POLITECNICO DI TORINO

MASTER's Degree in Mechatronic Engineering

MASTER's Degree Thesis

A Comparative Evaluation of LiDAR Odometry Algorithms in Urban and Unstructured Scenarios

Supervisors

Prof. M. CHIABERGE

M. MARTINI

M. AMBROSIO

G. FRANCHINI

G. AUDRITO

Candidate

Federica ZETTI

October 2025

Abstract

LiDAR-based odometry is a cornerstone of autonomous navigation, robotic perception, and mapping, offering precise pose estimation even when cameras or GPS become unreliable. However, most existing algorithms are designed and benchmarked for structured, urban-like environments, leaving their robustness in unstructured settings—such as vineyards, forests, and off-road terrains—insufficiently validated. This gap poses a critical limitation for deploying autonomous systems in agricultural robotics and other non-urban applications where localization failures can compromise safety and mission success.

Four state-of-the-art LiDAR odometry algorithms—KISS-ICP, Genz-ICP, MOLA-LO, and SiMpLe—were evaluated on two contrasting datasets. The KITTI benchmark represented structured urban roads, while a custom vineyard dataset collected by the Interdepartmental Center PiC4SER captured the irregular and repetitive geometry of agricultural fields. Each algorithm was first tested using its default configurations and then re-optimized for both environments. Performance was assessed through standard KITTI odometry metrics (absolute and relative pose error) and qualitative trajectory comparisons against GPS ground truth.

On the KITTI sequences, all methods demonstrated competitive accuracy, with KISS-ICP and MOLA-LO exhibiting slightly lower drift. In the vineyard dataset, performance differences became pronounced: Genz-ICP, which relies heavily on planar and edge features, experienced substantial accuracy degradation; SiMpLe achieved high computational efficiency but suffered frequent misalignments in repetitive row patterns; and KISS-ICP maintained stable estimates through its adaptive correspondence strategy.The findings reveal that benchmarking exclusively on urban data overestimates odometry robustness in unstructured environments. Hybrid strategies combining feature-rich techniques with adaptive correspondence models—such as those employed by MOLA-LO—show strong potential for enhancing reliability in agricultural robotics and similar contexts. Expanding benchmarking datasets to include diverse, real-world scenarios is essential for advancing LiDAR odometry toward dependable deployment beyond controlled urban settings.

Table of Contents

List of Figures

List of Tables

Acronyms

LiDAR	Light Detection And Ranging.
LOAM	LiDAR Odometry And Mapping.
Fast-LIO	Fast LiDAR-Inertial Odometry.
NDT	Normal Distributions Transform.
LIO-SAM	LiDAR-Inertial Odometry via Smoothing And Mapping.
ROS	Robot Operating System.
IMU	Inertial Measurement Unit.
GPS	Global Positioning System.
GNSS	Global Navigation Satellite System.
INS	Inertial Navigation System.
TOF	Time-Of-Flight.
SNR	Signal-to-Noise Ratio.
AMCW	Continuous-Wave Amplitude Modulated.
FMCW	Continuous-Wave Frequency Modulated.
ICP	Iterative Closest Point.

Chapter 1

Introduction

1.1 Background and motivation

Overview of robot localization

The main topic of the thesis represents the crucial problem of robot localization, that is fundamental for autonomous systems in order to achieve goals[1]. Robot localization can be divided into two subproblem: the determination of robot's position and orientation, that combined together is the problem of determining the robot's pose. Localization is the foundation for autonomous navigation, launching some more intricate tasks like obstacle avoidance, path planning and mapping. [1]

In this thesis, this problem will be addressed from two points of view: the problem of localization will be studied in a very structured environment and in a very challenging one, in order to study this issue under some major challenges like unsteady terrain, noisy data coming from the sensor and limited or sparse features of the environment. These challenges increase the probability of localization drift, i.e. small localization that accumulates over time. This has been done in order to highlight how performances of traditional methods become less and less promising in these latter conditions and to acknowledge the need to improve localization system in order to be used in the future in a wider range of applications.

LiDAR as a localization sensor

In the experiments and formulation of this thesis, the main sensors used for robot's odometry is the LiDAR (Light Detection and Ranging) sensor, which is a largely used technology in robotics. It generates high-resolution 3D point clouds, providing precise spatial data even in low-light or challenging environments. The LiDAR sensor is perfect for this type of double localization problem because of its leading accuracy, with a precision in the range of centimeters. The biggest dilemma regards the cost of the LiDAR sensor, because it can be prohibitively expensive in a lot of different scenario and this can limit their use, highlighting the need to develop more affordable options to make LiDAR more accessible.

1.2 Current State of LiDAR odometry algorithms

Review of existing solutions LiDAR Odometry (LO) is a fundamental component of modern autonomous navigation systems, dedicated to accurately estimating the robot's pose (position and orientation) through the sequential analysis of raw LiDAR point cloud data [2]. State-of-the-art algorithms primarily rely on scan matching techniques, such as the Iterative Closest Point (ICP) method, or on feature-based optimization approaches to effectively track motion. The following review focuses on recent algorithms notable for their efficiency and robustness:

- **GenZ-ICP**: This algorithm focuses on maximizing pose estimation accuracy, employing advanced ICP variants. It optimizes the selection and quality of features (e.g., point-to-point or point-to-plane correspondences) used for matching, aiming for excellent precision [3].
- **KISS-ICP**: Designed with the goal of keeping things simple and fast, KISS-ICP relies on a lean but solid point cloud matching strategy. It performs alignment operations efficiently without resorting to heavy or complex data structures for nearest neighbor computation, which makes it especially suitable for real-time use on systems with limited processing power [4].
- **MOLA**: MOLA is an architectural framework designed to fuse multiple odometry and sensor sources flexibly. It is distinguished by its robustness and scalability, often integrating LiDAR odometry with Inertial Measurement Units (IMU) and graph-based optimization to maintain accuracy over long distances and in dynamic environments [5].
- **SiMpLe**: By using efficient and selective mapping techniques, SIMPLE aims to maintain high localization precision especially when point clouds are dense, highlighting the limitations of traditional methods that demand an abundance of geometric data for effective scan matching [6].

Limitations in real-world applications The accuracy of these latter algorithms tends to drop when used irregular or poorly structured settings, even if they perform very well under ideal conditions, like urban areas with many structural features [2]. Specifically:

- Feature Scarcity or Ambiguity: In contexts such as tunnels, dense forests, or agricultural fields, the lack of distinct geometric features or the presence of repetitive structures (e.g., straight rows in the vineyard) compromises the robustness of scan matching methods (like ICP variants), leading to significant localization drift.
- Noisy Data and Unsteady Terrain: The combined effect of sensor noise and irregular robot motion over rough terrain (factors not typically present in urban driving) degrades input data quality. This disproportionately penalizes

algorithms that are not explicitly designed for robust noise handling, like SiMpLe.

In this study, these drawbacks are especially important because the tested environments often have few distinctive features and include dynamic elements. Such conditions make it clear that there is still a gap in current research: the need for LiDAR odometry methods that remain accurate and dependable even when the data are noisy or the structure of the scene is weak. Addressing this issue is the main reason and motivation behind the development of this thesis.

1.3 Thesis objective

The main goal of this thesis is to benchmark the performance of innovative LiDAR odometry algorithms against the distinct realities of challenging environments. While the field of robotics has witnessed great success in structured settings, particularly urban driving, where high-resolution 3D LiDAR sensors are standard, a critical deployment barrier remains: reliable operation outside these ideal conditions.

The existing body of work confirms that current state-of-the-art algorithms—such as KISS-ICP, MOLA, and SiMpLe—vacillate when confronted with feature-sparse settings such as tunnels or agricultural fields [2]. These scenarios introduce unique operational challenges, like severely limited sensor data, high levels of noise, and constraints on the computational power of the robot platform itself.

This thesis takes on this gap in robustness as one of its central challenges. My objective is double:

- Rigorous Performance Assessment: To establish an objective performance baseline, we will quantitatively evaluate the selected algorithms for accuracy, robustness, and computational efficiency.
- Real-World Applicability Insight: To determine which odometry methods are best suited for deployment in service robotics where reliable, low-cost localization is of greatest importance. This includes applications like autonomous inspection in hazardous areas, where sacrificing slight accuracy for massive gains in efficiency and robustness is a necessary trade-off.

By testing four different approaches with limited computational resources, this work aims to understand where current methods fail and what kind of design or algorithmic changes could make them more reliable. The results are intended to help guide in future works the creation of LiDAR odometry systems that are more resilient and robust in unstructured environments.

1.4 Organization of the thesis

This thesis is organized into eight chapters, each addressing critical components necessary to understand, implement, and evaluate LiDAR odometry algorithms.

Chapter 1 introduces the topic, providing background information, an overview of the current state of LiDAR odometry algorithms, and outlining the specific objectives of this work.

Chapter 2 presents a literature review covering fundamental concepts in mobile robot localization. This includes coordinate systems, localization techniques for mobile robots, and the main types of sensors used in this domain.

Chapter 3 focuses on the LiDAR sensor. It discusses the basic principles of LiDAR imaging, along with a detailed description of sources and detectors commonly used in modern LiDAR systems.

Chapter 4 is dedicated to the Robot Operating System (ROS). The chapter provides an overview of its architecture, graph-based communication model and useful development tools.

Chapter 5 reviews the state of the art in LiDAR-only odometry algorithms. It introduces two main categories: direct matching approaches and feature-based approaches.

Chapter 6 presents the experimental evaluation and benchmarking of the four selected LiDAR odometry algorithms. Initially, performance is tested on open-source datasets such as KITTI. Subsequently, data collected from the PIC4SeR platform is used to assess algorithm behavior in real-world scenarios.

Chapter 7 provides a detailed analysis of the results obtained in the previous chapter from both a quantitative and qualitative perspective. It compares algorithm performance in terms of accuracy and computational efficiency, identifying key strengths and limitations.

Chapter 8 concludes the thesis by summarizing the obtained results and outlining future research directions.

Chapter 2

Literature review

This chapter introduces the foundational concepts and methodologies that set up the analysis presented in this thesis. It begins with an overview of basic knowledge on coordinate systems and transformations in Sec. 2.1. Section 2.2 illustrates the core concept of localization next to a discussion of different methodological approaches. Finally, Sec. 2.3 lists the most common sensors used in mobile robotics, with the exception of the LiDAR sensor, which is detailed in Chapter 3.

2.1 Coordinate systems

2.1.1 Coordinate systems and frames of reference

A prerequisite for any mobile robotics application is the consistent and efficient representation of the robot's position and velocity in space [7]. To achieve this, the concept of coordinate frames was introduced to facilitate the accurate exchange of geometric information between interfacing systems [8].

The first step in this process is defining spatial conventions, as the initial arrangement of axes can produce two fundamentally incompatible coordinate systems (left-handed versus right-handed). Every robot's placement in the real world is described by its pose (position x,y,z and orientation roll, pitch, yaw) relative to the three major axes of a Cartesian Coordinate system [7].

Such a coordinate system is shown in Figure 2.1. While the directions and orientations of the coordinate axes are theoretically arbitrary, the right-handed coordinate system is the dominant convention in the world of robotics, including the Robot Operating System (ROS), and is thus the convention adopted in my thesis.

The required level of detail—specifically, the choice of origin, the type of coordinate system, and which degrees-of-freedom are tracked—depends entirely on the specific application. For instance, a mobile robot typically requires its representation relative to a building or the Earth's coordinate system, whereas a static manipulator uses its base as the origin. These local coordinate systems are known as Frames of

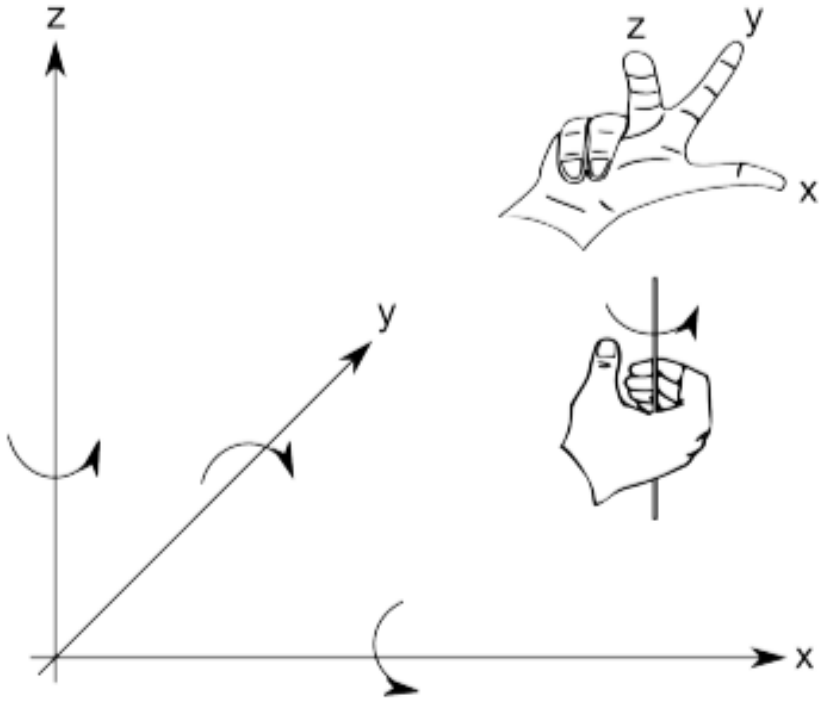


Figure 2.1: A coordinate system indicating the direction of the coordinate axes and rotation around them [7].

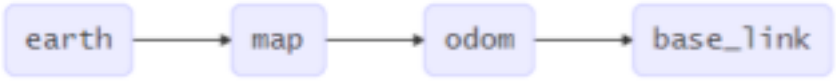


Figure 2.2: Relationship between frames, followed by the REP-105 convention. [9]

Reference. Furthermore, depending on the robot's degrees-of-freedom, it is customary to ignore any positional or orientational components that remain constant.

Following these definitions, the robot's pose—the common abbreviation for the combination of position and orientation—can be uniquely addressed. This description must always be made relative to a coordinate frame. In typical robotics applications, the tracking requires multiple coordinate frames, which are usually organized in a hierarchical tree, starting with the highest-level Inertial Reference Frame (or world frame). The remaining non-inertial and local frames follow established guidelines, specifically the REP-105 (Coordinate Frame Conventions) [9], which defines four principal frames: base_link, odom, map, and earth, as illustrated in Figure 2.2.

This tree architecture allows for the connection of multiple systems within the same global earth context. The base_link frame is rigidly affixed to the robot body. The map and odom frames are world-fixed frames whose origins are typically aligned with the robot's start position. While the map frame provides the most accurate position estimate and should be free of long-term drift, it may exhibit discontinuities (jumps). Conversely, the odom frame drifts over time but is guaranteed to be continuous

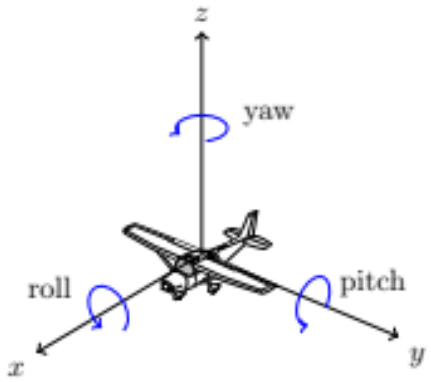


Figure 2.3: Euler Angles [10].

and is accurate enough for local planning and navigation [9]. Any secondary frames created by rigidly mounted sensors are typically connected to the base_link frame. Ideally, this frame tree should mirror the physical connections between all components involved.

2.1.2 Coordinate transformation

Having established a rigid transformation tree using the adopted conventions, the next step requires defining the mechanism to translate a point between any two coordinate frames [8]. To accomplish this, the representation of the pose needs to be defined together with the mechanism for converting coordinate representations from a parent frame to a child frame, and vice-versa [7]. This is the purpose of coordinate transformations [8].

The pose of an object, which may be a robot or another coordinate frame, must encapsulate both its position and orientation relative to its parent frame [7]. While position is simply represented by three coordinates representing a pure translation along the parent frame's axes, orientation requires more sophisticated and non-trivial conventions [7].

Translations. Since pose includes both position and orientation relative to the parent frame [7], the position component is straightforward. Three numbers represent the translation along each of the three axes [7]. When two coordinate frames are separated by a pure translational offset, transforming points is achieved simply by adding or subtracting the three coordinate offsets [10].

2.1.2.1 Euler Angles

Specifying orientation in three dimensions is mathematically more complex than position [7]. Among several available representations, each with its own trade-offs, Euler angles are often the most intuitive representation for visualization [7]. Figure 2.3 illustrates this concept. Orientation is expressed as a sequence of three successive

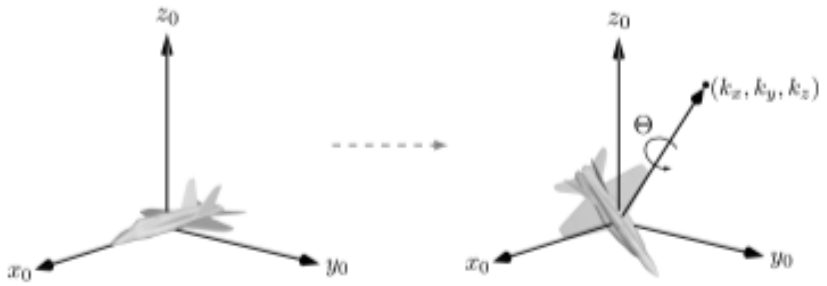


Figure 2.4: Axis-angle representation of orientation [10].

rotations around the coordinate axes, typically referred to as roll, pitch, and yaw [7]. A critical consideration when working with Euler angles is that the sequence in which the rotations are applied fundamentally alters the final result [7]. Furthermore, it is necessary to define whether rotations are applied relative to the parent frame or around the axes of the previously rotated frame. Combining these possibilities yields twenty-four possible conventions [7]. Although intuitive, Euler angles suffer from mathematical discontinuities (gimbal lock). In specific configurations, small changes in spatial orientation can cause large, non-smooth jumps in the numerical representation, making them inconvenient for applications requiring smooth kinematic updates [7].

2.1.2.2 Axis Angles

An alternative to encoding orientation via three successive Euler angles is to represent it as a single rotation θ around an arbitrary unit vector$[k_x, k_y, k_z]^T$ [7]. This is known as the axis-angle representation [7]. The geometric idea is illustrated in Figure 2.4.

2.1.2.3 Quaternions

The quaternion is one of the most widely used methods for encoding orientation in computational robotics [7]. There is a close, direct relationship between quaternions and the axis-angle representation [7]. If θ and $[k_x, k_y, k_z]^T$ describe an orientation in axis-angle form, the quaternion representation $q = (x, y, z, \omega)$ is defined as [7]:

$$x = k_x \sin\frac{\theta}{2}$$

$$y = k_y \sin\frac{\theta}{2}$$

$$z = k_z \sin\frac{\theta}{2}$$

$$\omega = \cos\frac{\theta}{2}$$

The resulting four-tuple q is referred to as a unit-quaternion because it holds the property $|q| = 1$ [7]. While quaternions are not geometrically intuitive, their popularity in robotics stems from their computational efficiency and, crucially, their ability

to avoid the singularities and discontinuities associated with Euler angles [7].

2.1.2.4 Rotation Matrices

Spatial coordinates are often represented as three-dimensional column vectors: $[xyz]^T$ [7]. This perspective is convenient because various spatial transformations can be performed using matrix operations [7]. Specifically, any rotation around the origin can be encoded as a 3×3 rotation matrix (R) [7]. Pre-multiplying the matrix by a point p performs the desired rotation: $p' = Rp$ [7].

The three elementary rotation matrices around the x, y, and z axes are [7]:

$$R_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{bmatrix}$$

$$R_y(\theta) = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix}$$

$$R_z(\psi) = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

The product of these three matrices, R_x, R_y, R_z, can represent any desired orientation [7]. This matrix will be responsible for the transformations from the robot's body frame to the world frame within the implementation of this work [7].

$$R_{xyz}(\phi, \theta, \psi) = \begin{bmatrix} \cos(\psi)\cos(\theta) & \sin(\psi)\sin(\theta) - \cos(\psi)cos(\theta) & \sin(\psi)sin(\theta)cos(\theta) + \cos(\psi)sin(\theta) \\ \sin(\psi)\cos(\theta) & sin(\psi)sin(\theta) + \cos(\psi)\cos(\theta) & -\sin(\phi)\sin(\theta)\cos(\theta) - \cos(\psi)\sin(\theta) \\ -\sin(\theta) & \cos(\theta)\sin(\phi) & \cos(\theta)\cos(\phi) \end{bmatrix}$$

The product of three elementary rotation matrices around the x, y, and z axes can represent any orientation [7]. For static rotations, the matrices must be multiplied in the order the rotations are intended to be applied [7]. The main advantage of rotation matrices is the convenient mechanism they provide for composing rotations and applying them to points , while the downside is the high redundancy of the nine-element representation [7].

2.2 Mobile robot localization

Mobile robot localization is the problem of determining the robot's pose relative to a given map of the environment. This process is commonly referred to as position estimation or position tracking [1]. Knowing the robot's pose is the first fundamental problem of autonomous navigation, addressing the core question:

"Where am I?"

Localization is one of the four essential building blocks of navigation, which include: perception (interpreting sensor data to extract meaningful information), localization (determining the robot's position within the environment), cognition (deciding how to act to achieve a goal), and motion control (modulating motor outputs to execute the desired trajectory) [11].

Localization can be framed as a problem of coordinate transformation [1]. Maps are defined in a global coordinate system, independent of the robot's current state. Localization is the act of establishing the correspondence between the map coordinate system and the robot's local coordinate system [1]. Knowing the robot's pose, expressed as $x_t = (x, y, \theta)^T$ in 2D space or as $x_t = (x, y, z, roll, pitch, yaw)^T$ in 3D space, is sufficient to determine this transformation, provided the pose is expressed in the same frame as the map [11].

However, a robot's exact pose cannot be measured directly, which is one of the main challenges in mobile robot localization [1]. Instead, the pose must be estimated over time from the data the robot collects. A single sensor reading usually does not contain enough information, so the system must combine data from multiple measurements, often using temporal or multi-sensor fusion methods [1]. This problem arises from the limited precision and incomplete nature of the robot's sensors and actuators.

Sensor noise Sensors provide the fundamental input for the perception process, making the quality of their readings critical [11]. Sensor noise imposes a fundamental limitation on the consistency of sensor readings, meaning the amount of useful information available from any single measurement is limited [1]. To counteract this reduction in information content, robots must integrate multiple readings, often using temporal or multi-sensor fusion to enhance the overall certainty of the input [1].

Sensor aliasing A second major limitation in mobile robot sensing is sensor aliasing, which refers to the non-uniqueness of a sensor reading (where multiple distinct positions yield the same measurement) [1]. This is a common occurrence in robotics. A classic example is navigating a uniform maze without unique landmarks: even noise-free sensor data may be insufficient to pinpoint the robot's position from a single measurement alone [1]. Consequently, localization techniques must be employed that base the robot's position recovery on a series of readings integrated over time, ensuring sufficient information is collected to resolve the ambiguity [1].

Effector noise Localization challenges are not limited to sensor technologies; robot effectors are also noisy, introducing uncertainty into the system [1]. A single movement command executed by a mobile robot can result in several possible outcomes, even when the initial state is well known [1]. This means that the simple act of moving tends to increase the robot's pose uncertainty over time. The primary

source of this inaccuracy often lies in an incomplete model of the environment, resulting in discrepancies between the intended motion, the physical motion, and the robot's proprioceptive estimates of motion [11].

2.2.1 Taxonomy of localization problems

Not every localization challenge presents the same degree of difficulty. To systematically understand the complexity of a specific problem, localization tasks can be categorized along several dimensions, particularly concerning the nature of the environment and the robot's initial knowledge. This brief taxonomy is primarily derived from the foundational work in probabilistic robotics [1].

- Local versus Global Localization: Localization problems can be grouped according to how much is known about the robot's initial position when the process begins:
  - **Position tracking**: assumes that the robot's starting pose is already known with reasonable accuracy. The uncertainty is therefore local, confined to a small region near the true position, and can usually be represented by a single, unimodal distribution such as a Gaussian.
  - **Global localization**: in this case, the initial pose is unknown. This case is harder than the first one because the pose uncertainty is unlimited and the algorithm must consider several different hypotheses.
  - **Kidnapped robot problem**: a special and difficult case of global localization where, after normal operation, the robot is suddenly moved to a new and unknown location. The challenge lies in detecting this major localization failure and quickly re-establishing the correct position using global methods. Tests involving this situation evaluate how well an algorithm can recover from unexpected errors.
- Static Versus Dynamic Environments: The persistence of objects within the environment defines another key dimension of difficulty:
  - **Static environments**: In these settings, the only variable component (state) is the robot's pose. All other objects remain permanently fixed in their location.
  - **Dynamic environments**: These environments contain objects, besides the robot itself, whose location or configuration changes over time (e.g., moving people or vehicles). Localization in dynamic environments is more complex than in static ones, as the solution must incorporate additional computational and modeling overhead to accommodate the moving elements.
- Passive Versus Active Approaches: This distinction describes whether the localization process can influence how the robot moves:

  - **Passive localization**: The system only observes what the robot is doing. The robot's path is planned by a separate module, and its motion is not adjusted to improve localization accuracy.
  - **Active localization**: In this kind of problems, the results are statistically better than passive ones because here the algorithm regulate the robot's motion in order to minimize localization error and/or reduce the risk of entering hazardous areas while poorly localized.

- Single-robot Versus Multi-Robot: The number of agents involved in the localization process:
  - **Single-robot localization**: Data collection and processing are centralized on a single robot platform, eliminating issues due to communication between robots.
  - **Multi-robot localization**: with this implementation, one robot's estimated position (or belief) can influence another's, as long as their relative positions are known; this usually involves handling complex communication links and data fusion between robots.

2.2.2 Probabilistic Map-based Localization

The core strategy for addressing the general robot localization problem involves continuously combining motion estimates with environmental observations within a known map [1]. Consider a mobile robot starting its movement from a precisely known location; it initially tracks its position using odometry (proprioceptive sensors). However, due to inherent odometry uncertainty, the pose uncertainty grows rapidly over time. To prevent this uncertainty from becoming unbounded, the robot must localize itself relative to its environment map using exteroceptive sensor data (e.g., LiDAR, vision, ultrasonic sensors) [1].

The information derived from the robot's odometry and the information provided by these environmental observations are combined to achieve the best possible pose estimate with respect to the map. This fusion process is typically separated into a general two-step update cycle:

- Prediction (Motion Update): Updating the robot's pose belief based on its motion (odometry).
- Correction (Measurement Update): Refining the estimated pose using the latest sensor readings from the environment..

The fundamental difference between probabilistic localization approaches lies in how the pose uncertainty (or belief state) is represented [1] [11]. This leads to distinct advantages and disadvantages between methods like Markov Localization and Kalman Filter Localization: Markov localization allows for localization starting from any unknown position and can thus recover from ambiguous situations because

1: **Algorithm Markov_localization(**$bel(x_{t-1}), u_t, z_t, m$**):**
2: *for all* x_t *do*
3: $\overline{bel}(x_t) = \int p(x_t \mid u_t, x_{t-1}, m)\ bel(x_{t-1})\ dx$
4: $bel(x_t) = \eta\ p(z_t \mid x_t, m)\ \overline{bel}(x_t)$
5: *endfor*
6: *return* $bel(x_t)$

Figure 2.5: Markov localization algorithm [1].

the robot can track multiple, completely disparate possible poses; on the other hand, the required memory and computational power can thus limit the precision and the size of the environment map that can be handled [1] [11].

Conversely, Kalman filter localization tracks the robot from an initially known position and is inherently both precise and computationally efficient [1] [11]. This approach is particularly well-suited for continuous world representations [11]. However, if the robot's uncertainty becomes too large and thus not truly uni-modal, the Kalman filter's assumption of a single Gaussian distribution can fail to capture the multitude of possible robot positions, potentially leading to localization failure [1] [11].

2.2.2.1 Markov localization

Markov localization tracks the robot's pose belief state by using an arbitrary probability density function to represent the robot's position [1]. It typically begins by fit together the robot's continuous configuration space into a finite, discrete number of possible poses within the map [1].

Given this general representation of the robot's position (which can be multi-modal), a robust update mechanism is necessary to compute the resulting belief state when new information is incorporated into the prior belief [1]. Probabilistic localization algorithms are fundamentally variants of the Bayes filter, and the direct application of Bayes filters to the localization problem is precisely what is termed Markov localization [1]. The algorithm, derived from the general Bayes filter structure, is depicted in Figure 2.5.

As shown in Figure 2.5, the algorithm requires a map (m) as input, which plays a role in the measurement model $p(z_t|x_t, m)$. Just like the Bayes filter, Markov localization transforms a probabilistic belief at time *t-1* into an updated belief at time t [1]. This method is capable of addressing the global localization problem, the position tracking problem, and the kidnapped robot problem within static environments [1].

The initial belief, $bel(x_0)$, reflects the robot's initial knowledge of its pose and is initialized differently depending on the specific localization task:

- **Position tracking** If the initial pose is precisely known (denoted $\bar{x_0}$), $bel(x_0)$ is initialized by a point-mass distribution:

$$bel(x_0) = \begin{cases} 1 & if x_0 = \bar{x_0} \\ 0 & otherwise \end{cases}$$

  In practical scenarios, the initial pose is often known only approximately. In such cases, the belief $bel(x_0)$ is usually initialized by a narrow Gaussian distribution centered around $\bar{x_0}$:

$$bel(x_0) = det(2\pi\Sigma)^{-\frac{1}{2}} exp[-\frac{1}{2}(x_0 - \bar{x_0})^T \Sigma^{-1}(x_0 - \bar{x_0})]$$

  where Σ is the covariance of the initial pose uncertainty [1].

- **Global localization** If the initial pose is unknown, $bel(x_0)$ must be initialized by a uniform distribution over the space of all legal poses in the map:

$$bel(x_0) = \frac{1}{|X|}$$

  where $|X|$ stands for the volume of the space of all poses within the map [1].

2.2.2.2 Kalman filter localization

The Kalman filter (KF) was initially developed as a technique for filtering and prediction in linear systems, implemented as a specific case of the Bayes filter [1]. The KF functions as an optimal estimator for systems that are linear and subject to zero-mean Gaussian noise [1]. It addresses the general problem of estimating the state X of a discrete-time controlled process governed by the linear system state transition equation:

$$\bar{x_{k+1}} = A_k x_k + B_k u_k + w_t$$

Here, A is the n×n system matrix, B is the n×m input gain matrix (where m is the dimension of the input vector u_k, and x_k is the n-dimensional state vector. The term w_k is an n-dimensional vector representing zero-mean Gaussian plant noise, included to model the uncertainty arising from the system dynamics and actuators [1].

The measurement equation is defined by

$$z_k = Hx_k + v_k$$

where H is a $q\ x\ n$ matrix that relates the state x_k to the measurement z_k (with q being the dimension of the measurement vector). The term v_k is a q-dimensional zero-mean Gaussian noise vector, known as the measurement noise vector, which incorporates the uncertainty of the measurement process [1].

The Kalman Filter is made up of two steps: a prediction phase followed by an

update (or correction) phase.

Prediction Step. In the prediction step, the state is projected forward using the state transition equation. Crucially, the uncertainty of the state must also be propagated forward in time. Since the state is modeled as a Gaussian distribution, fully characterized by its mean $\hat{x_k}$ and covariance P_k, the covariance is updated according to the following equation:

$$P_k^- = AP_kA^T + Q$$

In this equation, A is the same system matrix used to propagate the state mean, and Q represents the covariance matrix of the random Gaussian process noise. After this phase, the original Gaussian distribution defined by x_k and P_k is transformed into a new Gaussian, characterized by the predicted mean x_{k+1} and predicted covarianceP_{k+1}. This concludes the prediction step of the algorithm [1].

Update Step. The update step, also known as the correction step, fuses a new measurement (z_k) of an observable variable with the prior estimate from the prediction step [1]. First, the Kalman Gain (K) is calculated:

$$K = P^-H^T(HP^-H^T + Q)^{-1}$$

Next, the calculation of the difference between the expected observation and the actual observation is performed. This result is known as the residual or innovation equation [1]:

$$r_k = (z_k - H\hat{x}_k^-)$$

Finally, the mean and the covariance are corrected through these two final equations [1]:

$$\hat{x}_k = \hat{x}_k^- + r_k$$

$$P_k = (I - K_kH_k)P_k^-$$

The Figure 2.6 summarizes the sequence of these prediction and update steps.

2.2.2.3 Extended Kalman filter localization

The Extended Kalman Filter (EKF) is a widely used adaptation of the Kalman filter for nonlinear systems, and it remains fundamentally a Gaussian filter [1]. Its core mechanism involves applying a linearization around the current value of the state estimate whenever the system dynamics or measurement models are nonlinear [1]. In this context, the robot's dynamics are considered a nonlinear system, which is described by the nonlinear state transition function [1]:

$$x_{k+1} = f(x_k, u_k) + w_k$$

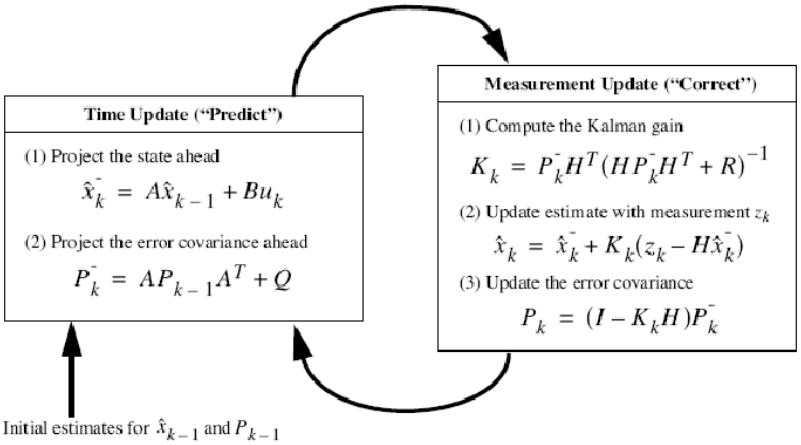


Figure 2.6: Kalman filter algorithm [12]

Here, x_{k+1} represents the robot's system state at time k+1, f is the nonlinear state transition function, and w_k is the process noise, which is assumed to be normally distributed. The term u_k represents the control inputs [1]. The measurements from the environment can be described by the following nonlinear sensor model [1]:

$$z_k = h(x_k) + v_k$$

In this equation, z_k represents the measurements at time k, h is the nonlinear sensor model that maps the state into the measurement space, and v_k is the normally distributed measurement noise [1].

The EKF algorithm proceeds through the standard two-step cycle:

Prediction Step. The first stage of the algorithm is the prediction step, which projects the current state estimate and its error covariance forward in time using the nonlinear function f [1]:

$$\hat{x}_k^- = f(\hat{x}_{k-1}, 0)$$

$$P_k^- = A_k P_{k-1} A_K^T + W_K Q_{k-1} W_k^T$$

The function f can often be described as a standard 3D forward kinematic model derived from Newtonian mechanics [1]. Crucially, the estimated error covariance (P) is propagated using the linearized models A and W, which are the Jacobians of the process evaluated at step k, perturbed by Q, the process noise covariance matrix [1].

Correction Step. The next phase is the correction step, where the latest measurement is fused with the predicted state. The first calculation involves determining the Kalman gain (K), which uses the measurement Jacobians H and V, the measurement covariance R, and the predicted covariance P_k^- [1]:

$$K_k = P_k^- H_k^T (H_k P_K^- H_k^T + V_k R_k V_k^T)^{-1}$$

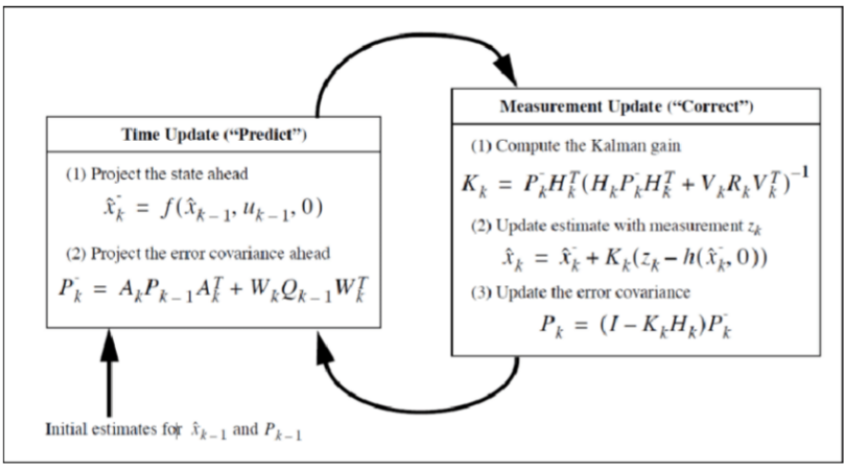


Figure 2.7: The Extended Kalman filter algorithm [13]

he calculated gain is then used to update the state vector by calculating the difference between the actual observation and the expected observation (innovation) and applying the gain [1]:

$$\hat{x}_k^- = \hat{x}_k^- + K_k(z_k - h(\hat{x}_K^-, 0)$$

As the final step, the Joseph form covariance update equation is used to promote filter stability by ensuring that the resulting covariance matrix P_K remains positive semi-definite [1]:

$$P_k = (I - K_k H_K) P_K^-$$

The entire process, including the prediction and correction phases of the Extended Kalman Filter, is summarized in Figure 2.7.

2.3 Sensors for mobile robots

A sensor is a device designed to convert a physical parameter or an environmental characteristic (such as temperature, distance, or speed) into a measurable signal that can be digitally processed to perform specific tasks [14]. Mobile robots rely on sensors to gather information about their environment, a prerequisite for safe navigation, complex perception, coordinated actions, and effective interaction with other agents [14].

Mobile robot sensors range from basic tactile devices, like bumpers, to advanced vision systems, such as structured light RGB-D cameras [14]. Regardless of the type, all sensors produce a digital output (e.g., a data string or a matrix) that the robot's computer can process. This output is typically created by discretizing one or more analog electrical signals using an Analog-to-Digital Converter (ADC) embedded within the sensor [14].

A sensor is inherently a type of input transducer. A transducer converts a signal

from one form of energy (e.g., thermal, mechanical) into another [14]. Transducers can either inject energy into the environment (called emitters or actuators) or capture energy from the environment (called receivers) [14]. A receiver that converts a measurable physical quantity, such as light or sound, into an electrical signal is precisely what is termed a sensor. In practical mobile robotics, the term "sensor" commonly refers to an integrated ensemble of transducers and associated devices packaged together to execute a specific sensing function [14].

2.3.1 Sensors Classifications

Sensors can be categorized in several ways. The most common classifications used in robotics are based on the source of the excitation signal and the domain of the measurement [14]:

- **Excitation Signal Source**:
  - Passive Sensors: These sensors generally do not require an external power supply to operate, except for minimal power needed for signal amplification and digital conversion [14].
  - Active Sensors: These sensors require an external power supply to self-generate and emit an excitation signal, measuring the environment's reaction to that signal (e.g., time-of-flight) [14].
- **Measurement Domain**:
  - Proprioceptive Sensors: These measure quantities dependent only on the robot's internal system and its current state (e.g., wheel position or joint angle) [14].
  - Exteroceptive Sensors: These measure quantities that depend on both the robot's state and the surrounding environment, such as the distance to the nearest obstacle [14].

Figure 2.8 illustrates a classification of common sensor types used in mobile robot applications.

Over recent years, researchers have developed numerous techniques and systems to detect and determine a robot's position [14]. No single approach is universally applicable; rather, every method possesses unique strengths and weaknesses [14]. These approaches are commonly categorized based on the primary sensors employed, such as Global Navigation Satellite Systems (GNSS), Wheel Odometry (Encoders), LiDAR or Ultrasonic Odometry, Inertial Navigation Systems (IMU), and Visual Odometry (Camera sensors) [14]. These techniques are usable for both indoors and outdoors, with the only exception of the Global Positioning System (GPS), which is generally unreliable indoors [14].

General classification (typical use)	Sensor Sensor System	PC or EC	A or P
Tactile sensors (detection of physical contact or closeness; security switches)	Contact switches, bumpers	EC	P
	Optical barriers	EC	A
	Noncontact proximity sensors	EC	A
Wheel/motor sensors (wheel/motor speed and position)	Brush encoders	PC	P
	Potentiometers	PC	P
	Synchros, resolvers	PC	A
	Optical encoders	PC	A
	Magnetic encoders	PC	A
	Inductive encoders	PC	A
	Capacitive encoders	PC	A
Heading sensors (orientation of the robot in relation to a fixed reference frame)	Compass	EC	P
	Gyroscopes	PC	P
	Inclinometers	EC	A/P
Ground-based beacons (localization in a fixed reference frame)	GPS	EC	A
	Active optical or RF beacons	EC	A
	Active ultrasonic beacons	EC	A
	Reflective beacons	EC	A
Active ranging (reflectivity, time-of-flight, and geometric triangulation)	Reflectivity sensors	EC	A
	Ultrasonic sensor	EC	A
	Laser rangefinder	EC	A
	Optical triangulation (1D)	EC	A
	Structured light (2D)	EC	A
Motion/speed sensors (speed relative to fixed or moving objects)	Doppler radar	EC	A
	Doppler sound	EC	A
Vision-based sensors (visual ranging, whole-image analysis, segmentation, object recognition)	CCD/CMOS camera(s)	EC	P
	Visual ranging packages		
	Object tracking packages		

A, active; P, passive; P/A, passive/active; PC, proprioceptive; EC, exteroceptive.

Figure 2.8: Classification of sensors used in mobile robotics applications [14].

2.3.1.1 Global Navigation Satellite System (GNSS)

A Global Navigation Satellite System is a satellite-based system that transmits signals to provide location information, including longitude, latitude, and altitude. Any receiver on Earth's surface can determine its position using GNSS, making it a highly popular technique for mobile robot localization [14]. The Global Positioning System (GPS), operated by the US, is a well-known satellite system consisting of twenty-four operational orbiting satellites.

Advantages and Disadvantages A major pro of GNSS is its ability to determine absolute position without error accumulation over distance, providing stable localization [14]. However, commercial GNSS accuracy is often limited to the meter level, which is insufficient for applications requiring high precision. Furthermore, GNSS signals are prone to blockage by walls and are sensitive to environmental factors (e.g., weather, metal structures), making them unsuitable for indoor applications [14].

2.3.1.2 Wheel Odometry

Odometry is an ancient concept derived from the Greek words hodos (journey or travel) and metron (measure). The term was first applied to wheel odometry, where a robot's motion is estimated by counting the number of wheel revolutions over a traveled distance to calculate linear displacement [14]. Wheel revolutions are typically

Figure 2.9: Optical encoder [15].

measured by Encoders, such as the optical encoder shown in Figure 2.9.

Advantages and Disadvantages While wheel odometry is simple, inexpensive, and provides a relative position, it is significantly affected by wheel slippage [14]. This slippage causes error accumulation that increases proportionally with the distance covered, making wheel odometry suitable only for short-term navigation tasks [14].

2.3.1.3 Inertial Navigation System (INS)

The Inertial Navigation System provides the mobile robot's position and orientation relative to a known starting point [14]. The core component is the Inertial Measurement Unit (IMU), which typically contains three accelerometers (providing linear accelerations) and three gyroscopes (providing rotational rates around three axes) [14]. Linear velocity and position are obtained by integrating the linear acceleration with respect to time, while the agent's orientation is obtained by integrating the rotational rates over time.

Advantages and Disadvantages Commercial inertial navigation systems are generally cheap but suffer from inherent noise and bias [14]. The integration process necessary to calculate position and orientation causes a large accumulation of error, leading to significant drift from the actual state [14]. Consequently, the inertial sensor alone is unreliable for determining position and orientation, but it is highly valuable when used to supplement other navigation systems, such as GPS, cameras, or LiDAR, to enhance overall reliability and accuracy [14].

2.3.1.4 Acoustic Systems

Acoustic systems utilize ultrasonic sensors to measure the distance between the system's location and obstacles by emitting acoustic energy [14]. The system transmits ultrasonic pulses and measures the time required for the reflected echoes (from surfaces or objects) to return, thereby calculating the distance traveled [14].

Advantages and Disadvantages Acoustic systems can provide high-accuracy motion estimation [14]. However, the accuracy is highly dependent on the reflection quality of the ultrasonic waves, which can be affected by the orientation of surfaces and the material properties of the objects [14]. Furthermore, the environment's acoustic noise can share the same frequency as the ultrasonic pulse, and the system is susceptible to receiving multiple echo reflections, which introduces complexity and error [14].

2.3.1.5 Visual Systems

Visual systems use a camera sensor to capture abundant information about the environment, making mobile robot localization one of their key applications [14]. Computer vision algorithms then analyze the captured images to determine the robot's position and orientation [14].

Advantages and Disadvantages The camera is considered a low-cost sensor compared to many other motion estimation sensors. Despite being inexpensive and providing rich data, camera sensors are sensitive to ambient conditions such as illumination changes, texture sparsity, motion blur, and shadows [14]. Moreover, most visual navigation systems assume a static scene; operating in a dynamic environment with frequently changing or moving objects presents a significant challenge. Lastly, computer vision algorithms are often computationally expensive due to the intensive processing required for feature extraction and image matching [14].

Chapter 3

LiDAR sensor

This chapter provides an introductory overview of the technology associated with LiDAR imaging systems and its current state of development [16].

The term LiDAR (Light Detection and Ranging) first appeared in 1963, created as a mix of the words "light" and "radar" [16]. Initially, its applications were concentrated in military and aerospace fields, with development primarily focused on remote sensing [16]. Today, LiDAR technology has matured significantly, largely driven by the recent surge in development toward autonomous and robotic vehicles [16].

3.1 Basic of LiDAR Imaging

The fundamental measurement principle utilized for LiDAR imaging is Time-of-Flight (TOF) [16]. This principle measures depth by calculating the time delay between the emission of light from a source and its return to a detector [16]. Consequently, LiDAR is an active, non-contact range-finding technique [16]. An optical signal is projected onto an object (the target), and the reflected or back-scattered signal is detected and processed to determine the distance [16]. This process enables the creation of a 3D point cloud representing a segment of the unit's environment [16]. Thus, the range R, or distance to the target, is derived from the light waves' round-trip delay time to the target [16].

LiDAR systems employ several different measurement principles [16]. The most straightforward is the Pulsed approach (Subsection 3.1.1.1), where a short light pulse is emitted, and the arrival time of its echo at the detector sets the distance [16]. A second common approach is based on Amplitude Modulation of a Continuous Wave (AMCW) (Subsection 3.1.1.2), which measures distance by comparing the phase difference between the emitted and the backscattered detected waves [16]. Because AMCW emission is continuous, the reflected signal is less strong than in the pulsed case, allowing the amplitude to remain below eye-safe limits at all times [16]. Finally, a third technique, Frequency-Modulated Continuous-Wave (FMCW) (Subsection 3.1.1.3), relies on the direct modulation and demodulation of signals in

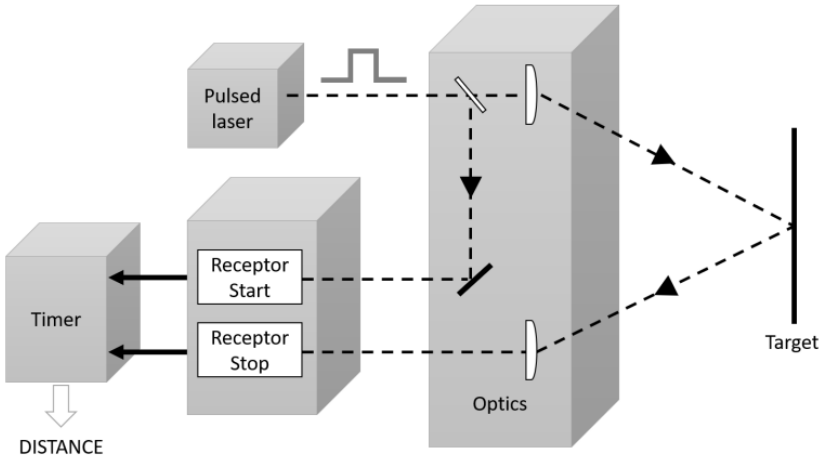


Figure 3.1: Pulsed time-of-flight (TOF) measurement principle [16].

the frequency domain, enabling distance detection through the coherent superposition of the emitted and detected waves [16].

3.1.1 Measurement Principles

3.1.1.1 Pulsed Approach

Pulsed TOF techniques use the simplest form of illumination modulation: distance is determined by multiplying the speed of light in the medium by the total time the light pulse takes to travel to the target and back [16]. Since the speed of light is constant in a given optical medium, the distance to the object is directly proportional to the traveled time [16].

The measured time represents twice the distance to the object (the light travels forth and back) and, therefore, must be halved to provide the actual range value to the target [16]:

$$R = \frac{c}{2} t_{oF}$$

where R is the range to the target, c is the speed of light ($c = 3x10^8 m/s$) in free space, and t_{oF} is the total time it takes for the pulse of energy to travel from its emitter to the observed object and then back to the receiver [16]. Figure 3.1 shows a simplified diagram of a typical implementation.

The attainable resolution in range (ΔR_{min}) is directly proportional to the available resolution in time counting (Δt_{min}) [16]. Consequently, the resolution in depth measurement is fundamentally limited by the precision of the time counting electronics [16]. A common depth resolution value is approximately 1.5 cm, based on a time interval measurement resolution in the 0.1 ns range, which is constrained by jitter and noise in the electronics [16].

Theoretically, the maximum range attainable (R_{max}) is limited only by the maxi-

mum time interval (t_{max}) the counter can measure [16]. In practice, however, this time interval is large enough that the maximum range is limited instead by the signal-to-noise ratio (SNR) due to laser energy losses during travel [16].

The pulsed principle involves the direct measurement of the round trip time between the emission of the light pulse and the return of the echo back-scattered from a target object [16]. Therefore, the pulses must be as short as possible (typically a few nanoseconds) with fast rise and fall times and high optical power [16]. Because the pulse irradiance power significantly exceeds the background (ambient) irradiance power, this method performs well outdoors under adverse environmental conditions and is suitable for longer distance measurements (it is also effective indoors, where the lack of solar background reduces power requirements) [16].

Advantages and Disadvantages The advantages of the pulsed approach include its simple measurement principle (direct TOF measurement), its long ambiguity distance, and the limited influence of background illumination due to the use of high-energy laser pulses [16]. However, its performance is limited by the signal-to-noise ratio (SNR) [16]. High laser energy is required while simultaneously adhering to strict eye-safety limits, necessitating the use of highly sensitive detectors, which can be expensive depending on the required detection range [16].

3.1.1.2 AMCW Approach

The Continuous Wave Amplitude Modulated (AMCW) approach utilizes the intensity modulation of a continuous lightwave instead of the discrete laser pulses used previously [16]. This principle is also known as CW modulation, phase-measurement, or amplitude-modulated continuous-wave [16]. The range value is obtained by measuring the phase-shift ($\Delta\phi$) induced in the intensity-modulated periodic signal during its round-trip journey to the target [16]. The optical power is modulated with a constant frequency f_M, typically in the tens of megahertz range, resulting in the emitted beam being a sinusoidal or square wave of frequency f_M [16]. The measurement of the distance R is thus derived from the phase shift $\Delta\phi$ between the reflected and the emitted signal [16]:

$$\Delta\Phi = k_M d = \frac{2\pi f_M}{c} 2R \longrightarrow R = \frac{c}{2}\frac{\Delta\Phi}{2\pi f_M}$$

where R is the range to the target, c is the speed of light in free space, k_M is the wavenumber associated with the modulation frequency, d is the total distance traveled, and f_M is the modulation frequency of the signal's amplitude [16]. Figure 3.2 illustrates the schematics of a conventional AMCW sensor.

Various techniques can be employed to demodulate the received signal and extract the phase information; for example, phase measurement can be achieved using signal processing techniques involving mixers and low-pass filters [16].

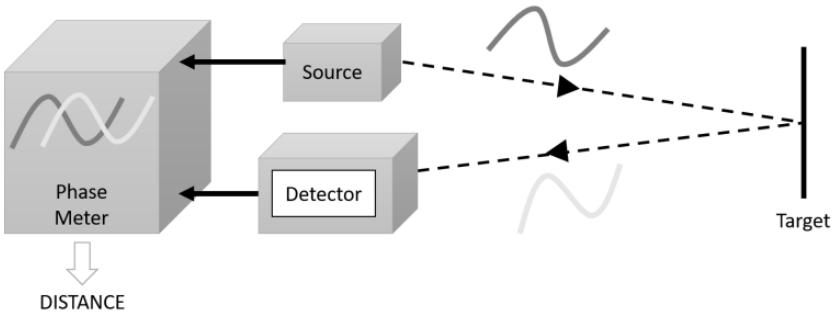


Figure 3.2: TOF phase-measurement principle used in amplitude modulation of a continous wave (AMCW) sensor [16].

In the AMCW approach, the range resolution is determined by the frequency f_M of the actual ranging signal (which can be adjusted) and the resolution of the phase meter, which is fixed by the electronics [16]. Increasing f_M also increases the resolution [16]. However, higher f_M frequencies lead to shorter unambiguous range measurements, meaning the phase value of the return signal begins to repeat itself after a 2π phase displacement [16]. Additionally, although phase measurement may be coherent in certain domains, the sensitivity of the technique remains constrained by the limited sensitivity of direct detection in the optical domain [16]. From the perspective of the Signal-to-Noise Ratio (SNR), a relatively long integration time is required over multiple time periods to achieve an acceptable signal rate, which, in turn, can introduce motion blur when dealing with moving objects [16].

In short, AMCW sensors are a simple and affordable option for indoor distance measurements, but they suffer from phase ambiguity and sensitivity limitations [16].

3.1.1.3 FMCW Approach

In the Continuous Wave Frequency Modulated (FMCW) approach, the emitted instantaneous optical frequency is periodically shifted, usually by varying the power applied to the source [16]. The source is typically a diode laser to enable coherent detection [16]. The reflected signal, which arrives at the receiver after a traveled time t_{OF}, is then mixed with a reference signal built from the emitter's output [16]. The resulting mixture creates a beat frequency (f_r) that is a measure of the probed distance [16]. For a static target, the delay between the collected light and the reference signal causes a constant frequency difference f_r in the mixed beams [16]:

$$f_r = slope \cdot \Delta\tau = \frac{B}{T} t_{oF} = \frac{B}{T} \frac{2R}{c} \longrightarrow R = f_r \frac{cT}{2B}$$

where B is the bandwidth of the frequency sweep, T denotes the period of the ramp, and $\Delta\tau$ equals the total traveled time t_{oF} [16]. Figure 3.3 depicts these main parameters.

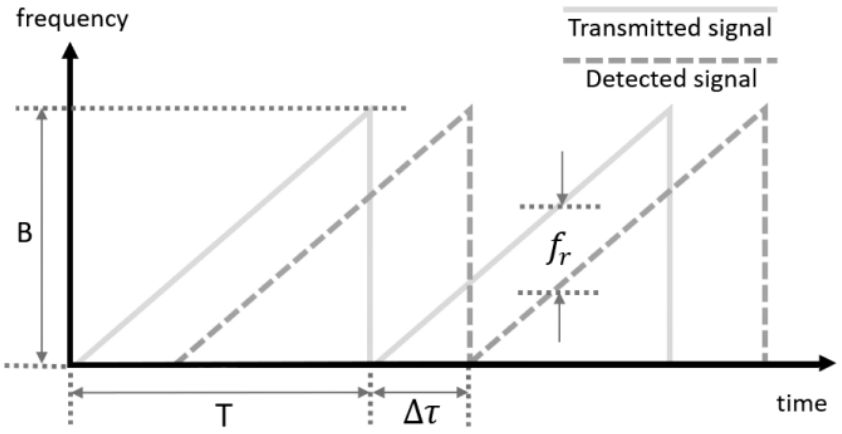


Figure 3.3: Frequency modulation and detection in the frequency-modulated continuous-wave (FMCW) method: main parameters involved [16].

In practice, the frequency difference between the outgoing and incoming components is translated into a periodic phase difference, which causes an alternating constructive and destructive interference pattern at the frequency f_r, resulting in a beat signal [16]. By employing the Fast Fourier Transform (FFT), the peak of the beat frequency can be easily translated into a distance measurement [16]. A triangular frequency modulation is typically used instead of a simple ramp because this detection method offers the significant advantage of simultaneously measuring not only the range but also the velocity and sign of the target [16].

The resolution of the FMCW technique is directly related to the total bandwidth (B) of the signal [16]. Since the ramp period T can be chosen arbitrarily, the FMCW method is capable of determining t_{oF} values in the picosecond range, equivalent to millimeter distances, by performing measurements in the kilohertz regime, which is perfectly feasible [16]. Unfortunately, it is generally difficult to realize a perfect linear or triangular optical frequency sweep by linearly modulating the control current, and the frequency-current curve is often nonlinear, especially near the slope change points [16].

	Pulsed	**AMCW**	**FMCW**
Parameter measured	Intensity of emitted and received pulse	Phase of modulated amplitude	relative beat of modulated frequency and Doppler shift
Measurement	Direct	Indirect	Indirect
Detection	Incoherent	Incoherent	Coherent
Use	Indoor/Outdoor	Only indoor	Indoor/Outdoor
Main advantage	Simplicity of setup; long ambiguity range	Established commercially	Simultaneous speed and range measures

Main limitation	Low SNR of returned pulse	Short ambiguity distance	Coherence length/ Stability in operating conditions (e.g.,thermal)
Depth resolution	1 cm	1 cm	0.1 cm

Table 3.1: Summary of Measurement Principles [16]

The FMCW technique, though more complex to implement, provides superior resolution and simultaneous velocity estimation, making it attractive for advanced LiDAR systems [16].

3.1.2 Imaging Strategies

The three main measurement strategies used in LiDAR imaging systems presented previously are inherently pointwise measurements [16]. However, useful LiDAR outputs are always 3D point clouds, which are necessary to accurately represent fields of view (FOV) as large as 360° around the object of interest [16]. A variety of strategies have been proposed to construct these LiDAR images from the repetition of single point measurements, but they can generally be grouped into three families: scanning components of different types, detector arrays, and mixed approaches [16].

3.1.2.1 Scanners

In the most common approach, a scanning element is used to reposition the laser spot on the target by modifying the angular direction of the outgoing beam, thereby generating a point cloud of the scene [16]. Numerous scanning strategies are found in commercial LiDAR products, which can be categorized into three main types [16]:

- **Mechanical scanners**
  LiDAR imaging systems based on mechanical scanners utilize high-grade optics and some type of rotating or galvanometric assembly, typically involving mirrors or prisms attached to mechanical actuators, to cover a wide FOV [16]. In these LiDAR units, the sources and detectors often rotate jointly around a single axis [16]. This process may involve sequentially pointing the beam across the target in 2D or rotating the entire optical configuration around a mechanical axis [16]. Mechanical scanners nearly always employ pulsed sources and are typically large and bulky [16]. When using rotating mirrors, they can achieve high spatial resolution in the direction of rotation (usually horizontal) but are often limited in the orthogonal direction (usually vertical), where the point cloud density depends on the number of parallel sources and detectors [16]. They are, however, highly effective for long-range applications [16].

- **Microelectromechanical scanners**
  Microelectromechanical systems (MEMS)-based LiDAR scanners enable programmable control of the laser beam position using tiny mirrors, only a few millimeters in diameter [16]. The tilt angle of these mirrors changes when a stimulus is applied, modifying the angular direction of the incident beam to direct the light to a specific point in the scene [16]. In LiDAR applications, the stimulus is most commonly voltage; the mirrors are steered by drive voltages generated from a digital representation of the scan pattern stored in memory [16]. These digital numbers are then converted to analog voltages using a digital-to-analog converter [16].

- **Optical Phased Arrays**
  An Optical Phased Array (OPA) is a novel, solid-state device that enables beam steering using a multiplicity of micro-structured waveguides [16]. Its operational principle is equivalent to that of microwave phased arrays: the beam direction is controlled by tuning the phase relationship between arrays of coherent transmitter antennas [16]. By aligning the phase of multiple coherent emitters, the emitted light constructively interferes in the far-field at specific angles, allowing the beam to be steered electronically [16]. OPAs offer very stable, rapid, and precise beam steering [16]. Because they contain no mechanical moving parts, they are robust, insensitive to external constraints like acceleration, highly compact, and can be integrated onto a single chip [16].

3.1.2.2 Detector Arrays

Due to the drawbacks of scanning approaches that rely on macroscopic moving elements, alternative imaging methods have been proposed to overcome their limitations, moving beyond MEMS scanners and OPAs [16]. These scannerless techniques typically combine specialized illumination strategies with arrays of receivers [16]. Transmitting optical elements flood-illuminate an entire scene, and a linear or matrix array of detectors receives the signals from separate angular subsections in parallel [16]. This parallelization allows range data of the target to be obtained in a single-shot, which greatly simplifies real-time applications [16].

The scene illumination used in these array systems may be pulsed (flash imagers) or continuous (AMCW or FMCW LiDARs) [16]:

- **Flash imagers**
  In a flash LiDAR, imaging is obtained by flood-illuminating the target scene, or a portion thereof, using a pulsed light source [16]. The backscattered light is collected by the receiver, which is divided among multiple detectors [16]. Each detector captures the image distance, and sometimes the reflected intensity, using the conventional time-of-flight principle [16].

- **AMCW cameras**

A second family of LiDAR imaging systems using detector arrays are the TOF cameras based on the AMCW measuring principle [16]. As previously explained in Subsection 3.1.1.2, these devices modulate the intensity of the source and then measure the phase difference between the emitter and the detected signal at each pixel on the detector [16]. This measurement is achieved by sampling the returned signal at least four times [16]. Detectors for these cameras are commonly manufactured using standard CMOS technology, making them small and low-cost [16].

Some successful LiDAR imagers have employed mixed imaging modalities, combining a scanning approach with a multiple detector arrangement [16]. The cylindrical geometry achieved by a single rotating axis paired with a vertical array of emitters and detectors has proven particularly successful [16].

3.2 Sources and Detectors for LiDAR Imaging Systems

A LiDAR system's core function is to illuminate a scene and use the returning backscattered signal for range-sensing and imaging [16]. Therefore, the basic system must include a light source (transmitter), a sensitive photodetector (receiver), the data processing electronics, and a strategy for acquiring information across the target area [16]. These components are essential for creating 3D maps and proximity data images [16]. Data processing and device operation are typically managed by combinations of Field Programmable Gate Arrays (FPGAs), Digital Signal Processors (DSPs), microcontrollers, or dedicated computers, depending on the complexity of the system architecture [16].

Given the strict requirements for frame rate, spatial resolution, and Signal-to-Noise Ratio (SNR) in LiDAR imaging, the light sources and photodetectors are critical, state-of-the-art components undergoing intensive development [16]. The overall performance of a LiDAR setup is intrinsically tied to the trade-offs and performance limits of these two key elements [16].

3.2.1 Sources

LiDAR systems commonly employ laser sources operating in the infrared (IR) region to leverage the atmosphere's transmission windows, particularly where water absorption is low, and to use beams invisible to the human eye [16]. Sources are primarily concentrated in three wavelength regions [16]:

- A band from 0.8 µm to 0.95 µm, dominated by diode lasers.
- Lasers operating at 1.06 µm, often built with fiber-based sources, can still be used with standard Silicon (Si) detectors.
- Lasers at 1.55 µm, readily available from the telecom industry, which require Indium Gallium Arsenide (InGaAs) detectors.

Wavelength selection is generally based on cost and eye safety considerations [16].

Sources are typically based on lasers or nonlinear optical systems driven by lasers, although other sources, such as LEDs, are sometimes found in short-range applications [16]. Several performance features must be considered, including peak power, pulse repetition rate (PRR), pulse width, wavelength (purity and thermal drift), emission characteristics (single-mode, beam quality, CW/pulsed), size, weight, power consumption, shock resistance, and operating temperature [16]. These features involve trade-offs; for example, achieving a larger peak power often compromises a large PRR or spectral purity [16].

Currently, the most popular sources for LiDAR technology are solid-state lasers (SSL) and diode lasers (DLs) [16]. The following table summarizes their key characteristics.

	Fibre Laser	**Microchip Laser**	**Diode Laser**
Amplifying media	Doped optical fiber	Semiconductor crystal	Semiconductor PN junction
Peak power (typ)	$>$ 10 kW	$>$ 1 kW	0.1 kW
PRR	$<$ 1 MHz	$<$ 1 MHz	$\approx$ 100 kHz
Pulse width	$<$ 5 ns	$<$ 5 ns	100 ns
Main advantage	Pulse peak power, PRR, beam quality, Beam delivery	Pulse peak power, PRR, beam quality	Cost, compact
Main disadvantage	Cost	Cost, beam delivery	Max output power and PRR. Beam quality

Table 3.2: Summary of the main features of sources for LiDAR sensors [16]

3.2.2 Photodetectors

Photodetectors are the other essential component of a LiDAR system, serving as the critical photon-sensing device in the active receiver that enables the Time-of-Flight (TOF) measurement [16]. A detector must possess high sensitivity for direct intensity detection and sufficient bandwidth to detect short pulses [16]. The material composition of the detector dictates its sensitivity to the specific wavelength of interest [16].

Light detection in LiDAR imaging systems typically relies on five different types of detectors [16]:

- PIN diodes: These are the most commonly used single-detectors. While they can be very fast in detecting light events if sensitive enough for the application, they do not provide internal gain; in optimal efficiency conditions, each photon generates only a single photoelectron [16].

- Avalanche photodiodes (APD): APDs are the most useful receivers for applications requiring moderate to high sensitivity, as their structure provides a certain level of multiplication of the current generated by incident light [16].
- Single-photon avalanche photodiodes (SPAD): SPADs are essentially APDs biased beyond their breakdown voltage, with their internal structure designed to withstand repetitive, large avalanche events [16]. Unlike APDs, where a single photon produces tens to hundreds of electrons, a SPAD produces a large electron avalanche of thousands of electrons, resulting in detectable photocurrents [16].
- Multi-pixel photon counters (MPPC): MPPCs are a recent, interesting arrangement of SPADs. They are pixelated devices formed by an array of SPADs where all pixel outputs are summed into a single analog output, effectively enabling photon-counting by measuring the combined intensity of the fired pixels [16].
- Photomultiplier tubes (PMT): PMTs operate based on the external photoelectric effect [16]. Electrons are emitted within a vacuum tube and collide with cascaded dynodes, resulting in a true electron avalanche and high gain [16].

Chapter 4

ROS: Robot Operating System

4.1 Overview

The Robot Operating System (ROS) is an open-source, widely adopted middleware platform for developing robotic applications [17]. Its official definition describes it as [18]:

"ROS is an open-source, meta-operating system for your robot. It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management."

ROS was initially created over a decade ago at Stanford and has since been maintained by Willow Garage and the Open Source Robotics Foundation (OSRF) [17]. It provides numerous libraries and tools that significantly aid software developers in creating complex robotics applications [17].

One of ROS's main strengths is how it simplifies system integration. In complex robots, managing different sensors—each with its own data format—can quickly become cumbersome [17]. ROS helps by providing a consistent data structure and many ready-to-use packages for common sensors, reducing the effort needed to process and analyze data. It also makes it easier to send control commands from software to the hardware motors of most mobile robots [17].

Due to the increasing demands of real-time performance and deployment on distributed embedded systems, ROS underwent a substantial re-write, leading to ROS 2 [17]. The older version, ROS 1, is still used in many legacy projects, but the community is migrating toward ROS 2, which was released in 2017 to solve the structural limitations of its predecessor [19].

As a meta-operating system, ROS still depends on an underlying host OS [19]. ROS 1 was limited to Linux systems, while ROS 2 expanded support to other platforms, including Windows and iOS [19]. From a programming point of view,

Figure 4.1: ROS logo [20]

ROS follows an Object-Oriented approach, and most of its code is written in C++ or Python [19]. Using ROS tools usually involves extensive interaction through the Command Line Interface (CLI) [19].

4.2 Graph Concepts

The foundation of a ROS 2 system is the ROS graph, defined as "the network of nodes in a ROS system and the connections between them by which they communicate" [21]. The graph is built from the following conceptual elements:

- **Nodes**: Independent computing processes that communicate with other nodes using client libraries.
- **Topics**: Unilateral communication channels used by nodes to publish or subscribe to messages.
- **Messages**: Simple data packets that nodes exchange with each other through topics.
- **Discovery**: The automatic mechanism by which nodes locate each other and establish initial connections.

Nodes

A node in the ROS graph is an independent computing process that leverages ROS client libraries to communicate with other processes [22]. An application, whether a single executable or multiple processes across different machines, can define one or more nodes [22]. Ideally, each node should be responsible for a single, modular purpose (e.g., a node to control wheel motors, another for reading laser range-finder data) [22]. Nodes communicate primarily through Topics (publisher-subscriber model), but ROS also provides Services and Actions for other communication patterns [22]. Additionally, nodes can be associated with configurable Parameters [22].

Messages and Communication Interfaces

Communication inside ROS is based on a few interface types: messages, services, and actions. Each uses a simple data format defined with an Interface Definition Language (IDL), from which ROS automatically generates code in different languages (such as C++ or Python) or DDS types [22].

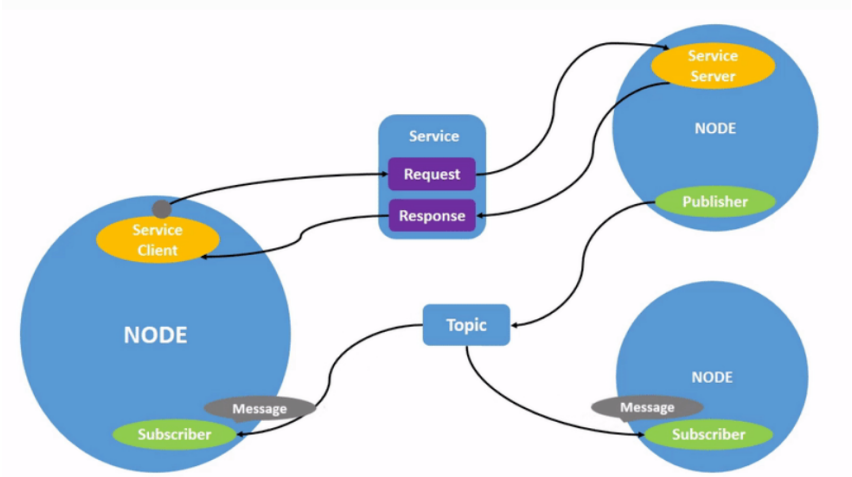


Figure 4.2: Nodes communication example in ROS 2 [22].

- **Messages**: These are simple data structures defined in a .msg file and exchanged via Topics [22]. A Topic is a unilateral channel: only one node can publish messages to it, but multiple subscribed nodes can receive the data stream [22].
- **Services**: Providing a request/response model, services offer a bilateral communication alternative to the publisher-subscriber pattern of topics [22]. Defined in a .srv file, the interface contains two parts: a request message and a response message. A server node offers the service, and a client node sends a request, triggering the server to compute and return a response [22].
- **Actions**: Intended for long-running tasks, actions are a communication type in ROS 2 that uses a client-server model similar to services but with the added feature of publishing a data stream of feedback [22]. Defined in a .action file, the structure comprises three message declarations: a goal, feedback, and a final result. The action client sends a goal to the action server, which acknowledges it and continuously publishes feedback until a final result is returned [22].

Discovery

The discovery of nodes in ROS 2 happens automatically via the underlying middleware and the process generally consists of three phases during a node's life cycle [22]:

- Startup: A node announces its presence on the network.
- Connection: Other nodes respond with the necessary information to establish appropriate communications through the correct interfaces.
- Execution: Nodes regularly announce their presence to the network, allowing new components to discover and connect to them.
- Shutdown: A node advertises its detachment from the graph before termination [22].

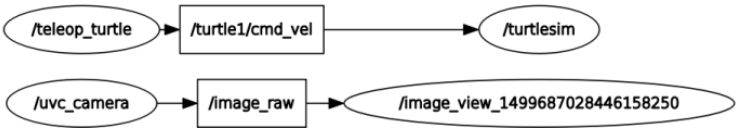


Figure 4.3: Example of rqt_graph [23].

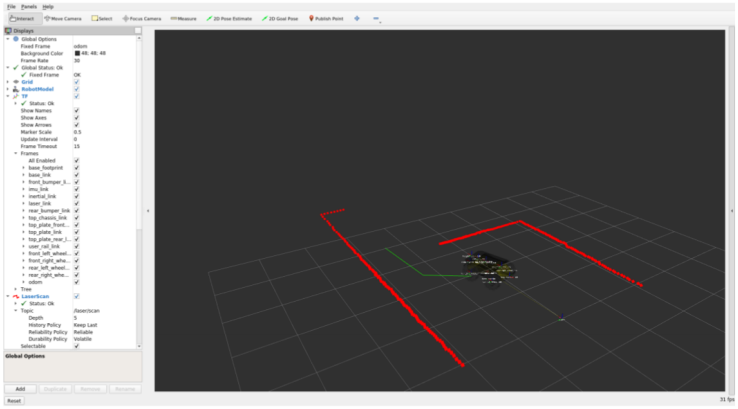


Figure 4.4: Example of RViz window [24].

4.3 ROS Useful Tools

ROS provides several tools for system analysis, debugging, and visualization, offering users feedback on the system's status [23].In this thesis, I primarly used:

- rqt_graph: This tool provides a graphical representation of the correlation between active processes (nodes and topics) during execution [23]. It is invaluable for debugging, allowing quick visualization of which nodes are acting as publishers and which are acting as subscribers, and ensuring correct operation [23].

- RViz: RViz is a powerful 3D visualization tool that can display any data published by the software through its topics [24]. It can visualize laser distance measurements (as points), Point Cloud Data from 3D sensors, camera imagery, and the robot's model and planned paths [24]. By representing the robot's point of view, RViz makes it easy to visualize what the robot is sensing and how it is interacting with the environment [24].

Chapter 5

State-of-the-art of LiDAR-Only Odometry algorithms

LiDAR sensors, alongside cameras, constitute one of the predominant sensing modalities employed within the fields of robotics and computer vision [25]. As previously explained in Chapter 3, LiDAR technology is founded upon the principle of active laser transmission; it sends out laser light pulses and subsequently measures the time delay of those pulses after they reflect off surfaces and return to the detector [25]. This process yields a direct measurement of the distance to those surfaces [25]. Consequently, a LiDAR sensor can be leveraged both to perceive the structural layout of its surrounding environment and to accurately estimate the sensor's own motion, or ego-location [25].

5.1 Introduction

Odometry represents a truly critical element of robot navigation, especially in operating environments where global positioning methods, such as the Global Positioning System (GPS), are either unavailable or unreliable [2]. Diverse sensors, including wheel encoders, Inertial Measurement Units (IMU), cameras, and LiDAR, are commonly integrated to perform odometry in robotics [2]. LiDAR, in particular, has garnered substantial attention because it furnishes rich three-dimensional (3D) data while remaining inherently immune to variations in ambient light [2].

The history of odometry in robotics shows a marked evolution and influential publications [2]. Initial approaches relied heavily on wheel encoders and dead reckoning; however, the accuracy of wheel odometry was intrinsically limited by sensor errors arising from wheel slippage and inherent algorithmic inaccuracies [2]. During this foundational phase, studies began to emerge focusing on obtaining odometry via range sensors, paralleled by advancements in scan registration algorithms, most notably the Iterative Closest Point (ICP) [2]. Subsequently, range sensor technology matured, and 3D LiDAR emerged as a transformative force capable of measuring the surrounding space in three dimensions, thereby vastly surpassing the capabilities of

traditional 2D measurements [2]. Recognizing that precise location data is paramount for autonomous robots' decision-making processes, researchers decisively shifted their focus toward LiDAR, which scans the surroundings in 3D without being affected by external lighting conditions [2].

5.2 LIDAR Odometry

The core objective of LiDAR odometry is to precisely estimate the incremental ego-motion of a robot or vehicle in real-time [25]. This estimation utilizes the current LiDAR scan alongside past observations, which may include a single previous scan or multiple scans aggregated into a local map [25]. A single scan typically encapsulates one complete rotation or full sweep of the sensor, thus providing a contextual snapshot of the surrounding environment at a specific moment [25]. At the very heart of LiDAR odometry is the technique of scan registration, also frequently termed scan matching [25]. Scan registration involves the meticulous alignment of a pair of scans to determine the precise relative transformation between them [25]. Since a scan fundamentally constitutes a set of points, or a point cloud, the well-established Iterative Closest Point (ICP) algorithm serves as a foundational technique for point cloud registration, capable of determining this relative transformation [25].

5.2.1 Foundations of Scan Registration

Scan registration constitutes a fundamental component of both LiDAR odometry and mapping systems, necessary for accurate alignment and environment mapping [25]. The objective is to identify the optimal transformation—specifically, the rotation $\mathbf{R} \in \Re^{3x3}$ and the translation $\mathbf{t} \in \Re^3$—that best brings the source scan (e.g., the most recent sensor reading) into alignment with the target scan (e.g., a local map or the previous scan) [25].

Iterative closest Point (ICP). A point cloud P is formally defined as a set of points existing within a three-dimensional coordinate system, mathematically represented as $P = p_i \in \Re^{3x3} \mid i = 1, 2, ..., N$, where each $p_i = (x_i, y_i, z_i)$ specifies the 3D coordinates of an individual point [25].

In the context of scan registration, the ICP algorithm is designed to minimize the total registration error between two point clouds, denoted P (source) and Q (target) [25]. ICP operates iteratively, determining the transformations $(\mathbf{R}^k, \mathbf{t}^k)$ for an optimization iteration k that will minimize the overall registration error [25]:

$$\mathbf{R}^k, \mathbf{t}^k = arg\ min_{\mathbf{R},\mathbf{t}} \sum_{(\mathbf{p},\mathbf{q})\ \in\ C} d(\mathbf{p}_i, \mathbf{R}\mathbf{q}_i + \mathbf{t})$$

Here, the set of correspondence C establishes the necessary links between points

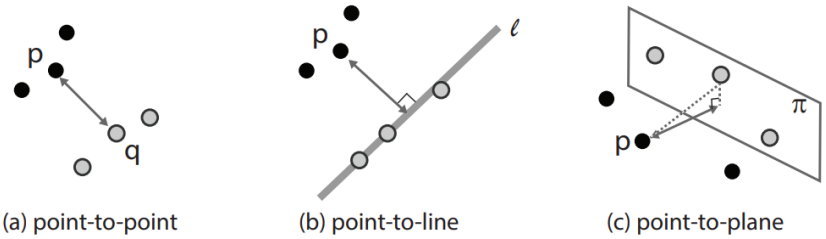


Figure 5.1: Typical distance metrics used in ICP. (a) Point-to-point distance is a straight-forward as the Euclidean distance between two points. (b) and (c) The point-to-higher-level feature (e.g. line or plane) is calculated as the shortest distance to the reconstructed line or plane using the target points [25].

in the source point cloud P and points in the target point cloud Q:

$$C = \{(\mathbf{p}, \mathbf{q}) \mid \mathbf{p} \in P, \mathbf{q} \in Q\}\,.$$

Within the ICP algorithm, the determination of the optimal transformation between the source and target is achieved iteratively by recomputing a new set of correspondences C in each subsequent iteration k [25]. This recomputation relies on the last determined transformation, given by the rotation $\mathbf{R}^{k-1}$ and translation $\mathbf{t}^{k-1}$, from the previous iteration k1 [25].

5.2.2 Distance measure in Registration Residual

The first essential component of an ICP implementation involves determining the specific geometric elements that will be used to define the error residual [25]. Points, lines, and planes are the most frequently chosen geometric features, as summarized in Figure 5.1. Point-to-point ICP represents the most basic implementation, minimizing the Euclidean distance between the corresponding points in the two clouds [25]. While this point-to-point cost function is simple and easy to implement, it can be highly susceptible to noise and outliers present in the data [25].

Distances to lines are also used as an error metric [25]. The point-to-line distance measures how far each point in the source cloud is from nearby linear features reconstructed in the target cloud [25]. This strategy often yields better results, especially in environments with clear structural patterns and strong linear geometry [25].

Distance can also be measured between points in the source cloud and planes (local surfaces) identified within the target point cloud [25]. This point-to-plane approach is generally more robust against noise and is capable of achieving higher alignment accuracy when operating in environments that feature numerous planar surfaces [25].

5.2.3 Determining Correspondences

The second core component common to all ICP algorithms is the process of data association, or the correspondence search, linking points between the source P and the target Q [25]. In its most elementary form, determining the correspondences between P and Q is achieved geometrically by finding the nearest neighbor q in the target Q for every point p in the source P [25]. This search specifically utilizes the iteratively updated transformation $(\mathbf{R}^{k-1}, \mathbf{t}^{k-1})$ and is formally expressed as:

$$C = \left\{ (\mathbf{p}, \mathbf{q}) \mid \mathbf{p} \in P, \mathbf{q} = arg\ min_{\mathbf{q}' \in Q} \left\| \mathbf{p} - \left(\mathbf{R}^{k-1}\mathbf{q}' + \mathbf{t}^{k-1} \right) \right\|_2 \right\}$$

However, executing this nearest-neighbor operation is typically computationally expensive [25]. To ensure that LiDAR odometry can operate in real-time, two widely adopted strategies are employed to mitigate the correspondence search time: the first involves reducing the set of potential candidates for a correspondence, and the second entails utilizing a search strategy fundamentally different from the traditional distance-based nearest neighbor approach [25].

5.3 Direct matching approach

As previously established, LiDAR-only odometry functions by meticulously analyzing successive LiDAR scans through the process of scan matching to ascertain a robot's current position [2]. LiDAR-only odometry methodologies can be broadly categorized into three distinct types, primarily based on the specific technique employed for scan matching: direct matching, feature-based matching, and deep learning-based matching [2]. Figure 5.2 illustrates the overarching summary of the literature concerning LiDAR-only odometry approaches.

The direct matching method represents the most straightforward approach in LiDAR-only odometry, as it explicitly computes the geometric transformation that aligns two consecutive scans [2]. Among the available techniques, the Iterative Closest Point (ICP) algorithm remains one of the most widely adopted solutions [2]. ICP iteratively minimizes an error metric—typically the sum of squared distances between corresponding points—until convergence [2]. The resulting transformations between successive scans are then used to estimate the robot's trajectory [2].

Notwithstanding its fundamental role, the ICP algorithm is known to have certain limitations; notably, it exhibits susceptibility to falling into local minima during optimization, a characteristic that makes a reliable initial pose guess crucial for success [2]. Furthermore, the algorithm is sensitive to the presence of noise, which includes data artifacts generated by dynamic objects within the scene [2]. Additionally, the inherent iterative nature of ICP can lead to substantial computational expense, potentially resulting in prohibitively slow computation speed in real-time applications. Consequently, considerable research effort has been focused on enhancing the

	Method	Year	Contributions
Direct	ICP [9]	1992	iteratively calculate closest point with point-to-point distance
	Chen and Medioni [25]	1992	point-to-plane ICP
	TrICP [26]	2002	improves ICP with trimmed squared method
	NDT [10]	2003	leverages normal distribution for registration
	Generalized-ICP [122]	2009	integrates point-to-point ICP and point-to-plane ICP
	NICP [123]	2015	extends Generalized-ICP by incorporating surface normals
	Hong and Lee [53]	2017	introduce probabilistic NDT representation
	IMLS-SLAM [32]	2018	IMLS representation for scan-to-map matching
	LiTAMIN [156], LiTAMIN2 [157]	2021	faster registration and modified cost function using KL divergence
	DLO [20]	2022	scan-to-map matching with selected keyframes using convex hull
	CT-ICP [31]	2022	interpolates the positions for continuous trajectory
	KISS-ICP [137]	2023	point-to-point ICP with adaptive thresholding
Feature	LOAM [162, 163]	2014	extract edge and planar feature points for registration
	LeGO-LOAM [125]	2018	leverages ground segmentation within LOAM framework
	SuMa [7]	2018	utilizes surface normals from surfel-based map
	SuMa++ [24]	2019	performs semantic ICP with semantic labels from RangeNet++ [95]
	F-LOAM [139]	2021	emphasizes horizontal features to minimize false detections
	Zhou et al. [176], π-LSAM [177]	2021	introduces plane adjustment in indoor situations
	MULLS [104]	2021	scan-to-map multi-metric linear least square ICP
	NDT-LOAM [22]	2021	combines weighted NDT and feature-based pose refinement
	E-LOAM [45]	2022	performs D2D-NDT with geometric and intensity features
	R-LOAM [101], RO-LOAM [102]	2022	extracts 3D triangular mesh features from reference object
	Wang et al. [141]	2022	coarse-to-fine odometry with NDT and PLICP
	VoxelMap [160]	2022	leverages probabilistic plane representation and adaptive voxel construction
Deep	LO-Net [80]	2019	scan-to-scan LiDAR odometry network
	LodoNet [173]	2020	select MKPs for odometry estimation
	Cho et al. [27]	2020	unsupervised learning with VertexNet and PoseNet

Figure 5.2: The overarching summary of LiDAR-only odometry. Direct, Feature, and Deep represent Direct, Feature-based, and Deep Learning-based matching each [2].

practical performance of the ICP algorithm to achieve improved odometry results [2].

As will be demonstrated in subsequent subsections, conventional LiDAR odometry typically calculates a discrete odometry estimate each time a new LiDAR point cloud is received [2]. Conversely, an increasing number of methods are now aiming to model a continuous trajectory, effectively emulating the smooth, continuous motion of an actual physical robot [2].

5.3.1 KISS-ICP

The KISS-ICP method proposes a notably simple yet highly resilient approach for incrementally estimating the trajectory of a moving LiDAR sensor, leveraging the established Iterative Closest Point (ICP) algorithm [4]. This strategy departs significantly from numerous contemporary odometry pipelines that frequently necessitate intricate feature extraction or intensive optimization routines [4]. Instead, KISS-ICP prioritizes both reliability and operational simplicity by judiciously integrating a compact set of well-conceived techniques that collectively enhance its stability and final pose accuracy. Key components incorporated into this framework include robust motion prediction, dedicated motion compensation (deskewing), efficient voxel-based subsampling, a sophisticated adaptive mechanism for correspondence association, and ultimately, a robust optimization framework [4].

The methodology is meticulously structured around four principal phases:

Motion Prediction and Scan Deskewing

To effectively counteract the geometric distortions inherent to the LiDAR acquisition process—distortions introduced by the sensor's continuous movement during the scan—KISS-ICP implements a constant velocity motion model [4]. This model is utilized to predict the robot's forthcoming pose based upon an analysis of its two most recent pose estimations. By employing this calculated model, the algorithm can derive both linear (v_t) and angular (ω_t) velocities, subsequently applying a temporal correction to each individual LiDAR point according to its relative acquisition timestamp. This critical process yields a deskewed point cloud, ensuring the geometric integrity of the data by eliminating motion-induced deformation [4].

Point Cloud Subsampling

Instead of relying on computationally heavy processes for the extraction of specific geometric features, KISS-ICP opts for a voxel-based downsampling of the point cloud [4]. This choice serves to drastically improve computational efficiency without sacrificing the underlying spatial structure crucial for registration quality. The process involves overlaying a 3D voxel grid onto the cloud, with each voxel constrained to retain only a limited number of representative points [4]. This necessary subsampling is strategically executed in a dual-stage manner, controlled by parameters α and β, distinguishing between the required filtering for local map updates and the filtering specifically optimized for the subsequent ICP registration step. This dual-filtering approach effectively balances processing speed with the preservation of alignment quality [4].

Local Map and Adaptive Correspondence Estimation

Following the deskewing and downsampling procedures, each processed scan is meticulously registered against a local map, which is itself dynamically constructed from the integration of multiple previous scans [4]. Point-to-point correspondences are initially identified via a nearest-neighbor search within the established voxel grid, but are then rigorously filtered using an adaptive threshold τ [4]. Crucially, this threshold is not static; rather, τ is determined dynamically by estimating the pose deviation (ΔT) observed between the predicted motion and the corrected pose during previous ICP iterations. This intelligent, data-driven mechanism ensures the robust rejection of outliers and maintains the algorithm's stability even when faced with sensor noise or sudden, abrupt changes in motion [4].

Alignment Through Robust Optimization

The final stage is dedicated to refining the alignment by calculating the corrective transformation ΔT_{icp} through a robust point-to-point ICP optimization [4]. The residual errors existing between the corresponding points are minimized using the

German–McClure robust kernel; this specific kernel is selected for its capacity to significantly limit the influence of severe outliers on the optimization process [4]. The algorithm continuously updates the overall global pose T_t by effectively compounding the predicted motion $T_{pred,t}$ with the calculated ICP correction $\Delta T_{icp,t}$ [4]. This iteration continues until the magnitude of the applied correction drops below a defined convergence threshold γ, notably without imposing any fixed or predetermined limit on the number of optimization iterations [4].

At its core, the design of KISS-ICP shows that focusing on a lean but carefully crafted processing pipeline can achieve both simplicity and precision in LiDAR odometry. This balance allows the algorithm to remain robust across diverse conditions while keeping computational costs remarkably low [4].

5.3.2 GenZ-ICP

he GenZ-ICP (Generalizable and Degeneracy-Robust LiDAR Odometry) framework introduces a methodology for LiDAR odometry that exhibits high generalizability and robustness against geometric degeneration conditions [3]. It overcomes the inherent limitations of conventional ICP approaches that rely on a single error metric [3]. Traditional systems can suffer significant performance degradation in geometrically poor contexts, such as long corridors or open spaces, because the efficacy of point-to-point and point-to-plane metrics varies considerably depending on the specific environmental structure [3].

GenZ-ICP tackles this challenge innovatively by jointly leveraging both error metrics within an adaptive, geometry-aware optimization scheme [3]. The algorithm dynamically balances the contribution of point-to-point and point-to-plane residuals using a weighting factor, α, which actively reflects the local planarity of the environment. This adaptive weighting allows the system to autonomously calibrate its behavior based on the context, thereby guaranteeing accurate and stable pose estimation across diverse scenarios [3].

Problem Formulation

The algorithmic objective is to estimate the optimal rigid-body transformation necessary for aligning consecutive LiDAR scans [3]. This goal is achieved by minimizing a cost function that explicitly incorporates the residuals of both metrics (point-to-plane and point-to-point) [3]. The adaptive weight α is the crucial element, as it determines the respective contribution of each residual based on the detected geometry [3].

Planarity Classification

Each individual correspondence is categorized as either planar or non-planar [3]. This distinction relies on analyzing the eigenvalue distribution of neighboring points, a method that quantifies the local surface variation using the expression $\frac{\lambda_3}{\lambda_1+\lambda_2+\lambda_3}$ [3]. If the calculated variation is below a defined threshold, the point is considered planar and utilizes the point-to-plane metric; otherwise, it is treated as non-planar and uses the point-to-point metric [3].

Residual and Jacobian Definition

Error residuals are defined differently for the two categories [3]:

- In planar regions, residuals are calculated as the dot product between the positional difference and the surface normal vector.
- In non-planar regions, the direct positional difference is used.

Both residuals are subsequently linearized to obtain their respective Jacobians, a step necessary for solving the least-squares optimization problem [3].

Adaptive Weighting Optimization

The weight α is automatically updated at every iteration, calculated as the ratio between the number of planar correspondences (N_{pl}) and the total number of correspondences ($N_{pl} + N_{po}$) [3]:

$$\alpha = \frac{N_{pl}}{N_{pl} + N_{po}}$$

This mechanism allows for seamless transitions between highly structured environments (where planar features like floors and walls dominate) and unstructured environments (such as vegetation or clutter) [3]. Consequently, the resulting cost function fuses the strengths of both error models, significantly elevating the accuracy and robustness in geometrically heterogeneous scenes [3]. Thanks to this adaptive mechanism, GenZ-ICP demonstrates notable generalizability and resistance to degeneration, maintaining high-precision pose estimation even in challenging or weakly constrained conditions [3].

5.3.3 MOLA-LO

MOLA-LO (MOLA LiDAR Odometry) functions as a central module within the broader MOLA (Modular Optimization for Localization and Mapping) framework, offering an open-source, flexible solution specifically designed for LiDAR-based motion estimation [5]. Developed using C++, the system places a strong emphasis on achieving configurability, modularity, and inherent robustness by integrating customizable point cloud processing pipelines with highly optimized variants of the

Iterative Closest Point (ICP) algorithm [5].

The operational sequence of MOLA-LO can be divided into the following stages:

Data Acquisition and Conditioning

MOLA-LO begins by processing raw 3D LiDAR point clouds [5]. To mitigate geometric distortions resulting from the sensor's motion during data capture, the system offers optional motion compensation (deskewing), provided that per-point timestamps are available [26]. Subsequent pre-processing steps, such as the removal of outliers and voxel grid downsampling, are then applied to simultaneously reduce noise and decrease the overall computational burden while critically preserving essential geometric features [26].

Local Map Maintenance

In a departure from simple scan-to-scan approaches, MOLA-LO actively maintains and utilizes a local map [5]. This map is compiled from previously integrated scans, thereby providing a robust and stable reference against which new scans can be accurately aligned. The local map undergoes dynamic updates as the robot traverses the environment, a feature that significantly enhances resilience in environments with sparse features. Furthermore, the system incorporates keyframe selection capabilities, facilitating the construction of sparser maps or supporting higher-level relocalization tasks [26].

Scan-to-Map Registration

An initial pose estimate for the registration process is first obtained through motion prediction (e.g., utilizing a constant velocity model or relying on the most recent odometry estimate) [5]. MOLA-LO then employs the specialized mp2p_icp filtering and ICP pipelines to align the current scan with the dynamic local map. This alignment is achieved by determining nearest neighbor correspondences and iteratively minimizing an error metric between the matched points [5].

The optimization routine seeks the rigid-body transformation (defined by rotation R and translation t) that minimizes the total alignment error E(R,t):

$$E(\mathbf{R},\mathbf{t}) = \sum_{i=1}^{N} ||\mathbf{R}\mathbf{p}_i + \mathbf{t} - \mathbf{q}_i||^2$$

Depending on the configuration, the core ICP process is highly flexible and may implement variants such as point-to-point, point-to-plane, Generalized ICP (GICP), or Normal Distributions Transform (NDT) [26]. The iterative optimization procedure continues until a defined convergence criterion is met, at which point the final

estimated transformation updates the robot's pose [5].

Pose Update and Covariance Estimation

The transformation obtained from ICP registration represents the relative motion between consecutive scans and is integrated over time to build the global trajectory [5]. In addition, MOLA-LO estimates the covariance matrix of the current pose, offering a quantitative expression of uncertainty. This uncertainty information is particularly useful when integrating MOLA-LO into higher-level systems such as SLAM or advanced sensor fusion frameworks [26].

System Output and Integration

The final 6-Degrees-of-Freedom (6-DOF) pose (position and orientation) is published in real-time, often utilizing ROS 2 interfaces, for immediate consumption by navigation and mapping systems [5]. MOLA-LO also furnishes dedicated visualization tools (e.g., mola_viz) that enable real-time monitoring of the robot's trajectory, the evolution of the local map, and the streaming LiDAR data [26].

In conclusion, MOLA-LO is a modular and high-performance LiDAR odometry framework designed for adaptability across different environments and system setups. Its main principles emphasize reliable motion estimation, reusability, and real-time compatibility, making it a strong foundation for broader localization and mapping pipelines [5, 26].

5.3.4 SiMpLe

SiMpLe (Simple Mapping and Localization Estimation) represents a LiDAR odometry algorithm distinguished by its minimal configuration requirements, offering an efficient and precise solution for motion estimation without necessitating complex front-end processing [6]. In common with all Simultaneous Localization and Mapping (SLAM) systems, SiMpLe confronts the interdependent challenges of localization and map construction, where the accuracy of one process inherently supports the other [6]. The algorithm's primary strength lies in its simplicity—it is intentionally designed for ease of comprehension, implementation, and parameter tuning—while consistently delivering robust performance in practical, real-world scenarios [6].

SiMpLe differentiates itself from traditional scan-to-scan registration techniques by employing a scan-to-map registration approach [6]. This strategic design choice is crucial, as it maximizes the spatial overlap between the current LiDAR measurements and the comprehensive, previously mapped data. This method effectively compensates for the inherent reduction in spatial resolution often observed at the sensor's longer sensing ranges [6].

The workflow of the algorithm is structured into three main phases:

Input Scan Spatial Subsampling

The procedure commences with the spatial subsampling of the incoming LiDAR scan $\mathbb{P}_k$ to generate a reduced scan, $\mathbb{P}'_k$, using a fixed radius r_{new} [6].

In contrast to random or conventional voxel-based sampling, this technique is designed to prevent aliasing artifacts that can arise from rigid discretization [6]. A KD-tree structure is utilized to execute the efficient removal of redundant points. Optionally, a minimum range filter (r_{min}) can be applied to discard points that are too proximal to the sensor [6]. This phase significantly curtails the point cloud size while preserving essential environmental information, thereby boosting computational performance. For applications where resources are abundant, such as offline processing, this subsampling step can be bypassed to allow for full-resolution analysis [6].

Point Cloud-to-Map Registration

Subsequently, SiMpLe registers the newly subsampled scan $\mathbb{P}'_k$ against the established subsampled map $\mathbb{M}'_{k-1}$ [6]. The objective is to estimate the optimal transformation $\mathbb{T}_{1\to k}$ that best aligns the two sets of points.

This registration relies on two core elements: an objective function and a search algorithm [6]. The objective function assigns a score to prospective pose hypotheses based on the quality of alignment between the new scan points and the existing map structure [6]. The search algorithm systematically explores the pose space to maximize this score [6].

The SiMpLe objective function rewards alignments where new scan points are located in close proximity to existing map points [6]. For any j-th pose estimate $\hat{T}_{1\to k,j}$, each point $p'_{k,i|j}$ from the current scan is paired with its nearest map point $m'_{k-1,i|j}$. The distance $d_{i|j}$ is calculated as:

$$d_{i|j} = ||\prime'_{k,i|j} - \gtrdot'_{k-1,i|j}||$$

corresponding reward $r_{i|j}$:

$$r_{i|j} \propto \exp\left(-\frac{d^2_{i|j}}{2\sigma^2_{reward}}\right)$$

The transformation that maximizes the cumulative reward is ultimately selected as the optimal estimate:

$$\hat{T}_{1\to k} = argmax_j \left\{ \sum_{i=1}^{n'_k} r_{i|j} \right\}$$

This reward-based formulation shares conceptual ties with ICP but substitutes explicit error minimization with a proximity-based reward [6]. This eliminates the requirement for complex pre-matching or feature extraction steps. The bandwidth parameter σ_{reward} allows for tuning the sensitivity to point distances, balancing precision against robustness [6].

To efficiently search for this optimal transformation, SiMpLe defaults to the Broyden–Fletcher–Goldfarb–Shanno (BFGS) quasi-Newton optimization method, although other gradient-based or stochastic alternatives can also be employed [6]. The optimization procedure halts when the improvement in the registration reward drops below a specified threshold, ϵ_{tol} [6].

A constant velocity motion model is employed to furnish an initial pose estimate, a critical step that both accelerates convergence and helps mitigate the risk of falling into local extrema:

$$\hat{\mathbb{T}}_{1\to k} = \mathbb{T}_{1\to(k-1)}\Delta$$

Where the incremental transformation Δ is derived from the previous two poses:

$$\Delta = \mathbb{T}^{-1}_{1\to(k-2)}\mathbb{T}_{1\to(k-1)}$$

Map update

Upon successful convergence, the existing map is updated using the latest registration result $\mathbb{T}_{1\to k}$ [6]. The newly transformed scan $\mathbb{P}'_k$ is integrated into the subsampled map $\mathbb{M}'_{k-1}$ while rigorously enforcing a minimum spatial separation r_{map} between the map points [6]. To prevent unbounded map growth, any points situated beyond the sensor's maximum operational range r_{max} from the robot's current pose are selectively pruned [6]. The fundamental scan-to-map registration employed by SiMpLe effectively mitigates alignment biases that are often introduced by LiDAR scan patterns. Furthermore, this approach guarantees the maximum possible geometric overlap between consecutive observations, leading to enhanced overall registration accuracy when compared to typical scan-to-scan matching techniques [6].

Thus, SiMpLe [6] offers a lightweight, efficient, and accurate LiDAR odometry approach. Its effectiveness stems from combining scan-to-map alignment with a reward-based optimization scheme. The algorithm's minimal design, low computational load, and proven robustness in field conditions make it a solid choice for standalone odometry or seamless integration within SLAM systems [6].

Chapter 6

Benchmark of the Selected Algorithms

This chapter details the methodological framework used for comparing the performance of four specific LiDAR odometry algorithms: KISS-ICP, GENZ-ICP, MOLA-LO, and SiMpLe. The central objective of this benchmark is to rigorously assess the effectiveness and resilience of each algorithm across two contrasting environments [27]. The first, represented by the KITTI Odometry dataset, offers a canonical, well-understood urban and rural scenario widely utilized by the research community [28]. The second, a proprietary dataset collected by the PIC4SeR research center, introduces unique challenges associated with minimal geometric texture, dense vegetation, and a significant scarcity of unique landmarks.

The evaluation relies on standard quantitative measures, mainly the Absolute Trajectory Error (ATE) and the Relative Pose Error (RPE), which assess positional and orientational accuracy [2]. Alongside these, computational efficiency is also analyzed to understand each method's overall practicality. This combined assessment helps clearly outline the strengths and limitations of each approach, identifying which algorithms are better suited for unstructured and demanding environments.

6.1 KITTI Odometry Dataset

The KITTI Vision Benchmark Suite, a seminal contribution from the Karlsruhe Institute of Technology and the Toyota Technological Institute of Chicago, has solidified its position as a foundational standard for evaluating perception and odometry algorithms in the context of autonomous driving [28]. The KITTI Odometry segment is particularly prevalent for testing the capabilities of both visual and LiDAR-based motion estimation systems [2, 27]. Data acquisition was performed using a synchronized suite of sensors mounted on a vehicle, which included stereo cameras (grayscale and color) and a Velodyne HDL-64E 64-beam LiDAR sensor [28].

The dataset is composed of 22 sequences. The initial 11 sequences (00 to 10)

Figure 6.1: Example images from the KITTI Odometry dataset (sequence 00, frame 000066). Left: grayscale version. Right: original color image. [28]

are designated for benchmarking, as they are accompanied by high-precision ground truth data derived from a GPS/IMU system. The remaining 11 sequences (11 to 21) are reserved for official online challenges and lack publicly released ground truth [28]. The data was collected across various driving and environmental conditions, encompassing diverse urban, rural, and highway settings [28], as it can be seen in Figure 6.1.

Despite its benchmark status, the KITTI dataset imposes significant constraints that rigorously test algorithm robustness [27]. The limited angular resolution of the Velodyne HDL-64E can compromise performance in environments where geometric detail is sparse, such as long highway stretches or expansive parking areas [2]. Furthermore, the sensor's 10 Hz scan rate, coupled with vehicle velocity, often results in low overlap between consecutive point clouds, thereby complicating the scan registration process. A critical aspect of many KITTI sequences is their open-loop trajectory; trajectories do not close, preventing algorithms from employing loop closure techniques essential for correcting the long-term accumulation of odometry errors [2]. This open-loop design means odometry errors, especially the Relative Pose Error (RPE), tend to compound significantly over the trajectory length.

The presence of dynamic objects—moving vehicles, pedestrians, and cyclists — introduces considerable complexity. These objects inject "outliers" or kinetic points into the point clouds, which must be effectively identified and filtered to prevent them from corrupting the inter-frame alignment [2]. This is a known vulnerability for many standard Iterative Closest Point (ICP)-based methods unless they integrate highly robust outlier rejection strategies. While the provided ground truth is generally reliable, its accuracy can degrade in areas with poor satellite signal reception (e.g., under bridges), a factor that must be considered during detailed error analysis [28], [29]. The need to rely on refined ground truth from sources like SemanticKITTI for the most accurate evaluations further highlights this issue [29].

The KITTI Odometry dataset provides a solid foundation for evaluating LiDAR odometry methods. However, its specific characteristics—such as sparse data, the presence of dynamic objects, and limited opportunities for loop closure—pose significant challenges that thoroughly test algorithm robustness.

Figure 6.2: Aerial view of the trajectory of the vehicle in the vineyard, indicating the start (red) and end (green) points.

6.2 Vineyard Dataset

This private dataset was collected by the PIC4SeR research center in a vineyard near Torino, Italy, as shown in Fig 6.2. Data acquisition spanned approximately 25 minutes using a Husky Unmanned Ground Vehicle (UGV) and was stored in a rosbag2 file, constituting a unique and highly challenging benchmark for localization systems. This environment presents a combination of factors that stand in sharp contrast to typical urban benchmarks like KITTI, focusing on the difficulties inherent to unstructured natural environments [30, 31].

The UGV is equipped with a rich and well-synchronized sensor suite. Its primary odometry sources are two 3D LiDARs, which increase point density and widen the field of view—an advantage in unstructured and complex environments. The UGV also carried:

- An Inertial Measurement Unit (IMU), providing high-frequency angular velocity and linear acceleration data.
- A high-precision GPS receiver, which supplies a reliable reference trajectory confirmed by topics like /husky/*global_localization*/odometry/gps, essential for performance evaluation.
- RGB and depth cameras, offering supplementary data on the visual characteristics of the environment.

The UGV's trajectory follows a repetitive back-and-forth pattern, characteristic of agricultural operations. The environment is defined by parallel rows of grapevines,

Figure 6.3: Screenshot from the onboard camera, showing the dense vegetation and uneven terrain of the vineyard environment.

mostly level ground (minor bumps notwithstanding), and a pronounced structural monotony [5].

The principal technical challenge is the structural degeneracy caused by the rows of grapevines [30, 32]. Unlike urban settings with diverse, rigid geometric features, the vineyard consists of repetitive, parallel structures. This severe lack of non-repeating landmarks makes it extremely difficult for standard LiDAR odometry (particularly ICP-based algorithms) to establish unique correspondences between successive scans [32]. This often leads to rapid pose divergence and accumulation of odometry drift [32, 33]. The sharp 180-degree turns at the end of each row also impose significant transient challenges. These environment's physical characteristics can be seen in the images Fig 6.2 and Fig 6.3.

Furthermore, the dense and complex vegetation is a major source of noise. The foliage, high grass, and irregular growth create a highly unstructured and "soft" point cloud [31]. LiDAR beams are frequently scattered or obstructed by leaves, resulting in incomplete and noisy data. The soft nature of the vegetation means points are not fixed to rigid bodies and can move with the wind, effectively adding a layer of dynamic noise to the scene. The visual data confirms a scarcity of unique visual landmarks, complicating multimodal odometry [33], [32] which must address obstacles like "irregular planes" and insufficient vertical constraints [33].

In conclusion, the vineyard dataset is a rigorous benchmark that forces algorithms to maintain accurate pose estimation in an environment characterized by repetitive geometric structures, elevated noise levels from complex vegetation, and the absence

of the clear, rigid features typically exploited by most localization algorithms. It represents a crucial test for generalizability to natural, unstructured domains, similar to the motivations behind Wild-Places [31].

6.3 Test Environment Setup

6.3.1 Hardware and Software Components

To guarantee a fair and reproducible evaluation of the selected odometry algorithms, precisely documenting the computational setup is paramount. All experiments were conducted on a desktop computer utilizing an AMD Ryzen 5 5600G with Radeon Graphics, which provided a base clock frequency of 3.90 GHz. The system featured 16 GB of RAM operating at 2133 MT/s, dedicating 15.3 GB to the operating system. Significantly, the experiments relied solely on CPU computation, lacking any dedicated GPU acceleration. The available 466 GB of storage easily accommodated the required datasets and all generated output files. As this hardware configuration falls within the mid-range category, it intrinsically curtailed computational throughput, meaning measured performance metrics will invariably appear lower than those reported in original publications that often leverage specialized computing resources.

The software environment was based on a dual-boot setup, with Windows 11 Pro (version24H2) as the host system. All experimental work was carried out in Ubuntu 22.04.5 LTS, running under the Windows Subsystem for Linux 2 (WSL2). This setup hosted the ROS 2 Humble distribution, chosen for its compatibility with the LiDAR drivers and stability under WSL2. Core dependencies for point cloud processing included `PCL`, `Eigen3`, `Ceres Solver`, and `yaml-cpp`. To prevent version conflicts and ensure consistent dependency management, all Python packages were isolated within a dedicated virtual environment (`venv`).

It must be stressed that the performance values presented here do not aim for perfect reproduction of the results reported in the original algorithm papers. The deliberate use of a CPU-only, mid-range setup, coupled with the virtualization overhead of WSL2, fundamentally restricted the achievable computational efficiency. Consequently, algorithms known for their high computational complexity, such as SiMpLe-LO, might show increased drift or reduced frame rates when compared to their official benchmarks. Therefore, readers must interpret these results as a balanced comparative analysis conducted under well-defined hardware limitations, rather than a direct validation of published performance metrics.

Vineyard Dataset Acquisition Platform: Sensors and Calibration

The proprietary Vineyard dataset was collected utilizing a Clearpath Husky, a robust, four-wheeled, teleoperated Unmanned Ground Vehicle (UGV) known for its

stability on uneven agricultural terrain. All sensors were firmly mounted onto a custom rigid frame to minimize vibration and guarantee fixed relative positioning during data collection. To ensure temporal integrity, the multi-sensor data streams were precisely synchronized and recorded in the `.mcap` format.

A Livox Mid-360 LiDAR (10 Hz) served as the main ranging device. Its 360° horizontal field of view and wide vertical coverage (from $-7°$ to $+52°$) were crucial for modeling both the ground and the dense canopy. With a range of 0.1–40 m, a 0.2 m resolution, and an angular accuracy of $\pm 0.15°$ (at 1σ), this LiDAR generated the dense, high-quality point clouds required for complex natural environments. Complementary visual context was provided by an Intel RealSense D435 RGB-D camera (30 Hz), which delivered 640×480 pixel resolution images and a depth sensing range up to 30 m.

Motion refinement was handled by a MicroStrain 3DM-GX5 Inertial Measurement Unit (IMU) (100 Hz), providing high-rate motion data crucial for mitigating drift (accelerometer resolution: 0.02 mg; gyroscope: 0.003°/s). Ground-truth positioning was established via a Swift Navigation Duro RTK-GPS receiver (5 Hz), delivering centimeter-level accuracy (typically 1–2 cm horizontal) under RTK correction, essential for reliable benchmarking. This combination of wide-FoV LiDAR, depth camera, high-precision IMU, and RTK-GPS enabled the generation of a consistent and richly detailed dataset, capturing both static vineyard geometry and dynamic vehicle motion.

6.3.2 Installation and Compilation of the Algorithms

Each odometry algorithm demanded a distinct installation process tailored to its specific technical dependencies.

We used two approaches for **KISS-ICP**. The straightforward `pip install kiss-icp` was used in KITTI trials. In order to decompress the.mcap.zstd files for the ROS 2-based vineyard dataset, we compiled and installed a specialized `zstd` plugin after cloning the official GitHub repository into the `ros2_ws` workspace.

GENZ-ICP installation began after strictly verifying prerequisites: Linux compatibility, CMake ≥ 3.14, a C++17 compiler, and libraries such as `Eigen3`, `PCL`, and `ros2bag`. The source code was then cloned and compiled using the `colcon build` tool within the ROS 2 workspace.

The **MOLA-LO** framework required confirming ROS 2 dependencies (`tf`, `sensor_msgs/PointCloud2`, `evo`, etc.). After activating the ROS 2 environment, we compiled the main MOLA modules and the odometry package, validating the setup with the `mm-viewer` visualization tool.

The **SiMpLe-LO** algorithm demanded the most extensive setup, including tools such as `git`, `g++`, `CMake`, and libraries like `Eigen`, `Intel TBB`, `nanoflann`, and `Dlib`. Following meticulous environment configuration, we cloned the repository and built the source code using the standard `make` procedure, generating a standalone executable.

In the end, the original laptop plan was shelved because virtualization limitations made it impossible to install Ubuntu. In order to comprehend why the observed computational efficiency is still marginally lower than the peak performance stated in the original algorithm references, this constraint compelled the consolidation of all tests onto the desktop computer.

6.4 Experimental Procedure

This section details the precise configurations and execution routines we adopted for each odometry algorithm. Our primary objective was to establish a reproducible, tightly controlled testing environment, essential for ensuring fair comparisons and minimizing external variability. We executed all algorithms using semi-automated, script-based workflows to guarantee consistency and maximize efficiency.

6.4.1 KISS-ICP

KITTI Odometry Dataset

For the KITTI dataset, we ran KISS-ICP following its official command-line workflow. To maintain a clean and consistent runtime, we first created and activated a dedicated Python virtual environment, isolating all dependencies for guaranteed reproducibility.

The entire batch of KITTI sequences (00–10) was processed via a comprehensive Bash script, achieving full automation. This script leveraged an iterative loop that precisely invoked the `kiss_icp_pipeline` command, supplying specific, custom parameters. Essential to this process was the use of the `-dataloader kitti` option, which designated the input format, alongside `-sequence "SEQ"` to facilitate dynamic sequence loading. Crucially, the `-visualize` flag was deliberately excluded to prevent graphical overhead from skewing the performance measurements. Upon each successful execution, the required trajectory estimates were automatically generated and persisted for subsequent analysis.

Vineyard Dataset

Execution on the proprietary vineyard dataset demanded a two-phase approach.

Unlike the simple KITTI setup, the custom dataset's ROS 2-based configuration required the coordination of multiple terminals to handle the distinct LiDAR, IMU, and GPS data streams.

The baseline evaluation used default parameters, coordinating several ROS 2 terminals simultaneously:

- **Data Playback:** We initiated streaming with `ros2 bag play 2025_06_25_vineyard_run1_0.mcap -clock -start-paused`.
- **TF Relay:** We utilized two distinct terminals to execute the `ros2 run relay` commands. This setup was essential for republishing and seamlessly consolidating both the static and dynamic TF data under one singular and unified `tf` topic.
- **Algorithm Node:** We launched the odometry node via `ros2 run kiss_icp kiss_icp_node -ros-args`, remapping the point cloud topic to `/husky/sensors/lidar3d_0/points` and correctly setting the appropriate frames.
- **Recording:** Finally, a separate terminal was deployed to execute `ros2 bag record`, ensuring the reliable capture of the odometry, GPS, and encoder outputs necessary for subsequent ground truth comparison.

Following the baseline, we conducted a refined run. We pre-processed the raw ground-truth data to remove known timestamp errors and outliers. Crucially, we supplied a tuned configuration file (`advanced.yaml`) to improve performance under vineyard conditions—specifically targeting robustness against dense vegetation, repetitive geometry, and uneven surfaces. The modular launch system (`odometry_launch.py`) managed node remapping, conditional RViz visualization, and parameter loading, ensuring the process remained fully reproducible.

6.4.2 GENZ-ICP

KITTI Odometry Dataset

A preliminary step was essential here: unlike KISS-ICP, GENZ-ICP strictly requires ROS 2-compatible `sensor_msgs/PointCloud2` messages as input. We addressed this by converting the KITTI data from `.bin` to the **`.db3` ROS bag format** using a custom Python script that serialized each scan and its timestamp into PointCloud2 messages.

The process required a multi-terminal ROS 2 setup to achieve completely synchronized execution:

- **Playback:** We ran `ros2 bag play` with flags `-clock -start-paused` for synchronization and `-read-ahead-queue-size 50` to handle buffering.

- **Algorithm:** We launched the odometry via `ros2 launch genz_icp odometry.launch.py`, mapping the topic to `/points_raw` and disabling visualization to secure accurate performance profiling.
- **Recording:** The `ros2 bag record` command was initiated to capture all necessary outputs, including the `/genz_odometry` topic and other relevant debug streams.

Due to memory constraints on our mid-range hardware, process termination ("Killed") occasionally occurred on larger sequences (02, 08, 09); for these, we recorded only essential topics. Post-processing required converting the output through three stages: ROS bag → YAML → TUM → KITTI format, utilizing custom scripts for quaternion to rotation matrix handling. Parameter tuning via the **`kitti.yaml`** configuration was performed to ensure efficient data processing.

Vineyard Dataset

The evaluation using the vineyard data also proceeded via a two-phase protocol: we first established a baseline using the default system parameters, followed by a dedicated, optimized configuration. Synchronization was vital, achieved by aligning multiple terminals to manage data replay, the republishing of TF frames, the launch of the odometry node (`ros2 launch genz_icp odometry.launch.py`), and the output recording. The subsequent optimization phase involved careful refinement of the `outdoor.yaml` configuration file. This crucial step significantly reduced drift and greatly enhanced the system's robustness against the repetitive, dense vegetation patterns characteristic of the vineyard environment.

6.4.3 MOLA-LO

KITTI Odometry Dataset

We executed MOLA-LO using the standalone command-line interface `mola-lidar-odometry-cli`, which conveniently processes KITTI's native `.bin` files directly. We used a scripted loop for all sequences (00–10), invoking:

```
mola-lidar-odometry-cli -input-kitti-seq SEQ -output-tum-path

PATH-config lidar3d-0-default.yaml
```

Environment variables such as `MOLA_INITIAL_VX` and `MOLA_SIMPLEMAP_MIN_XYZ=10` were set to precisely control the mapping behavior. We also included the `-kitti-correction-angle-deg` flag, essential for correcting the known LiDAR mounting offset. The final trajectories were exported directly in TUM format for objective evaluation.

Vineyard Dataset

Testing on the vineyard dataset began with the GUI tool `mola-lo-gui-ros bag2`, which allowed us to perform quick, visual parameter tuning. The environment variable `MOLA_USE_FIXED_LIDAR_POSE=true` was necessary due to the dataset lacking automated TF data. Following the visual tuning phase, final, quantitative runs were performed using the CLI version with the optimized configuration file `lidar3d-0-default-vigna.yaml`. This approach yielded quantitative reproducibility while successfully bypassing any computational overhead associated with the GUI.

6.4.4 SiMpLe

KITTI Odometry Dataset

We executed SiMpLe-LO using its executable:

```
./simple -config config_KITTI_offline.yaml
```

Each KITTI sequence was processed sequentially, producing pose estimates and timing logs. Since these poses are generated in the LiDAR reference frame, we had to apply a custom Python script to convert them to the KITTI standard camera frame:

$$T_{\text{cam}} = T_{\text{velo}\to\text{cam}}\, T_{\text{velo}}\, T_{\text{velo}\to\text{cam}}^{-1}$$

Here, $T_{\text{velo}\to\text{cam}}$ is the rigid transformation matrix sourced from KITTI's `calib.txt`. After transformation, we evaluated the results using both the internal *devkit* and the official KITTI toolkit for a comprehensive accuracy assessment.

Vineyard Dataset

The original `.mcap` format of the vineyard dataset was incompatible with the SiMpLe framework, which required standard `.bin` files. To address this, we developed a custom script, `convert_mcap_to_kitti_lidar.py`. This tool efficiently extracted point cloud data from the LiDAR topic and saved them in a format that replicated the precise KITTI file structure. Simultaneously, we exported the ground-truth information into a separate `poses.txt` file for streamlined comparison later in the analysis.

Two critical verification steps confirmed data integrity:

1. **Sanity Check:** `check_values_pointcloud2.py` verified non-`NaN` values and range limits.
2. **Visual Check:** `visualization3d_filebin.py` rendered 3D point clouds using Matplotlib, confirming the quality of the converted data visually.

Finally, we performed intensive parameter optimization within a custom `config.yaml`, a step necessary to ensure the algorithm's stability and precision in the uniquely challenging vineyard environment.

6.5 Evaluation Metrics

To conduct a comprehensive and standardized comparison of the selected LiDAR odometry algorithms, we established a robust evaluation framework. We quantified the performance of each algorithm using established metrics that assess both global localization accuracy and computational efficiency. The `evo` library, recognized as a standard tool for evaluating odometry and SLAM algorithms, was utilized for all error calculations across both the KITTI and the private vineyard datasets.

6.5.1 Odometry Error Analysis

We assessed the accuracy of the estimated trajectories primarily through two metrics: the Absolute Trajectory Error (ATE) and the Relative Pose Error (RPE).

The **ATE** serves as a direct measurement of the global path consistency achieved by the estimation process. It functions by calculating the Root Mean Square Error (RMSE) between the estimated poses and the corresponding ground truth poses, following an initial rigid body alignment. This particular metric is indispensable for accurately assessing the long-term accuracy and the total accumulated drift inherent in an algorithm across the full trajectory length. To facilitate this analysis, the ATE for each respective algorithm was computed using the specialized `evo_ape` tool.

Conversely, the **RPE** captures the local accuracy of the odometry. It calculates the error in the pose transformation between consecutive or fixed-interval keyframes. The RPE is particularly valuable for evaluating short-term drift and local consistency. We derived the RPE for each algorithm using the `evo_rpe` tool.

Before computing either of these metrics, we ensured all estimated trajectories were aligned with the ground truth using a rigid body transformation to neutralize any initial positional or rotational offsets, thereby focusing the error analysis purely on relative movement and drift.

6.5.2 Computational Efficiency

Measuring computational efficiency was vital for assessing each algorithm's viability for real-time applications. Our primary metric was the average processing time per LiDAR scan. We measured this directly within the ROS 2 environment by monitoring the time elapsed between an algorithm receiving a single point cloud and outputting the corresponding odometry message. To guarantee the accuracy of these measurements, we executed all tests without any graphical user interface (GUI), such

as RViz, eliminating potential performance overhead from visualization. We also recorded the total runtime for the complete trajectory, providing broader context for the algorithm's overall efficiency profile.

6.5.3 Robustness Evaluation

We assessed the robustness of each algorithm based on both its theoretical design and its empirical performance within the challenging vineyard environment. This analysis relied not on a single numerical metric but on a qualitative assessment of how effectively each pipeline managed specific environmental stressors.

A key focus of this evaluation was the analysis of algorithm behavior when confronted with structural monotony and a lack of distinctive features. Although the algorithms generally demonstrated stability, certain parameter choices resulted in significant drift, particularly evident during the 180-degree turns between vineyard rows. This finding underscored that an algorithm's practical robustness is highly dependent on proper configuration. Furthermore, we assessed the algorithms' ability to manage occlusions and sparse data resulting from irregular foliage. We partially mitigated this by restricting the LiDAR range, a tactic that reduced the influence of distant, less informative data and forced a greater focus on close-range geometry. The final parameter set adopted for each algorithm represents the best empirically determined trade-off between accuracy and inherent robustness across the selected datasets.

Chapter 7

Experimental Results

This chapter presents the experimental findings derived from the evaluation of the selected LiDAR odometry algorithms. Our goal is twofold: to deliver both a quantitative assessment and a qualitative analysis of their performance across two starkly contrasting datasets. The first is the highly familiar **KITTI odometry dataset**, which epitomizes a structured urban setting. The second is a proprietary dataset meticulously collected within a **Vineyard dataset**, a domain that introduces unique complexities such as dense vegetation, irregular terrain, and dynamic occlusions.

The chapter is organized to build a clear and logical narrative. I begin by describing the parameters chosen for each algorithm, outlining both the tuning procedure and the reasoning behind each configuration. Next, I present the results obtained on the KITTI dataset, followed by those derived from the vineyard experiments. Each section includes summary tables of error statistics, visual trajectory plots comparing estimates to ground truth, and an interpretative discussion of the outcomes. The chapter concludes with a comparative overview that highlights the most significant differences across datasets, examining robustness, computational cost, and the overall suitability of each approach under distinct environmental conditions.

7.1 Parameter Selection

The performance obtained from any LiDAR odometry algorithm is critically dependent on the choice of its internal parameters, which directly mediate both localization accuracy and computational efficiency. Consequently, before detailing the experimental results, it is imperative to provide an explicit and exhaustive overview of the setup and parameter configurations adopted throughout this evaluation. This section maps out the systematic strategy we followed for parameter tuning, covering initial attempts and subsequent refinements, alongside the specific configurations ultimately selected for each algorithm. Documenting these choices ensures full experimental reproducibility and clearly explains the rationale underpinning the final parameter selection, emphasizing the inherent trade-offs encountered between precision and efficiency when moving between different datasets and challenging environmental

contexts.

7.1.1 Parameter tuning strategy

Parameter selection for each odometry algorithm followed a structured and iterative process aimed at balancing accuracy and runtime efficiency. The first tests were carried out using the default settings from the official implementations on both the KITTI and vineyard datasets. As expected, the default configurations did not always perform well—particularly in the vineyard dataset, where dense vegetation and uneven ground introduced both geometric and computational difficulties. Adjustments were therefore made progressively, based on repeated trials and error pattern observation.

We then initiated a second iteration of testing, designated the "second attempt," specifically to refine the parameter values. During this intensive phase, we incrementally adjusted individual parameters while meticulously observing their resulting effects on both trajectory estimation and computational time. For instance, we fine-tuned parameters such as the voxel grid resolution, the maximum number of iterations, and the convergence thresholds to minimize drift and enhance robustness, especially in regions characterized by sparse or noisy point cloud data. The final parameter set adopted for each algorithm represents an optimal compromise: achieving minimal trajectory errors while maintaining execution times that allow for a fair and meaningful comparison across all tested methods and datasets.

This systematic tuning method not only ensured fairness and repeatability across experiments but also provided a transparent framework for comparing algorithm performance in diverse environments and sensor conditions.

7.1.2 Algorithm-Specific Configurations

7.1.2.1 KISS-ICP

For the KISS-ICP algorithm, we employed two distinct configurations tailored to the radically different characteristics of the KITTI and vineyard environments.

KITTI Odometry Dataset We ran the algorithm using the default `basic.yaml` configuration provided by the repository [4]. Key settings included:

- `deskew`: `True` to compensate for motion distortion within the LiDAR scans.
- *max_range*: 100.0 m and *min_range*: 0.0 m, defining the LiDAR's effective sensing interval.
- *max_points_per_voxel*: 20, limiting the number of points used for registration within each voxel.

- Adaptive thresholding parameters were set as *initial_threshold*: 2.0 and *min_motion_th*: 0.1, which provided an effective balance between sensitivity and robustness for standard urban sequences.

Vineyard Dataset — First Attempt For the vineyard environment, we initially used the `advanced.yaml` configuration without modification. Key parameters here included:

- *voxel_size*: 1.0 m in the mapping module and *max_points_per_voxel*: 20.
- The adaptive threshold was substituted with a **fixed threshold of** 0.3, which improved stability when operating in highly vegetated areas.
- Registration parameters used *max_num_iterations*: 500 and *convergence_criterion*: 0.0001, relying on automatic thread allocation.

Vineyard Dataset — Second Attempt To critically enhance performance in the vineyard, we adopted a refined set of parameters:

- *max_range*: 20.0 m and *min_range*: 1.0 m to aggressively filter out distant or noisy points.
- *voxel_size*: 0.5 m and *max_points_per_voxel*: 40 to increase map resolution and capture finer details within the vineyard rows.
- The adaptive threshold was adjusted to $fixed\_threshold$: 0.5 with *min_motio n_th*: 0.1.
- Registration parameters were kept unchanged (*max_num_iterations*: 500, *convergence_criterion*: 0.0001) to maintain reliable convergence despite using smaller voxel sizes.

These careful adjustments allowed KISS-ICP to maintain a strong trade-off between accuracy and computational efficiency, especially crucial in the challenging vineyard environment where occlusions and vegetation otherwise severely degrade performance.

7.1.2.2 GENZ-ICP

GENZ-ICP similarly required distinct parameter sets for the two datasets, including a crucial refinement in the second testing phase for the vineyard scenario.

KITTI Odometry Dataset The algorithm utilized the default `kitti.yaml` configuration, from the repository [3]:

- `deskew`: `false`
- *max_range*: 100.0 m, *min_range*: 5.0 m
- *voxel_size*: 0.6 m, *desired_num_voxelized_points*: 3000

- *planarity_threshold*: 0.18, *max_points_per_voxel*: 3
- *initial_threshold*: 2.0, *min_motion_th*: 0.1
- *max_num_iterations*: 100, *convergence_criterion*: 0.0001

These parameters were initially tuned to match the urban characteristics of KITTI sequences, where structured roads and limited occlusions permit a coarser voxelization scheme.

Vineyard dataset — First Attempt The first set of vineyard tests was executed using the unmodified `outdoor.yaml` configuration file.

- `deskew`: `false`, *max_range*: 100.0 m, *min_range*: 0.5 m
- *voxel_size*: 0.6 m, *desired_num_voxelized_points*: 3000
- *planarity_threshold*: 0.2, *max_points_per_voxel*: 3
- *initial_threshold*: 2.0, *min_motion_th*: 0.1, *max_num_iterations*: 100, *convergence_criterion*: 0.0001

Vineyard Dataset — Second Attempt To achieve a significant performance improvement, we adjusted the parameters as follows:

- *max_range*: 20.0 m, *min_range*: 1.0 m to effectively remove distant and potentially noisy points.
- *voxel_size*: 0.15 m and *desired_num_voxelized_points*: 5000 to increase point cloud density and capture finer environmental details.
- *planarity_threshold*: 0.1 and *max_points_per_voxel*: 10 to better preserve critical surface features within the dense vegetation.
- *initial_threshold*: 2.0, *min_motion_th*: 0.1 remained consistent.
- Iteration and convergence settings: *max_num_iterations*: 100, *convergence_criterion*: 0.0001.

This critical optimization allowed GENZ-ICP to manage the vineyard environment much more robustly, successfully reducing drift and significantly improving the accuracy of trajectory estimation.

7.1.2.3 MOLA-LO

We evaluated MOLA-LO on both datasets, basing our configurations on the repository's default
`lidar3d - default.yaml` file [5].

KITTI Odometry Dataset For the KITTI experiments, we only adjusted a small number of parameters to better suit the dataset's characteristics. Specifically, *min_motion_model_xyz_cov_inv* was reduced to 0.5 (from 1.0), the adaptive threshold initial sigma was lowered to 1.5 m (from 2.0 m), and the local map's voxel size was fixed at 0.5 m. We configured the ICP solver to use a Gauss-Newton method with a maximum of 300 iterations per step. Adaptive thresholding was enabled, featuring a minimum motion of 0.1 m and a maximum sigma of 3 m, thereby allowing the algorithm to dynamically adjust registration sensitivity based on the observed motion. Keyframe selection and local map updates were controlled by minimum translation and rotation thresholds, with a maximum distance of 100 m for keyframe retention, which ensured efficient mapping and reliable trajectory estimation.

Vineyard Dataset For the vineyard dataset, we further adapted the `lidar3d-default.yaml` configuration to account for the specific sensor setup and environmental context. We updated the LiDAR sensor topic to `/husky/sensors/lidar3d_0/points` and appropriately set the corresponding IMU and GNSS topics. Providing a fixed sensor pose was necessary to improve initial alignment stability. While the local map's voxel size remained 0.5 m, we set the minimum rotation between keyframes to 45° to better capture the sparse and irregular structure of the vineyard rows. The adaptive threshold maintained the same settings as KITTI, and the ICP solver remained configured with the Gauss-Newton method and 300 maximum iterations. These crucial adjustments enabled MOLA-LO to handle the challenging geometry and density variations of the vineyard dataset, ultimately yielding robust odometry despite the highly unstructured environment.

7.1.2.4 SiMpLe

We evaluated SiMpLe on both the KITTI and vineyard datasets, starting with the default configurations provided in the repository [6].

KITTI Odometry Dataset For the KITTI experiments, we employed the `config_kitti_offline.yaml` setup without alterations. The main parameters were $\sigma = 0.3$, $r_{Map} = 0.5$, $r_{New} = 0.3$, $convergence_{Tol} = 1e-6$, $minSensorRange = 0$, and $maxSensorRange = 120$. This configuration enabled efficient processing of dense urban LiDAR scans while retaining strong registration precision.

Vineyard Dataset For the vineyard dataset, we initially selected a baseline configuration adapted for a less structured environment ($\sigma = 0.5$, $r_{Map} = 1.5$, $r_{New} = 0.3$, $convergence_{Tol} = 1e-4$, $minSensorRange = 3$, $maxSensorRange = 80$). However, the resulting performance was insufficient due to the inherent sparsity and irregularity of the vineyard LiDAR scans. This necessitated a series of parameter tuning experiments, which ultimately led to the final optimized configuration: $\sigma = 0.5$, $r_{Map} = 1.5$, $r_{New} = 0.3$, $convergence_{Tol} = 1e-4$, $minSensorRange = 1$,

$maxSensorRange = 10$. These final adjustments significantly improved odometry estimation by limiting the influence of distant points and drastically increasing the sensitivity to nearby features, which proved far more informative in the vineyard environment.

7.2 Results on the KITTI Odometry Dataset

The KITTI odometry benchmark [28] stands as one of the most established references for assessing LiDAR-based odometry and SLAM methods. It offers a comprehensive range of realistic driving conditions, spanning structured urban areas and semi-rural environments, each with its own diverse sensor trajectories and motion profiles. To maintain consistency and ensure the reproducibility of our evaluation, we tested all algorithms using the official KITTI development kit for odometry, which served as our primary source for computing standard error metrics.

Alongside the core KITTI tools, we made extensive use of the `evo` framework to extract additional insights. The `evo_ape` and `evo_rpe` commands were applied to compute the Absolute Pose Error (APE) and Relative Pose Error (RPE), respectively. Meanwhile, `evo_traj` provided an intuitive visualization of the estimated trajectories compared to the ground truth. This visual confirmation helped ensure that numerical findings matched the observed motion patterns.

In the subsequent pages, I will lay out both the quantitative and qualitative outcomes achieved by the four evaluated LiDAR odometry algorithms on the KITTI dataset. Our discussion emphasizes the distinct strengths inherent in each approach, alongside any limitations observed under specific parameter choices and motion conditions. We dedicate particular attention to identifying sources of drift, evaluating robustness against dynamic objects, and assessing how sensitive each method proves to careful parameter tuning. Collectively, these results offer a balanced, realistic picture of algorithm behavior on this mature benchmark, establishing a clear and necessary baseline for comparison with the challenging vineyard dataset discussed in the next section.

7.2.1 Quantitative Results

7.2.1.1 Development Kit Analysis

To guarantee a fair and reproducible evaluation of the LiDAR odometry algorithms on the KITTI odometry benchmark [28], we initiated our analysis using the official development kit provided by the dataset authors. This toolkit strictly adheres to the benchmark's standard protocol, computing two essential indicators for every trajectory segment: the average translational error, expressed as a percentage of the total distance traveled, and the average rotational error, measured in degrees per meter. Crucially, we repeated the evaluation over increasing path lengths (from

	00	01	02	03	04	05	06	07	08	09	10
KISS-ICP	0.528	**0.786**	0.537	0.677	0.385	0.342	0.281	0.376	0.820	**0.534**	0.512
GENZ-ICP	0.617	0.956	0.717	0.967	1.004	0.579	0.597	0.618	**0.629**	0.757	0.860
MOLA-LO	0.562	0.695	**0.525**	**0.671**	**0.331**	0.419	0.284	0.421	0.821	0.545	0.612
SiMpLe	**0.514**	0.858	0.526	0.697	0.397	**0.297**	**0.278**	**0.299**	0.804	0.549	**0.506**

Table 7.1: Average Translational Error (%) of KISS-ICP, GENZ-ICP, MOLA-LO, SiMpLe, on sequences 00-10. In **bold** characters are reported the lowest value of the error per sequence.

	Average	Paper
KISS-ICP	0.56	0.50
GENZ-ICP	0.68	0.51
MOLA-LO	0.58	0.55
SiMpLe	0.55	0.55

Table 7.2: Overall average translational error (%) of the evaluated algorithms on KITTI sequences 00–10, compared with the reference average values reported in their original publications.

100 m, 200 m, up to 800 m), which allowed us to observe the evolution of the error as a direct function of distance.

To deepen the analysis, we complemented the standard metrics with a full statistical summary using the `evo` toolkit. For each KITTI sequence, we calculated both the APE and RPE, then summarized them with key statistical measures—mean, median, RMSE, maximum, and minimum. This allowed a balanced view of overall accuracy and local consistency throughout the trajectories.

The following tables and figures summarize the complete evaluation results, making it straightforward to compare the different algorithms. Table 7.1, for instance, lists the average translational errors (%) for all 11 KITTI odometry sequences. Each row corresponds to an algorithm, while the columns represent the sequences. The last two columns contrast our experimental averages with the values reported in the original papers.

To provide a concise overview of the comparative results, Table 7.2 reports the mean translational error (%) obtained across KITTI sequences 00–10 for all evaluated algorithms. For completeness, the reference averages drawn from the corresponding original publications are also included. This side-by-side presentation allows an immediate verification of how closely our experimental outcomes align with previously reported benchmarks [28].

After assessing translational accuracy, we shifted focus to rotational performance, measured in degrees per meter (deg/m). Table 7.3 compiles these data, organizing the algorithms by rows and the KITTI sequences by columns. Each entry gives the average rotational error for a given algorithm–sequence pair.

	00	01	02	03	04	05	06	07	08	09	10
KISS-ICP	0.0019	0.0010	0.0015	0.0016	**0.0013**	0.0013	0.0008	**0.0016**	0.0019	0.0014	0.0019
GENZ-ICP	0.0031	0.0014	0.0026	0.0033	0.0009	0.0033	0.0033	0.0057	0.0031	0.0031	0.0029
MOLA-LO	0.0022	0.0014	0.0017	0.0017	0.0014	0.0019	**0.0008**	0.0024	0.0021	0.0017	0.0021
SiMpLe	**0.0018**	**0.0009**	0.0013	0.0014	0.0010	0.0013	0.0008	0.0018	**0.0016**	**0.0012**	**0.0015**

Table 7.3: Average rotational error (deg/m) of the four evaluated algorithms across KITTI sequences 00–10. The lowest error value for each sequence among the four methods is highlighted in **bold**.

	Average	**Paper**
KISS-ICP	0.0016	0.0017
GENZ-ICP	0.0029	N.A.
MOLA-LO	0.0019	0.0017
SiMpLe	0.0014	0.0015

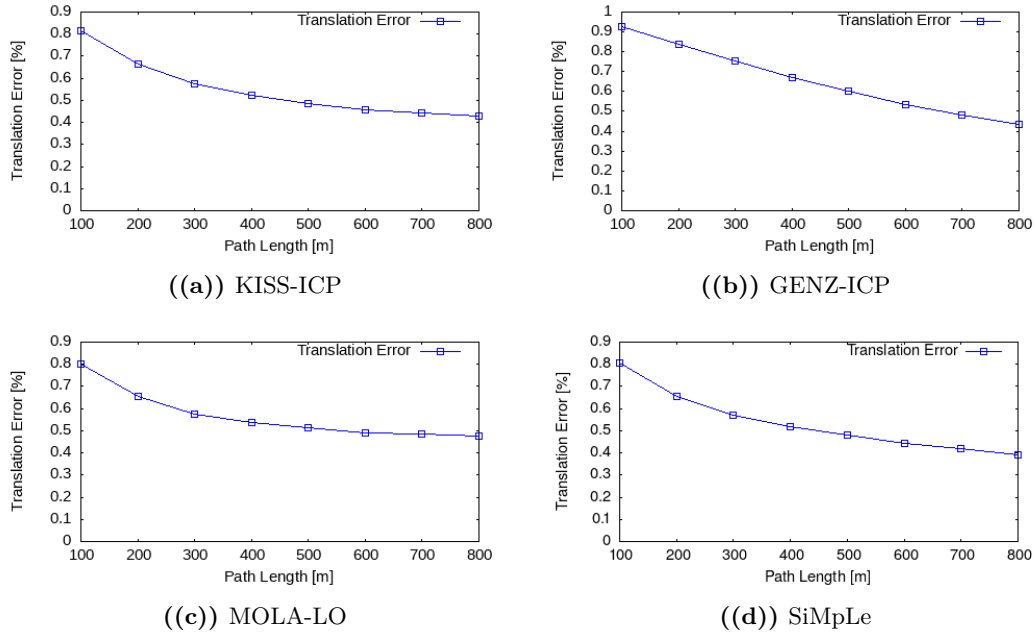
Table 7.4: Overall average rotational error (deg/m) of the evaluated algorithms on KITTI sequences 00–10, compared with the reference average values reported in their original publications.

For a broader perspective, Table 7.4 reports the mean rotational errors (deg/m) computed across sequences 00–10 for all algorithms. Reference values from the respective publications are also included, allowing a direct comparison between our findings and prior benchmarks.

To further characterize the performance profile of the evaluated LiDAR odometry methods, we analyzed the average translational and rotational errors as a function of segment length, ranging from 100 m up to 800 m. Each resulting curve represents the mean error across all KITTI sequences for a single algorithm, intuitively revealing how drift evolves with increasing trajectory length. These visualizations offer a direct comparison of both inherent accuracy and overall robustness. It is particularly interesting to note that certain algorithms exhibit a stabilizing or slightly decreasing trend in error for longer segments, a phenomenon which suggests that accumulated drift tends to distribute more uniformly along extended trajectories.

Figure 7.1 explicitly illustrates the average translational error (%) as a function of segment length for KITTI sequences 00–10. The figure utilizes four dedicated subplots to present the individual algorithm data—Figure 7.1(a) (KISS-ICP), Figure 7.1(b) (GENZ-ICP), Figure 7.1(c) (MOLA-LO), and Figure 7.1(d) (SiMpLe)—with each curve depicting the mean error averaged over all sequences.

Figure 7.2 illustrates the mean rotational error (deg/m) as a function of segment length for KITTI sequences 00–10. Each subfigure—7.2(a) (KISS-ICP), 7.2(b) (GENZ-ICP), 7.2(c) (MOLA-LO), and 7.2(d) (SiMpLe)—shows the averaged error trends for the corresponding method.



((a)) KISS-ICP

((b)) GENZ-ICP

((c)) MOLA-LO

((d)) SiMpLe

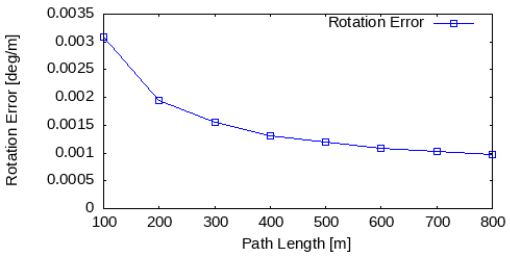
Figure 7.1: Average translational error (%) as a function of segment length for KITTI sequences 00–10. The four subfigures show the results for the individual algorithms: (a) KISS-ICP, (b) GENZ-ICP, (c) MOLA-LO, and (d) SiMpLe.

7.2.1.2 Evo Tools

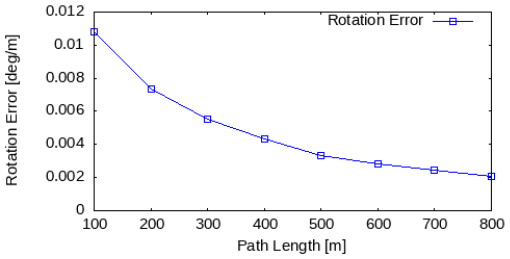
To effectively complement the preceding benchmark evaluation, we further scrutinized the performance of the four LiDAR odometry algorithms using the `evo` framework, focusing specifically on the Absolute Pose Error (APE) and the Relative Pose Error (RPE) metrics.

We selected two representative KITTI sequences to highlight the significant influence of environmental structure on odometry accuracy. **Sequence 01** traverses an open highway stretch where the inherent scarcity of distinctive landmarks and geometric references typically exacerbates localization drift. Conversely, **Sequence 06** navigates a densely built urban area, offering a rich collection of reference features—buildings, vegetation, and parked vehicles—that generally boost motion estimation reliability. While comparing sequences with inherently different trajectories can introduce bias, Sequences 01 and 06 are directly comparable given that both comprise exactly 1100 frames.

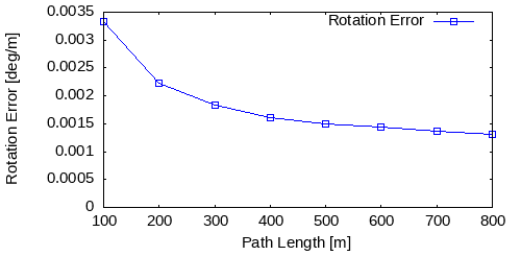
For every algorithm, we computed APE and RPE using `evo_ape` and `evo_rpe`, respectively. The analysis included all standard statistical descriptors (mean, median, RMSE, minimum, and maximum) to provide a comprehensive quantitative comparison. We visually depict the contrasting scenes of these two sequences using representative grayscale images from the KITTI Odometry Dataset, shown in Figure 7.3(a) and Figure 7.3(b). Finally, the resulting tables and plots clearly summarize the APE and RPE outcomes, providing an accessible overview of how each algorithm



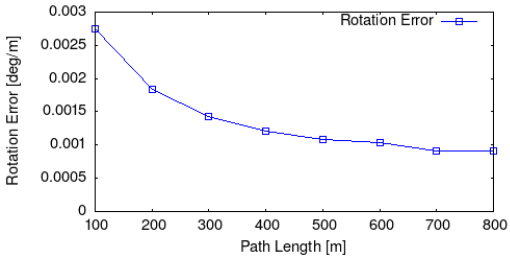
((a)) KISS-ICP



((b)) GENZ-ICP



((c)) MOLA-LO



((d)) SiMpLe

Figure 7.2: Average rotational error (deg/m) as a function of segment length for KITTI sequences 00–10. The four sub-figures show the results for the individual algorithms: (a) KISS-ICP, (b) GENZ-ICP, (c) MOLA-LO, and (d) SiMpLe.

behaves under both structured and unstructured driving scenarios.

((a)) Sequence 01 - Frame 000336/001100

((b)) Sequence 06 - Frame 000767/001100

Figure 7.3: Representative grayscale frames from the KITTI Odometry Dataset. (a) Sequence 01 with sparse landmarks and open road; (b) Sequence 06 in a structured urban environment with multiple reference features.

For a consistent and thorough comparison of odometry performance, we analyzed the translational and rotational components of both the Absolute Pose Error (APE) and the Relative Pose Error (RPE) separately. Treating these quantities independently makes it easier to see how each method manages position accuracy versus orientation drift, rather than relying on a single mixed metric that can obscure their individual contributions.

In practice, six complementary statistical indicators were used to describe each algorithm's behavior. The `max` and `min` values capture the extremes, while the `mean` provides an overview of global accuracy. The `median` helps filter occasional outliers, the `std` (standard deviation) shows the consistency around the mean, and the `RMSE` emphasizes larger deviations. Combined, these indicators offer a complete and nuanced picture of each method's precision and reliability.

	Max	Mean	Median	Min	Std	Rmse
KISS-ICP	5.366	2.959	2.980	1.110	1.076	3.149
GENZ-ICP	6.232	3.112	3.154	**0.427**	1.164	3.322
MOLA-LO	8.157	4.662	4.856	1.426	1.806	4.999
SiMpLe	**4.912**	**2.333**	**2.299**	0.989	**0.888**	**2.496**

Table 7.5: Absolute Pose Error w.r.t translational part (m) for sequence 01 (sparse-road scenario). Reported statistics: mean, median, RMSE, standard deviation, minimum, and maximum. Lowest values are highlighted in **bold**.

	Max	Mean	Median	Min	Std	Rmse
KISS-ICP	0.938	0.3710	0.366	0.066	0.193	0.418
GENZ-ICP	2.288	1.043	1.039	0.143	0.349	1.100
MOLA-LO	2.022	0.353	0.332	0.069	0.204	0.408
SiMpLe	**0.868**	**0.293**	**0.269**	**0.009**	**0.172**	**0.340**

Table 7.6: Absolute Pose Error w.r.t translational part (m) for sequence 06 (urban scenario). Reported statistics: mean, median, RMSE, standard deviation, minimum, and maximum. Lowest values are highlighted in **bold**.

Tables 7.5–7.12 report the APE and RPE statistics for both translation and rotation, computed on the two selected KITTI sequences (01 and 06). These correspond to a sparse-road scene and a structured urban scene, respectively. Presenting them side by side allows an immediate visual comparison between algorithms under contrasting conditions. For each statistic, the lowest error is highlighted in bold to mark the best-performing method.

Overall, these numerical results provide a strong quantitative foundation for the following qualitative and trajectory-based analysis, helping to interpret how each odometry algorithm maintains accuracy across different driving contexts.

7.2.2 Qualitative Results

Although numerical statistics are essential for objective evaluation, visual inspection can reveal aspects that numbers may overlook. Observing reconstructed trajectories often helps explain specific algorithmic behaviors—such as how methods cope with feature sparsity or recover after temporary localization loss. For this reason, we

	Max	Mean	Median	Min	Std	Rmse
KISS-ICP	1.822	0.891	0.791	**0.464**	0.298	0.940
GENZ-ICP	178.992	93.845	90.585	0.630	39.935	101.987
MOLA-LO	2.452	1.261	1.026	0.630	0.458	1.318
SiMpLe	**1.337**	**0.791**	**0.780**	0.552	**0.155**	**0.806**

Table 7.7: Absolute Pose Error w.r.t rotational part (deg) for sequence 01 (sparse-road scenario). Reported statistics: mean, median, RMSE, standard deviation, minimum, and maximum. Lowest values are highlighted in **bold**.

	Max	**Mean**	**Median**	**Min**	**Std**	**Rmse**
KISS-ICP	0.986	**0.303**	**0.252**	0.026	0.166	0.346
GENZ-ICP	179.823	96.085	90.871	0.917	39.935	104.049
MOLA-LO	**0.688**	0.504	0.510	**0.017**	0.200	0,544
SiMpLe	0.875	0.303	0.267	0.070	**0.152**	**0.340**

Table 7.8: Absolute Pose Error w.r.t rotational part (deg) for sequence 06 (urban scenario). Reported statistics: mean, median, RMSE, standard deviation, minimum, and maximum. Lowest values are highlighted in **bold**.

	Max	**Mean**	**Median**	**Min**	**Std**	**Rmse**
KISS-ICP	3.097	1.799	1.725	0.436	0.818	1.976
GENZ-ICP	6.788	2.841	2.726	0.226	1.268	3.111
MOLA-LO	0.122	0.033	**0.030**	0.004	0.016	0.037
SiMpLe	**0.098**	**0.032**	0.030	**0.002**	**0.015**	**0.035**

Table 7.9: Relative Pose Error w.r.t translational part (m) for sequence 01 (sparse-road scenario). Reported statistics: mean, median, RMSE, standard deviation, minimum, and maximum. Lowest values are highlighted in **bold**.

	Max	**Mean**	**Median**	**Min**	**Std**	**Rmse**
KISS-ICP	0.936	0.564	0.607	0.160	0.252	0.617
GENZ-ICP	5.269	2.055	1.993	0.221	0.809	2.209
MOLA-LO	1.229	0.021	0.016	**0.001**	0.052	0.056
SiMpLe	**0.079**	**0.015**	**0.013**	0.001	**0.010**	**0.018**

Table 7.10: Relative Pose Error w.r.t translational part (m) for sequence 06 (urban scenario). Reported statistics: mean, median, RMSE, standard deviation, minimum, and maximum. Lowest values are highlighted in **bold**.

	Max	**Mean**	**Median**	**Min**	**Std**	**Rmse**
KISS-ICP	0.274	0.041	0.039	0.004	0.023	0.047
GENZ-ICP	388.937	162.794	156.162	12.966	72. 626	178.259
MOLA-LO	0.268	**0.040**	0.038	0.005	**0.002**	0.048
SiMpLe	**0.244**	0.041	**0.038**	**0.003**	0.022	**0.046**

Table 7.11: Relative Pose Error w.r.t rotational part (deg) for sequence 01 (sparse-road scenario). Reported statistics: mean, median, RMSE, standard deviation, minimum, and maximum. Lowest values are highlighted in **bold**.

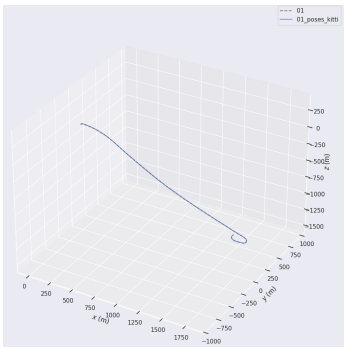
	Max	**Mean**	**Median**	**Min**	**Std**	**Rmse**
KISS-ICP	0.185	0.034	0.029	**0.002**	0.021	0.040
GENZ-ICP	179.987	116.538	119.134	15.292	40.434	123.358
MOLA-LO	0.200	0.035	0.029	0.002	0.022	0.041
SiMpLe	**0.170**	**0.030**	**0.025**	0.002	**0.020**	**0.036**

Table 7.12: Relative Pose Error w.r.t rotational part (deg) for sequence 06 (urban scenario). Reported statistics: mean, median, RMSE, standard deviation, minimum, and maximum. Lowest values are highlighted in **bold**.

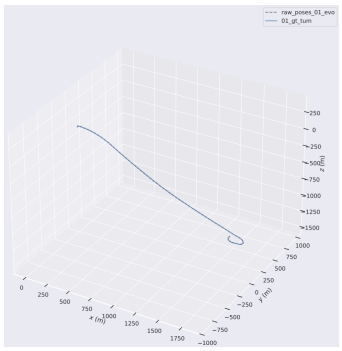
generated representative sequences from the KITTI dataset using `evo_traj`, visually comparing estimated and ground truth trajectories.

Among the available sequences, we focused on Sequences 01 and 06, chosen for their strong environmental contrast. **Sequence 01** represents a long, mostly straight road with very few static landmarks and minimal texture—conditions that tend to magnify small pose errors and challenge LiDAR-based odometry. In contrast, **Sequence 06** takes place in a dense urban area, surrounded by buildings, trees, and parked vehicles. This richer structure generally improves registration stability and helps algorithms maintain alignment over time.

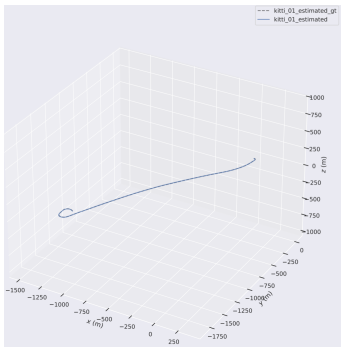
To preserve readability, only these two representative trajectories are shown out of the full set of eleven KITTI sequences. Figures 7.4 and 7.5 illustrate the results for both cases, displaying the paths estimated by the four evaluated algorithms. These side-by-side plots make it easy to see where each method performs best—or struggles—when transitioning from a feature-poor open road to a highly structured city scene. The visual comparison adds an intuitive perspective to the quantitative analysis presented earlier.



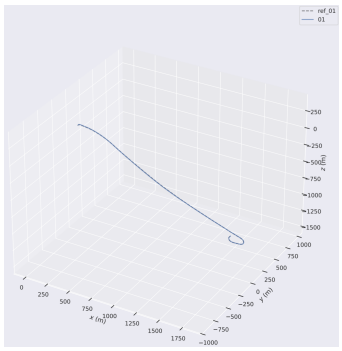
((a)) KISS-ICP



((b)) GENZ-ICP

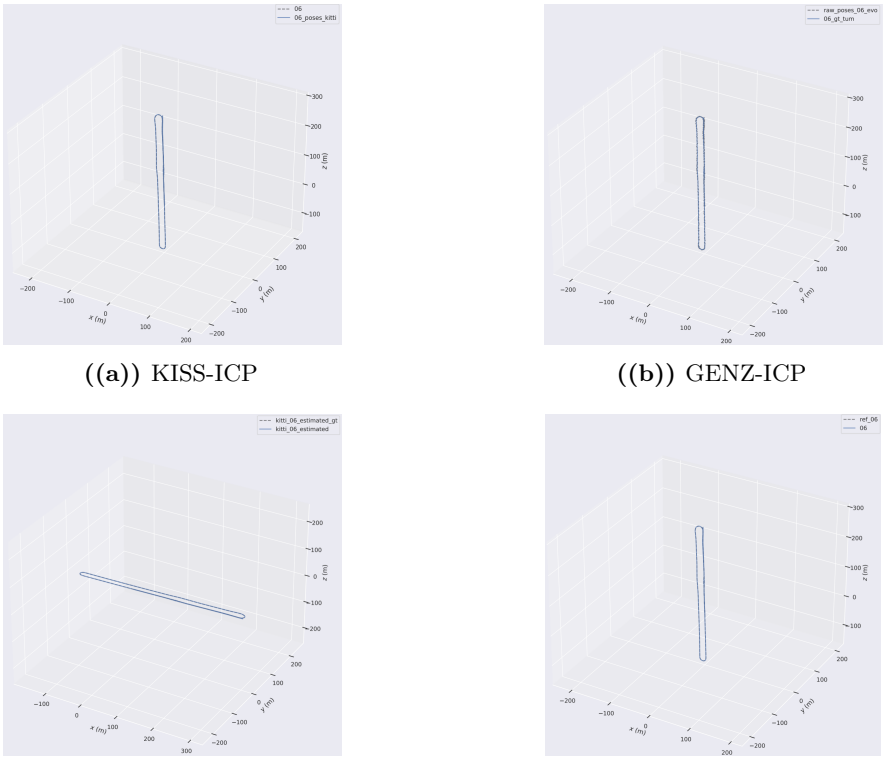


((c)) MOLA-LO



((d)) SiMpLe

Figure 7.4: Trajectory comparison for Sequence 01 (sparse-road scenario). The four subfigures show the paths reconstructed by (a) KISS-ICP, (b) GENZ-ICP, (c) MOLA-LO, and (d) SiMpLe, overlaid with the ground-truth trajectory from the KITTI Odometry Dataset.



((a)) KISS-ICP

((b)) GENZ-ICP

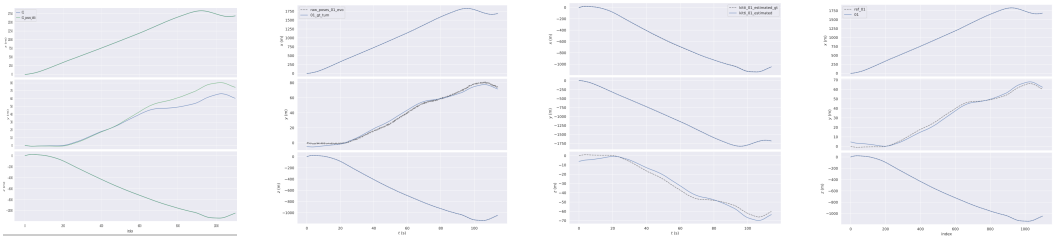
((c)) MOLA-LO

((d)) SiMpLe

Figure 7.5: Trajectory comparison for Sequence 06 (urban scenario). The four subfigures display the paths reconstructed by (a) KISS-ICP, (b) GENZ-ICP, (c) MOLA-LO, and (d) SiMpLe, overlaid with the ground-truth trajectory from the KITTI Odometry Dataset.

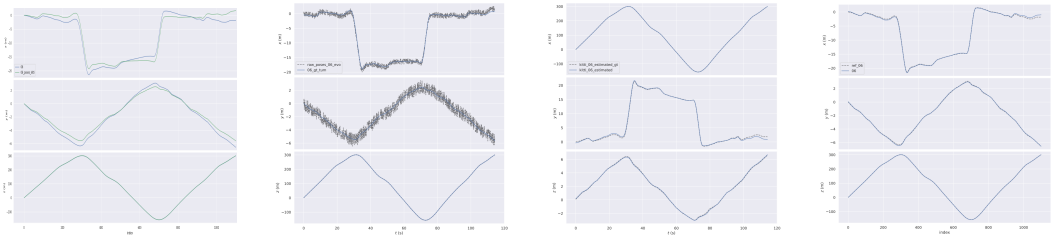
In addition to the 3D trajectory visualizations, it proved useful to look more closely at how each spatial coordinate evolves over time. By plotting the estimated and ground-truth x, y, and z components against the frame index, we can directly observe where drift tends to accumulate. In practice, this view helps identify whether the largest deviations occur in **lateral displacement**, **longitudinal motion**, or **vertical drift**. For example, in several cases the vertical component shows a slow bias build-up, even when the horizontal path appears well aligned. Figures 7.6 and 7.7 present these coordinate-wise plots for Sequences 01 and 06, allowing a detailed side-by-side comparison across all four algorithms.

To link numerical and visual analysis, we also introduced time–error plots (Figure 7.8) summarizing the main performance patterns observed. These plots show, for example, the low cumulative drift achieved by SiMpLe in steady-motion sequences and the rotational instability occasionally seen in GENZ–ICP during complex urban paths like Sequence 06. Such visualizations make error evolution easier to interpret frame by frame.



((a)) KISS-ICP ((b)) GENZ-ICP ((c)) MOLA-LO ((d)) SiMpLe

Figure 7.6: Evolution of estimated and ground-truth x, y, and z coordinates over frame index for sequence 01, shown for all four algorithms. Highlights axis-specific deviations in the sparse-road scenario.

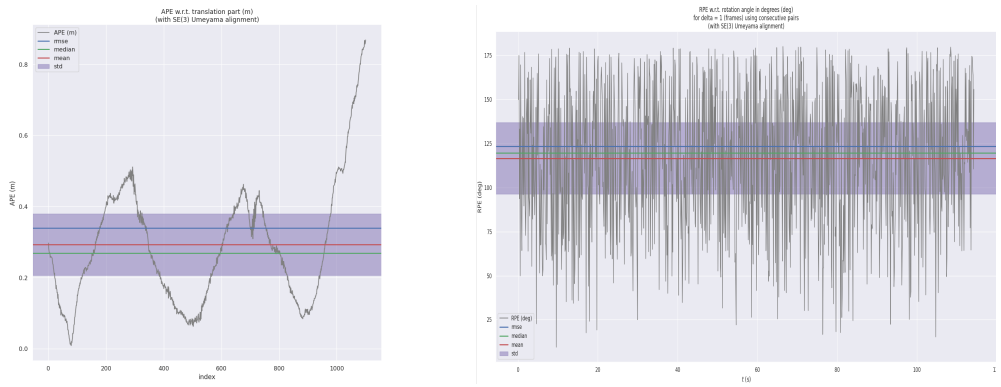


((a)) KISS-ICP ((b)) GENZ-ICP ((c)) MOLA-LO ((d)) SiMpLe

Figure 7.7: Evolution of estimated and ground-truth x, y, and z coordinates over frame index for sequence 06, shown for all four algorithms. Highlights axis-specific deviations in the sparse-road scenario.

7.2.3 Critical Analysis

In this section, we provide a critical analysis of the odometry results obtained on the KITTI dataset, integrating both the quantitative metrics and the qualitative trajectory observations presented earlier. Our discussion aims to fully explain the observed performance trends, including any slight deviations from the results reported in the original publications. We examine the relative performance of the different algorithms, explicitly highlighting why certain methods consistently outperform others under specific driving and environmental conditions. e also investigated the variability



((a)) APE w.r.t. translational part by SiMpLe ((b)) RPE w.r.t. rotational part by GenZ

Figure 7.8: Detailed Qualitative Analysis of Best- and Worst-Case Lidar Odometry Performance on the KITTI Odometry Dataset.

in performance across different sequences, analyzing how factors like environmental complexity, moving objects, or limited structural features affect trajectory accuracy. This examination helped clarify each algorithm's inherent strengths and weaknesses and provided the context needed to interpret both numerical metrics and visual outcomes.

7.2.3.1 Interpreting Quantitative Trends and Error Dynamics

This subsection provides a detailed interpretation of the quantitative results derived from the official KITTI odometry metrics and the `evo` toolkit, clarifying key differences in overall performance and assessing robustness among the four evaluated algorithms.

SiMpLe consistently demonstrates the best overall balance and statistical consistency. Its lowest average rotational error (0.0014 deg/m) across the entire dataset emerges as the single most critical factor contributing to its high reliability. This superior rotational stability strongly suggests a sophisticated and robust feature weighting strategy that effectively minimizes the influence of less constrained degrees of freedom, particularly the yaw axis, which is highly prone to drift. This outcome translates directly into highly reproducible results that align closely with the original publication's benchmarks.

KISS-ICP follows immediately, achieving near-optimal translational accuracy (0.56%). This high performance is a hallmark of the Generalized-ICP (G-ICP) approach, which efficiently capitalizes on the prevalent planar and structured features of the KITTI environment by performing a weighted minimization that robustly considers local surface geometry. Its accuracy powerfully confirms its efficacy as a fast, feature-preserving, geometry-aware solution.

MOLA-LO provides solid, mid-range performance, with its error metrics clustering near the group average. This behavior is typical of filter-based or slightly more complex optimization architectures: they attain necessary stability but require highly specific, scenario-tuned parameters to maximize performance.

GENZ-ICP emerged as a clear outlier, producing the largest translational and rotational errors among all tested methods. This behavior indicates deep issues in its optimization or outlier-handling mechanisms and confirms that it is not well-suited for general-purpose odometry tasks, particularly outside controlled conditions.

Analyzing the error accumulation across increasing segment lengths (Figures 7.1 and 7.2) allows us to assess the algorithms' capacity for long-term drift suppression. The SiMpLe and KISS-ICP curves are characterized by the flattest slope for both translational and rotational errors. This linearity confirms that their incremental er-

rors (RPE) are consistently managed and prevented from propagating uncontrollably, demonstrating strong frame-to-frame convergence and stable geometric constraint management.

On the contrary, **GENZ-ICP** displayed the most rapid accumulation of error, suggesting that its local registration errors were not being properly corrected in subsequent iterations. As a result, local misalignments quickly evolved into global drift, making the approach unsuitable for longer trajectories.

7.2.3.2 Outlier Detection and Catastrophic Failure Modes

The detailed `evo` statistics, particularly the Maximum values of the RPE Rotational metric, shift the focus of our analysis from smooth drift accumulation to acute, catastrophic failure modes and the effectiveness of outlier rejection.

The extremely high RPE rotational maximums observed for **GENZ-ICP** (for example, 179.987 deg on Sequence 06) clearly indicate a solver breakdown. This failure can be traced to weaknesses in its robust weighting strategy or the use of an overly optimistic motion prediction. When faced with ambiguous correspondences, the least-squares optimization converged to a false local minimum, effectively flipping the LiDAR frame by 180 deg and causing a complete loss of alignment.

In sharp contrast, the low rotational Standard Deviation (`Std`) and Max values of **SiMpLe** clearly demonstrate its success in outlier suppression. SiMpLe's superior performance is a direct result of its pipeline prioritizing the quality of correspondences and employing robust statistics to aggressively down-weight points that induce high instantaneous errors. This methodological design preserves the integrity of the transformation matrix, even when localized noise or transient occlusions occur.

7.2.3.3 Algorithmic Response to Feature Density

The comparison between the feature-sparse Sequence 01 and the feature-rich Sequence 06 provides the essential acid test for each algorithm's intrinsic feature dependency and modeling strategy.

In the feature-rich urban setting of Sequence 06 (Figure 7.3(b)), the environment provides an abundance of distinctive geometric features. Under these conditions, both KISS-ICP and SiMpLe perform exceptionally well, reaching near-optimal APE Translational RMSE values (for example, 0.340 m for SiMpLe). This result confirms that both methods make effective use of clear planar and corner features. Interestingly, their similar performance suggests that in well-structured environments, the efficient voxel management of KISS-ICP can rival the more elaborate feature extraction used by SiMpLe.

The sparse-road environment (Sequence 01, Figure 7.3(a)) fundamentally challenges the algorithms by offering ambiguous correspondences over long, straight segments, primarily starving the yaw constraint.

MOLA-LO is the most severely penalized, recording the highest APE Translational RMSE (4.999 m). This strong performance degradation suggests that MOLA-LO's feature extraction or filtering is highly sensitive to the spatial distribution of features. When points are scarce and distant, the algorithm clearly lacks the necessary local geometric diversity to accurately constrain all six degrees of freedom.

SiMpLe, while showing some performance degradation in the vineyard data, still achieves the lowest translational APE RMSE (2.496 m). This relative success underlines the importance of its feature evaluation process. The algorithm's conservative weighting and filtering steps appear to limit the influence of uncertain correspondences, preventing them from distorting the final estimate and allowing SiMpLe to maintain greater overall stability than the other methods tested.

7.2.3.4 Summary and Implications

The critical analysis of the KITTI benchmark confirms that the performance of modern LiDAR Odometry is ultimately determined by its architectural response to optimization instability.

- **SiMpLe**'s leading performance stems from its robust feature handling and superior rotational stability, making it the most reliable solution for generalized application, as it effectively manages both high-feature complexity and low-feature ambiguity.
- **KISS-ICP** is confirmed as the most computationally efficient option for environments with structured, abundant features, leveraging its optimized G-ICP implementation.
- **GENZ-ICP** clearly illustrates how fragile odometry systems can become when they lack strong solver robustness. In this case, temporary motion irregularities or ambiguous data points can trigger severe rotational failures that the algorithm cannot recover from.

Ultimately, this study underscores that for practical, long-term autonomous navigation, the rigor of the feature validation and optimization strategy (low RPE Max) is a far more critical determinant of overall system reliability than achieving marginal gains in mean translational accuracy.

7.3 Results on the Vineyard Dataset

We further evaluated the performance of the four LiDAR odometry algorithms on the Vineyard Dataset, a private and exceptionally challenging benchmark. This environment is uniquely characterized by repetitive vegetation rows, a scarcity of distinctive landmarks, and several tight, sharp turns. Unlike the structured, urban-like KITTI benchmark, this agricultural setting introduces specific, difficult challenges such as structural monotony, dense vegetation noise, and non-rigid features, all of which substantially impair robust scan matching and accurate pose estimation.

This section presents both the numerical and visual results obtained on the vineyard dataset. Quantitative analyses were carried out using the `evo` framework, focusing on Absolute Pose Error (APE) and Relative Pose Error (RPE), for both translational and rotational components. These metrics capture not only accumulated drift but also short-term pose accuracy, providing a solid basis for comparing algorithm performance under realistic agricultural conditions.

The qualitative evaluation complements these numerical results through trajectory visualizations generated with `evo_traj`, including 3D paths, 2D projections, and overlays with the ground truth trajectory. We also leveraged `evo_res` to generate box plots and aggregated statistics, summarizing performance trends across different trajectory segments.

Additionally, we include specific comparisons between straight vineyard rows and sharp turning maneuvers to clearly highlight how environmental geometry significantly affects algorithm behavior. Presenting these detailed results within this section ensures that the vineyard-specific challenges and their impact on LiDAR odometry are thoroughly understood before we proceed to the broader cross-dataset comparisons in the subsequent chapter.

7.3.1 Quantitative Results

7.3.1.1 Overview of Evaluation Metrics and Methodology

We carried out the quantitative evaluation of LiDAR odometry performance on the vineyard dataset using the `evo` toolkit, which provides an indispensable, robust framework for trajectory comparison and error analysis. Specifically, we employed `evo_ape` (Absolute Pose Error) and `evo_rpe` (Relative Pose Error) to compute both the translational and rotational components of the error across the entire trajectory. These complementary metrics are essential for assessing both global consistency (APE) and local drift behavior (RPE).

To provide a broader and deeper statistical perspective, we used `evo_res` to generate aggregated statistics and comparative visualizations among all four algorithms.

For each error type—translational error and rotational error in degrees—*evo_res* produced:

- Time-series plots of error evolution over the full trajectory
- Histograms comparing the four algorithms for key statistics (mean, median, minimum, maximum, RMSE, and standard deviation)
- Box plots and violin plots to highlight the distribution, spread, and outliers.

All metrics were computed along the full trajectory using a high-precision GPS reference, ensuring consistent and fair comparison across methods. To better understand performance in different motion scenarios, the trajectory was divided into ten straight rows and nine turning segments. Each metric (`evo_ape` and `evo_rpe`, for both components) was recalculated separately for these two motion types. This segmentation allows a clearer view of how each algorithm behaves in repetitive linear paths versus sharp directional changes.

7.3.1.2 Absolute Pose Error (APE)

This subsection presents the quantitative evaluation of the four LiDAR odometry algorithms on the vineyard dataset. We performed all computations using the `evo` toolkit to ensure both consistency and reproducibility [34]. We evaluated the errors over the entire recorded trajectory, critically using the exact same GPS ground truth reference and trajectory length for all methods, thus guaranteeing direct comparability [34]. Our quantitative analysis focuses on the Absolute Pose Error (APE) and Relative Pose Error (RPE), considering both their translational and rotational components [34].

Two summary tables are provided for the APE results. Table 7.13 lists the translational errors, expressed in meters, while Table 7.14 reports the rotational components in degrees. Each table includes standard statistical indicators such as mean, median, minimum, maximum, standard deviation, and RMSE. Together, they give a concise yet complete picture of the absolute positioning accuracy achieved by the evaluated algorithms.

69) To complement these tables, we also present the corresponding `evo_res` visualizations, which show the detailed behavior of the APE errors. For both translational and rotational components, four plots are displayed: the **time evolution** of the error, a **histogram** showing its distribution, a **box plot** summarizing spread and outliers, and a **violin plot** depicting the distribution's overall shape. These visualizations offer a more intuitive grasp of the error dynamics and effectively support the quantitative results.

Figures 7.9 and 7.10 present the `evo_res` visualizations of the Absolute Pose Error (APE) for the translational and rotational components, respectively. For each

	Max	Mean	Median	Min	Std	Rmse
MOLA-LO	**1.970**	**0.682**	**0.652**	**0.123**	**0.241**	**0.723**
KISS-ICP	24.677	5.672	5.666	0.283	2.967	6.402
GENZ-ICP	30.259	6.779	4.452	0.618	5.905	8.989
SiMpLe	68.445	21.505	17.714	0.563	15.231	26.352

Table 7.13: Absolute Pose Error (APE) statistics for the translational component of the error (in meters) on the vineyard dataset. The table reports mean, median, minimum, maximum, standard deviation, and RMSE for each of the four evaluated algorithms. Lowest values are highlighted in **bold**.

	Max	Mean	Median	Min	Std	Rmse
KISS-ICP	**27.890**	**19.049**	**20.569**	**0.255**	**6.868**	**20.249**
MOLA-LO	38.663	24.518	31.775	8.610	12.062	27.324
GENZ-ICP	74.874	35.839	26.886	2.121	27.589	45.228
SiMpLe	173.232	54.962	51.375	25.463	18.148	57.881

Table 7.14: Absolute Pose Error (APE) statistics for the rotational component of the error (in degrees) on the vineyard dataset. The table reports mean, median, minimum, maximum, standard deviation, and RMSE for each of the four evaluated algorithms. Lowest values are highlighted in **bold**.

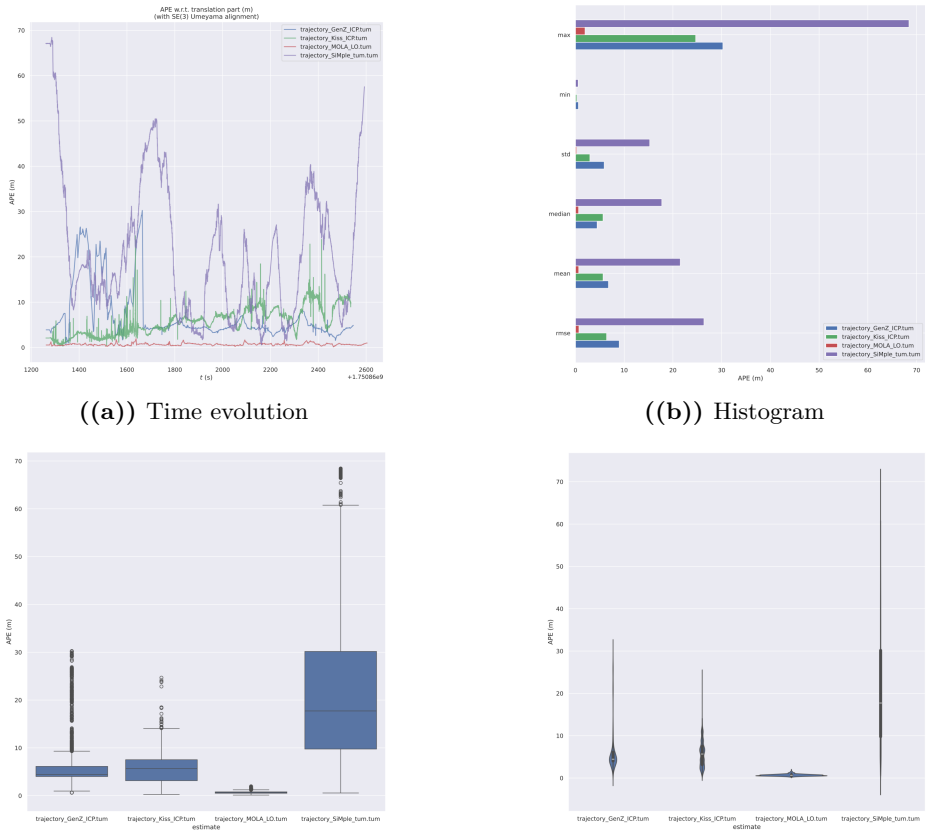
metric, we provide four essential, complementary plots: the temporal evolution of the error along the trajectory, a histogram detailing the error distribution, a box plot summarizing the spread and the outliers, and a violin plot illustrating the underlying distribution shape. These comprehensive visualizations enable a detailed, multi-faceted assessment of the error behavior and clearly highlight the performance differences among the four LiDAR odometry algorithms.

7.3.1.3 Relative Pose Error (RPE)

This subsection delivers the quantitative evaluation of the four LiDAR odometry algorithms, specifically focusing on the Relative Pose Error (RPE) within the vineyard dataset. We performed all computations using the `evo` toolkit to ensure consistency and full reproducibility. As with the APE analysis, we evaluated the errors over the entire recorded trajectory using the same ground-truth GPS reference, ensuring all methods remain directly comparable. This quantitative assessment concentrates on the RPE, examining both its translational and rotational components.

70) Two additional tables summarize the RPE analysis. Table 7.15 presents the translational results (in meters), while Table 7.16 reports the rotational values (in degrees). Each table lists the key statistical descriptors—mean, median, minimum, maximum, standard deviation, and RMSE—providing a clear overview of local pose accuracy and short-term stability for each algorithm.

To complement these numerical summaries, we include the corresponding `evo_res` visualizations, presented in Figures 7.11 and 7.12. These figures are essential for



((a)) Time evolution

((b)) Histogram

((c)) Box plot

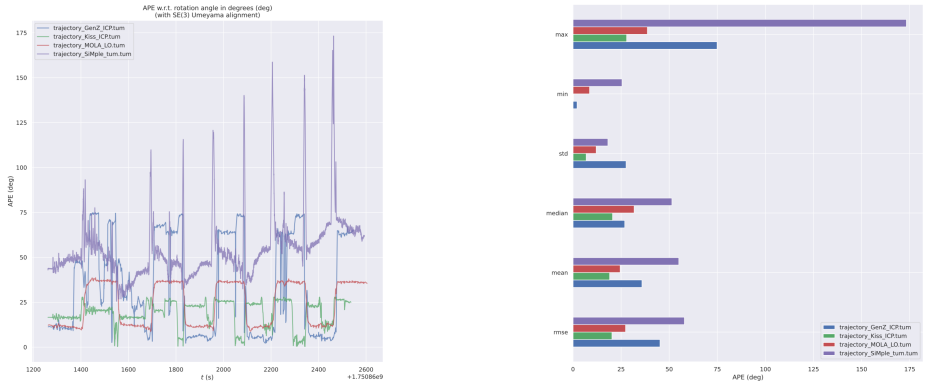
((d)) Violin plot

Figure 7.9: Translational Absolute Pose Error of the four algorithms, shown with (a) temporal evolution, (b) histogram, (c) box plot, and (d) violin plot.

moving beyond tabular data and visually assessing the algorithms' local stability. For each of the two RPE metrics (rotational and translational), we provide four key complementary plots:

- The temporal evolution of the error along the trajectory,
- A histogram detailing the error distribution,
- A box plot summarizing the spread and outliers, and
- A violin plot illustrating the underlying distribution shape.

These visualizations allow a detailed assessment of the error behavior and clearly highlight the differences in performance among the four LiDAR odometry algorithms under the repetitive vineyard conditions.



((a)) Time evolution

((b)) Histogram

((c)) Box plot

((d)) Violin plot

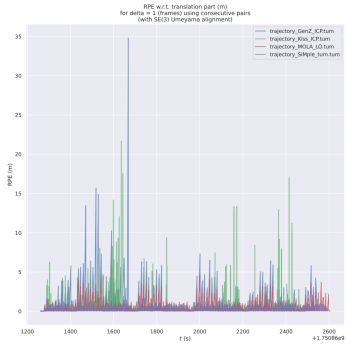
Figure 7.10: Rotational Absolute Pose Error of the four algorithms, shown with (a) temporal evolution, (b) histogram, (c) box plot, and (d) violin plot.

	Max	Mean	Median	Min	Std	Rmse
MOLA-LO	4.607	0.192	**0.074**	0.002	0.424	0.466
KISS-ICP	21.691	0.292	0.146	**0.000**	0.743	0.798
GENZ-ICP	34.812	0.371	0.121	0.002	1.143	1.202
SiMpLe	**2.305**	**0.189**	0.129	0.003	**0.181**	**0.261**

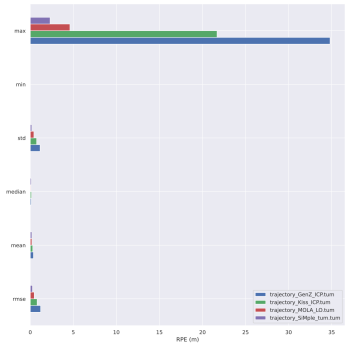
Table 7.15: Relative Pose Error (RPE) statistics for the translational component of the error (in meters) on the vineyard dataset. The table reports mean, median, minimum, maximum, standard deviation, and RMSE for each of the four evaluated algorithms. Lowest values are highlighted in **bold**.

	Max	Mean	Median	Min	Std	Rmse
MOLA-LO	14.658	0.725	0.565	0.001	0.755	1.047
KISS-ICP	**7.993**	**0.309**	**0.196**	**0.000**	**0.495**	**0.583**
GENZ-ICP	73.665	1.411	0.656	0.001	5.140	5.330
SiMpLe	8.836	0.986	0.824	0.006	0.700	1.210

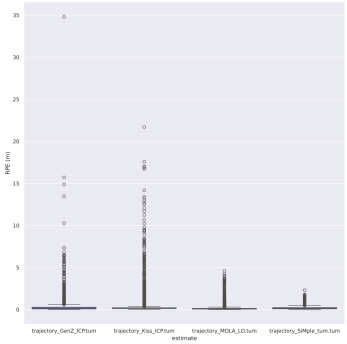
Table 7.16: Relative Pose Error (RPE) statistics for the rotational component of the error (in degrees) on the vineyard dataset. The table reports mean, median, minimum, maximum, standard deviation, and RMSE for each of the four evaluated algorithms. Lowest values are highlighted in **bold**.



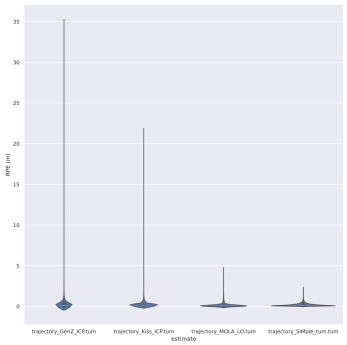
((a)) Time evolution



((b)) Histogram

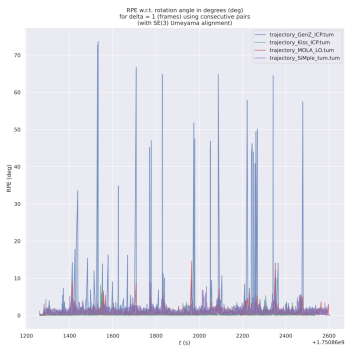


((c)) Box plot

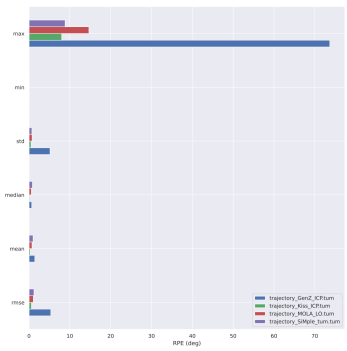


((d)) Violin plot

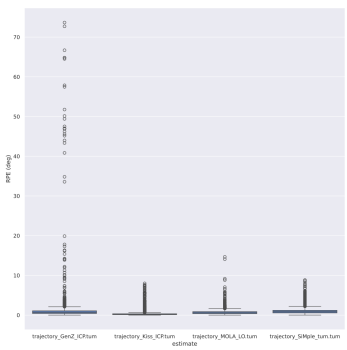
Figure 7.11: Translational Relative Pose Error of the four algorithms, shown with (a) temporal evolution, (b) histogram, (c) box plot, and (d) violin plot.



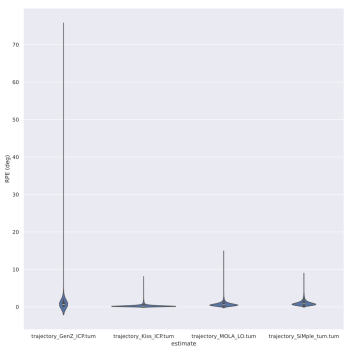
((a)) Time evolution



((b)) Histogram



((c)) Box plot



((d)) Violin plot

Figure 7.12: Rotational Relative Pose Error of the four algorithms, shown with (a) temporal evolution, (b) histogram, (c) box plot, and (d) violin plot.

7.3.1.4 Straight Rows vs Turning Maneuvers Analysis

To comprehensively analyze how trajectory structure influences LiDAR odometry performance, we segmented the vineyard dataset into distinct **straight row segments** and **turning maneuvers**. Straight rows feature highly repetitive visual patterns that can dramatically amplify drift, whereas turning maneuvers involve sharp directional changes that severely challenge scan-matching algorithms. Segmenting the trajectory in this manner allows for a focused, controlled evaluation of algorithmic behavior under these two fundamentally distinct motion patterns.

For each motion category, all APE and RPE metrics were computed separately for both translational and rotational components. To ensure clarity, we aggregated the statistics across all segments within each category, averaging the six key indicators—mean, median, minimum, maximum, standard deviation, and RMSE—independently for straight paths and turning maneuvers.

Tables 7.17 through 7.20 present these aggregated results. Each table corresponds to a specific metric: translational APE (7.17), rotational APE (7.18), translational RPE (7.19), and rotational RPE (7.20). Within each table, the four algorithms are reported as rows, with alternating background shading used to clearly distinguish straight row segments (white) from turning maneuvers (gray). Columns report the six statistical metrics, facilitating a clear and immediate comparison across both algorithms and motion scenarios.

The final column in Tables 7.17 and 7.18 introduces the **relative average error**, a key metric for normalizing performance across segments. We calculated this value by dividing the root mean square error (RMSE) for each segment by its average path length. For the translational APE table, the relative error is expressed as a percentage of the path length, while for the rotational APE table, it is given in degrees per meter. This normalization is critical because it accounts for the significant difference in length between the long straight rows and the short curved turns. By normalizing the error, the metric allows for a direct comparison of algorithmic performance on a per-meter basis, eliminating any bias introduced by segment length variations and providing a more robust measure of long-term drift. We assume a constant speed throughout the trajectory to ensure consistency in this analysis.

	Max	Mean	Median	Min	Std	Rmse	Relative (%)
MOLA-LO row	0,801	0,230	0,199	0,042	0,136	0,269	0,266
MOLA-LO turn	130,244	0,526	0,500	0,248	0,177	0,559	2,747
KISS-ICP row	10,714	2,353	1,903	0,547	1,558	2,853	2,790
KISS-ICP turn	2,697	0,929	0,820	0,129	0,527	1,074	5,604
GENZ-ICP row	7,622	3,122	2,736	0,338	1,842	3,631	3,589
GENZ-ICP turn	1,772	0,875	0,862	0,172	0,385	0,965	4,741
SiMpLe row	24,932	10,098	9,269	0,190	6,347	11,969	11,881
SiMpLe turn	6,250	2,418	2,213	0,630	1,229	2,722	12,255

Table 7.17: Aggregated translational Absolute Pose Error (APE) for straight row and turning maneuver segments, with alternating row shading to differentiate segment types. Columns report mean, median, minimum, maximum, standard deviation, and RMSE.

	Max	Mean	Median	Min	Std	Rmse	Relative (deg/m)
MOLA-LO row	59,086	57,236	57,288	55,222	0,703	57,242	0,566
MOLA-LO turn	36,277	24,602	25,596	14,628	9,086	24,628	1,210
KISS-ICP row	54,145	49,181	50,976	42,702	3,672	51,854	0,507
KISS-ICP turn	25,846	14,384	11,652	5,810	7,329	16,305	0,851
GENZ-ICP row	85,016	77,827	79,257	59,968	5,652	78,437	0,775
GENZ-ICP turn	69,789	32,679	27,852	15,466	18,751	39,451	1,938
SiMpLe row	93,870	71,134	71,322	60,845	5,57	71,506	0,710
SiMpLe turn	92,127	52,906	54,593	16,985	23,307	51,887	2,355

Table 7.18: Aggregated rotational Absolute Pose Error (APE) for straight row and turning maneuver segments, with alternating row shading to differentiate segment types. Columns report mean, median, minimum, maximum, standard deviation, and RMSE.

	Max	Mean	Median	Min	Std	Rmse
MOLA-LO row	2,675	0,198	0,008	0,011	0,391	0,439
MOLA-LO turn	1,189	0,128	0,066	0,013	0,210	0,247
KISS-ICP row	9,961	0,306	0,152	0,015	0,669	0,741
KISS-ICP turn	1,423	0,207	0,098	0,007	0,26	0,337
GENZ-ICP row	8,757	0,393	0,135	0,018	0,969	1,051
GENZ-ICP turn	1,332	0,191	0,105	0,017	0,253	0,319
SiMpLe row	1,494	0,203	0,144	0,011	0,184	0,274
SiMpLe turn	0,785	0,122	0,093	0,01	0,104	0,161

Table 7.19: Aggregated translational Relative Pose Error (RPE) for straight row and turning maneuver segments, with alternating row shading to differentiate segment types. Columns report mean, median, minimum, maximum, standard deviation, and RMSE.

	Max	Mean	Median	Min	Std	Rmse
MOLA-LO row	3,411	0,681	0,577	0,055	0,471	0,830
MOLA-LO turn	8,464	1,058	0,647	0,127	1,507	1,862
KISS-ICP row	2,409	0,280	0,192	0,008	0,326	0,435
KISS-ICP turn	2,151	0,482	0,292	0,026	0,494	0,703
GENZ-ICP row	21,900	1,008	0,586	0,067	2,434	2,689
GENZ-ICP turn	50,670	2,485	0,744	0,146	7,961	8,372
SiMpLe row	5,093	0,900	0,794	0,061	0,578	1,073
SiMpLe turn	4,468	1,489	1,313	0,084	0,926	1,754

Table 7.20: Aggregated rotational Relative Pose Error (RPE) for straight row and turning maneuver segments, with alternating row shading to differentiate segment types. Columns report mean, median, minimum, maximum, standard deviation, and RMSE.

7.3.2 Qualitative Results

Beyond the numerical scores, it is often the visual analysis of trajectories that reveals how each odometry system truly behaves in the field. In the vineyard environment, visual comparisons help to capture effects that numbers alone cannot—such as how the algorithms cope with repetitive vine rows, dense foliage, and the abrupt turning maneuvers typical of agricultural navigation.

To better visualize these outcomes, representative trajectory plots were produced using *evo_traj*, directly comparing the estimated paths with the GPS-based ground truth. Both 3D views and 2D top-down projections were included, offering complementary perspectives that emphasize global accuracy as well as localized drift. Particular attention was paid to sections where repetitive geometry or sharp direction changes typically cause alignment errors.

These visualizations work as the natural counterpart to the quantitative results, helping to expose subtle error patterns that aggregate metrics may overlook. Observing full trajectories from different perspectives also makes it easier to understand where and why deviations appear, turning the statistical summaries into tangible visual evidence of each algorithm's behavior.

Figure 6.2 presents an overhead, satellite-based view of the robot's route across the vineyard. This path, recorded via GPS, acts as the ground truth baseline for all subsequent odometry evaluations. The start and end points are clearly labeled, and the coordinate axes correspond to latitude and longitude, with North indicated for spatial reference. This overview establishes the physical context in which the LiDAR trajectories should be interpreted.

Figures 7.13 to 7.16 show the estimated trajectories produced by the four LiDAR odometry algorithms. Each figure contains two panels: the left side displays the 2D top-down projection, while the right side offers the full 3D trajectory overlaid with the GPS reference path. This dual view highlights both planar deviations and vertical inconsistencies, providing a more complete picture of performance under the vineyard's complex terrain.

The combined use of 2D and 3D representations allows for immediate recognition of localized errors while keeping the visual comparison straightforward and readable. Together, these qualitative results complement the numerical analyses, clarifying how each algorithm adapts—or fails to adapt—to the subtle, repetitive geometry of real-world agricultural environments.

Figures 7.17–7.20 depict how each LiDAR odometry algorithm evolves over time across all six degrees of freedom. For every method, two subplots are provided: the

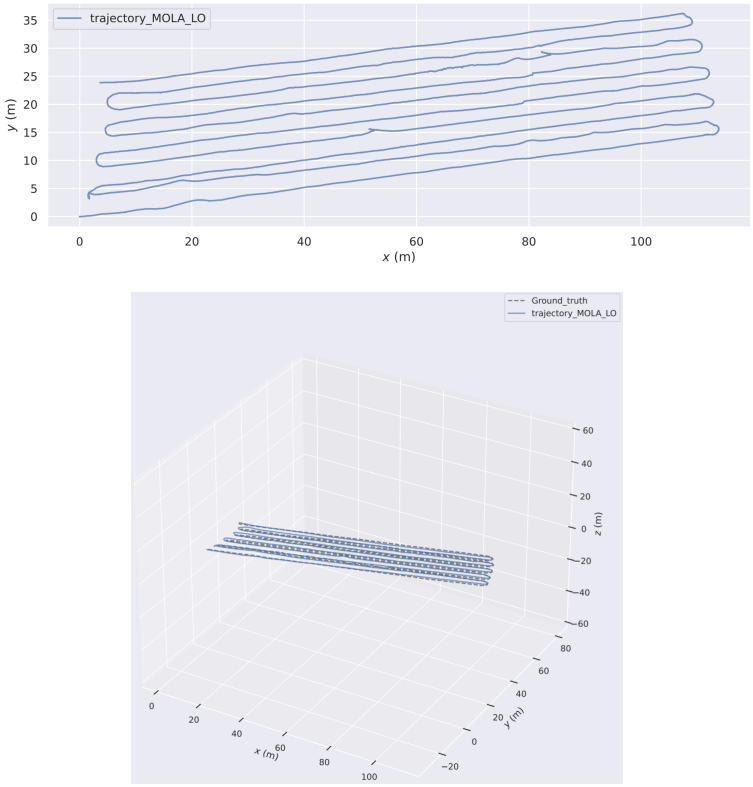


Figure 7.13: Estimated trajectory of MOLA-LO. Top: 2D top-down projection; Down: 3D trajectory overlaid on the ground truth.

first presents the translational components $x(t)$, $y(t)$, and $z(t)$, while the second displays the rotational components—roll, pitch, and yaw—across the same temporal window.

Analyzing these plots makes it possible to see where errors concentrate along specific spatial axes. Deviations from the ground truth in the translational curves indicate the directions most affected by drift or misalignment, while discrepancies in the rotational plots reveal orientation instabilities that may disrupt overall trajectory consistency.

To enable a direct visual comparison among the four LiDAR odometry algorithms, Figure 7.21 overlays all estimated trajectories with the GPS-based ground truth. This combined visualization clearly shows how closely each method aligns with the reference path, while also exposing the regions where deviations emerge. In a single view, it offers a compact yet informative summary of overall trajectory accuracy and spatial consistency.

In addition to the trajectory overlays, Figure 7.22 shows how translational and rotational components evolve over time for all four algorithms. This combined view allows a direct comparison of how pose estimates diverge across the six degrees of freedom, offering a clear and compact understanding of translation and rotation behavior side by side.

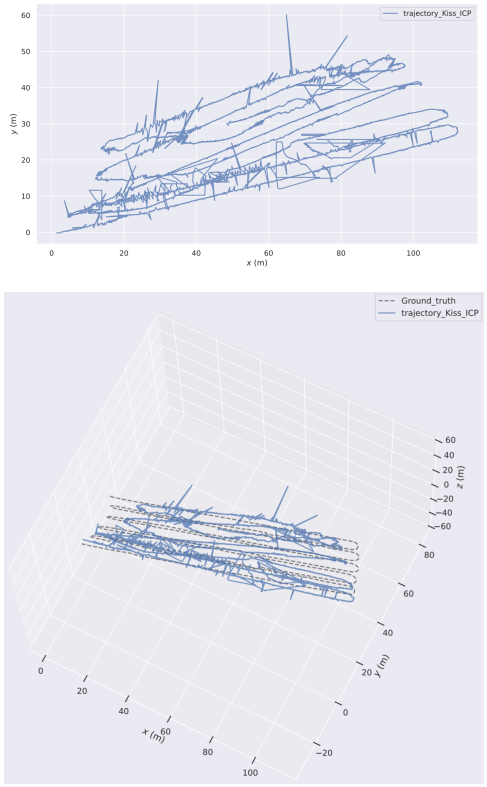


Figure 7.14: Estimated trajectory of KISS-ICP. Top: 2D top-down projection; Down: 3D trajectory overlaid on the ground truth.

7.3.3 Critical Analysis

The critical analysis presented in this section evaluates the performance of the four LiDAR odometry algorithms in the complex agricultural setting of the vineyard dataset. Unlike the highly structured KITTI benchmark, the vineyard introduces distinct challenges that can strongly influence scan matching and pose estimation. These include long, repetitive vine rows that promote cumulative drift, dense foliage that adds irregular noise and non-rigid features, and sharp U-turns that severely test each algorithm's ability to adapt to sudden direction changes.

This discussion connects the quantitative indicators—such as translational and rotational APE and RPE, aggregate statistics, and relative error measures—with the qualitative visualizations and component-wise comparisons. By correlating numerical trends with visual deviations, it provides a comprehensive explanation of why certain methods sustain stable performance while others suffer localized failures or systematic drift when exposed to agricultural conditions.

7.3.3.1 Interpretation of Quantitative Trends

This subsection interprets the quantitative outcomes obtained from the vineyard dataset, focusing on how each algorithm balances translational and rotational accuracy over time. The analysis extends beyond numerical metrics to identify patterns

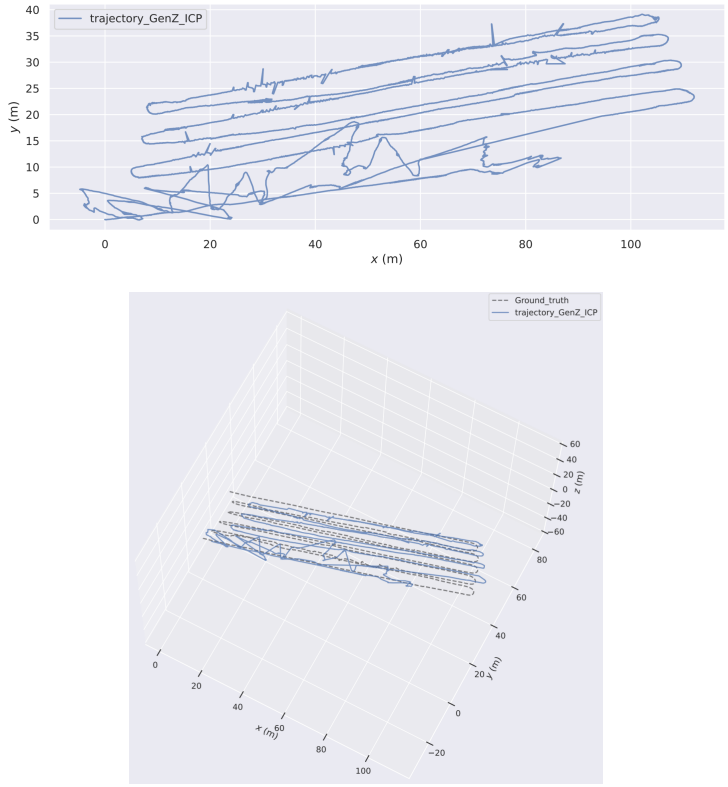


Figure 7.15: Estimated trajectory of GENZ-ICP. Top: 2D top-down projection; Down: 3D trajectory overlaid on the ground truth.

in robustness, cumulative drift, and error dispersion, revealing how specific vineyard characteristics—such as repetitive geometry and abrupt turns—influence performance.

MOLA-LO stands out as the best-performing algorithm overall. It achieves the lowest translational APE (mean 0.682 m) and RPE (mean 0.192 m), with moderate rotational APE and RPE values (24.518 deg and 0.725 deg/m). Even in curved sections, errors increase only slightly—about 0.266% on average. This consistency stems from MOLA-LO's effective filtering and optimization steps, which minimize the impact of vegetation noise and keep the estimated path closely aligned with the ground truth.

KISS-ICP delivers intermediate results, with an average translational APE of 5.672 m and translational RPE of 0.292 m. Rotational errors remain moderate (APE = 19.049 deg, RPE = 0.309 deg/m). Although stable overall, the algorithm tends to accumulate drift over long, straight paths, and yaw deviations become more noticeable during sharp turns. This pattern aligns with the pairwise nature of ICP, which performs well under mild noise but lacks strong global correction, making it more sensitive to fast orientation changes.

GENZ-ICP exhibits high initial error that progressively decreases. Average translational APE reaches 6.779 m, with RPE = 0.371 m, and rotational APE = 35.839 deg, RPE = 1.411 deg/m. Errors peak at the start—where repetitive vine

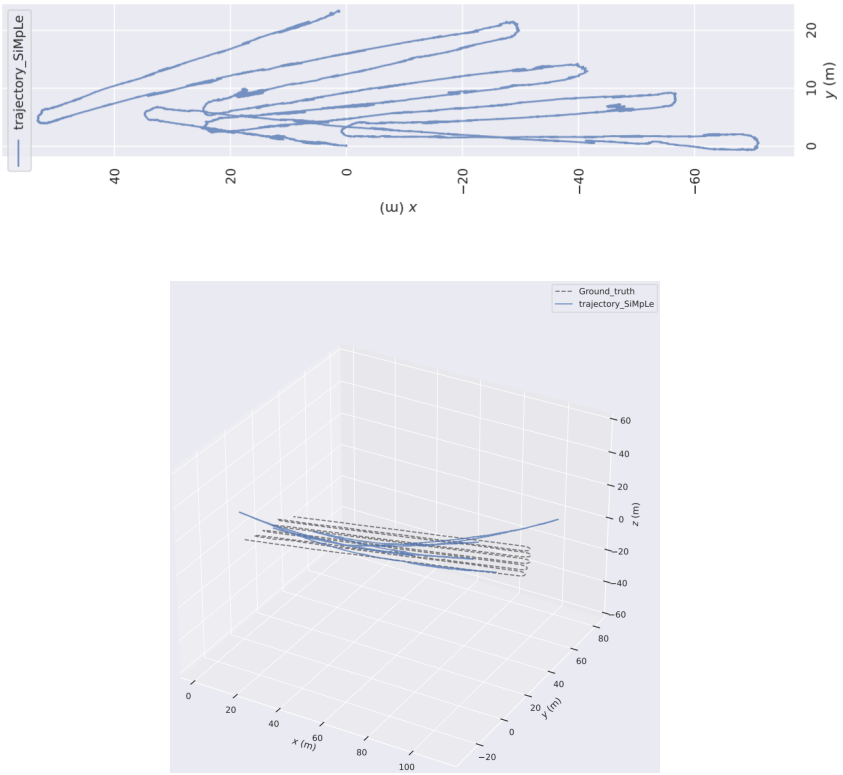
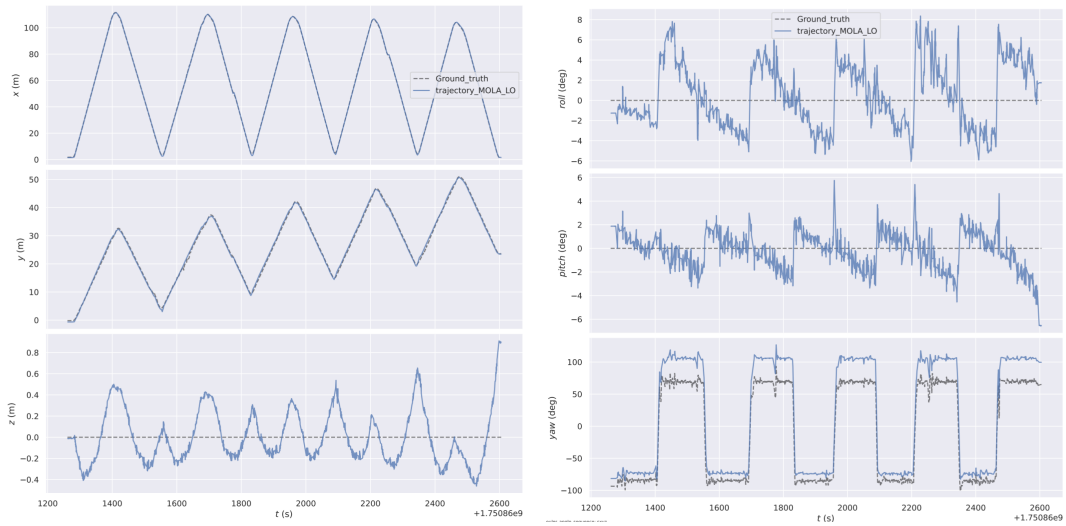


Figure 7.16: Estimated trajectory of SiMpLe. Top: 2D top-down projection; Down: 3D trajectory overlaid on the ground truth.

structures trigger misalignments—but later stabilize as the algorithm converges toward the ground truth. Despite partial recovery, residual yaw offsets persist during curves. This early instability underscores GENZ-ICP's dependence on accurate initial alignment and its vulnerability to ambiguous geometric patterns.

SiMpLe performs poorest overall. Its translational APE averages 21.505 m (RPE = 0.189 m), and rotational APE and RPE reach 54.962 deg and 0.986 deg/m, respectively. Drift accumulates along straight rows and becomes extreme during turns, with vertical deviations up to ±15 m and roll/pitch errors surpassing ±40–60 deg. These issues indicate strong sensitivity to environmental noise and the absence of robust filtering or global correction.

In summary, the vineyard experiment clearly highlights how crucial it is for odometry systems to handle repetitive patterns, dense vegetation, and frequent turns. Among the tested methods, MOLA-LO stands out thanks to its strong filtering and optimization framework. KISS-ICP remains generally stable, though it accumulates drift gradually over time. GENZ-ICP manages partial recovery after early misalignments, while SiMpLe experiences severe difficulties when confronted with complex and irregular environments.



((a)) Translational Components vs Time

((b)) Rotational Components vs Time

Figure 7.17: Temporal evolution of the Translational and Rotational Components of the trajectory estimated by MOLA-LO.

7.3.3.2 Interpretation of Error Dynamics Over Time

The temporal evolution of APE and RPE clarifies how each algorithm reacts to the vineyard's main challenges—long rows, tight turns, and recurring structures. Examining the time series of translational (x, y, z) and rotational (roll, pitch, yaw) components reveals the dynamics of error growth and recovery.

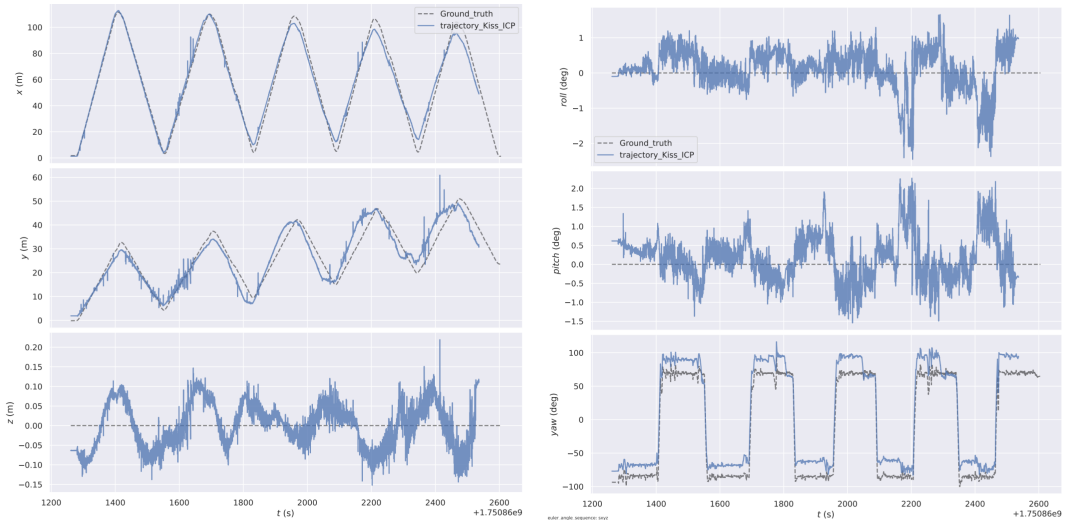
MOLA-LO remains remarkably stable, with translational APE nearly constant and vertical deviations within ± 0.4 m, peaking at 0.8 m. These small fluctuations confirm effective drift suppression and robust handling of repetitive geometry.

KISS-ICP shows moderate translational errors that grow gradually along x and y, while z stays within ± 0.1 m. High-frequency fluctuations suggest sensitivity to local geometry, without a global correction mechanism to smooth residual noise.

GENZ-ICP displays early instability: large initial errors that diminish after roughly 400 s. The algorithm stabilizes once sufficient spatial information is accumulated, though residual yaw offsets persist through turns.

SiMpLe is highly unstable, with substantial translational deviations and vertical drift up to ± 15 m. Peaks align with turning segments, indicating an inability to compensate for cumulative drift.

From a rotational perspective, all algorithms reveal a similar "square-wave" pattern in yaw during 180° turns. MOLA-LO limits errors to about $\pm 7°$ in roll, $\pm 4°$ in pitch, and roughly 30° in yaw. KISS-ICP shows smaller deviations—around $\pm 2°$ in roll and pitch, with yaw drift reaching about 20°. GENZ-ICP initially suffers from



((a)) Translational Components vs Time ((b)) Rotational Components vs Time

Figure 7.18: Temporal evolution of the Translational and Rotational Components of the trajectory estimated by KISS.

large $\approx 90°$ yaw offsets that gradually diminish, whereas SiMpLe shows fluctuations between ± 40 and $60°$, especially when turning. These results confirm that orientation changes remain a major challenge in natural, vineyard-like environments.

7.3.3.3 Data Analysis and Outlier Detection

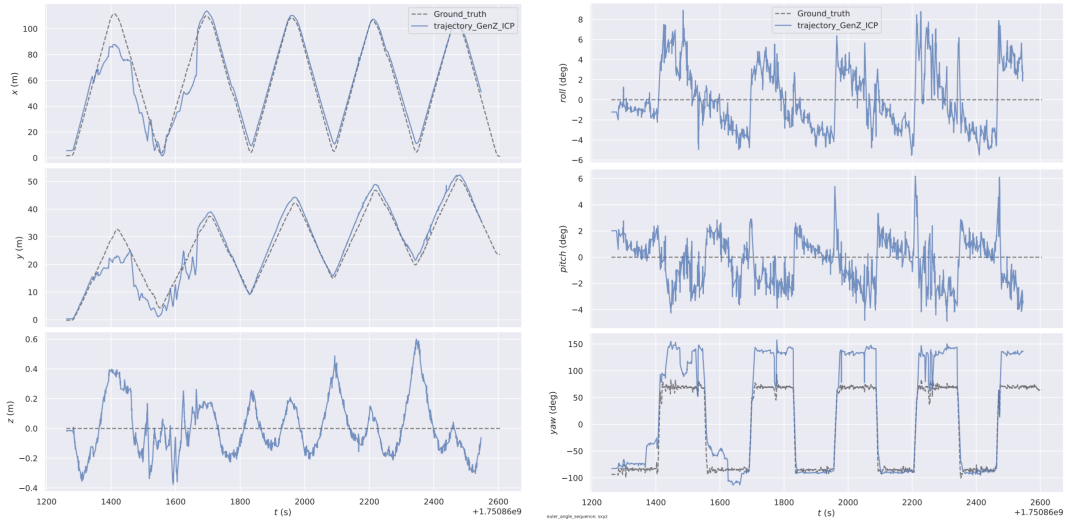
The statistical box and violin plots offer deeper insight into algorithmic stability by revealing variance, skewness, and outlier presence beyond mean metrics. The box plots summarize spread and central tendency, while violin plots display the full error density.

In terms of APE, **MOLA-LO** is the most consistent: both translational and rotational distributions are narrow, symmetric, and centered near zero, confirming minimal drift and strong reliability.

SiMpLe, conversely, shows extreme dispersion with a long right tail—clear evidence of large, infrequent error spikes that distort the mean. This asymmetric spread exposes major stability issues over time.

GENZ-ICP and **KISS-ICP** occupy intermediate positions. GENZ's rotational APE violin plot is bimodal, implying two distinct operational states—one accurate and one prone to severe error bursts around 70 deg. Both algorithms display more outliers and wider distributions than MOLA, signifying reduced robustness.

Looking at the RPE behavior, the situation changes slightly. MOLA-LO and SiMpLe produce compact, nearly symmetric error plots centered close to zero, suggesting strong short-term consistency. By contrast, GENZ-ICP and KISS-ICP display wider,



((a)) Translational Components vs Time

((b)) Rotational Components vs Time

Figure 7.19: Temporal evolution of the Translational and Rotational Components of the trajectory estimated by GENZ.

right-skewed distributions, indicating occasional local failures despite acceptable frame-to-frame accuracy. Overall, these observations imply that MOLA-LO maintains stability on both global and local scales, while the other methods—particularly SiMpLe—are more prone to cumulative drift.

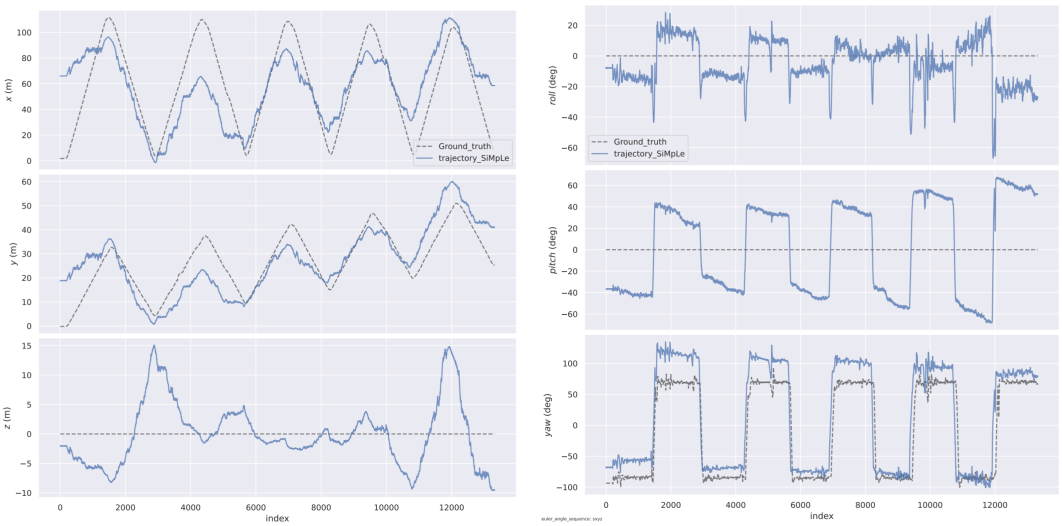
7.3.3.4 Segmented Error Analysis: Straight vs. Curved Paths

Path segmentation reveals how motion type affects performance. Straight segments dominate the trajectory length and thus heavily influence average metrics, whereas turns impose sharper dynamic loads that expose weaknesses in rotational handling.

For translational APE, **MOLA-LO** again leads: despite slightly higher errors in turns, its results remain far superior, with only one outlier linked to a specific maneuver. Surprisingly, **KISS-ICP** and **GENZ-ICP** sometimes perform better in turns than on straight paths, likely because turns introduce richer geometric constraints. **SiMpLe**, however, fails in both contexts, with severe degradation during rotations.

Rotationally, the trend intensifies. MOLA-LO preserves accuracy even under abrupt changes, while GENZ-ICP and SiMpLe experience steep error increases, demonstrating their limited resilience to fast orientation shifts.

RPE analysis complements these findings: MOLA-LO and SiMpLe maintain excellent local accuracy with low dispersion, even during curves. KISS-ICP and GENZ-ICP, instead, show higher variability, pointing to unstable short-term alignment.



((a)) Translational Components vs Time

((b)) Rotational Components vs Time

Figure 7.20: Temporal evolution of the Translational and Rotational Components of the trajectory estimated by SiMpLe.

Overall, the segmented results reinforce MOLA-LO's leading performance. It consistently converts local reliability into globally precise trajectories, unlike GENZ-ICP and KISS-ICP, which lose accuracy under complex motion, or SiMpLe, whose drift rapidly grows over time.

7.3.3.5 Summary of Results and Implications

The comprehensive analysis confirms clear performance hierarchies. **MOLA-LO** proves the most reliable, achieving low APE and RPE, minimal variance, and symmetric error distributions. Its ability to preserve stability across both straight and curved sections demonstrates effective drift control—essential in agricultural navigation, where straight paths dominate total travel.

SiMpLe displays large dispersion and positive skewness, confirming instability and noise sensitivity. Its good short-term accuracy fails to translate into global consistency. **GENZ-ICP** and **KISS-ICP** perform moderately but remain prone to drift on long paths and instability during turns; GENZ-ICP's bimodal error behavior particularly reflects abrupt degradation rather than gradual decline.

In conclusion, while several algorithms show acceptable local precision, only **MOLA-LO** successfully converts local robustness into globally coherent trajectories. Its combination of filtering, optimization, and resilience to repetitive geometry makes it the most suitable solution for LiDAR-based odometry in complex, unstructured agricultural environments.

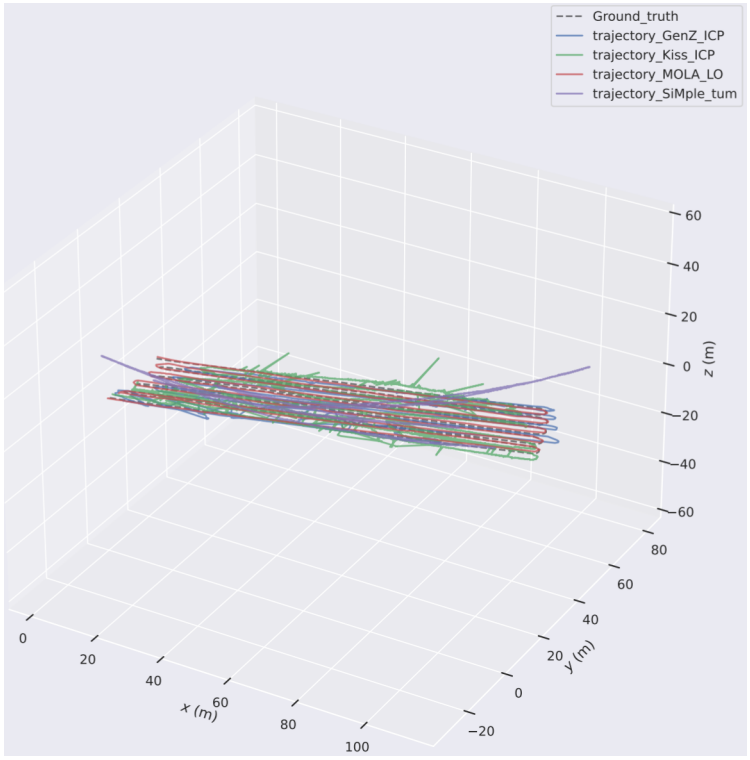
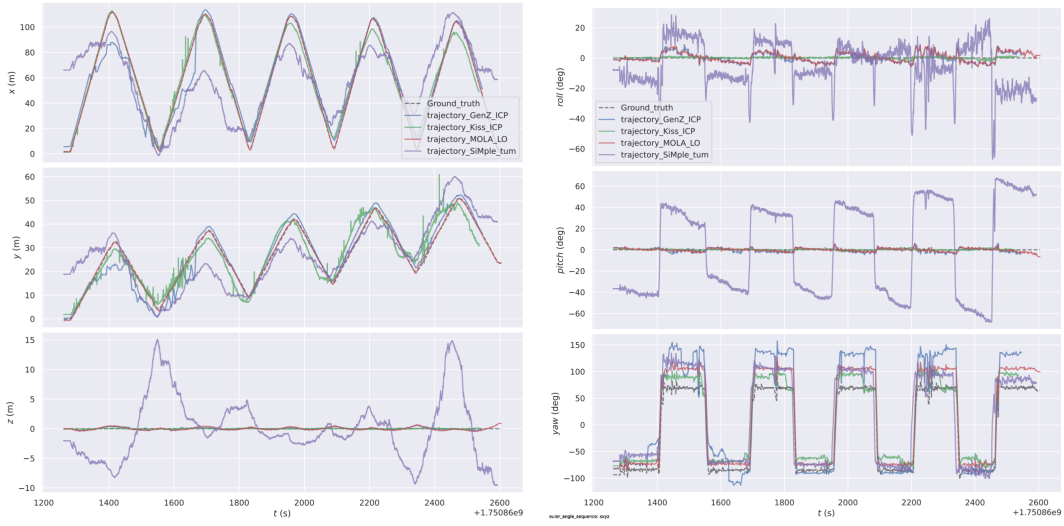


Figure 7.21: Overlay of all four algorithm trajectories with the ground truth in the Vineyard, showing overall alignment and localized deviations.



((a)) Translational Components vs Time **((b))** Rotational Components vs Time

Figure 7.22: Overlay of all four algorithms showing (a) translational components (x, y, z) over time and (b) rotational components (roll, pitch, yaw) over time, highlighting deviations relative to the ground truth.

7.4 Comparative Discussion

The preceding chapters examined the performance of four LiDAR odometry frameworks—**KISS-ICP**, **GENZ-ICP**, **MOLA-LO**, and **SiMpLe**—tested across two environments that differ radically in structure and feature density: the *KITTI Odometry Benchmark*, representing a well-organized urban setting, and a *Vineyard Dataset*, characterized by repetitive geometry and significant vegetation noise. This comparative discussion integrates the outcomes from both datasets to provide a unified interpretation of each method's reliability, adaptability, and efficiency across contrasting conditions. This chapter moves beyond the separate dataset evaluations and focuses on three main aspects. First, a cross-dataset analysis highlights how each algorithm adapts—or fails to adapt—to changes in scene structure and feature variability. Second, a robustness-versus-efficiency study explores how accuracy and computational demand trade off under limited hardware resources. Finally, the discussion draws general conclusions to guide algorithm selection for both autonomous navigation and agricultural robotics, emphasizing real-world applicability rather than ideal laboratory conditions.

7.4.1 Cross-dataset comparison

Comparing the results from the *KITTI* and *Vineyard* datasets reveals how environmental structure strongly influences algorithm behavior. The KITTI sequences feature rich geometry—with roads, facades, and sharp corners—while the vineyard environment introduces repeating patterns, vegetation noise, and narrow turns that amplify drift and misalignment. This contrast clearly shows that approaches performing well in structured, urban settings often struggle to generalize in unstructured or semi-natural contexts.

SiMpLe achieves exceptional accuracy in KITTI due to its advanced feature-weighting and filtering pipeline, which benefits from dense and varied geometry. However, this same complexity becomes a weakness in the vineyard scenario. When confronted with repetitive vine rows and high vegetation noise, its optimization process misinterprets redundant features, leading to severe pose drift and instability. In short, the algorithm's precision in feature-rich contexts translates poorly to environments with low structural variability.

MOLA-LO, in contrast, adopts a more conservative approach. Its emphasis on global optimization and noise suppression results in only average accuracy on KITTI, but proves advantageous in the vineyard. The algorithm's resilience stems from its reduced reliance on distinctive features and its ability to maintain long-term trajectory consistency. In highly repetitive conditions, this design prevents the accumulation of drift and maintains better overall stability.

KISS-ICP occupies an intermediate position. On KITTI, it efficiently exploits

local planar geometry, achieving strong translational accuracy at a modest computational cost. Yet, in the vineyard, it gradually accumulates drift and yaw offsets, particularly during turning maneuvers. The lack of a strong global correction mechanism limits its robustness under noisy, repetitive conditions.

GENZ-ICP demonstrates particularly uneven performance across both datasets. On KITTI, it is highly sensitive to initialization errors and outliers, leading to unstable optimization. In the vineyard tests, it sometimes manages partial recovery after misalignment but remains inconsistent overall. Its weak error suppression and reliance on clear geometric cues make it unsuitable for environments dominated by vegetation or repetitive layouts.

Minor discrepancies between our KITTI results and the values reported in the original algorithm papers are attributable not to methodological issues, but to computational limitations. All experiments were conducted on a CPU-only setup (Ryzen 5 5600G, 16 GB RAM, no dedicated GPU) within WSL2 on Windows. Such hardware restricts correspondence searches, voxel filtering, and optimization efficiency. Consequently, algorithms with heavy computational pipelines—especially SiMpLe—experience degraded performance relative to published benchmarks, which typically rely on high-end GPU-enabled systems. In conclusion, the cross-dataset analysis shows that algorithmic complexity does not guarantee robustness. Sophisticated models like SiMpLe perform exceptionally in structured benchmarks but degrade in unstructured domains. Simpler, globally consistent designs like MOLA maintain accuracy across diverse conditions. KISS-ICP offers a balanced trade-off, while GENZ-ICP remains unstable under real-world constraints. This reinforces the importance of evaluating odometry systems across heterogeneous datasets and under realistic hardware limitations.

7.4.2 Robustness and efficiency trade-offs

Selecting an appropriate LiDAR odometry system involves balancing two competing demands: robustness, defined as stability and accuracy under noise and drift, and efficiency, which depends on computational cost and real-time feasibility. The present evaluation highlights how algorithmic design and environmental complexity interact with hardware performance. All experiments were executed on a mid-range CPU system (Ryzen 5 5600G, 16 GB RAM, Ubuntu on WSL2). This setup lacks GPU acceleration and thus emphasizes CPU efficiency. In such conditions, algorithms that rely heavily on iterative optimization, complex filtering, or feature extraction incur significant runtime penalties.

Within this context, **SiMpLe** stands out for its precision in structured datasets like KITTI but at the cost of high computational load. Its detailed feature-weighting and filtering pipeline, while powerful, increases latency and reduces frame rates.

When exposed to the repetitive geometry and vegetation noise of the vineyard, this computational overhead amplifies drift and reduces stability, illustrating that heavy algorithmic design does not guarantee robustness in resource-limited scenarios.

MOLA-LO, in contrast, operates efficiently with relatively low computational demand. Thanks to its simpler matching strategy and global optimization scheme, it achieves stable results even without GPU acceleration. While its accuracy on KITTI is slightly below the top performers, its resilience in the vineyard data highlights excellent generalization and makes it a practical choice for systems running on mid-range hardware.

KISS-ICP achieves a middle ground. Its G-ICP formulation balances efficiency and accuracy, avoiding heavy feature extraction while maintaining solid translational estimates. It performs near-optimally on KITTI and only moderately degrades in the vineyard, confirming that lightweight ICP variants can sustain acceptable robustness under constrained computational conditions.

GENZ-ICP shows how lack of robust initialization and outlier handling can compromise performance, especially when computation per scan is limited. Its unstable convergence and sensitivity to noise suggest that increased iteration counts do not compensate for weak data association. As a result, the algorithm becomes less efficient and more prone to cumulative drift.

Similar findings are echoed in recent literature, where feature-based methods often trade computational efficiency for robustness, while lightweight ICP formulations achieve higher frame rates but require careful initialization [27]. Overall, our experiments suggest that, under CPU-only conditions, algorithms emphasizing global optimization, outlier rejection, and minimal per-frame computation—as in MOLA-LO—yield the best balance between robustness and efficiency. SiMpLe excels in idealized conditions but struggles under hardware and environmental constraints, while KISS-ICP remains a balanced, reliable alternative for general-purpose use.

Algorithm	**Robustness**	**Efficiency**	**Trade-off**
SiMpLe	High on KITTI, fails in vineyard	Low	Accurate in structured, poor generalization
MOLA-LO	Stable KITTI/vineyard	Moderate–High	Balanced, robust and efficient
KISS-ICP	Good KITTI, moderate vineyard drift	High	Efficient, moderate robustness
GENZ-ICP	Low, unstable	Moderate	Neither robust nor efficient

Table 7.21: Robustness vs Efficiency trade-offs on mid-range CPU hardware.

Chapter 8

Conclusion and Future Work

This chapter concludes the benchmark analysis by summarizing the principal findings, interpreting performance tendencies across datasets, and discussing their implications for practical deployment. It also outlines the limitations of this study and identifies future research directions that could extend and refine these results.

8.1 Summary of the Results

This work has benchmarked four LiDAR odometry frameworks—**SiMpLe**, **MOLA-LO**, **KISS-ICP**, and **GENZ-ICP**—across two complementary datasets: the structured, urban-oriented *KITTI Odometry Benchmark* and a private *Vineyard Dataset* designed to capture repetitive and unstructured agricultural geometry. These two datasets collectively represent the contrast between feature-rich urban scenes and environments dominated by vegetative noise and geometric repetition.

On the **KITTI dataset**, SiMpLe demonstrated the highest overall accuracy, particularly in minimizing rotational drift. Its complex filtering and feature-weighting modules effectively exploited the diversity of geometric cues present in structured city environments. KISS-ICP closely followed, achieving strong translational consistency through its efficient G-ICP formulation. MOLA-LO achieved intermediate results, providing smoother yet slightly less optimized trajectories, while GENZ-ICP exhibited severe instability and large-scale drift, confirming weak robustness under structured conditions. Minor discrepancies with previously published values (Tables 7.2 and 7.4) are explained by the mid-range CPU-only setup used here, which lacks GPU acceleration and thus limits computational throughput.

In contrast, the **Vineyard dataset** emphasized the drawbacks of complex, feature-dependent algorithms. SiMpLe, despite its strength on KITTI, struggled in the presence of repetitive rows and high vegetation noise, leading to alignment failures and severe drift. KISS-ICP maintained moderate stability but gradually accumulated error along straight segments and turns. MOLA-LO, however, emerged as the most stable method, benefiting from its simpler feature handling and emphasis

on global consistency. GENZ-ICP remained inconsistent, occasionally recovering after initial misalignments but suffering persistent yaw drift and curve errors.

The cross-dataset analysis suggests that excessive algorithmic complexity can become a drawback in poorly structured settings. SiMpLe performs well in urban, feature-rich scenes but loses adaptability in environments with noisy or ambiguous geometry. MOLA-LO, with its more conservative design, proves to be more versatile, striking a balance between simplicity and robustness. KISS-ICP occupies a middle ground—efficient, fairly accurate, and reasonably stable—while GENZ-ICP consistently falls short in both domains. These insights underline the importance of evaluating algorithms across multiple, contrasting environments to obtain a realistic view of their performance.

The trade-off between robustness and computational efficiency was also evident. SiMpLe, although highly accurate on KITTI, incurs a substantial runtime cost under limited hardware resources. MOLA-LO achieved the best compromise, maintaining reliable accuracy across environments with manageable computation time. KISS-ICP proved highly efficient but less tolerant to noise and repetitive structures, while GENZ-ICP was both unstable and inefficient. Table 7.21 summarizes these relationships, underscoring that practical deployment requires balancing algorithmic precision with hardware feasibility.

8.2 Limitations and Future Work

Despite providing valuable insights into LiDAR odometry performance under contrasting conditions, this benchmark has several limitations that inform directions for future research.

- **Hardware limitations.**
  All experiments were performed on a mid-range CPU-only system (Ryzen 5 5600G, 16 GB RAM, no dedicated GPU). This setup constrained the processing rate and may have affected the numerical stability of computationally intensive algorithms, particularly SiMpLe. Consequently, part of the observed performance gap with respect to published results likely arises from limited computational resources rather than inherent algorithmic deficiencies.

- **Hardware limitations.**
  All experiments were performed on a mid-range CPU-only system (Ryzen 5 5600G, 16 GB RAM, no dedicated GPU). This setup constrained the processing rate and may have affected the numerical stability of computationally intensive algorithms, particularly SiMpLe. Consequently, part of the observed performance gap with respect to published results likely arises from limited computational resources rather than inherent algorithmic deficiencies.

- **Dataset constraints.**
  The vineyard dataset, though representative of unstructured agricultural environments, covers only a single crop type and configuration, with fixed row spacing and limited trajectory diversity. This restricts the generalization of the conclusions to other agricultural or off-road contexts. Factors such as seasonal variation, lighting changes, or moving obstacles were not incorporated, potentially masking dynamic effects that could further influence odometry stability.

- **Sensor modality limitations.**
  This evaluation focused solely on LiDAR-based odometry. In practical robotic applications, most navigation systems combine multiple sensors—such as cameras, IMUs, or GNSS—to improve reliability and precision. As a result, this study does not account for the potential benefits achievable through sensor fusion or hybrid SLAM approaches, which often deliver more robust localization in real-world conditions.

- **Scenario and sampling diversity.**
  The datasets used operate at fixed scanning frequencies and trajectory complexities. Algorithms may behave differently with higher-density LiDARs, faster motion, or dynamic scenes. Additionally, while this benchmark examined both straight and curved motion segments, extreme maneuvers and irregular paths were not included, limiting insight into performance under aggressive motion dynamics.

Future Work

Extending this benchmark should include a broader set of datasets covering diverse crops, terrains, and environmental conditions to evaluate adaptability and long-term drift behavior. Implementing GPU-accelerated versions of computation-heavy algorithms would yield a more realistic estimate of runtime efficiency. Future work could also explore hybrid LiDAR–visual–inertial pipelines to mitigate weaknesses of single-modality odometry in repetitive environments. Finally, large-scale, long-duration experiments are needed to study cumulative drift behavior, loop closure potential, and real-time deployability in both structured and natural environments.

Bibliography

[1] D. Fox S. Thrun W. Burgard. *Probabilistic Robotics*. Cambridge, Massachusetts: MIT Press, 2005 (cit. on pp. 1, 9–17).

[2] W. Yang D. Lee M. Jung and A. Kim. "LiDAR Odometry Survey: Recent Advancements and Remaining Challenges". PhD thesis. Seoul, 08826, Republic of Korea: Seoul National University, Dec. 2023. URL: https://arxiv.org/abs/2312.17487 (cit. on pp. 2, 3, 36, 37, 39, 40, 48, 49).

[3] Daehan Lee, Hyungtae Lim, and Soohee Han. "GenZ-ICP: Generalizable and Degeneracy-Robust LiDAR Odometry Using an Adaptive Weighting". In: *IEEE Robotics and Automation Letters (RA-L)* 10.1 (2025), pp. 152–159. DOI: 10.1109/LRA.2024.3498779 (cit. on pp. 2, 42, 43, 62).

[4] B. Mersch I. Vizzo T. Guadagnino et al. "KISS-ICP: In Defense of Point-to-Point ICP - Simple, Accurate, and Robust Registration If Done the Right Way". In: https://arxiv.org/abs/2209.15397 (July 2023) (cit. on pp. 2, 40–42, 61).

[5] Jose Luis Blanco-Claraco. "A flexible framework for accurate LiDAR odometry, map manipulation, and localization". In: *The International Journal of Robotics Research* 0.0 (2025), p. 02783649251316881. DOI: 10.1177/02783649251316881 (cit. on pp. 2, 43–45, 51, 63).

[6] Phillips TG Bhandari V. "Minimal configuration point cloud odometry and mapping". In: *The International Journal of Robotics Research* 43.11 (2024), pp. 1831–1850. DOI: 10.1177/02783649241235325 (cit. on pp. 2, 45–47, 64).

[7] Nikolaus Correll. *Introduction to autonomous Robots*. Magellan Scientific, 2016 (cit. on pp. 5–9).

[8] Elias Maia Azevedo dias Ferreira. "Improving LiDAR Odometry and Mapping in real-time using Inertial Measurements". MA thesis. Porto: Universidade do Porto, 2021 (cit. on pp. 5, 7).

[9] Wim Meeussen. "Coordinate frames for Mobile Platform". In: *ROS.org* (Oct. 2010) (cit. on pp. 6, 7).

[10] Tim Whiteman. "Frames". In: *Academia* (2016) (cit. on pp. 7, 8).

[11] I.R. Nourbakhsh R. Siegwart. *Introduction to Autonomous Mobile Robots*. Cambridge, Massachusetts: MIT Press, 2004 (cit. on pp. 10–13).

[12] S. Esmaeelpourfard M. A. Sharbafi M. Hoshyari et al. “MRL Team Description 2010 - Small Size Robot League”. In: *International RoboCup Competition*. June 2010 (cit. on p. 16).

[13] G. Dobie B. Das and S. G. Pierce. “AS-EKF: a delay aware state estimation techniquefor telepresence robot navigation”. In: *2019 Third IEEE International Conference on Robotic Computing (IRC)*. February 2019 (cit. on p. 17).

[14] T. Stoyanov H. Andreasson G. Grisetti et al. “Sensors for Mobile Robots”. In: *Encyclopedia of Robotics. Springer*. September 2023 (cit. on pp. 17–21).

[15] John Farid Nasry Henawy. “Visual inertial odometry and LiDAR inertial odometry for mobile robot”. MA thesis. Singapore: Nanyang Technological University, 2021 (cit. on p. 20).

[16] Santiago Royo and Maria Ballesta-Garcia. “An Overview of Lidar Imaging Systems for Autonomous Vehicles”. In: *Applied Sciences, MDPI* (Sept. 2019) (cit. on pp. 22–31).

[17] T.Azumi Y. Maruyama S. Kato. “Exploring the performance of ROS2”. In: *Proceedings of the 13th International Congference on Embedded Software*. EMSOFT ’16, Pittsburgh, Pennsylvania: Association for Computing Machinery, 2016.’ (Cit. on p. 32).

[18] Jason M. O’Kane. *A gentle introduction to ROS*. Columbia, South Carolina: University of South Carolina, 2014 (cit. on p. 32).

[19] Paolo Vanella. “Implementation of ROS-based Multi-Agent SLAM Centralized and Decentralized Approaches”. MA thesis. Torino: Politecnico di Torino, 2023 (cit. on pp. 32, 33).

[20] “ROS Brand Guide”. In: *ROS.org* Branding Guidelines (2020) (cit. on p. 33).

[21] ROS.org. “ROS Concepts”. In: *https://wiki.ros.org/ROS/Concepts* (2022) (cit. on p. 33).

[22] ROS.org. “Understanding nodes”. In: *ROS 2 Documentation: Humble* () (cit. on pp. 33, 34).

[23] R. Y Y. Pyo H. Cho et al. *ROS Robot Programming*. GeumCheon-gu, Seoul, Republic of Korea: ROBOTIS Co.,Ltd., 2017 (cit. on p. 35).

[24] Riccardo Tassi. “Design of a Behavior-Based Navigation Algorithm for Autonomous Tunnel Inspection”. MA thesis. Torino: Politecnico di Torino, 2022 (cit. on p. 35).

[25] M. Fallon J. Behley et al. *SLAM handbook Chapter 10: LiDAR SLAM*. Cambridge, United Kingdom: Cambridge University Press, March 2025 (cit. on pp. 36–39).

[26] J. L. Blanco-Claraco and R. Aguilera Lopez. *GitHub*. 2025. URL: `https://github.com/MOLAorg/mola_lidar_odometry/tree/develop` (cit. on pp. 44, 45).

[27] Jianke Zhu Xin Zheng. "Efficient LiDAR Odometry for Autonomous Driving". In: *IEEE Robotics and Automation Letters* 6 (2021), pp. 8458–8465. DOI: 10.1109/LRA.2021.3110372 (cit. on pp. 48, 49, 99).

[28] Andreas Geiger, Philip Lenz, and Raquel Urtasun. "Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2012 (cit. on pp. 48, 49, 65, 66).

[29] HKUST-Aerial-Robotics. *Benchmark for LiDAR-based 3D global registration*. https://github.com/HKUST-Aerial-Robotics/LiDAR-Registration-Benchmark. Accessed: 2025-10-09. Cites SemanticKITTI for more accurate ground truth. 2024 (cit. on p. 49).

[30] Z. Zhao, Y. Zhang, J. Shi, and L. Long. "Robust Lidar-Inertial Odometry with Ground Condition Perception and Optimization Algorithm for UGV". In: *Sensors (MDPI)* 22.19 (2022), p. 7424. DOI: 10.3390/s22197424 (cit. on pp. 50, 51).

[31] J. Knights, K. Vidanapathirana, et al. *Wild-Places: A Large-Scale Dataset for Lidar Place Recognition in Unstructured Natural Environments*. 2023. arXiv: 2211.12732 [cs.RO]. URL: https://arxiv.org/abs/2211.12732 (cit. on pp. 50–52).

[32] Jingyi Zhou Zhiqiang Dai and Tianci Li. "An intensity-enhanced LiDAR SLAM for unstructured environments". In: *Measurement Science and Technology* 34 (2023). DOI: 10.1088/1361-6501/acf38d (cit. on p. 51).

[33] Bing Zhang, Xiangyu Shao, Yankun Wang, Guanghui Sun, and Weiran Yao. "R-LVIO: Resilient LiDAR-Visual-Inertial Odometry for UAVs in GNSS-denied Environment". In: *Drones* 8.9 (2024). ISSN: 2504-446X. DOI: 10.3390/drones8090487. URL: https://www.mdpi.com/2504-446X/8/9/487 (cit. on p. 51).

[34] T. Kern, L. Tolksdorf, and C. Birkner. *Comparison of Localization Algorithms between Reduced-Scale and Real-Sized Vehicles Using Visual and Inertial Sensors*. 2025. DOI: 10.48550/arXiv.2507.11241 (cit. on p. 79).