. The vessel was tested in various environments, including calm waters, rough seas, and adverse weather conditions, to assess the robustness and accuracy of the integrated systems. By using the AURORA vessel as a testbed, the simulation framework provides a controlled yet realistic environment for validating the performance of autonomous navigation technologies.

3.3.2.6 Modular and scalable design

The environment was developed with modularity and scalability in mind,
enabling future additions and refinements. Each component of the simulation the map, water, AURORA, and sensors was designed to function independently, allowing for adjustments or replacements as needed. This modular approach ensures that the environment can adapt to new requirements or expanded use cases, making it a flexible tool for research and development.

3.3.3 Sensor integration

To enable meaningful experimentation, the simulation environment includes a suite of sensors mounted on the AURORA vessel [52, 53]. These sensors, including LIDAR and IMU were integrated into the Unity scene and calibrated to replicate their real-world counterparts. Each sensor was carefully positioned and configured to ensure accurate data generation, allowing for the simulation of realistic conditions such as sensor noise, occlusions, and environmental interferences. This setup supports the collection of comprehensive datasets for further analysis and validation.

Figure 3.12: IMU and LiDAR sensors integrated into Unity. The sensors are positioned relative to the AURORA vessel's structure, ensuring accurate alignment with the ship's coordinate system. Their placement is designed to be easily adjustable, allowing for repositioning to different locations on the vessel. This flexibility enables testing various configurations and scenarios, enhancing the versatility of the simulation environment.

3.3.3.1 Inertial Measurement Unit

The Inertial Measurement Unit (IMU) is a critical sensor for estimating the vessel's orientation, angular velocity, and acceleration. In the simulation, a virtual IMU was integrated into the AURORA vessel to replicate the functionality of its real-world counterpart. The IMU was configured to generate data corresponding to six degrees of freedom (6-DoF), including linear acceleration along the x, y, and z axes, as well as angular rates around these axes.

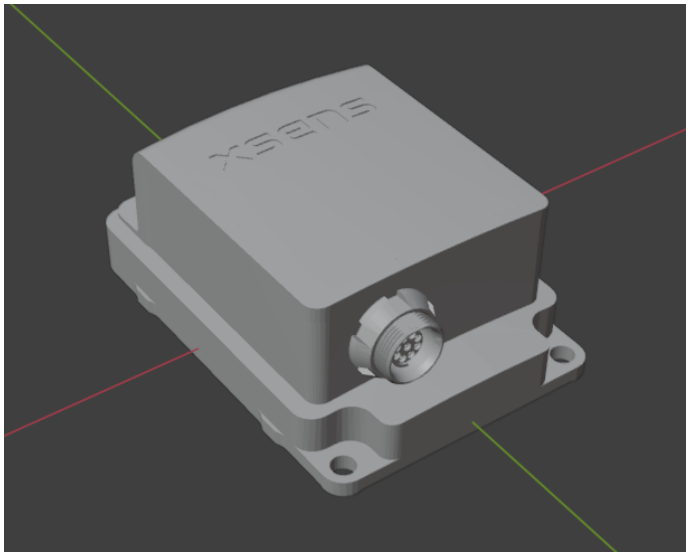


Figure 3.13: 3D model of the IMU sensor integrated into Unity.

Positioning and calibration The IMU was carefully positioned within the virtual vessel to ensure its measurements aligned with the vessel's reference frame. Calibration was performed to match the characteristics of a typical IMU sensor, including the introduction of sensor-specific parameters such as bias, scale factor, and random noise. These factors were tuned to replicate real-world inaccuracies, allowing for realistic testing of sensor fusion algorithms.

Simulation of environmental effects To enhance realism, environmental effects such as wave-induced vibrations and ship motion dynamics were incorporated into the IMU's data stream. These effects simulate the challenges faced by real IMUs operating on vessels in dynamic maritime environments. The simulated data captures the complex interactions between the vessel and its surroundings, providing valuable input for navigation and control algorithms.

IMU message structure in ROS 2 In the simulation framework, the virtual IMU uses the `sensor_msgs/Imu` message format, as defined in ROS. This message provides the necessary fields to represent the orientation, angular velocity, and linear acceleration of the vessel. The structure of the message is as follows:

List 3.2: IMU Message Definition in ROS 2 Format.

```
Header header
uint32 seq
time stamp
string frame_id
geometry_msgs/Quaternion orientation
float64 x
float64 y
float64 z
float64 w
float64[9] orientation_covariance
geometry_msgs/Vector3 angular_velocity
float64 x
float64 y
float64 z
float64[9] angular_velocity_covariance
geometry_msgs/Vector3 linear_acceleration
float64 x
float64 y
float64 z
float64[9] linear_acceleration_covariance
```

Fields description

- **Header:** Contains metadata such as the timestamp and frame of reference.
  - `seq`: Sequential ID of the message.
  - `stamp`: Timestamp indicating when the data was generated.
  - `frame_id`: Reference frame for the IMU data (e.g., the vessel's coordinate frame).
- **Orientation:** A quaternion (`x, y, z, w`) representing the IMU's orientation in the specified frame.
- **Angular Velocity:** A vector (`x, y, z`) representing the rotational velocity in radians per second.
- **Linear Acceleration:** A vector (`x, y, z`) representing the acceleration in meters per second squared.
- **Covariances:** Three 3×3 covariance matrices (orientation, angular velocity, and linear acceleration) that quantify the uncertainty of each measurement.

3.3.3.2 Light Detection And Ranging

The LiDAR was integrated into Unity and configured to publish data using the `sensor_msgs/PointCloud2` message format, which is the standard for LiDAR data in ROS 2.

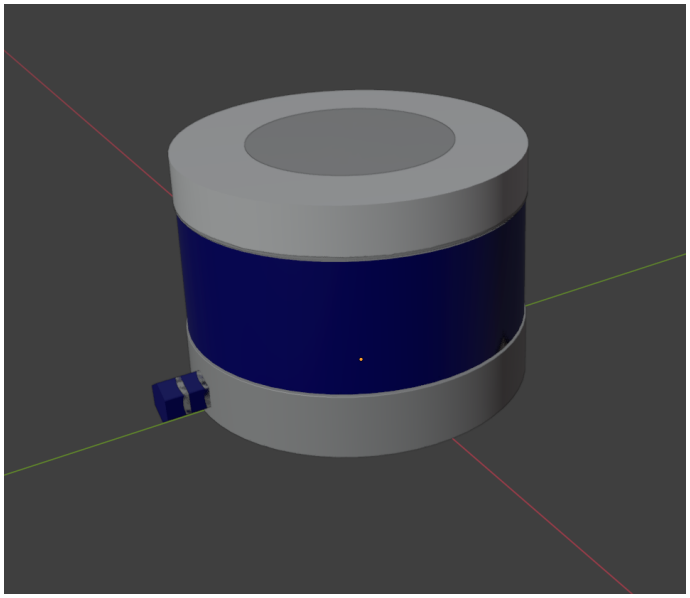


Figure 3.14: 3D model of the Velodyne LiDAR sensor integrated into Unity.

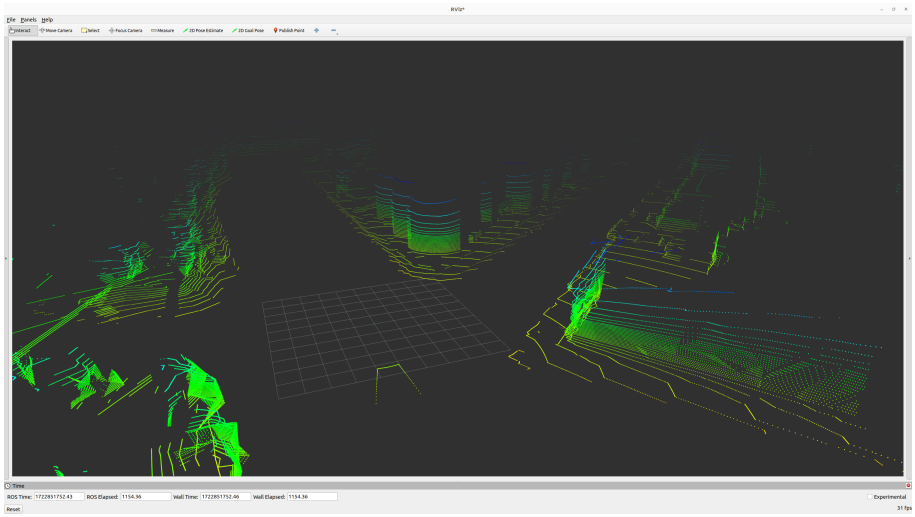


Figure 3.15: Point cloud data generated by the LiDAR sensor, visualized in RViz, captured in an open water environment. This visualization demonstrates the sensor's ability to map unobstructed surroundings.

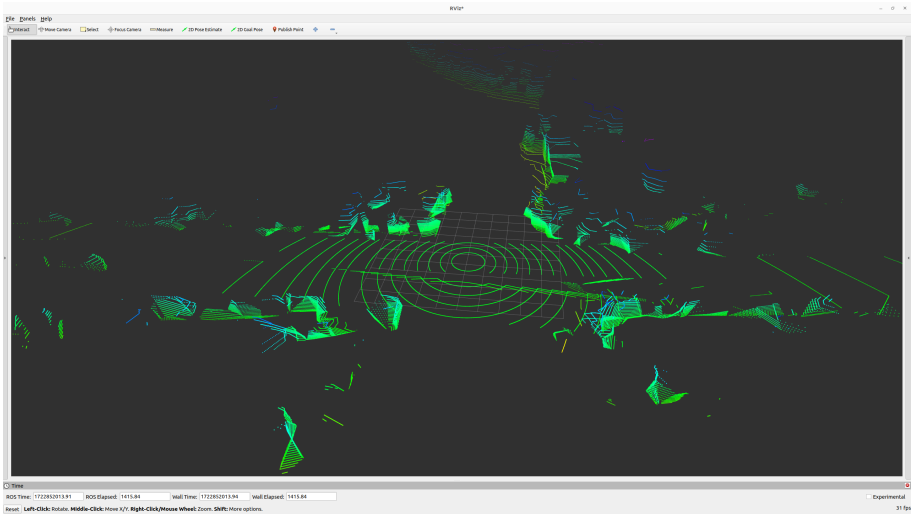


Figure 3.16: Point cloud data generated by the LiDAR sensor, visualized in RViz, captured under a bridge. This visualization highlights the detailed mapping of complex structures and the simulation's environmental mapping capabilities.

PointCloud2 message structure in ROS The virtual LiDAR sensor in this project publishes point cloud data using the `sensor_msgs/PointCloud2` message format, which is a standard in ROS for representing 3D point cloud data.

In this project, the `PCD` format has been selected due to its suitability for real-time processing and its ability to efficiently handle modifications to point cloud data. The `PCD` format is specifically designed for point cloud storage, offering flexibility in managing both geometric (`x`, `y`, `z`) and attribute data, such as `intensity`. Its binary encoding ensures compact storage and high-speed read/write operations, which are critical for real-time simulations.

One of the objectives of this work is to simulate the effects of adverse weather conditions, particularly rain, on LiDAR measurements. This involves systematically adding noise to the original point cloud data by altering the positions (`x`, `y`, `z`) and intensities of points according to the developed rain simulation model. The `PCD` format facilitates this process by allowing seamless integration of modified data, ensuring consistency and precision during updates. Furthermore, its compatibility with various tools and libraries, such as Point Cloud Library (PCL), provides advanced capabilities for efficient data manipulation, visualization, and analysis. These features make `PCD` the optimal choice for simulating and analyzing the effects of rain on LiDAR data in a computationally efficient manner.

The structure of the message is as follows:

List 3.3: Point Cloud Message Definition in ROS 2 Format.

```
Header header
uint32 seq
time stamp
string frame_id
uint32 height
uint32 width
sensor_msgs/PointField[] fields
string name
uint32 offset
uint8 datatype
uint32 count
bool    is_bigendian
uint32  point_step
uint32  row_step
uint8[] data
bool    is_dense
```

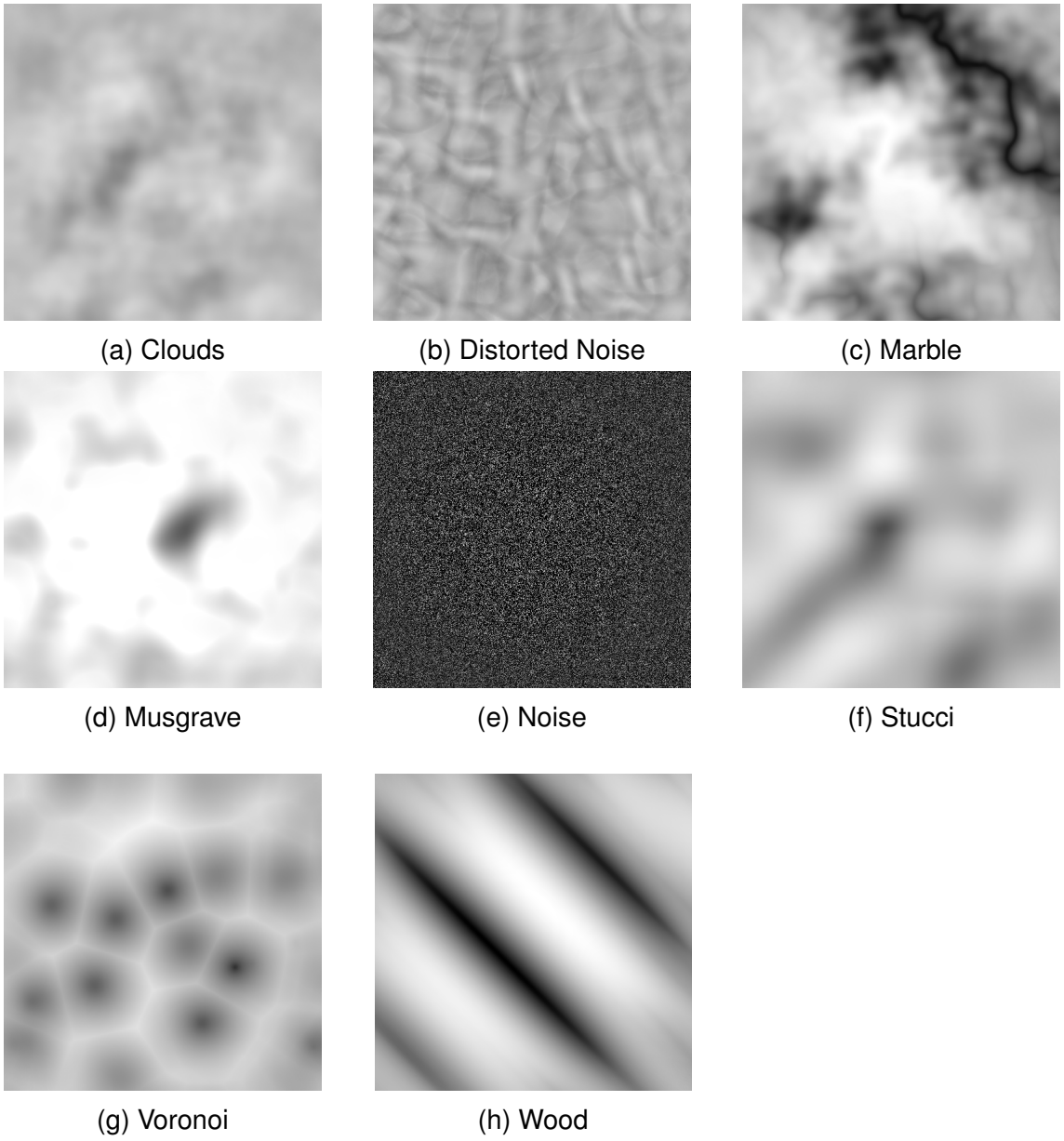
Fields description

- **Header:** Contains metadata, such as the timestamp and reference frame for the point cloud data.
  - `seq`: Sequential ID of the message.
  - `stamp`: Timestamp indicating when the point cloud was captured.
  - `frame_id`: The coordinate frame in which the point cloud is expressed (e.g., the LiDAR sensor's frame).
- **Height and Width:** Define the dimensions of the point cloud data. For unstructured point clouds, `height` is 1 and `width` is the number of points.
- **Fields:** An array of `PointField` structures, describing the layout of the point cloud data. Each `PointField` specifies:
  - `name`: Name of the field (e.g., `x`, `y`, `z`, `intensity`).
  - `offset`: Byte offset of the field in each point record.
  - `datatype`: Type of data stored in the field (e.g., float32).
  - `count`: Number of elements in the field (e.g., 1 for `x`, `y`, and `z`).
- **Data:** The actual point cloud data stored as a byte array.
- **is_bigendian:** Specifies the byte order of the data.
- **Point and Row Step:** Define the size of a single point and a row of points, respectively, in bytes.
- **is_dense:** Indicates whether the point cloud contains invalid (NaN) points.

3.3.3.3 Interaction of LiDAR with Unity textures

In this project, the virtual LiDAR sensor interacts with Unity textures to simulate the intensity of laser returns based on the properties of the surfaces in the environment. This interaction is facilitated using `RGLUnityPlugin`, which enables realistic LiDAR simulations by incorporating texture-based intensity calculations.

Texture integration A texture in Unity represents a 2D image mapped to a surface (Mesh) in the virtual environment. For the LiDAR, the texture is used to calculate the intensity of a hit point when a laser beam interacts with a surface. The texture is expected to be a grayscale image with 8-bit red channel data, representing the reflectivity of the surface. If the texture is omitted, the intensity of the return is set to zero, indicating no reflectivity.



(a) Clouds (b) Distorted Noise (c) Marble

(d) Musgrave (e) Noise (f) Stucci

(g) Voronoi (h) Wood

Figure 3.17: Various grayscale textures used in the simulation for representing different surface properties [6].

Calculation of intensity The intensity of a hit point is determined by sampling the texture at the texture coordinates of the mesh. These coordinates are interpolated from the triangle vertices of the hit point, which are provided during the mesh creation process. The texture coordinates must be in the range [0, 1]; otherwise, the texture will be tiled across the surface. This approach ensures that the LiDAR sensor accurately captures surface properties, such as material reflectivity, and simulates varying return intensities based on surface characteristics.

Flexibility of texture assignment In Unity, textures are assigned to entities (e.g., meshes or objects) in the environment. Multiple entities can share the same texture, allowing for efficient resource usage and consistency in reflectivity across similar surfaces.

By integrating texture-based intensity calculations, the LiDAR sensor in this project is able to simulate real-world interactions with different materials and surface properties. This capability is essential for several applications. Additionally, it supports material differentiation by simulating how the LiDAR reacts to surfaces with varying reflectivity, such as asphalt, metal, or water.

The texture-based intensity calculations also introduce variability in intensity returns based on surface textures. This provides critical input for advanced perception and classification algorithms, which rely on intensity patterns for object detection and material identification. Furthermore, the LiDAR data is vital for testing Simultaneous Localization and Mapping (SLAM) algorithms under realistic conditions, including scenarios with adverse weather, enhancing the reliability of navigation and perception systems in diverse environments.

3.4 Introducing The Rain Simulation Model

In this section, we address the technical decisions made in the development of the rain simulation framework for LiDAR sensors. These decisions are crucial for accurately modeling the effects of rain on LiDAR scans and ensuring the reliability of the simulated data for testing and validation purposes.

Rain introduces challenges for LiDAR sensors, such as backscattering, absorption, and occlusion caused by raindrops [54, 55, 56]. These effects significantly impact the performance of LiDAR systems by introducing noise, reducing signal accuracy, and increasing the complexity of data interpretation. To address these issues, the framework uses a probabilistic approach to simulate the interaction between rain and LiDAR beams, balancing computational efficiency and physical accuracy.

The following subsections outline the specific considerations and methodologies used to implement the rain simulation, along with a justification for the modeling choices made.

Rain primarily affects LiDAR sensors in two key ways:

- **Optical Interference:** Raindrops scatter and absorb the laser beams emitted by the LiDAR, leading to erroneous distance measurements and increased noise in the data.
- **Occlusion:** Dense rain can obscure objects or features in the environment, reducing the effective range of the sensor and introducing challenges for algorithms that depend on clear and consistent data.

Evaluating the resilience of perception and Simultaneous Localization and Mapping (SLAM) algorithms under such conditions is a necessary step toward creating robust autonomous systems. SLAM algorithms, which depend heavily on accurate environmental mapping and localization, may fail or produce suboptimal results when the input data is corrupted by noise caused by rain. A robust SLAM system must be capable of compensating for these challenges to maintain reliability.

Simulating rain in a controlled virtual environment provides an efficient and cost-effective way to test and improve these systems. Unlike real-world testing, where replicating specific rain conditions can be costly and unpredictable, simulation allows developers to:

- Introduce configurable and repeatable rain scenarios.
- Analyze the specific impact of different rain intensities on LiDAR data.
- Develop and validate sensor fusion and filtering techniques to mitigate rain-induced noise.

Moreover, incorporating a rain simulation enhances the validity of autonomous system testing by exposing the algorithms to realistic and diverse conditions. This ensures that the systems are prepared for the complexities of operating in the real world, improving their robustness and reliability.

In summary, rain simulation is a critical step in advancing autonomous technology. It enables developers to identify and address the limitations of perception systems and SLAM algorithms under adverse weather conditions, contributing to the overall safety and efficiency of autonomous vehicles in challenging environments.

3.4.1 Model structure

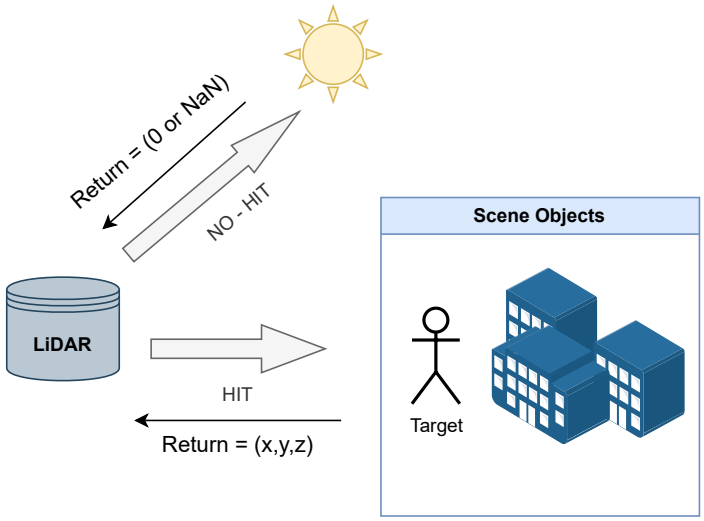


Figure 3.18: Graphical representation of LiDAR return classification.

Workflow of the rain noise simulation process The rain noise simulation model processes each point in the LiDAR data as follows:

1. **Point classification:** Each point is classified as a "hit" or "non-hit" based on its distance from the LiDAR sensor. Points exceeding the maximum distance threshold are labeled as "non-hits."
2. **Calculation of μ (mean number of raindrop interactions):** For each point classified as a "hit," the model calculates μ, which represents the mean number of raindrop interactions within the laser beam's volume. This calculation relies on raindrop size distribution equations that account for the rain intensity, laser beam radius, and distance traveled by the beam. Based on the μ value, the number of raindrops that intersect the laser beam is sampled using a Poisson distribution.
3. **Evaluation of Impact Intensity:** Using the LiDAR's initial returned intensity and the number of raindrops intersecting the laser beam, the model evaluates whether the impact is strong enough to be classified as a valid detection. This process involves RIRD, RIO and DT discussed below.
    - If $RIRD > DT$, the interaction is classified as a false positive, adding a spurious point to the cloud.
    - If $RIO > DT$ and $RIRD$ is negligible, the point is classified as a regular detection.

- If neither $RIRD$ nor RIO exceed DT, the point is excluded as a false negative.

4. **Position adjustment for impacted points:** For points that interact with a significant number of raindrops, the position is adjusted slightly to reflect the scattering effects caused by water droplets.

5. **Final intensity adjustment:** The final intensity of the point is adjusted using Beer-Lambert's Law to account for the scattering and absorption effects. This ensures the intensity is consistent with physical reflectivity properties.

6. **Output the modified point cloud:** After processing all points, the model outputs a modified point cloud containing:

   - **Normal returns:** Points that represent valid detections with their adjusted intensity and position.
   - **False returns:** Simulated points caused by raindrop interactions, with adjusted intensity and position to reflect the scattering effects.
   - **Eliminated points:** Points removed from the cloud due to high attenuation caused by rain, where the intensity fell below the detection threshold.

The model dynamically adjusts rain intensity (RR) during simulation, allowing for real-time changes in weather conditions. Parallel processing using OpenMP [57] ensures computational efficiency, enabling the simulation to handle large point clouds in real-time.

The noisy point cloud generated by the model retains the same format and frame rate as commercial LiDAR sensors, making it suitable for integration into autonomous vehicle testing platforms. The output simulates realistic environmental effects, including attenuation, scattering, and false detections caused by rainfall.

3.4.2 Detailed technical justification

The proposed model meticulously addresses these challenges through a combination of deterministic and probabilistic approaches [58, 59, 60, 15, 61, 62, 63, 64].

To achieve this, the rain model incorporates key factors such as drop size distribution, beam interaction modeling, and dynamic rain intensity variations, ensuring a realistic representation of real-world behavior in the simulated point cloud data.

3.4.2.1 Mathematical representation of rain effects

Drop size distribution: The model uses the Marshall-Palmer exponential distribution to define raindrop sizes:

$$N(D) = N_0 \cdot \exp(-\Lambda D), \tag{3.1}$$

where:

- $N(D)$ represents the number of raindrops per unit volume with a diameter D (units: $\mathrm{m}^{-3}\,\mathrm{mm}^{-1}$),
- D is the raindrop diameter (units: mm),
- N_0 is the intercept parameter, typically $8000\,\mathrm{m}^{-3}\,\mathrm{mm}^{-1}$ for rain, and
- Λ is the slope parameter (units: mm^{-1}), which is a function of rain intensity R (units: mm/h):

$$\Lambda = 41 \cdot R^{-0.21}. \tag{3.2}$$

To calculate the total density of raindrops per unit volume, N_{total}, we integrate $N(D)$ over the range of possible raindrop diameters:

$$N_{\text{total}} = \int_{D_{\text{min}}}^{D_{\text{max}}} N(D)\, dD, \tag{3.3}$$

where:

- D_{min} and D_{max} are the minimum and maximum diameters of raindrops considered (units: mm) [12].

This integral is performed because $N(D)$ describes the droplet density as a function of diameter. By integrating over all possible droplet sizes, we obtain the total number of raindrops per unit volume (m^{-3}), which can then be used to calculate the expected number of raindrops within the LiDAR beam.

Beam interaction and probabilistic modeling: The expected number of raindrops, μ, within a LiDAR beam path is calculated as:

$$\mu = V_{\text{beam}} \cdot N_{\text{total}}, \tag{3.4}$$

where:

- V_{beam} is the volume of the LiDAR beam (units: m^3), approximated as:

$$V_{\text{beam}} = \pi r_{\text{beam}}^2 \cdot \text{distance}, \tag{3.5}$$

  where:

  - r_{beam} is the radius of the beam (units: m),
  - distance is the range of the beam (units: m).

The **Poisson distribution** is used to model the number of raindrops intercepted by the LiDAR beam:

$$p(n) = \frac{\mu^n \exp(-\mu)}{n!}, \tag{3.6}$$

where:

- $p(n)$ is the probability of encountering n raindrops (dimensionless),
- n is the number of raindrops (dimensionless), and
- μ is the expected number of raindrops (dimensionless, as demonstrated above).

This distribution is particularly suitable because:

- **Rare Events:** The interaction between the LiDAR beam and individual raindrops is infrequent, given the relatively low droplet density compared to the beam's volume. The expected number of raindrops, $\mu = V_{\text{beam}} \cdot N_{\text{total}}$, is dimensionless, resulting from the multiplication of the beam's volume (in m^3) and the total droplet density (in m^{-3}).
- **Independence of Events:** Each droplet's presence along the beam path is independent of others, as their distribution within the atmosphere is random and uniform. This independence aligns with the assumptions of the Poisson distribution, making it a suitable model for this scenario.

The attenuation of the LiDAR beam in rain is governed by the Lambert-Beer law [65, 66, 42, 67], which models the exponential reduction of signal intensity as:

$$P_r(z) = P_0 \cdot \exp\left(-2 \int_0^z \alpha(r) dr\right), \tag{3.7}$$

where $P_r(z)$ is the received power at distance z, P_0 is the initial power, and $\alpha(r)$ is the attenuation coefficient at a given distance r.

This model accounts for the fact that as the LiDAR beam travels through the atmosphere in rainy conditions, it encounters water droplets that scatter and absorb the beam's energy. The attenuation coefficient $\alpha(r)$ quantifies the loss of intensity per unit distance due to these interactions. It depends on the properties of the rain, including the size, shape, and distribution of raindrops, as well as the wavelength of the LiDAR beam.

The integral $\int_0^z \alpha(r) dr$ represents the cumulative effect of attenuation over the beam's path from the sensor to the distance z. Since the LiDAR beam travels

both to the target and back to the sensor, the total attenuation is doubled, hence the factor of 2 in the exponent.

This relationship highlights the *exponential decay*. The received power $P_r(z)$ decreases exponentially with distance, meaning that as the beam travels further, the signal weakens more rapidly due to the accumulation of scattering and absorption effects.

The attenuation coefficient α comprises scattering (μ_s) and absorption (μ_a) components:

$$\alpha = \mu_s + \mu_a. \tag{3.8}$$

The coefficients μ_s and μ_a are derived based on the raindrop size distribution and material properties, as shown in Table 3.2.

Rainfall Rate (mm/h)	**Scattering Coefficient μ_s (m^{-1})**	**Total Attenuation Coefficient α (m^{-1})**
5.0	0.0013	0.00132
12.5	0.0024	0.00244
25.0	0.0038	0.00387
100.0	0.0097	0.00991

Table 3.2: Attenuation Coefficients for Different Rainfall Rates

The table illustrates the relationship between rainfall intensity and attenuation coefficients, highlighting the significant impact of heavy rain on signal loss. These coefficients were derived from experimental data and theoretical analyses based on the work in [61].

3.4.2.2 Illustrative models of light attenuation in rain

The two diagrams presented in Figures 3.19 and 3.20 provide a comprehensive visual interpretation of the Lambert-Beer law and its practical implications for LiDAR signal attenuation in rain.

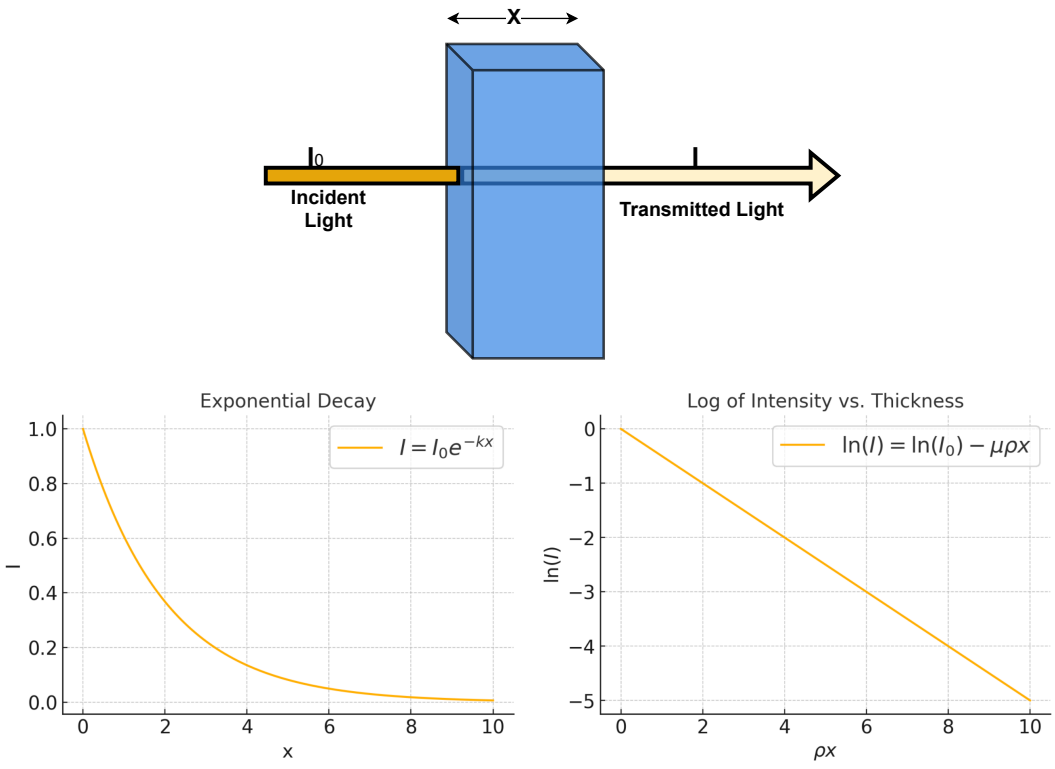


Figure 3.19: Theoretical Model of Absorption and Scattering Effects.

Figure 3.19 illustrates the theoretical framework for attenuation. The diagram highlights the Lambert-Beer equation, where the initial intensity I_0 diminishes as the beam passes through an absorbing medium of thickness x. This exponential behavior is fundamental for quantifying the extent of attenuation. The linear relationship between $\ln(I)$ and the distance x is particularly useful for deriving the coefficients μ_a and μ_s through experimental calibration.

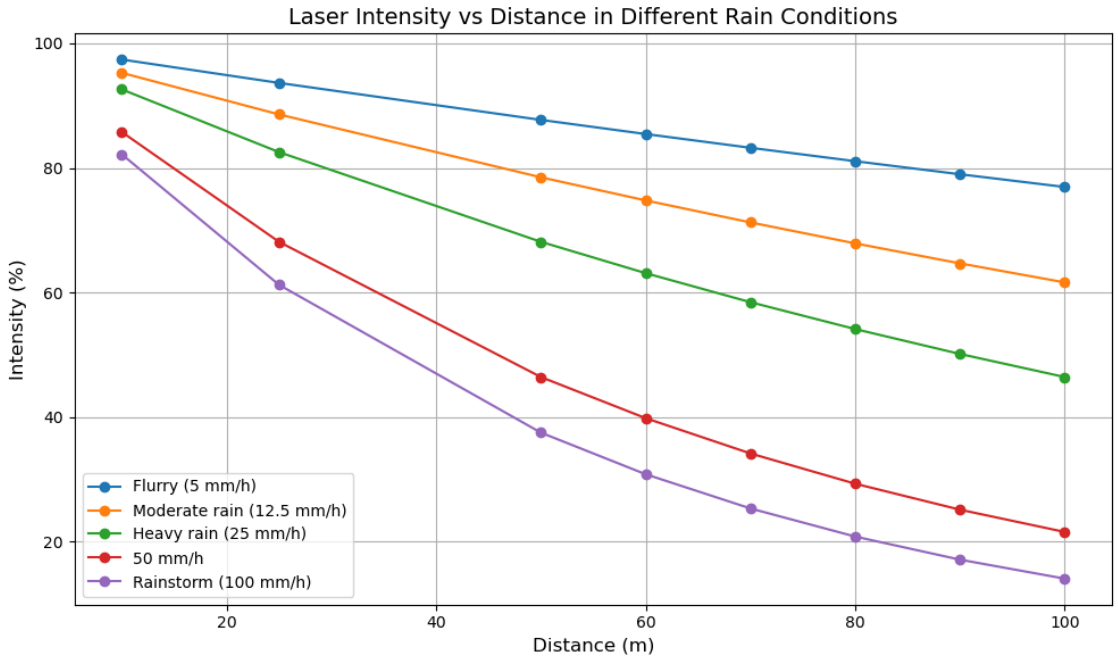


Figure 3.20: Laser intensity vs distance under different rain conditions. Data obtained from the rain simulation model.

Figure 3.20 shows the reduction in LiDAR signal intensity as a function of distance for various rainfall rates. This chart emphasizes the exponential decay described by the Lambert-Beer law, where higher rainfall rates (e.g., 50 mm/h and 100 mm/h) correspond to a steeper decline in signal intensity. The attenuation is primarily influenced by the scattering and absorption properties of water droplets, which directly affect the coefficients μ_s and μ_a. The visualization aids in understanding the practical implications for LiDAR systems, where the transmitted power $P_r(z)$ determines the effective detection range under varying environmental conditions.

Point classification and modification process Given the number of water droplets intersected by each laser beam and the corresponding intensity loss along its path, each point in the LiDAR cloud is classified based on specific criteria as *Regular Detection*, *False Negative*, or *False Positives*.

This classification process not only reflects the impact of rain on LiDAR performance but also accounts for spatial adjustments. For every point, the position is modified based on the cumulative impact of the rain droplets along the laser's path. The degree of adjustment is proportional to the scattering effect caused by the droplets, which is derived from the calculated interaction rate (μ) and the number of droplets interacting with the beam. This ensures that the final position reflects the physical deflection caused by rain-induced interference.

3.4.2.3 Integration with parallel processing

To achieve real-time simulation, the model employs parallel processing using OpenMP. Each point in the LiDAR cloud is processed independently, with key tasks including:

- Classifying points as hits or non-hits based on distance thresholds.
- Calculating the interaction probabilities using the Poisson distribution.
- Modifying point coordinates and intensities to simulate the effects of rain.

This parallelized approach ensures compatibility with high-frame-rate LiDAR systems.

3.4.2.4 Dynamic rain intensity adjustment

The rain model supports dynamic adjustments to rain intensity during simulation. This feature enables testing under varying weather conditions, with the rain rate R updated in real time. The system recalculates the raindrop distribution and interaction probabilities accordingly, ensuring consistency in the simulated noise effects.

This detailed justification highlights the technical foundations and innovations of the proposed rain noise model, ensuring its relevance for autonomous vehicle testing and validation in adverse weather conditions.

3.4.3 Model assumptions

The proposed rain simulation model incorporates several assumptions to balance complexity and computational efficiency. These assumptions simplify the implementation while maintaining a high degree of realism for testing autonomous vehicle LiDAR systems under rainy conditions. Below are the key assumptions and their implications:

Uniform rainfall distribution The model assumes that rainfall is uniformly distributed throughout the simulation area. This simplification ensures consistent interactions between LiDAR beams and raindrops, avoiding the need for complex spatial variability calculations. While this assumption aligns with standard meteorological models for moderate rainfall, it may not capture localized variations in rain intensity observed in real-world conditions such as wind-driven rain or microbursts.

Independent raindrop interactions Each raindrop's interaction with a LiDAR beam is treated as an independent event. This assumption allows the use of probabilistic models, such as the Poisson distribution, to calculate the likelihood of raindrop encounters within the beam path. It simplifies the computation but does not account for potential clustering effects or correlated raindrop behaviors.

No beam divergence The LiDAR beam is modeled as a cylindrical path with a constant radius along its length. This ignores beam divergence, which occurs in real-world LiDAR systems due to diffraction and optical design. While this assumption simplifies volume calculations for raindrop interactions, it may underestimate the effects of scattering and attenuation at longer distances.

Fixed detection thresholds The model uses fixed detection thresholds for signal intensity to classify points as valid detections, false positives, or false negatives. This does not consider the dynamic adjustments that real-world LiDAR systems might apply based on environmental conditions or adaptive algorithms.

Homogeneous raindrop properties Raindrop sizes and optical properties are derived from the Marshall-Palmer distribution, assuming homogeneous material properties for all raindrops. This neglects potential variations in raindrop composition (e.g., pollutants or mixed precipitation) that could affect their scattering and absorption characteristics.

3.4.3.1 Impact of assumptions

These assumptions enable the model to efficiently simulate large-scale point clouds in real time while preserving key rain effects such as attenuation, scattering, and noise. However, they introduce limitations that may affect the model's accuracy in highly variable or extreme weather conditions. Future work could address these limitations by incorporating:

- Spatially variable rain intensity to simulate localized weather phenomena.
- Beam divergence effects to enhance the realism of scattering and attenuation.
- Dynamic detection thresholds to mimic real-world LiDAR system adaptability.

By understanding and addressing these assumptions, the model can serve as a robust foundation for testing and improving autonomous vehicle perception in adverse weather.

3.4.4 Validation model

Validating the proposed rain simulation model is critical to ensure its accuracy and effectiveness in replicating real-world conditions. The validation process involves comparing the simulated outputs against theoretical predictions, experimental data, and real-world observations. Below are the key approaches for validation:

3.4.4.1 Qualitative assessment

Beyond numerical metrics, visual inspection of simulated point clouds provides valuable qualitative insights. This involves identifying patterns of false positives and negatives within the simulated point cloud and observing the spatial distribution of noise to evaluate its consistency with real-world rain effects.

3.4.4.2 Performance testing

Real-time performance is crucial for the model's integration into simulation environments. Validation involves:

- Measuring the computation time per frame to ensure compatibility with high-frame-rate LiDAR systems.
- Evaluating the scalability of the model for large-scale point clouds.
- Ensuring that the simulation maintains real-time performance under varying rain intensities.

Chapter 4

Experiments and results

Contents

4.1 Initial Tests and Validation

To evaluate the performance and realism of the proposed rain simulation model, a series of initial tests were conducted. These tests aimed to validate the model's ability to accurately replicate the effects of rain on LiDAR point clouds and assess its computational efficiency under real-time constraints.

4.1.1 Simulation setup

The initial tests were performed using the simulated environment with the following configuration:

- **LiDAR Specifications:** A virtual 128-channel LiDAR sensor with a maximum range of 100 meters and an angular resolution of 0.2 degrees (see Figure 4.1).
- **Rain Intensity:** Rainfall rates ranging from 5 mm/h (light rain) to 100 mm/h (heavy rain).
- **Point Cloud Parameters:** Point clouds generated at 10 frames per second, consistent with real-world LiDAR systems. Each frame contains approximately 28,800 points, resulting in a total of 288,000 points per second at full 360-degree coverage.

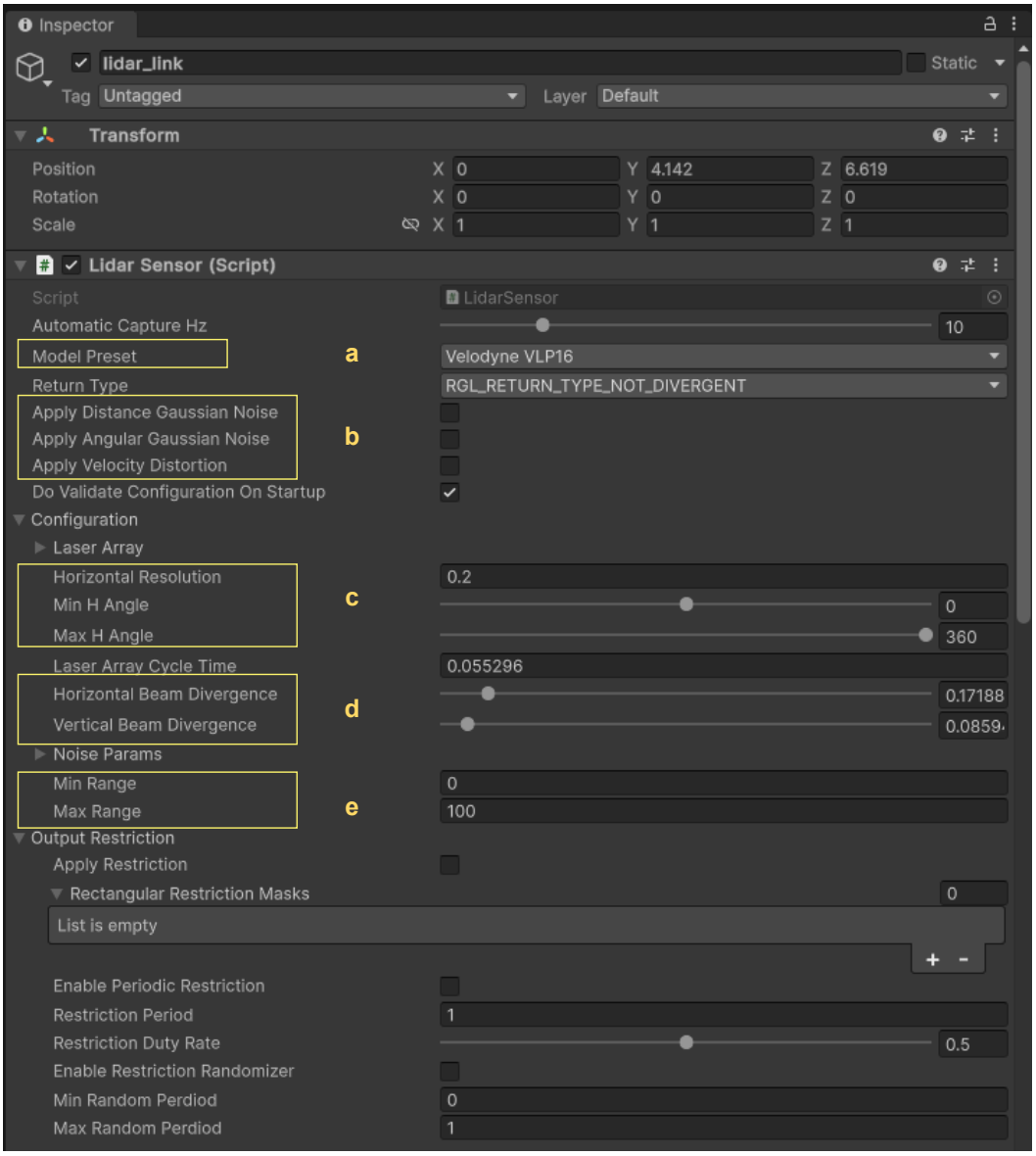


Figure 4.1: LiDAR sensor configuration showing details of the Velodyne VLP16 parameters in the simulation. a)Sensor model, b) Noise addition, c) Resolution parameters, d) Divergence, e) Range settings.

4.1.2 Qualitative results

To evaluate the effectiveness and realism of the rain simulation model on LiDAR systems, a controlled test scene was utilized. This scene was carefully designed to obtain different returns, including a variety of objects such as buildings, vehicles, and different objects. The test environment provides a structured layout to analyze the impact of rain intensity on LiDAR performance.

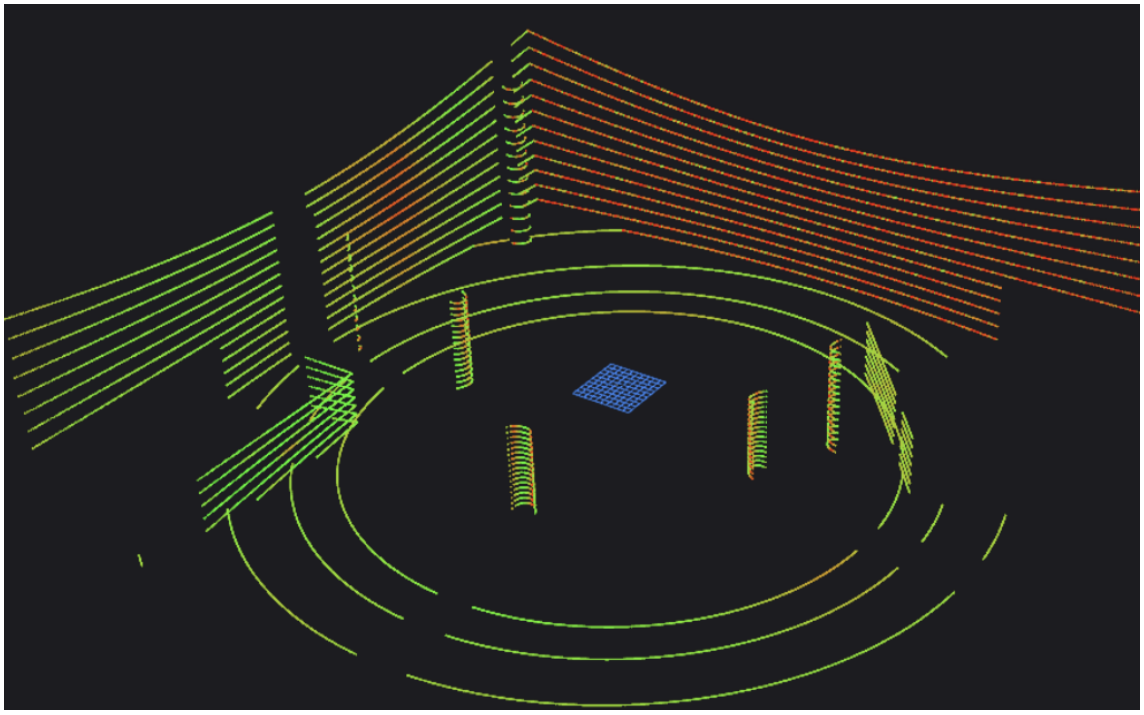


Figure 4.2: Test scene visualized in Foxglove Studio [7] with no rain. The intensity gradient ranges from 0 (red) to 255 (green), highlighting the variation in LiDAR return intensities.

Figure 4.2 illustrates the test scene used in the experiments, showcasing the spatial arrangement, object diversity, and the visual representation of LiDAR intensity values critical for comprehensive evaluation. Each object is configured with one of the textures showed on 3.17.

Incorporating rain into the simulation Visual inspection of the simulated point clouds under varying Rain Rates (RR) revealed the following patterns and behaviors:

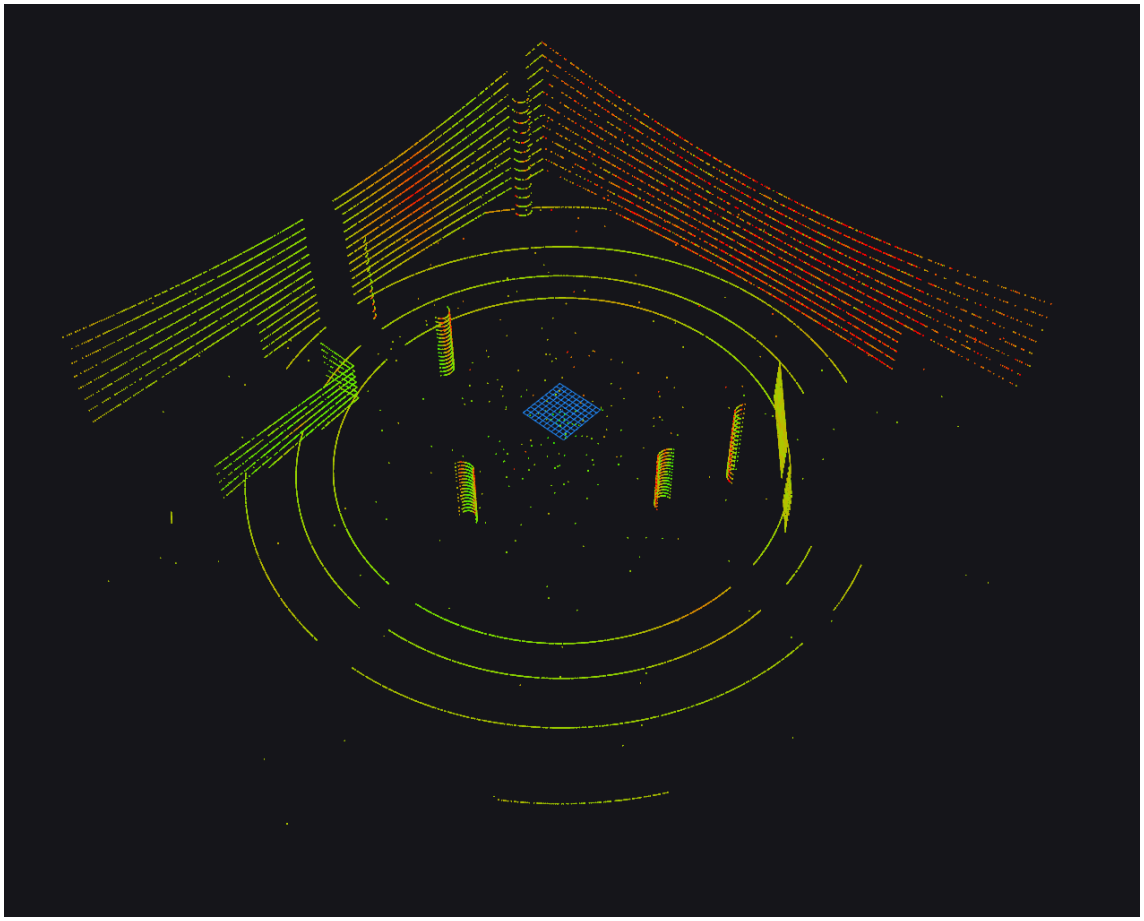


Figure 4.3: Rain Rate: 5-25 mm/h.

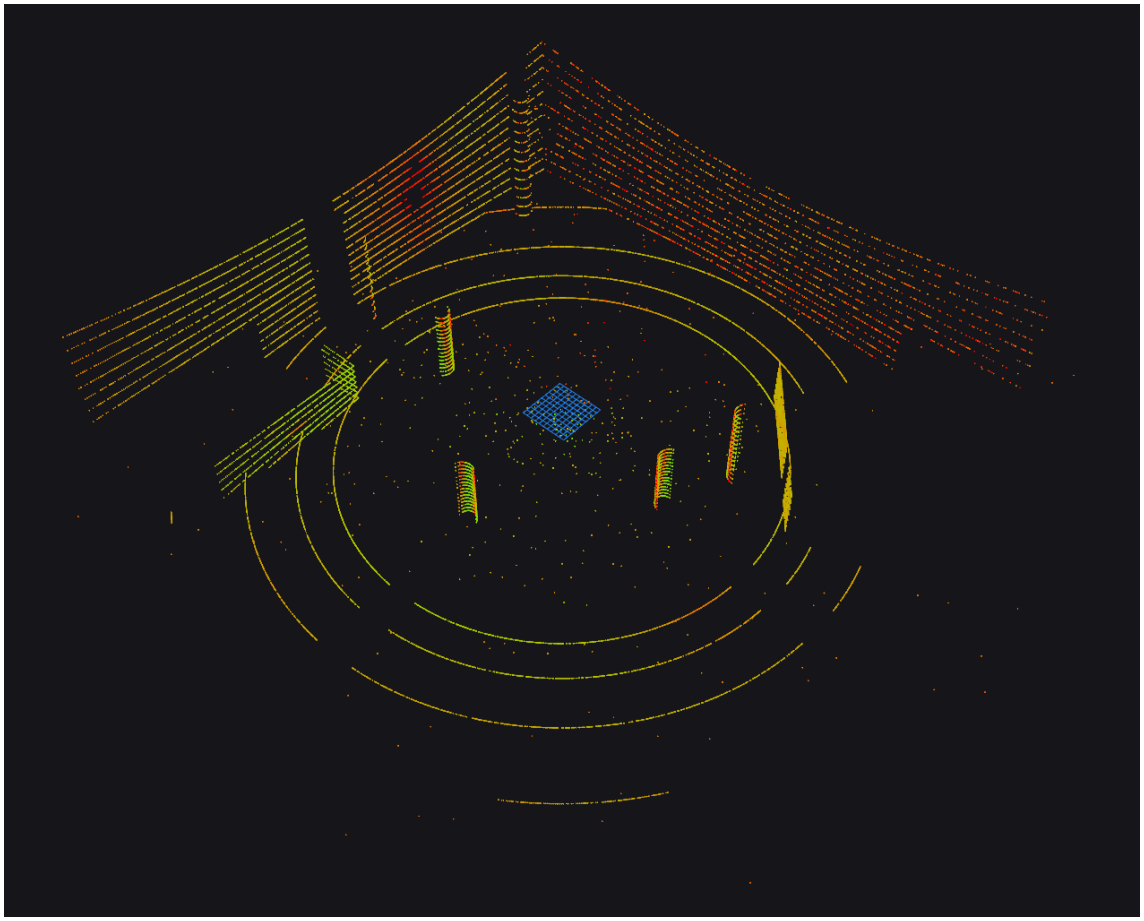


Figure 4.4: Rain Rate: 25-50 mm/h.

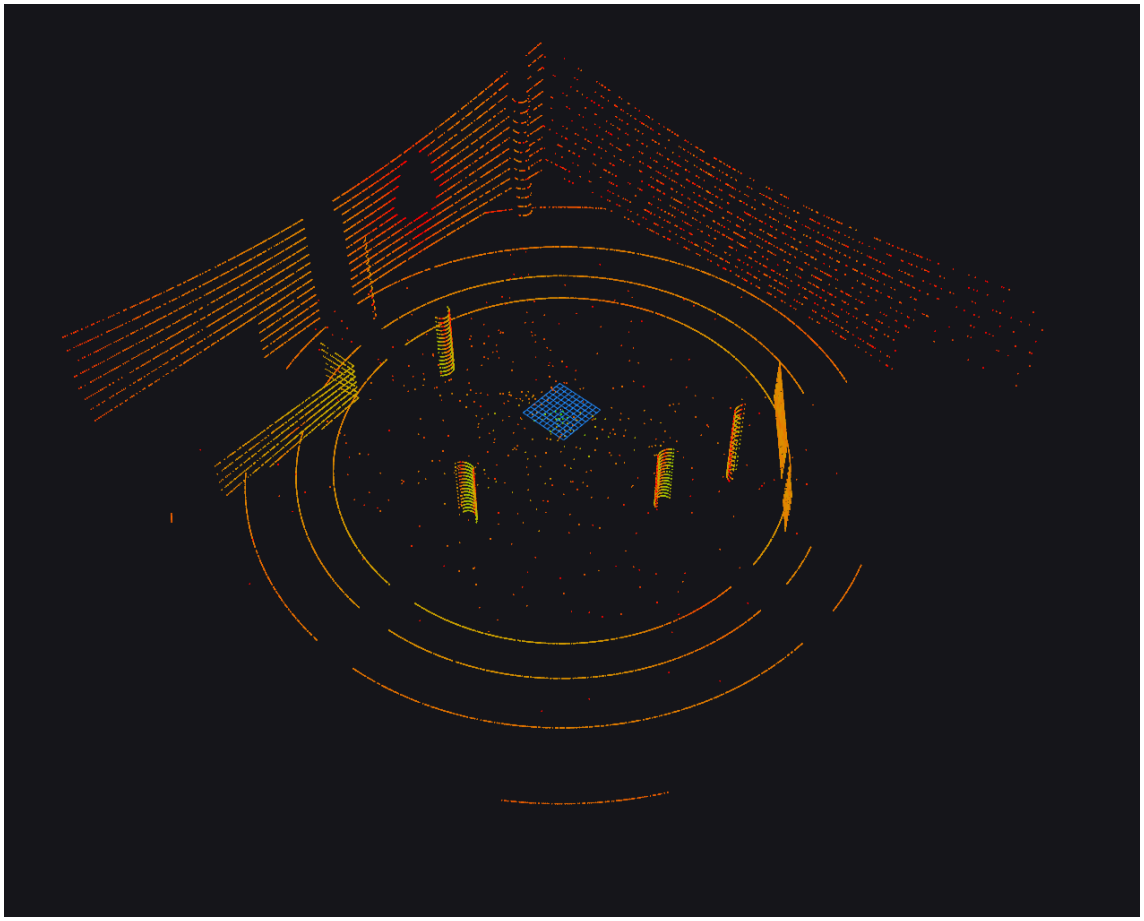


Figure 4.5: Rain Rate: 50-75 mm/h.

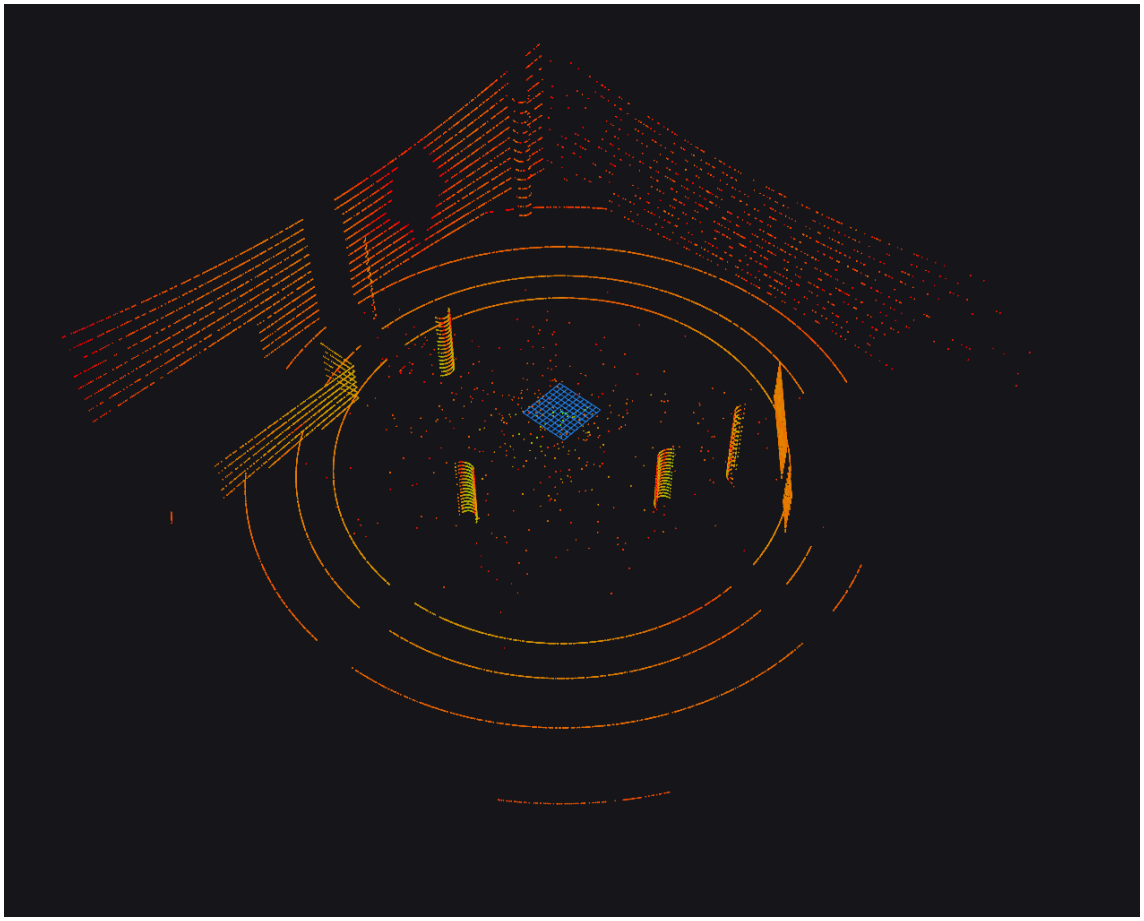


Figure 4.6: Rain Rate: 75-100 mm/h.

- Points classified as "non-hits" were redistributed with realistic noise patterns, avoiding clusters or artifacts, as observed across all Rain Rates.
- False positives caused by raindrop reflections were distributed spatially along the beam path, increasing in density with higher Rain Rates, consistent with expected rain effects.
- False negatives were more prominent in objects partially obscured by dense rain, especially under heavy rain conditions (RR > 50 mm/h), replicating real-world attenuation phenomena.

Figures 4.3 through 4.6 illustrate the simulated LiDAR point clouds under increasing Rain Rates (RR), ranging from 5-25 mm/h to 75-100 mm/h. These figures highlight how higher rain intensities affect the LiDAR data, introducing more noise, false positives, and attenuation. The intensity gradient, ranging from 0 (red) to 255 (green), represents the variation in return intensities across all scenarios.

4.1.3 Quantitative metrics based on raindrops hit distribution

To further analyze the rain simulation model, we evaluated the **number of raindrops in the LiDAR beam path**, which is derived from using μ as the mean of the Poisson distribution. This parameter reflects the variability in the number of raindrop interactions for each laser beam, capturing the stochastic nature of rain interactions and their influence on the LiDAR's performance.

The experiment involved counting the number of raindrops that impacted each laser beam of the LiDAR while sweeping the Rain Rate (RR) from 5 to 100 mm/h. The results are visualized in Figure 4.7.

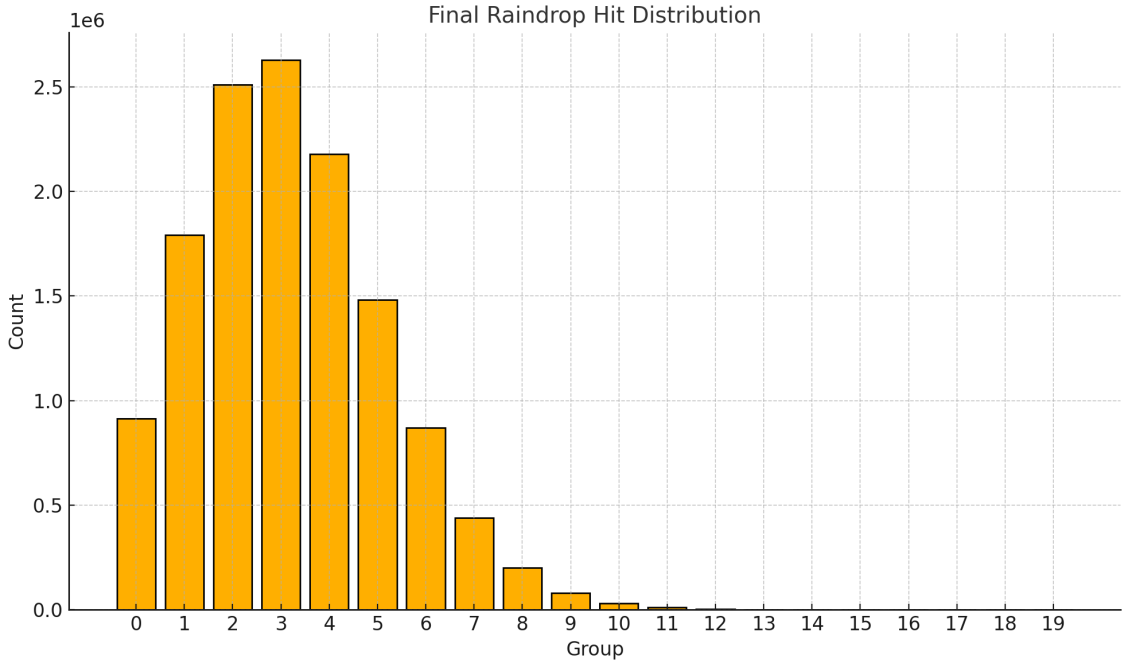


Figure 4.7: *Number of raindrops in the LiDAR beam path* distribution across a full sweep of RR from 5 to 100 mm/h. The x-axis represents the number of raindrops that intersect a single laser beam, while the y-axis indicates the total count of laser beams experiencing that specific number of raindrop interactions.

The orange graph represented earlier provides a full sweep of RR values from 5 to 100 mm/h, offering an overview of raindrop impacts across a wide range. In contrast, the yellow graphs, presented subsequently, focus on narrower ranges, enabling a more detailed analysis of raindrop behavior and offering additional insights into the effects of rain intensity on hit distribution.

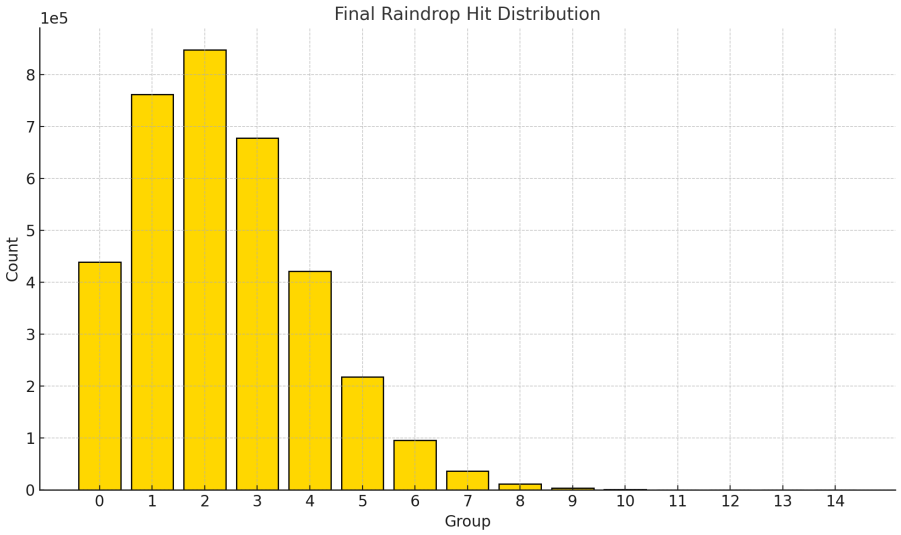


Figure 4.8: Raindrop hit distribution for Rain Rate (RR) in the range of 5-25 mm/h.

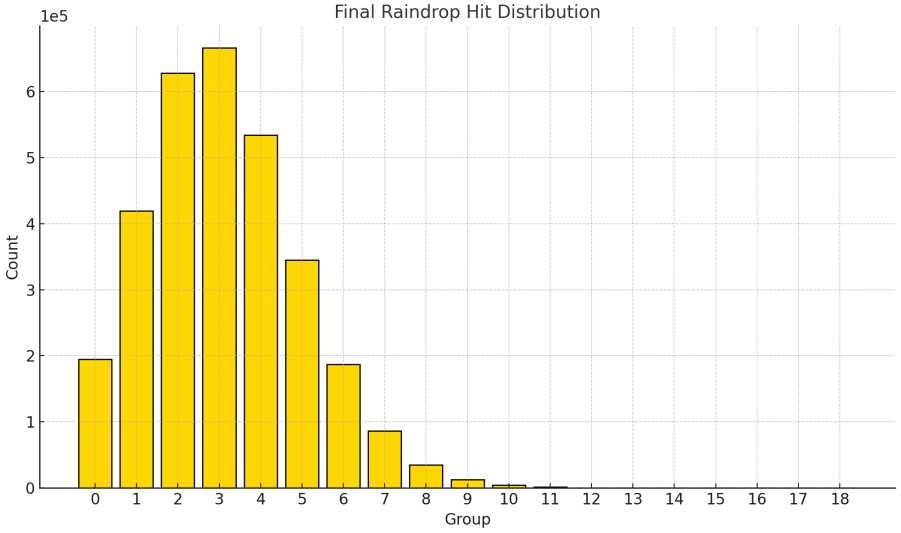


Figure 4.9: Raindrop hit distribution for Rain Rate (RR) in the range of 25-50 mm/h.

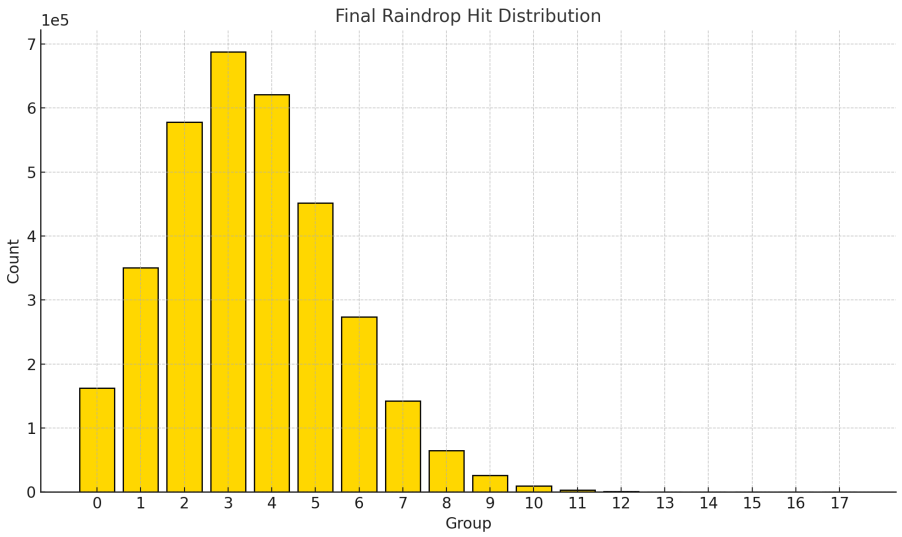


Figure 4.10: Raindrop hit distribution for Rain Rate (RR) in the range of 50-75 mm/h.

Figures 4.8 through 4.10 illustrate how increasing rain intensity shifts the raindrop hit distribution towards higher values. This shift represents a greater number of raindrops encountered within the beam path, increasing the noise and attenuation effects observed in the LiDAR data. These results provide quantitative support for the model's realism in simulating rain conditions.

4.1.4 Publication rate analysis

The proposed rain simulation model is designed with computational efficiency in mind to ensure compatibility with real-time LiDAR systems. This section highlights the technical characteristics of the algorithm and evaluates its performance in terms of processing time and output frequency.

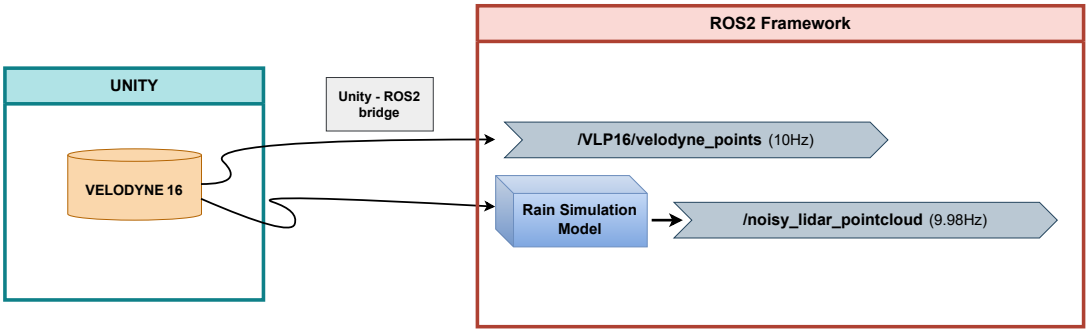


Figure 4.11: Flowchart illustrating the real-time testing process, including the Unity environment, Velodyne VLP-16 LiDAR, ROS 2 framework, and the rain simulation model. The system bridges Unity and ROS 2 to evaluate pointcloud publication rates under simulated rain conditions.

4.1.4.1 Technical characteristics

The algorithm leverages the following features to achieve high performance:

- **Parallel Processing:** The use of OpenMP enables parallel computation of point cloud data, allowing the simulation to efficiently process large-scale LiDAR point clouds.
- **Optimized Data Structures:** Custom data structures and memory-efficient operations are employed to minimize computational overhead.
- **Dynamic Rain Adjustment:** The model supports real-time updates to rain intensity, recalculating relevant parameters without interrupting the simulation.

4.1.4.2 Publication frequency

To validate the real-time capabilities of the algorithm, the frequency of the rainy pointcloud topic was measured during the simulation in the ROS 2 environment. The system was tested on a computational platform with the following specifications:

- **Processor:** 12th Gen Intel® Core™ i9-12950HX × 24.
- **Memory:** 64.0 GiB RAM.
- **System:** Dell Inc. Precision 7670.

The LiDAR system operates at a native frequency of 10 Hz (Table 4.1). Under the rain simulation model, the publication frequency of the rainy pointcloud topic was observed to be 9.98 Hz (Table 4.2). This minor reduction in frequency, corresponding to the rate at which the pointcloud messages are published in ROS 2, is caused by the computational overhead introduced by the rain simulation model applied to the LiDAR system. The overhead, approximately 0.2%, is considered acceptable for real-time applications.

Average Rate (Hz)	Min (s)	Max (s)	Std Dev (s)	Window
9.999	0.094	0.105	0.00436	11
10.009	0.093	0.106	0.00484	22
10.033	0.089	0.110	0.00538	33
9.998	0.089	0.110	0.00554	43
10.018	0.089	0.110	0.00523	54
10.015	0.089	0.110	0.00483	64

Table 4.1: Publishing rate of the Velodyne 16 ROS 2 topic simulated in Unity and published without modifications. For detailed terminal output, refer to Appendix A.16.

Average Rate (Hz)	Min (s)	Max (s)	Std Dev (s)	Window
9.940	0.095	0.103	0.00309	11
9.963	0.095	0.104	0.00326	21
9.996	0.095	0.104	0.00353	32
9.993	0.095	0.104	0.00362	42
9.988	0.095	0.104	0.00352	52
9.990	0.095	0.104	0.00353	63
9.992	0.094	0.104	0.00349	74

Table 4.2: Publishing rate of the Velodyne 16 rainy points after applying the Rain Model. For detailed data, refer to Appendix A.17.

The results demonstrate that the proposed algorithm maintains near-native performance levels while simulating complex rain effects and achieves publication frequencies comparable to commercial LiDAR systems. It is important to note that the reported average is not based solely on the initial data points but rather on a significantly larger dataset.

4.2 Overall Results of the proyect

4.3 Validation against Real-World Data

The simulated point clouds were visually compared to real LiDAR data collected under rainy conditions. Key observations from this qualitative comparison include:

- Similar visual patterns of noise, including false positives and negatives, in both simulated and real-world data.
- Comparable attenuation effects and intensity reductions observed across various rainfall rates.

A more in-depth review of the validation process using real-world rain models is discussed in Appendix B.

4.4 Summary of Project Results and Conclussion

This section provides an overview of the global outcomes achieved throughout the project, highlighting the main findings and showcasing visual representations of the results. The images and visualizations presented here illustrate the performance and impact of the rain simulation model on LiDAR data processing on real-time.

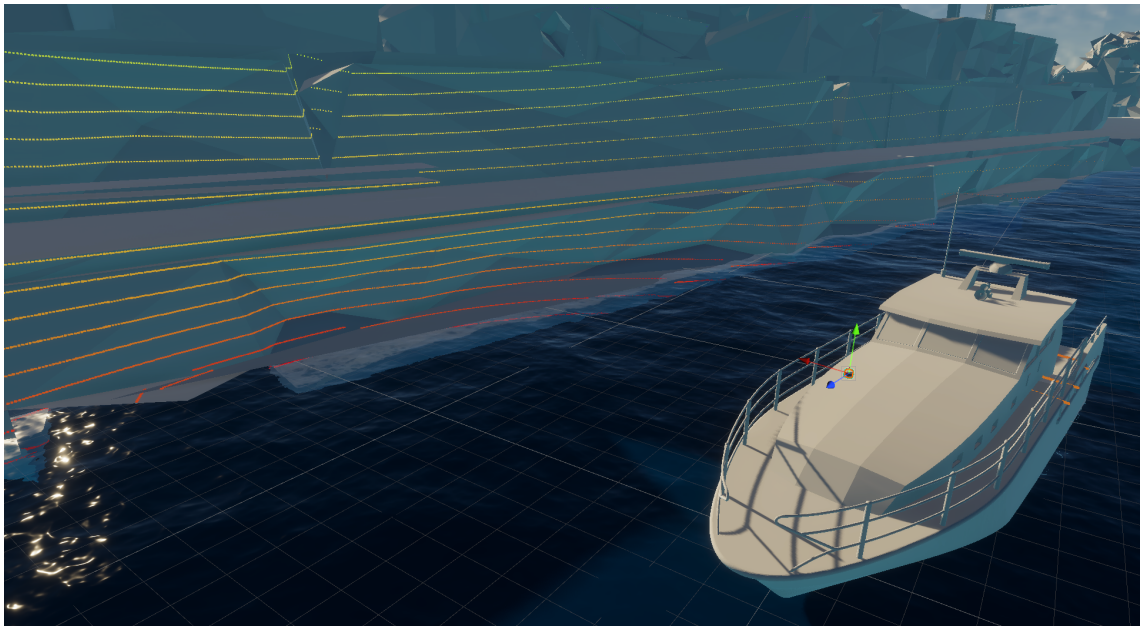


Figure 4.12: Visualization of the LiDAR points during a simulated scene in Unity, showing the interaction between the laser beams and the environment.

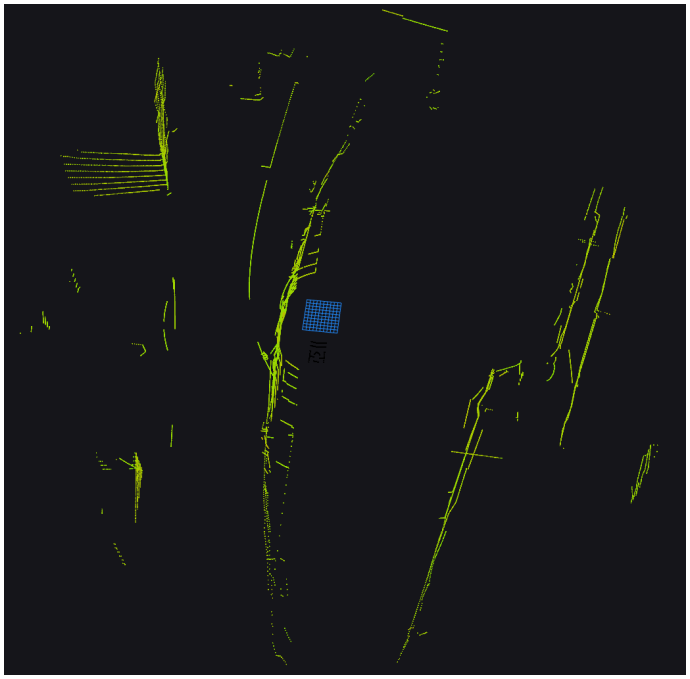


Figure 4.13: Point cloud published on the `/VLP16/velodyne_points` topic without applying the rain model, showcasing the LiDAR's original output in a simulated environment.

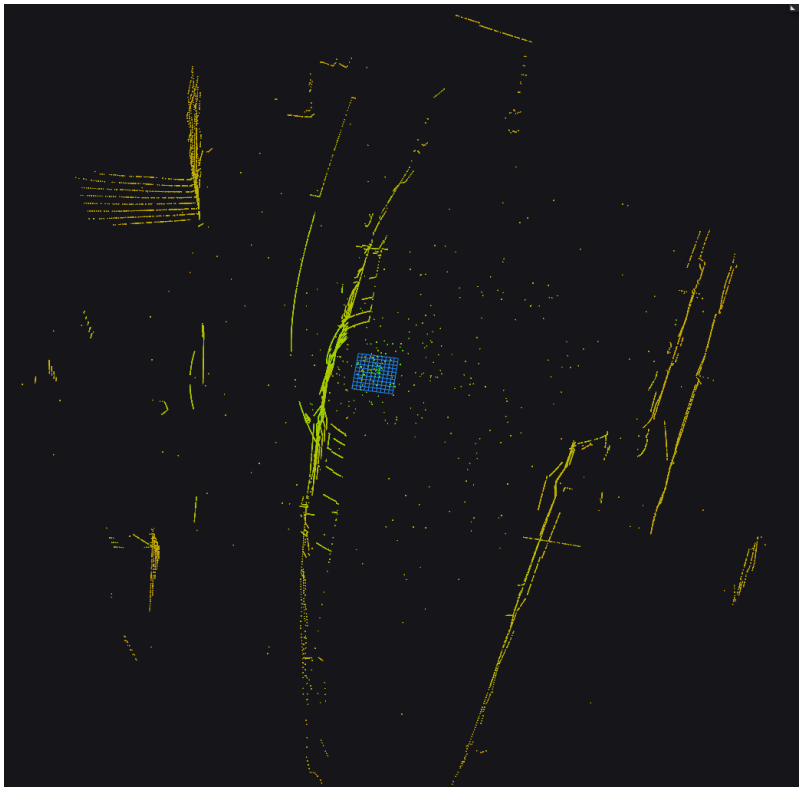


Figure 4.14: Point cloud published on the `/noisy_lidar_pointcloud` topic after applying the rain model to the `/VLP16/velodyne_points` data, demonstrating the simulated rain effects on LiDAR perception.

The entire process, including scene management in Unity, point cloud publication to ROS 2, and rain model simulation, is executed simultaneously in real-time. This seamless integration ensures the system's suitability for testing autonomous driving applications under dynamic and adverse weather conditions.

Part III

Conclusions and Future Work

Chapter 5

Conclusions

Evaluation of Results with Respect to Objectives

Throughout the course of this project, significant progress was achieved in addressing the primary objectives outlined at the beginning. Specifically, the project successfully developed a simulation environment that accurately replicates real-world navigation conditions for a small sensorised vessel traversing the canals of Berlin. This environment integrates key components, including an IMU and LiDAR sensors, both of which were seamlessly integrated with the ROS 2 framework to ensure robust communication and real-time data processing.

One of the key achievements of this project was the implementation of a novel rain simulation model, which effectively replicates the impact of rainfall on the LiDAR sensor. This model introduces noise and attenuation effects consistent with real-world rain conditions, enhancing the realism and utility of the simulation. By enabling both visual and quantitative evaluation of sensor data under varying rain intensities, the model provides a valuable tool for autonomous navigation system testing. The successful simulation of rain is a significant step forward; however, the simulation currently lacks representations of other environmental factors, such as wind, fog, or snow, which are equally relevant for real-world navigation scenarios.

The results obtained from the experiments provide a comprehensive understanding of the rain simulation model's impact on LiDAR performance. The quantitative metrics demonstrate how increasing rain intensity directly affects the LiDAR point clouds. Figures 4.8 through 4.10 illustrate a clear shift in the raindrop hit distribution towards higher values as rain intensity increases. This shift signifies a greater number of raindrops intersecting the LiDAR beam paths, which in turn amplifies noise and attenuation effects. These findings quantitatively validate the realism of the rain simulation model in replicating adverse weather conditions and their effects on LiDAR data fidelity.

The qualitative analysis corroborates these quantitative findings. Visual in-

spection of the point clouds under different rain intensities revealed degradation patterns consistent with real-world behavior. Specifically, point clouds generated under heavy rain showed reduced point density and increased angular scattering, both of which align with expectations for LiDAR systems operating in rainy conditions. This visual confirmation supports the credibility of the model in simulating realistic rain effects.

The performance tests further validate the efficiency and reliability of the proposed model in maintaining near-real-time performance. The publication frequency of the rainy pointcloud topic was consistently measured at 9.98 Hz, with a standard deviation below 0.0035 Hz across multiple runs. This represents a negligible computational overhead of 0.2%, demonstrating that the model is well-optimized for high-performance hardware. This consistency is essential for real-time applications, where stable and predictable sensor outputs are crucial for autonomous systems.

These achievements directly address the primary objective of creating a versatile and realistic simulation model for testing sensorized vehicles under diverse conditions. The proposed solution effectively bridges the gap in the availability of datasets and environments required for evaluating autonomous navigation systems and Simultaneous Localization and Mapping (SLAM) algorithms. The results, supported by visual and quantitative analyses, highlight the simulation's ability to emulate real-world conditions and its consistency in providing reliable sensor data.

Additionally, the modular design of the system ensures that the simulation environment can be expanded in the future. This includes the integration of new sensor modalities, such as cameras or radar systems, as well as the inclusion of more complex environmental factors. These potential enhancements would further broaden the scope and applicability of the project, enabling researchers to test and validate autonomous systems in increasingly complex scenarios.

Conclusion In conclusion, this project has made a significant contribution to addressing the challenges in autonomous navigation and SLAM research. By developing a robust simulation environment and incorporating a rain simulation model, it provides a practical and realistic solution for testing and validating autonomous systems. While limitations remain, particularly in the need for validation against real-world rain data and the inclusion of additional weather conditions, the proposed future enhancements offer promising directions for expanding the model's scope. This work lays a solid foundation for continued research and innovation, offering researchers and engineers a powerful tool to push the boundaries of autonomous navigation and mapping technologies.

5.1 Limitations and Future Work

While the project demonstrates the ability to accurately simulate rain effects and adapt to dynamic environments, certain limitations remain. For instance, the rain simulation has not yet been validated against real-world rain data. This step is critical for ensuring the model's fidelity and for aligning its outputs with real-world conditions. Additionally, the simulation does not currently include other weather phenomena, such as wind, fog, or snow, which could have significant impacts on sensor performance. Incorporating spatial and temporal variability in the rainfall model could also improve the simulation's realism and its applicability across a broader range of scenarios.

Future work could focus on addressing these limitations by validating the rain simulation against physical LiDAR data collected under real rainy conditions and by expanding the range of environmental factors simulated. The modularity of the current framework allows for the seamless integration of additional sensors, such as cameras, radar, or ultrasonic devices, which would enable comprehensive multimodal testing. Investigating the specific effects of these environmental variables on different sensor modalities and their interaction would further enhance the simulation's value for autonomous navigation and SLAM research.

Moreover, extending the simulation to support complex scenarios, such as urban environments with pedestrian and vehicle traffic or off-road conditions, would make the tool even more versatile. By doing so, researchers and engineers could fine-tune their systems to operate reliably under a wide range of conditions, ultimately contributing to safer and more efficient autonomous navigation.

Finally, the collaborative potential of this framework is significant. Its modular and extensible nature opens the door for contributions from the research community, enabling the creation of a shared repository of scenarios and datasets. This collective effort would drive further innovation and enable autonomous systems to tackle increasingly complex challenges in navigation and mapping.

Part IV

Appendix

Appendix A

Configuration and parameters of the simulation tool

Contents

A.1 World settings

A.1.1 Directional light configuration

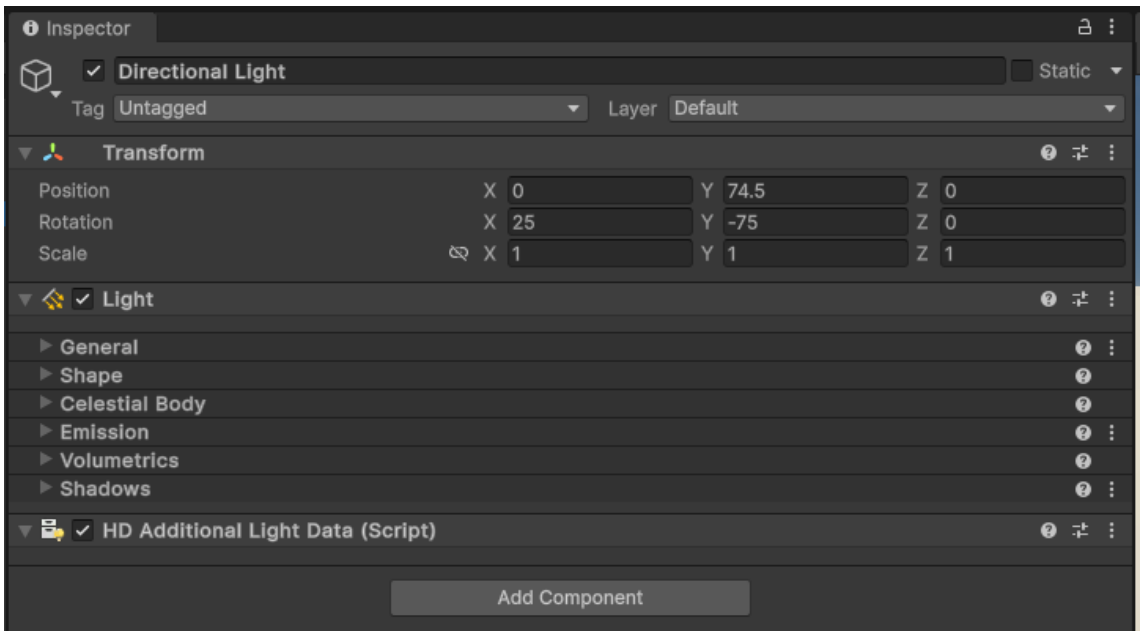


Figure A.1: Configuration of the directional light in the simulation environment.

A.1.2 HDRP asset configuration

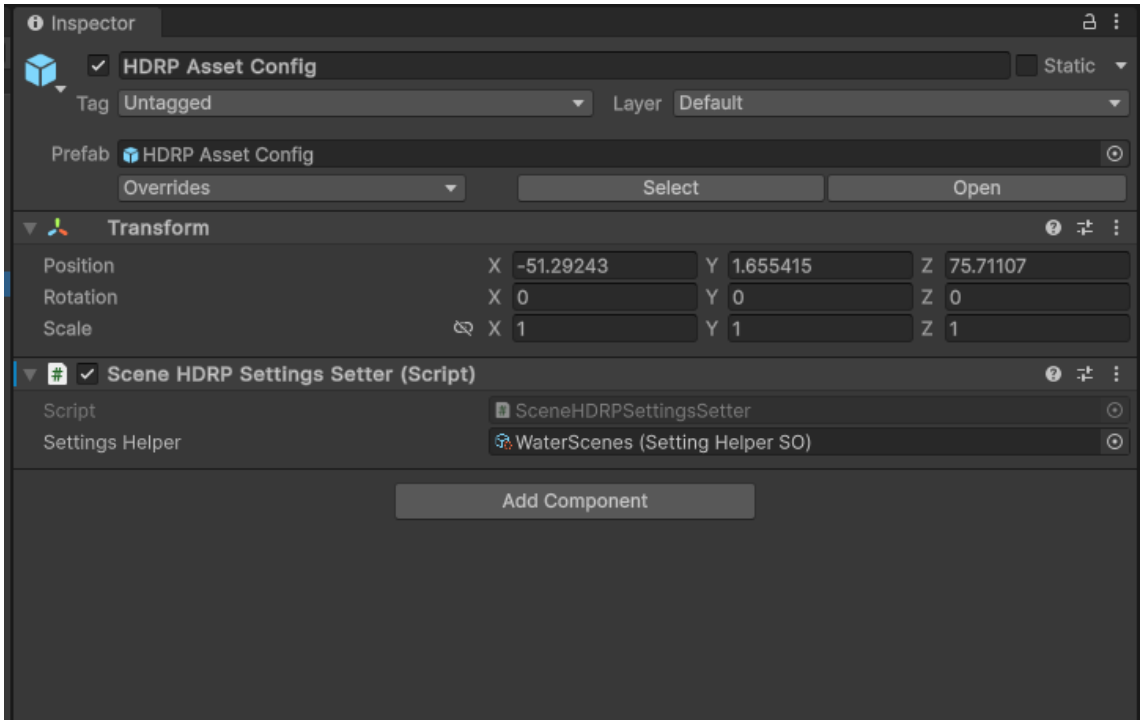


Figure A.2: HDRP asset configuration for rendering settings.

A.1.3 Ocean settings (general)

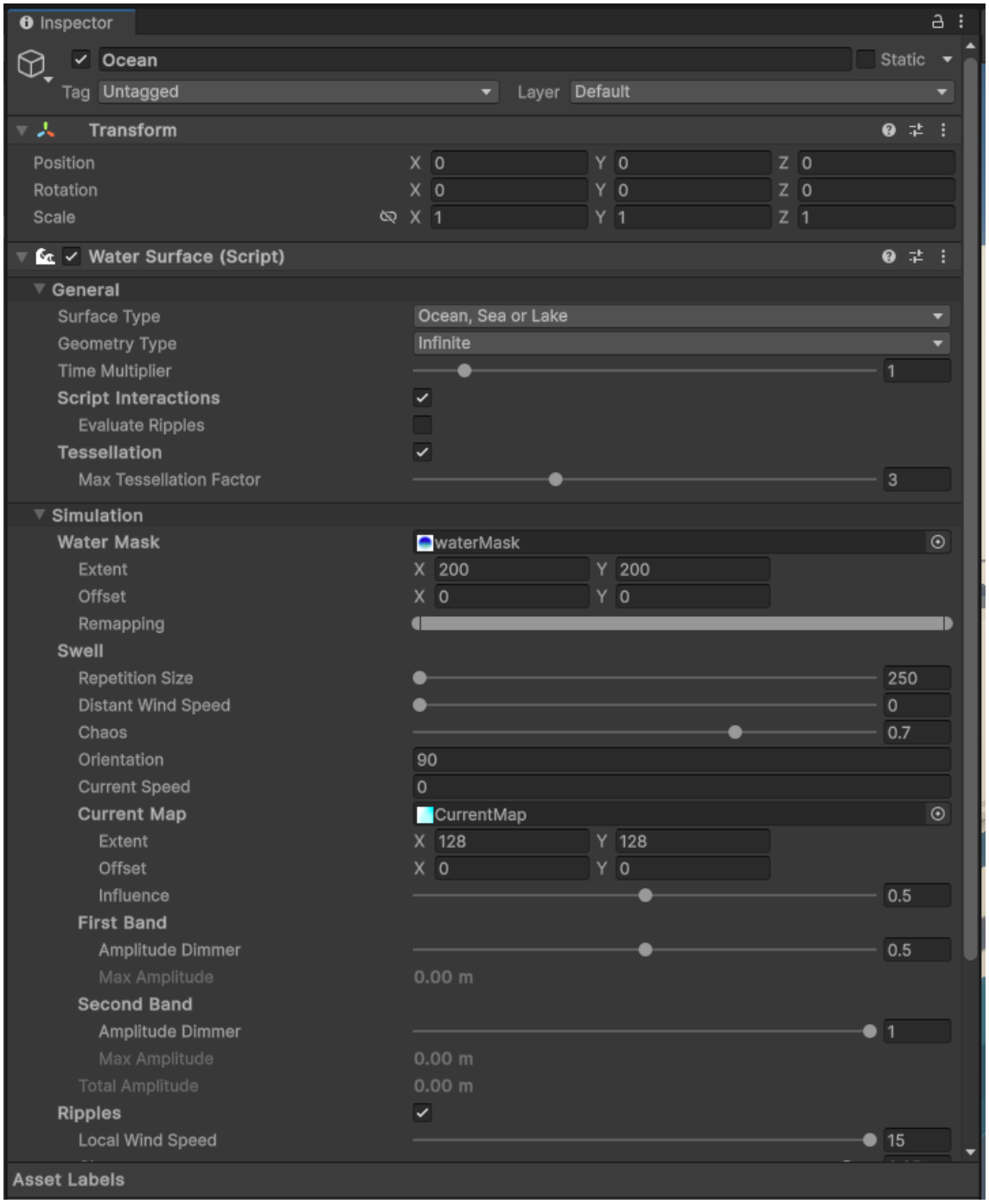


Figure A.3: General settings for the ocean surface in the simulation.

A.1.4 Ocean settings (deformation and appearance)

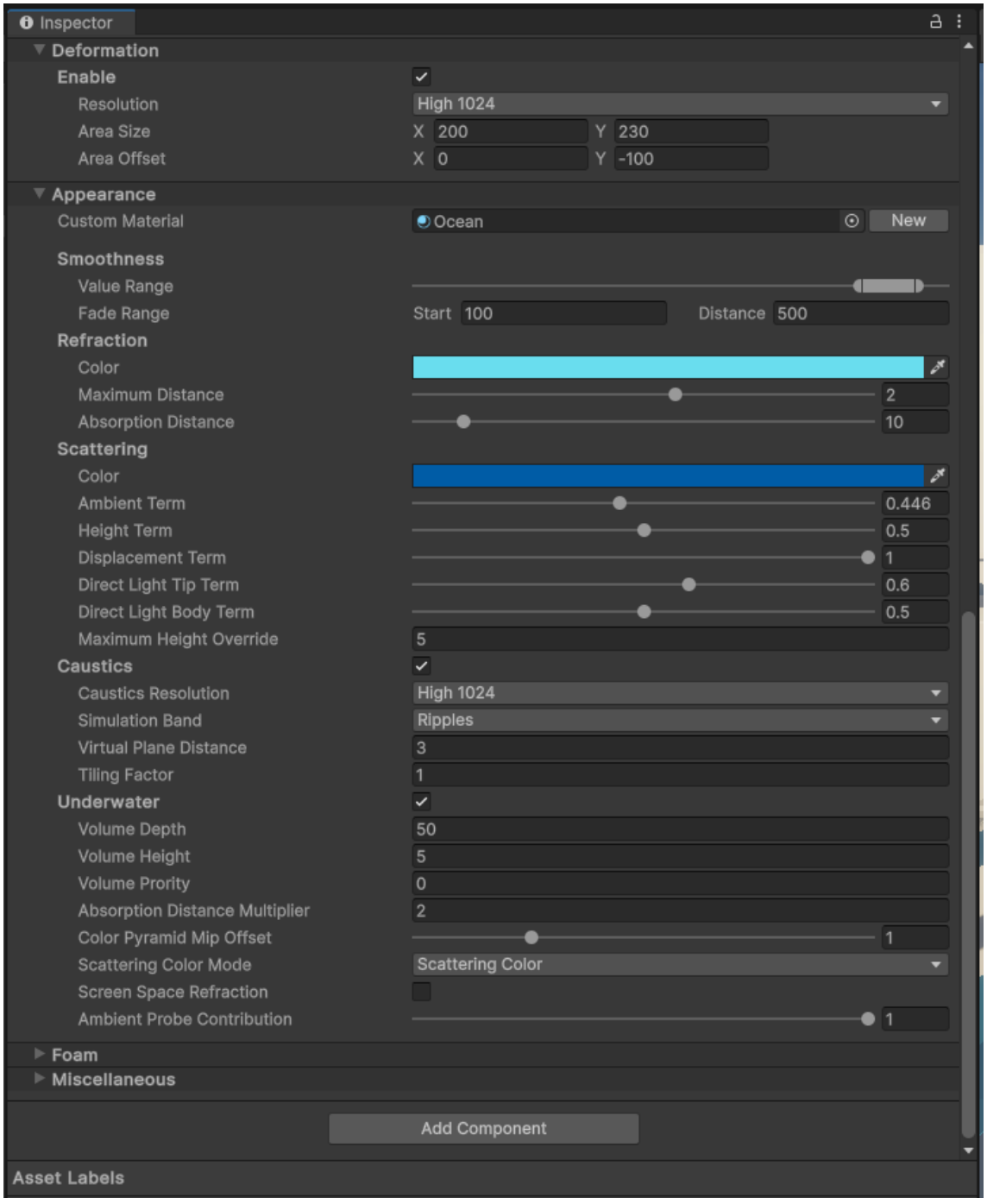


Figure A.4: Deformation and appearance settings for the ocean in the simulation.

A.1.5 Ocean settings (foam and miscellaneous)

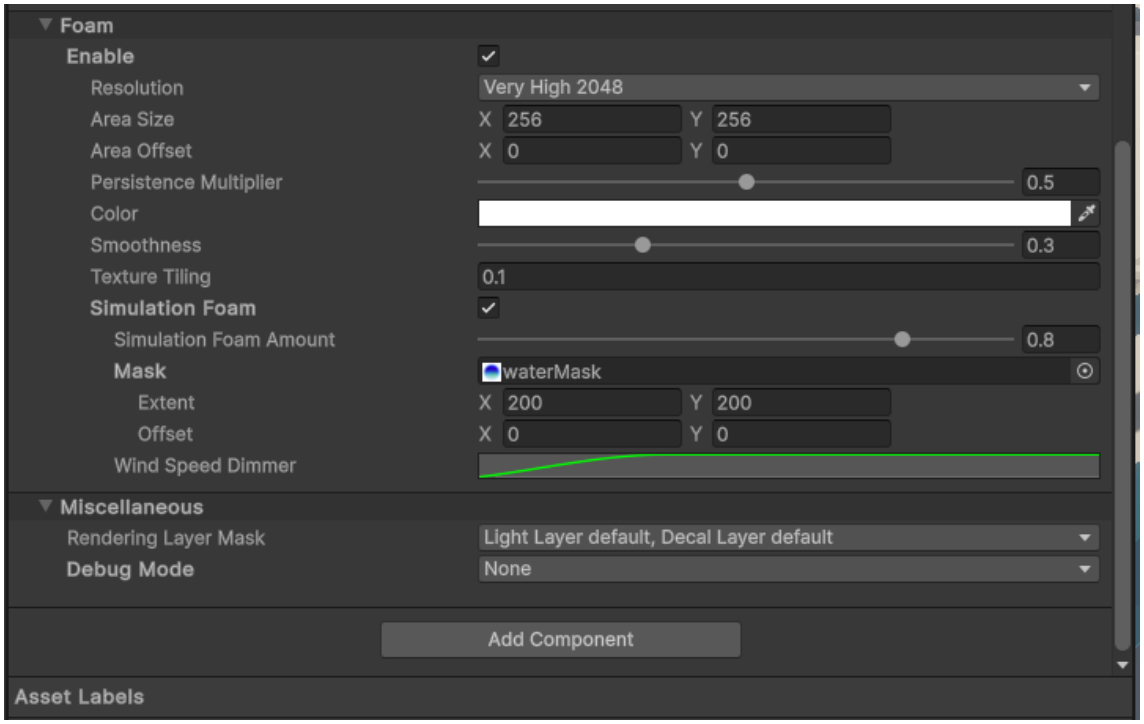


Figure A.5: Foam and miscellaneous settings for the ocean in the simulation.

A.1.6 Ocean volume configuration

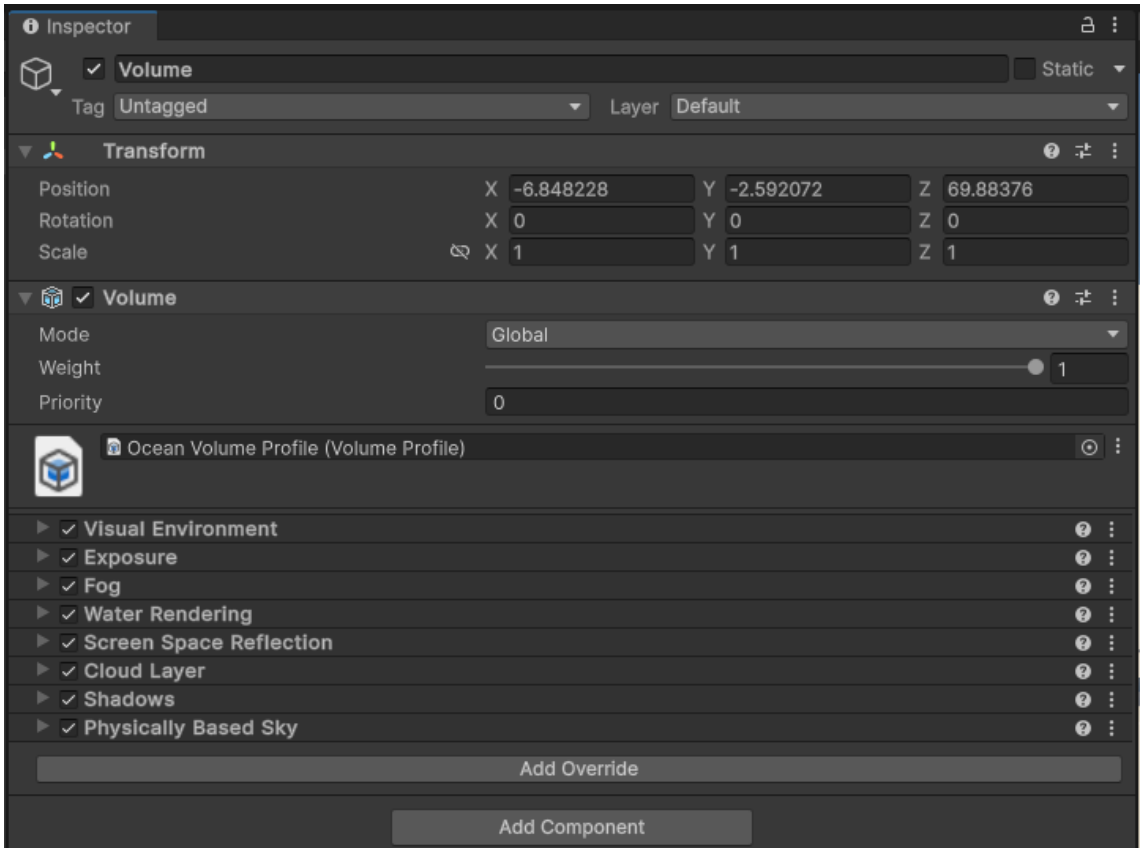


Figure A.6: Ocean volume profile configuration for environmental effects.

A.1.7 Map settings

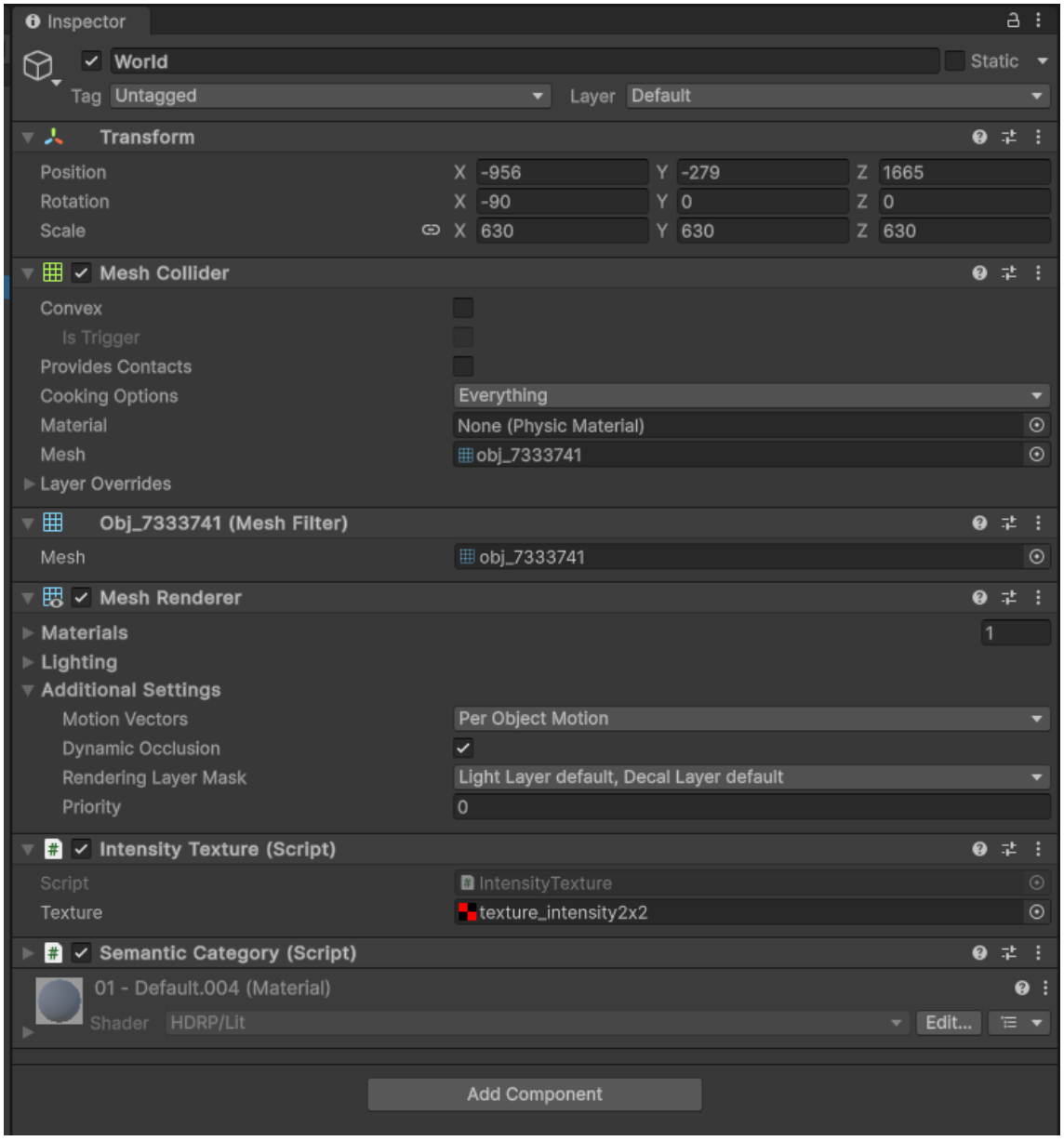


Figure A.7: General Westhafen map configuration and material settings in the simulation.

A.1.8 Hierarchy overview

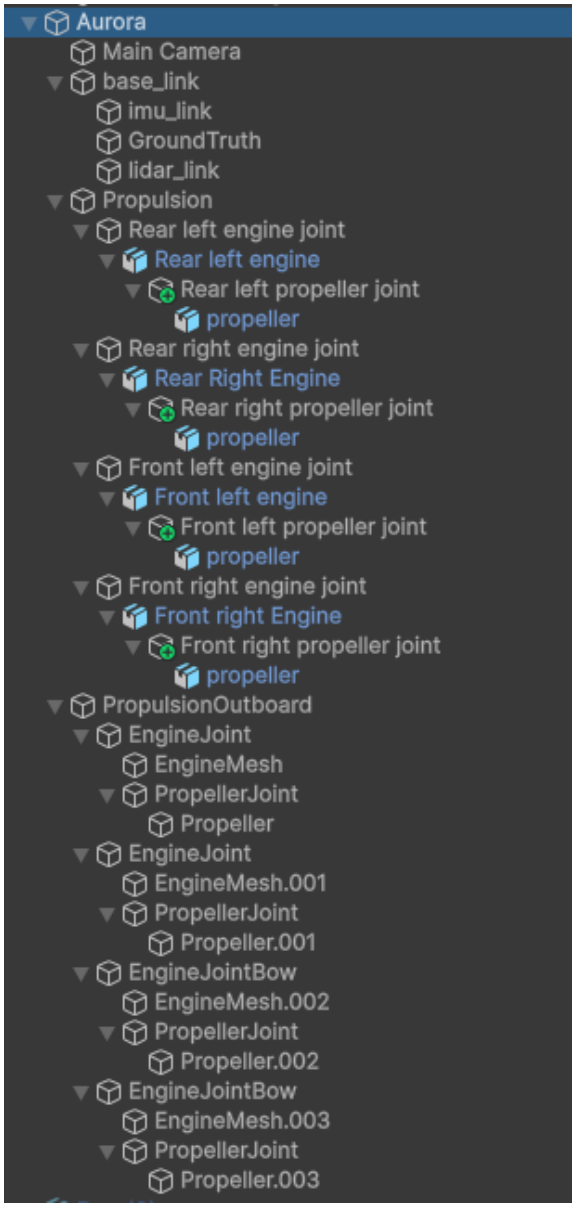


Figure A.8: Detailed hierarchy of components in the AURORA simulation environment.

A.1.9 AURORA configuration

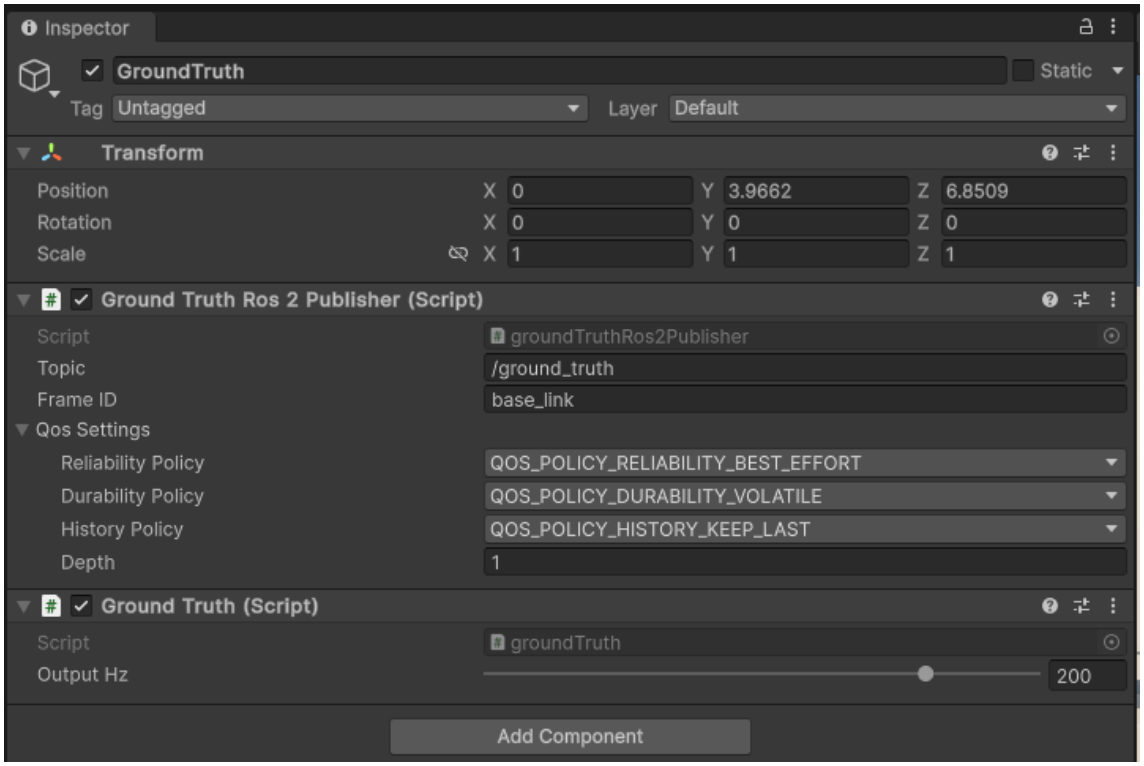


Figure A.9: Configuration of the Ground Truth publisher in the AURORA environment.

A.1.10 Propulsion system configuration

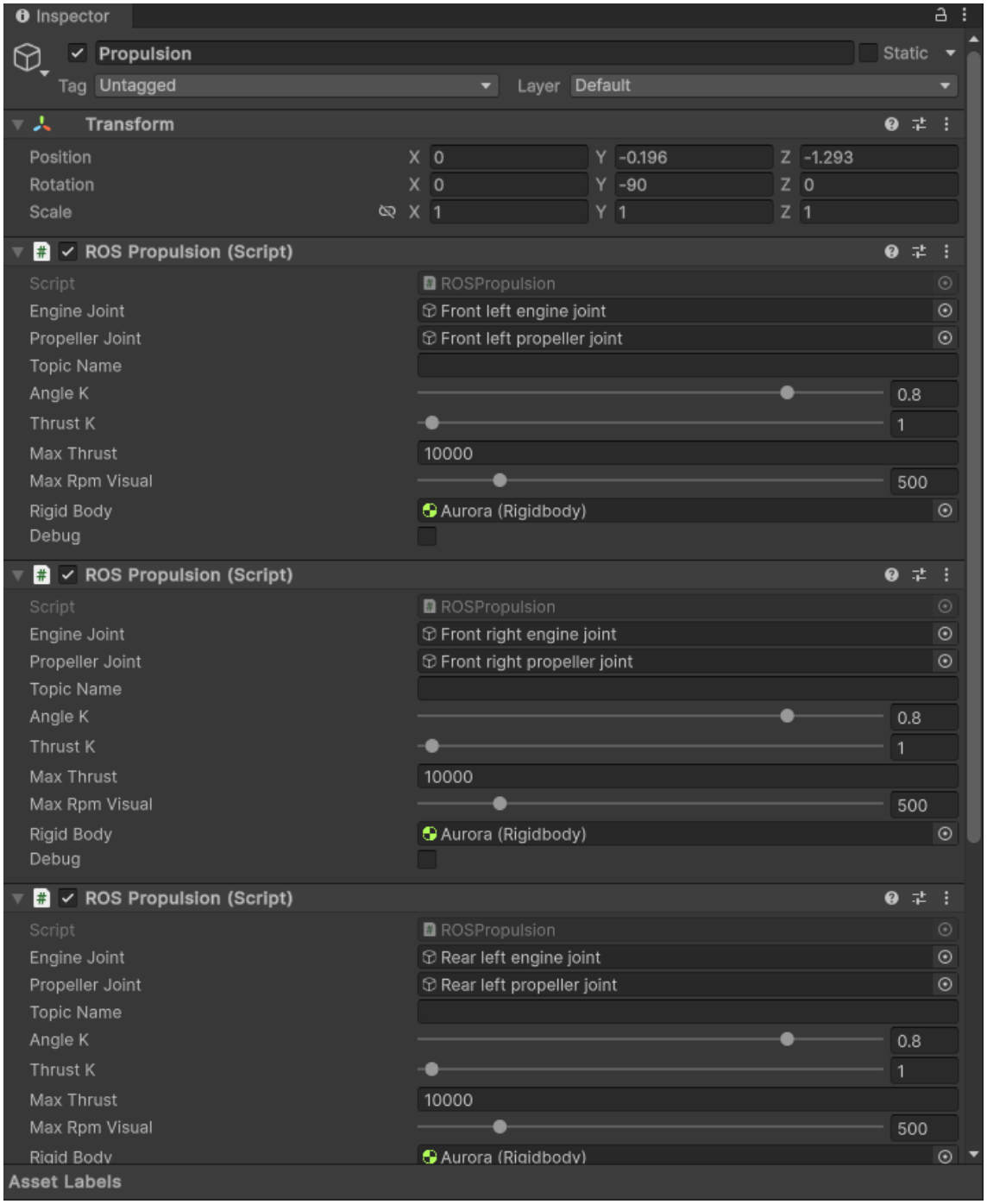


Figure A.10: Propulsion system settings for the AURORA vessel.

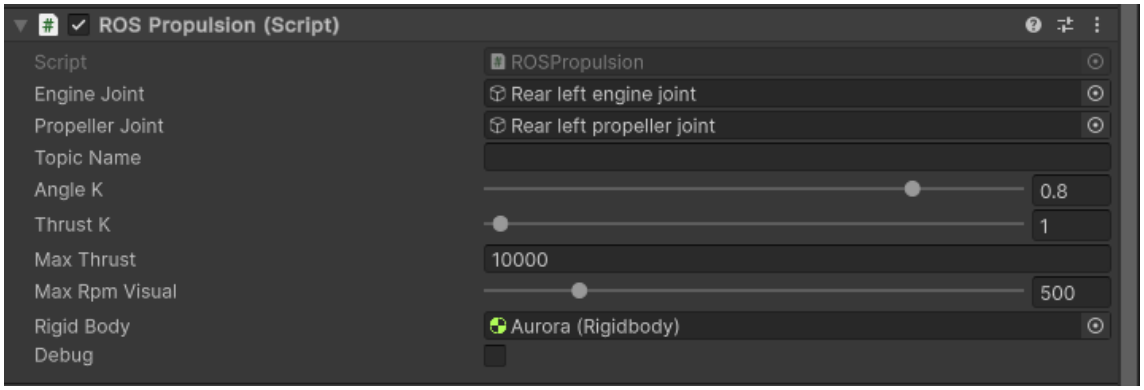


Figure A.11: ROS-based propulsion configuration in the simulation.

A.1.11 IMU configuration

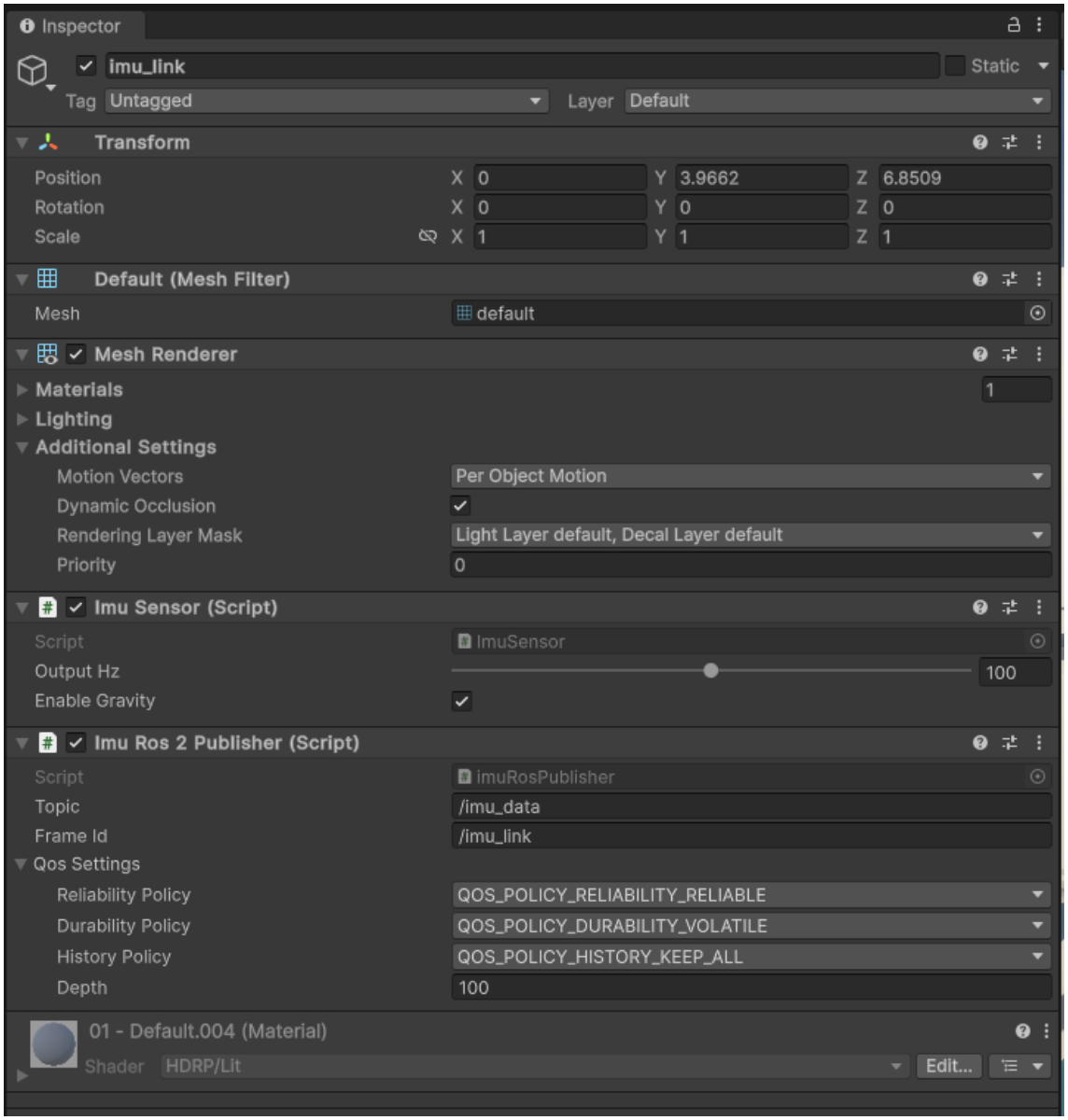


Figure A.12: Configuration of the IMU sensor and ROS 2 publisher.

A.1.12 LiDAR configuration (Velodyne VLP-16)

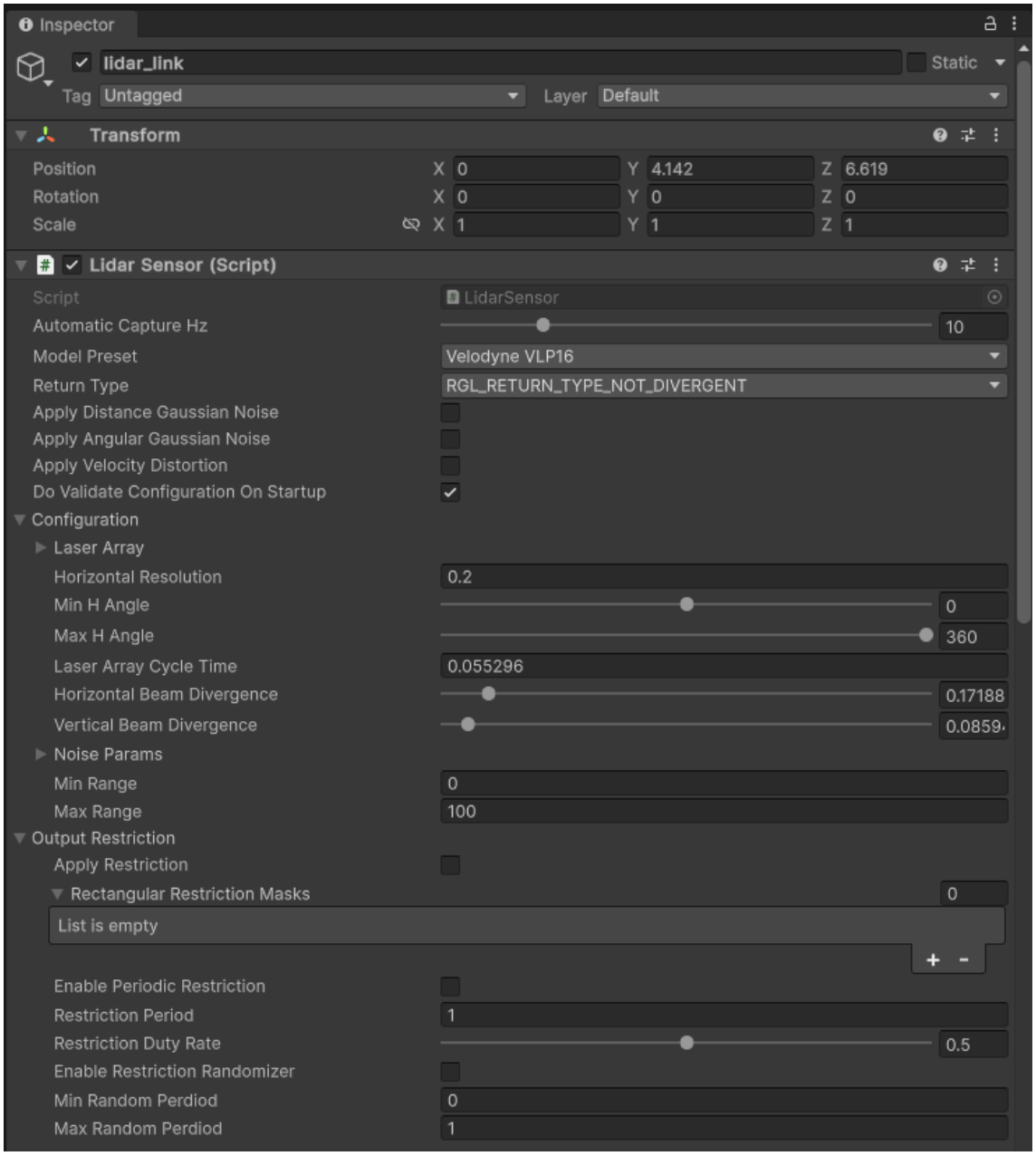


Figure A.13: General configuration of the Velodyne VLP-16 LiDAR sensor.

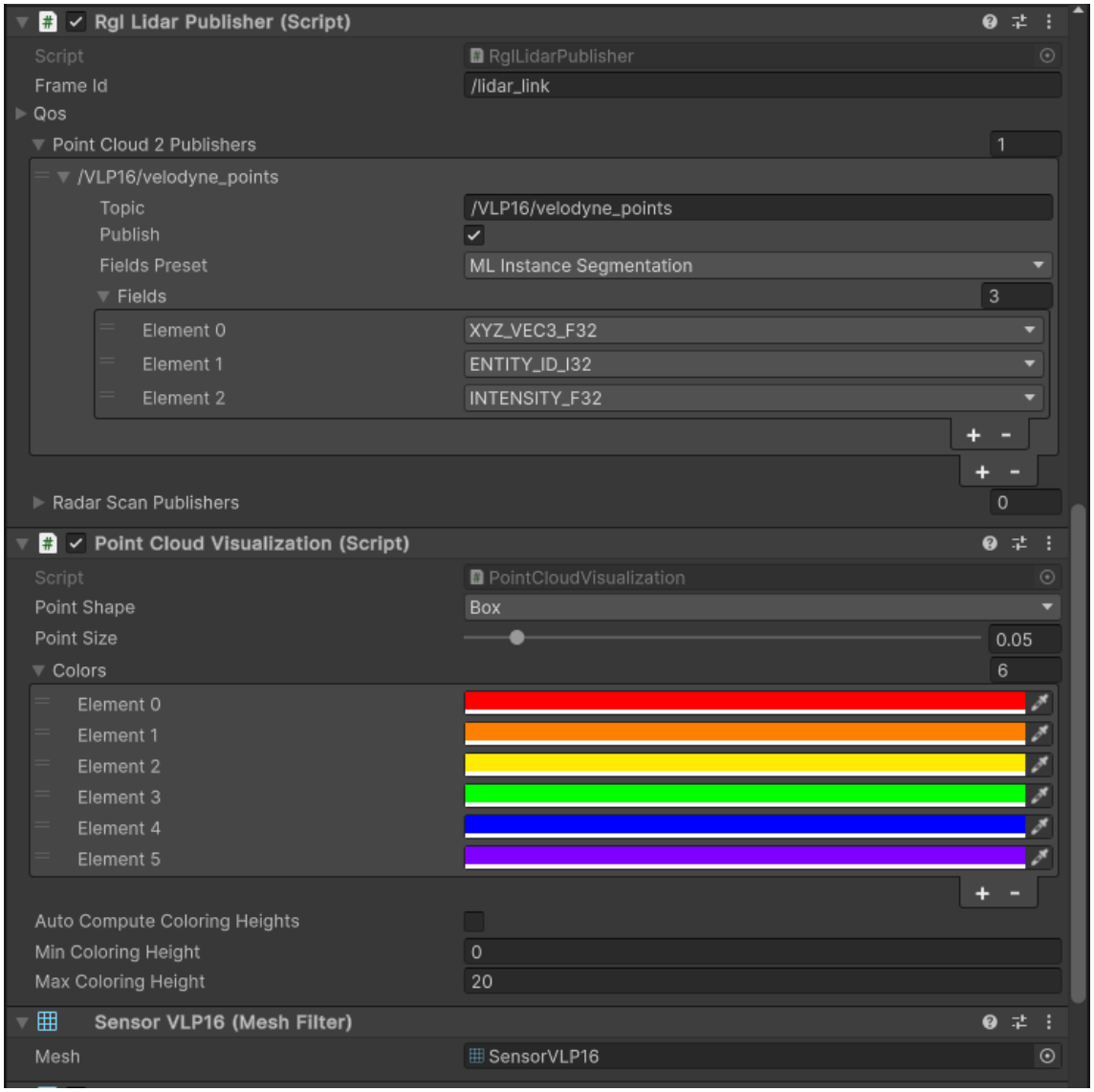


Figure A.14: ROS 2 publisher and visualization settings for the Velodyne VLP-16.

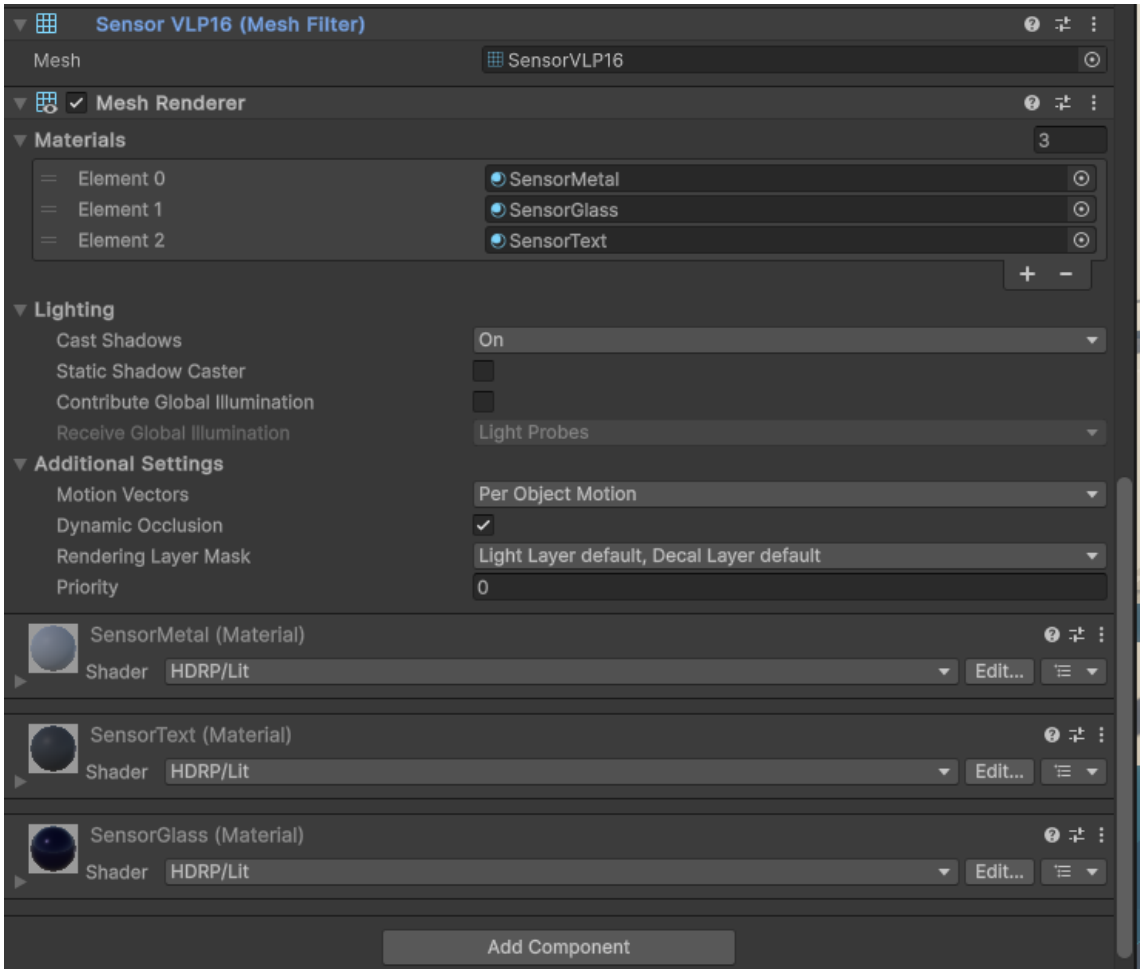


Figure A.15: Detailed material and rendering settings for the Velodyne VLP-16 model.

A.2 Performance Information

```
knadmin@lidar2:~/Luis/Unity/Projects$ ros2 topic list
/VLP16/velodyne_points
/ground_truth
/imu_data
/parameter_events
/rosout
knadmin@lidar2:~/Luis/Unity/Projects$ ros2 topic hz /VLP16/velodyne_points
average rate: 9.999
        min: 0.094s max: 0.105s std dev: 0.00436s window: 11
average rate: 10.009
        min: 0.093s max: 0.106s std dev: 0.00484s window: 22
average rate: 10.033
        min: 0.089s max: 0.110s std dev: 0.00538s window: 33
average rate: 9.998
        min: 0.089s max: 0.110s std dev: 0.00554s window: 43
average rate: 10.018
        min: 0.089s max: 0.110s std dev: 0.00523s window: 54
average rate: 10.015
        min: 0.089s max: 0.110s std dev: 0.00483s window: 64
```

Figure A.16: Publishing rate of the Velodyne 16 simulated in Unity.

```
^Cknadmin@lidar2:~/Luis/TFG/ros2_ws$ ros2 topic hz /noisy_lidar_poi
ntcloud
average rate: 9.940
        min: 0.095s max: 0.103s std dev: 0.00309s window: 11
average rate: 9.963
        min: 0.095s max: 0.104s std dev: 0.00326s window: 21
average rate: 9.996
        min: 0.095s max: 0.104s std dev: 0.00353s window: 32
average rate: 9.993
        min: 0.095s max: 0.104s std dev: 0.00362s window: 42
average rate: 9.988
        min: 0.095s max: 0.104s std dev: 0.00352s window: 52
average rate: 9.990
        min: 0.095s max: 0.104s std dev: 0.00353s window: 63
average rate: 9.992
        min: 0.094s max: 0.104s std dev: 0.00349s window: 74
```

Figure A.17: Publishing rate of the Velodyne 16 rainy points after applying the Rain Model.

Appendix B

Rain Model Validation Campaign

Contents

B.1 Introduction

This appendix presents the preliminary framework for validating the rain simulation model developed in this thesis against real-world rain conditions. The objective of this campaign is to assess the fidelity and accuracy of the simulated rain effects by comparing them with data collected from actual LiDAR sensor readings under varying rainfall scenarios. While the experimental data collection is still pending due to the necessity of suitable weather conditions, this document outlines the foundational work already completed, including the literature review, experimental setup, validation framework, and proposed evaluation metrics.

The validation of simulation models with real-world data is a critical step to ensure that the developed framework provides realistic and reliable results. The rain model's accuracy will be tested by analyzing its ability to replicate the behavior of LiDAR signals in rain, considering factors such as signal attenuation, point cloud density reduction, and noise levels.

This appendix is structured as follows:

- **Section B.2:** A review of relevant studies that have validated environmental simulation models using real-world data, with a particular focus on LiDAR sensors.
- **Section B.3:** A detailed description of the experimental setup, including the physical configuration of the LiDAR sensor, the environmental conditions required for data collection, and the criteria for selecting test sites.
- **Section B.4:** An overview of the validation framework, including the metrics and statistical methods that will be used to compare simulated and real-world results.

This document serves as a placeholder and roadmap for the full validation campaign, which will be completed once the required real-world data is collected. The findings from this campaign will play a crucial role in refining the simulation model and advancing the understanding of LiDAR performance under adverse weather conditions.

B.2 Review of Validation Methods for Rain Simulation Models

The validation of simulation models for rain effects on LiDAR systems is a critical step in ensuring their reliability and accuracy. However, existing approaches in the literature differ significantly in the type and rigor of their validation methods. This section reviews various validation methodologies, categorizing them into qualitative or quantitative approaches, and highlights their strengths and limitations.

B.2.1 Validation approaches in the literature

Several works focus on validating rain simulation models for sensors, but their approaches vary widely in methodology and rigor. Hasirlioglu and Riener [59] proposed an early model-based approach to simulate rain effects on automotive surround sensors, including LiDAR, radar, and cameras. However, the validation in this work was primarily qualitative, relying on visual assessments to judge the plausibility of the simulated effects. While this provides useful insights into the visual fidelity of the simulation, the lack of quantitative metrics and real-world data limits the robustness of their validation, particularly for LiDAR-specific effects like point cloud degradation.

Building on this foundation, Hasirlioglu and Riener [58] expanded their approach by incorporating data from a real rain simulation facility alongside virtual

rain simulations. This later work included a more rigorous quantitative validation, employing metrics such as Match Distance to compare cumulative distance distributions, Structural Similarity Index to evaluate image similarity for grayscale camera data, and Mean Squared Error to compare radar profiles. However, while their validation for radar and camera data was extensive, their treatment of LiDAR remained limited, focusing primarily on general signal attenuation rather than LiDAR-specific characteristics like point density or noise effects.

Espineira et al. [63] introduced a probabilistic rain model integrated into real-time simulations using Unreal Engine. Their validation is primarily qualitative, relying on visual assessments of the realism of simulated scenes and controlled virtual tests to demonstrate the model's capabilities. However, the absence of comparisons to real-world data limits the ability of their model to reliably represent real rain effects on LiDAR sensors.

Teufel et al. [68] simulated various adverse weather conditions, including rain, snow, and fog, to generate extended datasets for neural network training. Their validation focused on evaluating the robustness of object detection algorithms trained with these datasets, emphasizing improvements in neural network performance. However, this approach does not include a direct comparison between simulated and real rain effects.

Hahner et al. [15] concentrated on fog simulation but provided a robust validation framework that could inform future rain simulation studies. Their approach involved a quantitative comparison of simulated fog data with real-world measurements using metrics such as signal attenuation and SNR. While focused on fog rather than rain, this direct validation with real-world data makes their methodology one of the more rigorous reviewed.

Haider et al. [64] present a novel approach for modeling the effects of rain and fog on LiDAR sensor performance, integrating simulations based on Mie scattering theory. Their model enables the analysis of signal attenuation and SNR in both time domains and point cloud data. The validation process includes quantitative comparisons between simulated and real measurements conducted in controlled environments, utilizing KPIs such as detection rate (DR), false detection rate (FDR), and distance error (d_{error}). Results demonstrate a strong correlation between simulations and real-world measurements when raindrop distributions are consistent, with low MAPE, such as 2.1% for DR and 14.7% for FDR. This approach integrates high-fidelity simulations with experimental measurements, providing a rigorous framework for validating rain models in virtual environments.

B.3 Validation Methodology

To validate the rain simulation model, a structured approach is implemented, starting with the configuration of the experimental setup, followed by data collection, and concluding with the use of quantitative metrics to compare simulated

and real-world results.

B.3.1 Validation setup

The validation experiment uses a state-of-the-art LiDAR sensor, the Ouster OS0, configured to replicate real-world applications. Key specifications of the sensor include (see Table B.1):

Table B.1: Specifications of the Ouster OS0 LiDAR Sensor

Specification	Value
Horizontal (FoV)	360°
Vertical (FoV)	90°
Angular Resolution	0.35° (horizontal), 0.7° (vertical)
Maximum Range	120 meters
Scanning Frequency	10 Hz

The details of the sensor setup and environment are as follows:

B.3.1.1 LiDAR configuration

To implement the rain model on a real sensor (Ouster OS0), it was necessary to adjust several parameters. In simulation, the sensor placed in the Unity environment provides all the required information, including the orientation (azimuth and elevation angles) of each beam fired by the sensor. However, for real sensors, this information is either unavailable or varies across devices. Three sensors were evaluated to determine if they met the necessary requirements for applying the model: Livox, Velodyne, and Ouster. Among these, only the Ouster sensor could be modified to provide the required data.

Unlike in simulation, the real Ouster sensor does not directly provide the orientation of each beam, as this information is intrinsic to its internal mechanisms. Instead, the sensor outputs the origin point (0,0,0) and the detected points (x, y, z). Initially, no-hits were represented as NaN (Not a Number), making them unusable for computations. To address this limitation, the driver was modified to replace NaN values with a fixed distance. This modification allowed no-hits to be represented as standard points (x, y, z), making them easily distinguishable from hits.

With this adjustment, we can calculate the unit vector that defines the orientation of each beam using the origin point and the final point (hit or no-hit). This unit

vector, combined with the distance to the final point, provides all the parameters necessary to apply the rain model.[1]

B.3.2 Data acquisition system

The experimental setup for data acquisition consists of the Ouster OS0 sensor, power supply, cabling, and a set of computers configured for remote operation. This configuration ensures the system can be controlled remotely, allowing data recording during precipitation events without requiring physical presence.

[1] For further details, refer to the *ouster-lidar/ouster-ros/issues/396* GitHub repository.

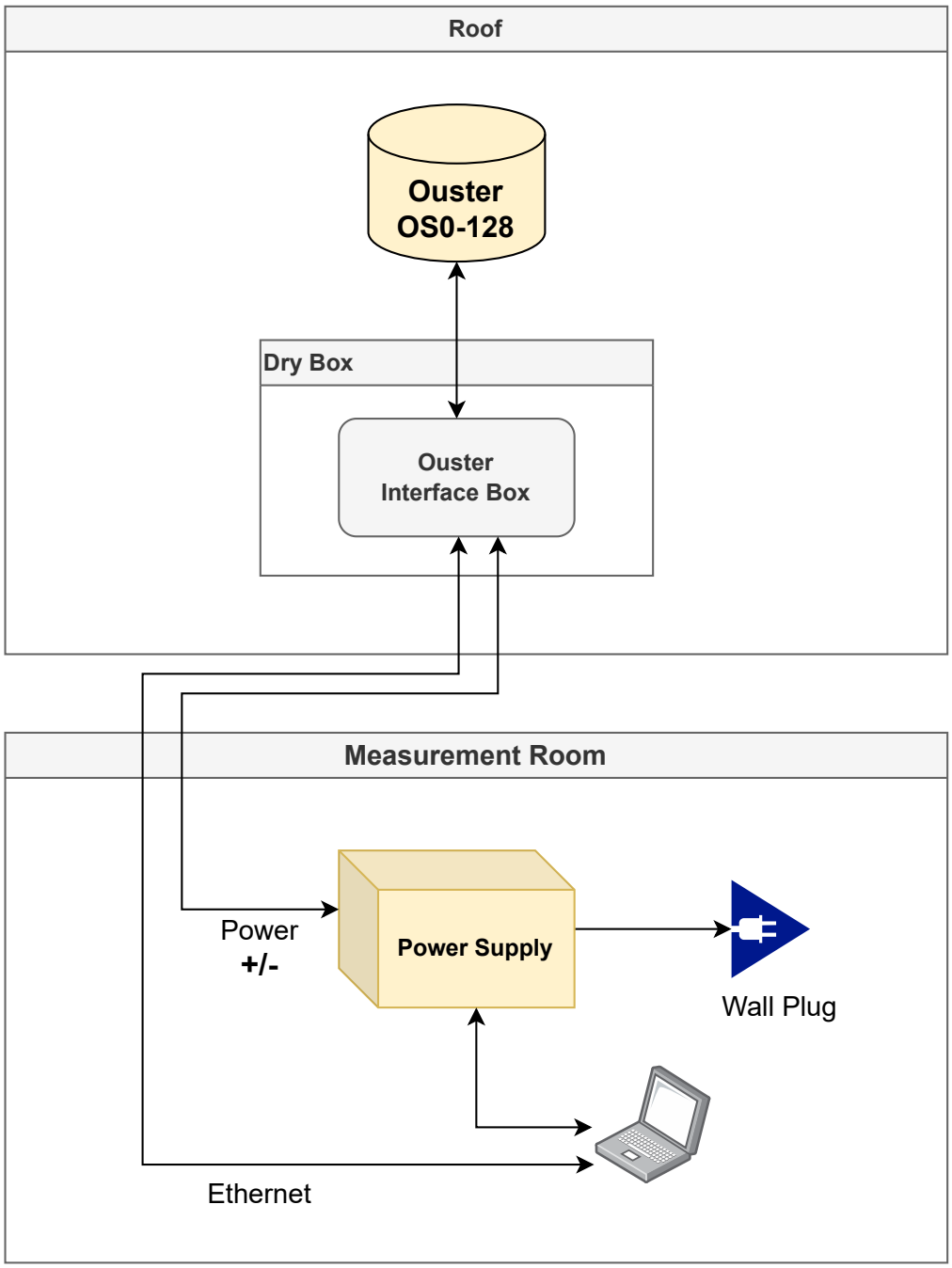


Figure B.1: Schematic of the experimental setup showing the placement of the Ouster OS0-128 sensor, interface box, dry box, and connections to the measurement room, including the power supply and Ethernet cabling.

Sensor and cabling The Ouster OS0 LiDAR sensor is mounted securely at a fixed height and orientation to ensure consistent data recording. To protect the sensitive components from environmental conditions, all the important components, including the power supply and network equipment, are stored in a drybox. The cables are routed from the drybox to the measurement room, ensuring a secure and reliable connection. The sensor is connected to the system using robust Ethernet cabling, which minimizes interference and supports high-speed data transmission from the sensor to the recording computer.

Power supply To enable remote operation, the sensor is powered through a programmable power supply. This power supply is connected to the Ouster sensor and can be remotely turned on or off via commands. This feature allows the sensor to remain inactive when not in use, conserving energy, and ensures that the system can be activated on-demand whenever precipitation is detected.

Computers for data recording The data acquisition system includes a set of computers connected to the sensor via a local network. These computers are equipped with software capable of recording the LiDAR data in real-time. Each computer is configured to be accessible remotely, enabling users to:

- Connect to the system via a secure remote desktop.
- Turn on the programmable power supply.
- Start and stop the data recording process.

Remote monitoring and operation This setup allows for complete remote control of the system. By monitoring weather conditions, the system can be activated whenever precipitation occurs, ensuring efficient data collection during rain events. The flexibility of the remote configuration eliminates the need for constant manual supervision, making the data acquisition process more efficient and adaptable to unpredictable weather patterns.

B.3.3 Data collection

The experiments are conducted in a controlled environment to ensure repeatability. For a precise evaluation, a static surrounding is necessary around the sensor.

Figure B.2: Configuration and positioning of the Ouster sensor for the validation campaign.

To facilitate the analysis of the recorded data, the Ouster sensor's horizontal field of view (FoV) was reduced, as the OOI should be a solid and static surface. The adjusted FoV approximately corresponds to the area depicted in Fig. B.2.

Data is collected under both real-world rain conditions and simulated scenarios for direct comparison.

B.3.3.1 Real-world data collection

Real LiDAR measurements are captured in the experimental setup, with each test repeated multiple times to ensure consistency. The data collection process covers varying rain intensities and lighting conditions. For clarity, the recordings are divided into two main scenarios:

Real rainy scene In this scenario, data is collected during active precipitation to capture the impact of rain on the LiDAR sensor's performance. The experimental setup ensures controlled rain intensities. These recordings provide critical information on rain-induced signal attenuation, scattering effects, and point cloud density degradation.

Dry scene This scenario involves collecting data under identical environmental conditions but without any precipitation. Data collected in this scenario helps establish reference metrics such as point cloud density, distance accuracy, and SNR.

Simulated rainy scene To complement the real-world data, the rain model is applied to the data collected in the Dry Scene to generate a Rainy Simulated Scene. The generated Rainy Simulated Scene provides a controlled environment to compare simulated and real rainy conditions, enabling a thorough evaluation of the rain model's accuracy and reliability.

B.4 Validation Metrics

To evaluate the performance of the rain simulation model, the following KPIs are used:

- **Detection Rate:** The percentage of LiDAR points corresponding to correctly detected objects under rain conditions.

$$\text{DR} = \frac{\text{Correctly Detected Points}}{\text{Total Detected Points}} \times 100 \tag{B.1}$$

- **False Detection Rate:** The percentage of LiDAR points classified as rain droplets.

$$\text{FDR} = \frac{\text{False Points}}{\text{Total Detected Points}} \times 100 \tag{B.2}$$

- **Distance Error (d_{error}):** The mean absolute error in the distances measured by the LiDAR under real and simulated rain conditions.

$$d_{\text{error}} = \frac{\sum_{i=1}^{N} |d_{\text{real},i} - d_{\text{sim},i}|}{N} \tag{B.3}$$

Signal-to-noise ratio The SNR measures the quality of the LiDAR signal under varying conditions. It is defined as:

$$\text{SNR} = 20 \cdot \log_{10}\left(\frac{\mu}{\sigma}\right) \tag{B.4}$$

where:

- μ: The mean value of the LiDAR signal,
- σ: The standard deviation of the noise.

A higher SNR indicates better signal quality, as the LiDAR sensor is able to distinguish valid returns more effectively despite interference from rain.

Point cloud density Point cloud density represents the total number of valid points detected by the LiDAR sensor under varying rain intensities. It is calculated as:

$$\text{Density} = \frac{\text{Number of Valid Points}}{\text{Scan Area}} \tag{B.5}$$

where:

- **Number of Valid Points:** The total number of points successfully returned by the LiDAR sensor,
- **Scan Area:** The area scanned by the sensor, determined by its Field of View (FoV) and range.

B.4.1 Comparison and analysis

The validation compares real and simulated data across all defined metrics:

Correlation of results Quantitative metrics such as DR, FDR, and d_{error} are compared between real and simulated data to assess the accuracy of the rain simulation model. Additionally, the MAPE is calculated for the SNR and the signal attenuation ($\sigma_{\text{ext,rain}}$) to evaluate the consistency between real-world and simulated results. The MAPE is defined as:

$$\text{MAPE} = \frac{1}{n} \sum_{i=1}^{n} \left| \frac{y_i - x_i}{y_i} \right| \times 100 \tag{B.6}$$

where y_i is the measured value, x_i is the simulated value, and n represents the total number of data points. The MAPE provides a quantitative measure of the accuracy of the simulated data compared to the real-world data, making it a critical metric for validating the model's performance under different rain intensities.

These metrics provide a comprehensive framework for evaluating the performance of the rain simulation on LiDAR and enables a robust validation of the model.

Bibliography

[1] Michael A. Lefsky, Warren B. Cohen, Geoffrey G. Parker, and David J. Harding. Lidar remote sensing for ecosystem studies: Lidar, an emerging remote sensing technology that directly measures the three-dimensional distribution of plant canopies, can accurately estimate vegetation structural attributes and should be of particular interest to forest, landscape, and global ecologists. *BioScience*, 52(1):19–30, 01 2002.

[2] Jeffrey S. Deems, Thomas H. Painter, and David C. Finnegan. Lidar measurement of snow depth: a review. *Journal of Glaciology*, 59(215):467–479, 2013.

[3] Open Robotics. *ROS 2 Documentation*, 2025.

[4] Amjed Almousa, Belal Sababha, Nailah Al-Madi, Amro Barghouthi, and Rimah Younisse. Utsim: A framework and simulator for uav air traffic integration, control, and communication. *International Journal of Advanced Robotic Systems*, 16:172988141987093, 09 2019.

[5] Unity Technologies. *Unity Documentation*, 2025.

[6] Blender Foundation. *Blender: The Free and Open Source 3D Creation Suite*, 2025.

[7] Foxglove Technologies. Foxglove studio. `https://foxglove.dev/`, 2025. Visual debugging and data visualization tool for robotics.

[8] David Van Krevelen and Ronald Poelman. A survey of fmcw radar: Principles and applications. *IEEE Aerospace and Electronic Systems Magazine*, 33(3):28–35, 2018.

[9] Yang Shen, Yitian Wang, Hai Wang, and Lei Liu. Event cameras: Emerging technology for high-speed visual sensing. *Nature Electronics*, 3(7):323–334, 2020.

[10] Bo Liu and Wenhua Chen. Advances in frequency-modulated continuous-wave (fmcw) lidar for autonomous vehicles. *Sensors*, 22(5):1123, 2022.

[11] Michael Johnson and Emily Smith. Lidar penetration in fog and rain: A study on performance for autonomous systems. *Journal of Autonomous Robotics*, 12(4):123–145, 2021.

[12] Sinan Hasirlioglu and Andreas Riener. Introduction to rain and fog attenuation on automotive surround sensors. pages 1–7, 10 2017.

[13] Christian P. Robert and George Casella. *Monte Carlo Statistical Methods*, volume 2. Springer, 1999.

[14] Alonso Llorente. *Monte Carlo simulation tool to assess SLAM performance*. PhD thesis, 06 2024.

[15] Martin Hahner, Christos Sakaridis, Dengxin Dai, and Luc Van Gool. Fog simulation on real lidar point clouds for 3d object detection in adverse weather. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 15283–15292, October 2021.

[16] Jie Shan and Charles K. Toth. *Topographic Laser Ranging and Scanning: Principles and Processing*. CRC Press, 2009.

[17] Orazio Svelto. *Principles of Lasers*. Springer, 5th edition, 2010.

[18] Monte D. Turner and Gary W. Kamerman. *Laser Radar: Progress and Opportunities in Active Electro-Optical Sensing*. Springer, 2016.

[19] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011. IEEE.

[20] Andreas Geiger, Philipp Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3354–3361. IEEE, 2012.

[21] Holger Caesar, Varun Bankiti, Alexander H Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11621–11631, 2020.

[22] Baidu Inc. Apollo: An open autonomous driving platform, 2022.

[23] Greg Turk. The ply polygon file format, 1994.

[24] American Society for Photogrammetry and Remote Sensing (ASPRS). LAS Specification Version 1.4 – R15, 2019.

[25] Niantic Labs. Spz file format for 3d gaussian splats, 2023.

[26] Hugging Face. Introduction to 3d gaussian splatting, 2023.

[27] Iraj Sadegh Amiri, Ahmed Rashed, Fatma Houssien, and Abd Mohammed. Temperature effects on characteristics and performance of near-infrared wide bandwidth for different avalanche photodiodes structures. *Results in Physics*, 14, 06 2019.

[28] M. Lemmens. Airborne lidar sensor: Product survey. *GIM International*, 21(2), 2007.

[29] J.R. Ridgway and et al. Airborne laser altimeter survey of long valley, california. *Geophysical Journal International*, 133:267–280, 1997.

[30] A. Shaker, W. Y. Yan, and N. El-Ashmawy. The effects of laser reflection angle on radiometric correction of the airborne lidar intensity data. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XXXVIII-5/W12:213–217, 2011.

[31] Timo Hanke, Alexander Schaermann, Matthias Geiger, Konstantin Weiler, Nils Hirsenkorn, Andreas Rauch, Stefan-Alexander Schneider, and Erwin Biebl. Generation and validation of virtual point cloud data for automated driving systems. pages 1–6, 10 2017.

[32] Xiangyu Yue, Bichen Wu, Sanjit Seshia, Kurt Keutzer, and Alberto Vincentelli. A lidar point cloud generator: from a virtual world to autonomous driving. pages 458–464, 06 2018.

[33] Chenqi Li, Yuan Ren, and Bingbing Liu. Pcgen: Point cloud generator for lidar simulation. pages 11676–11682, 05 2023.

[34] Hanfeng Wu, Xingxing Zuo, Stefan Leutenegger, Or Litany, Konrad Schindler, and Shengyu Huang. Dynamic lidar re-simulation using compositional neural fields, 2024.

[35] Richard Marcus, Niklas Knoop, Bernhard Egger, and Marc Stamminger. A lightweight machine learning pipeline for lidar-simulation. pages 176–183, 01 2022.

[36] Joseph Mom, Silas Tyokighir, and Gabriel Igwue. Evaluation of some raindrop size distribution models for different rain rates. *International Journal of Engineering and Technical Research*, 10:131–135, 09 2021.

[37] J. S. Marshall and W. Mc K. Palmer. The distribution of raindrops with size. *Journal of Atmospheric Sciences*, 5(4):165–166, 1948.

[38] Graham Feingold and Zev Levin. The lognormal fit to raindrop spectra from frontal convective clouds in israel. *Journal of Applied Meteorology*, 25:1346–1364, 09 1986.

[39] D. Deirmendjian. *Electromagnetic Scattering on Spherical Polydispersions*. The RAND Corporation, Santa Monica, California, April 1969. Report R-456-PR, prepared for United States Air Force Project RAND.

[40] Lord Rayleigh. On the light from the sky, its polarization and colour. *Philosophical Magazine*, 41(271):107–120, 1871. Seminal work on Rayleigh scattering.

[41] Gustav Mie. Beiträge zur Optik trüber Medien, speziell kolloidaler Metallösungen. *Annalen der Physik*, 330(3):377–445, January 1908.

[42] August Beer. Bestimmung der absorption des rothen lichts in farbigen flüssigkeiten. *Annalen der Physik und Chemie*, 86(2):78–88, 1852. Established the relationship between absorption, concentration, and path length in solutions.

[43] Joe Hocking. *Unity in Action: Multiplatform Game Development in C# with Unity 2021, Third Edition*. Manning Publications, Shelter Island, NY, 3rd edition, 2021. Explores advanced Unity concepts including scene management, component-based architecture, and optimization techniques.

[44] GameDev Beginner. Raycasts in unity made easy, 2023.

[45] Enrique Fernández, Aaron Martinez, and Luis Sánchez. *Learning ROS for Robotics Programming*. Packt Publishing, 2nd edition, 2015. A comprehensive guide to ROS and Gazebo for robotic simulations and development.

[46] Femexrobotica.org. *Simulación de Robots Móviles en Gazebo*, 2016. Detailed guide for simulating mobile robots using Gazebo and ROS.

[47] Open Robotics. *Gazebo Documentation*, 2025. Official documentation of Gazebo, including tutorials and API references.

[48] Stephen Seth Ulibarri. *Unreal Engine C++: The Ultimate Developer's Handbook*. Self-published, 2020. Comprehensive guide covering advanced game development in Unreal Engine using C++.

[49] Robotec.AI. *ROS2 for Unity Documentation*, 2025. Comprehensive documentation for integrating ROS2 with Unity3D, including installation, configuration, and usage examples.

[50] Unity Technologies. Ros–tcp connector: Integration of ros/ros2 with unity3d, 2025. Facilitates communication between Unity3D and ROS/ROS2 environments.

[51] Siemens. Ros#: Robot operating system integration for unity3d, 2025. A library for integrating Unity3D with ROS using C#.

[52] Robotec.AI. Robotecgpulidar: Gpu-accelerated lidar simulation library, 2025.

[53] Field Robotics Japan. Unitysensors: Ros/ros2-enabled sensor models for unity3d, 2025. Provides virtual sensor models (LiDAR, IMU, Camera, GNSS) with ROS/ROS2 integration.

[54] John Doe and Jane Smith. A methodology to model the rain and fog effect on the light detection and ranging (lidar) sensor performance for the simulation-based testing of lidar systems. *Sensors*, 23(15):6891, 2023. Provides simulation-based insights into how rain and fog affect LiDAR performance.

[55] Alice Johnson and Robert Brown. Predicting the influence of rain on lidar in adas. *Electronics*, 8(1):89, 2023. Develops mathematical models to assess rain impact on LiDAR for ADAS.

[56] Emily White and Michael Green. Investigation of automotive lidar vision in rain from material and optical perspectives. *Sensors*, 24(10):2997, 2024. Examines signal attenuation and false reflections caused by rain in automotive LiDAR.

[57] OpenMP Architecture Review Board. OpenMP application program interface version 3.0, May 2008.

[58] Sinan Hasirlioglu and Andreas Riener. A general approach for simulating rain effects on sensor data in real and virtual environments. *IEEE Transactions on Intelligent Vehicles*, PP:1–1, 12 2019.

[59] Sinan Hasirlioglu and Andreas Riener. A model-based approach to simulate rain effects on automotive surround sensor data. pages 2609–2615, 11 2018.

[60] Christopher Goodin, Daniel Carruth, Matthew Doude, and Christopher Hudson. Predicting the influence of rain on lidar in adas. *Electronics*, 8:89, 01 2019.

[61] Jing Guo, He Zhang, and Xiang-jin Zhang. Propagating characteristics of pulsed laser in rain. *International Journal of Antennas and Propagation*, 2015:1–7, 09 2015.

[62] Laurent Hespel, Nicolas Riviere, Thierry Huet, B. Tanguy, and Romain Ceolato. Performance evaluation of laser scanners through the atmosphere with adverse condition. *Proc SPIE*, 8186, 10 2011.

[63] Juan Espineira, Jonathan Robinson, Jakobus Groenewald, Pak Chan, and Valentina Donzella. Realistic lidar with noise model for real-time testing of automated vehicles in a virtual environment. *IEEE Sensors Journal*, PP:1–1, 02 2021.

[64] Arsalan Haider, Marcell Pigniczki, Shotaro Koyama, Michael Köhler, Lukas Haas, Maximilian Fink, Michael Schardt, Koji Nagase, Thomas Zeh, Abdulkadir Eryildirim, Tim Poguntke, Hideo Inoue, Martin Jakobi, and Alexander Koch. A methodology to model the rain and fog effect on the performance of automotive lidar sensors. *Sensors*, 23:6891, 08 2023.

[65] Pierre Bouguer. *Essai d'optique sur la gradation de la lumière*. Claude Jombert, 1729. Introduced the concept of light attenuation in a medium.

[66] Johann Heinrich Lambert. *Photometria sive de mensura et gradibus luminis, colorum et umbrae*. Eberhardt Klett, 1760. Formalized the relationship between light intensity and path length in absorbing media.

[67] T. G. Mayerhöfer, S. Pahlow, and J. Popp. The bouguer–beer–lambert law: Shining light on the obscure. *ChemPhysChem*, 21(19):2020–2040, 2020. Comprehensive review of the historical and scientific evolution of the Bouguer-Beer-Lambert law.

[68] Sven Teufel, Georg Volk, Alexander von Bernuth, and Oliver Bringmann. Simulating realistic rain, snow, and fog variations for comprehensive performance characterization of lidar perception. pages 1–7, 06 2022.