Eötvös Loránd University

Faculty of Informatics

Erasmus Mundus Joint Master in
Intelligent Field Robotic Systems

LiDAR Odometry and Mapping Beyond RTK Accuracy

Author:
Liviu-Daniel Florescu
MSc Intelligent Field Robotic Systems (IFRoS)

Internal supervisor:
Iván Eichhardt
Assistant Professor

External supervisor:
Maximilian Fenkart
CTO, Sodex Innovations GmbH

Budapest, 2025

Title of the thesis

LiDAR Odometry and Mapping Beyond RTK Accuracy

Topic of the thesis

Out of the five basic senses that humans use to experience the world, vision accounts for 80% of the information input that our brains operate with [1]. Naturally, robotics research aims to replicate this and develop systems that can not only collect high-quality visual data but also create rich artificial representations of the world, enabling autonomous systems to confidently reason about their environment.

This work will focus on the use of 3D Light Detection and Ranging (LiDAR) sensors in outdoor settings. When mounted on an arbitrary mobile base and moved around a target environment, the sensor collects information about the geometry of the scene, which can be merged in order to create a general 3D model. However, this process depends on accurate displacement measurements that are not trivial to obtain. An existing solution relies on Global Navigation Satellite System (GNSS) localization, corrected using Real-Time Kinematic positioning (RTK), and orientation from an Inertial Measurement Unit (IMU). Together, these create an Inertial Navigation System (INS) whose output can be interpreted as the 3D transformation between sensor poses at discrete time steps. Even though this represents the state-of-the-art technology for outdoor localization, with centimeter-level position error, its accuracy is unsatisfactory when it comes to high-quality pointcloud registration. Additionally, this is not feasible for all outdoor scenes, because GNSS accuracy varies heavily depending on surroundings and signal strength.

We will investigate methods that address the limitations of the INS-based registration, by using the visual information in the scene, such that the system is less reliant on a sensor with fluctuating uncertainty. Previous research in this area [2] indicates that visual cues alone should be enough to achieve reliable displacement estimation, enabling the computation of odometry from LiDAR data, as well as creating a 3D map of the explored environment. A comparison between LiDAR odometry and GNSS localization is also within the scope of this work. The research questions that we aim to answer are the following:

- What metrics exist for measuring the accuracy of pointcloud registration?
- Can methods that use only visual information achieve higher quality pointcloud registration (3D mapping) than RTK-based merging?
- To what extent is LiDAR-based odometry an alternative to GNSS localization?

The work will be carried out in collaboration with Sodex Innovations GmbH, who provide the sensor rig (LiDAR, RGB cameras, GNSS + RTK + IMU) as well as several existing datasets consisting of sensor data collected while exploring outdoor rural environments. The project will span approximately 15 weeks, and is tentatively structured as shown below, subject to change brought by results at different stages.

Stage	Estimated Duration	Description
Introduction	2 weeks	Familiarization with the existing sensors, datasets and related processes
Background investigation	3 weeks	Exploring existing solutions, initial experimentation on available data
Design and Development	4 weeks	Iterative implementation and testing of new approaches
Analysis	4 weeks	Evaluation and comparison between the solutions developed at different iterations
Final write-up	2 weeks	Reporting methods, implementation and results

Keywords
LiDAR odometry, LiDAR mapping, pointcloud registration, GNSS, RTK

References
[1] Man, Dariusz & Olchawa, Ryszard. (2018). The Possibilities of Using BCI Technology in Biomedical Engineering. 10.1007/978-3-319-75025-5_4.
[2] Lee, Dongjae & Jung, Minwoo & Yang, Wooseong & Kim, Ayoung. (2024). LiDAR odometry survey: recent advancements and remaining challenges. Intelligent Service Robotics. 17. 1-24. 10.1007/s11370-024-00515-8.

Contents

Chapter 1

Introduction

Like many scientific terms that we encounter in our daily activities, *robotics* describes a broad collection of technologies and stands at the intersection of key research directions. Motivated by practicality, disciplines that appear completely unrelated find themselves building upon advancements in other fields, diluting boundaries and joining forces to enable otherwise-impossible advancements.

What constitutes a *robot*, then? In its inceptive use by Karel Čapek [1] in 1921, the word stemmed from the Slavic *robota*, meaning "servitude" or "forced work", and referred to a human-like mechanical system working on factory assembly lines. This concept, however, dates from earlier centuries. Around 1495, Leonardo Da Vinci envisioned a mechanical knight controlled by a series of pulleys, that was able to perform simple movements [2]. Testifying to the industrial revolution and the computational breakthroughs of the last few decades, modern-day humanoid robots can perform acrobatics [3], interact with humans in constrained scenarios [4, 5, 6] and even replicate human facial expressions [7]. Still, a device ought not necessarily appear human-like in order for it to be labeled as a robot. Autonomous vacuum cleaners, space rovers or crop-monitoring drones fall under the same category. At this stage, a complete taxonomy would have to address dozens of physical (size, shape, mobility, locomotion system, etc.) and non-physical (autonomy, perception abilities, use-case, etc.) characteristics, and none of these would independently convey the meaning that we intuitively associate to the notion of *robot*. Without assessing whether an exhaustive, generic definition is even possible, we can synthesize the above by affirming that a *robot* is an artificial system that performs one or more tasks and is able to evaluate its state or gather information from its environment.

1.1 Robotic perception

This formulation highlights two essential aspects of a robotic agent (Fig. 1.1): actuation, seen as some form of dynamic physical ability, allowing the agent to enact the desired behavior, and perception, the ability to observe changes in the environment.

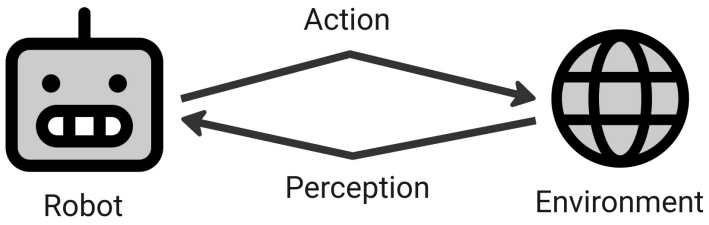


Figure 1.1: A simplified interpretation of the robotics paradigm. The interaction between an agent and its environment can be seen as a two-way flow: the agent alters the environment through its actions, and uses perception to observe it.

Sensing modalities have largely different contributions to the perception mechanism. To this extent, the phrase *visual dominance* was introduced by F. Colavita [8], whose study demonstrated that humans focus more on the visual component when presented with an audiovisual stimulus, and following research has strongly confirmed this tendency [9, 10]. Unsurprisingly, a similar pattern is emerging in the case of robots, thanks to the reduced cost, familiarity and wide availability of cameras. In many situations, visual stimuli provide most of the necessary information, and this has motivated the development of various image processing algorithms.

Technological innovation in the last century has led to the situation in which artificial sensors surpass humans in both the range of signals that are perceived, as well as the accuracy of the measurements. A relevant example is the class of Light Detection and Ranging (LiDAR) sensors which retrieve three-dimensional information about the environment at a very high frequency and with rather negligible measurement errors, in the form of *point clouds* (Fig. 1.2). Among many applications, this type of sensors can be used to construct virtual representations of a specific environment, enabling engineers to experiment with a realistic model, evaluate construction progress or validate a finished project.

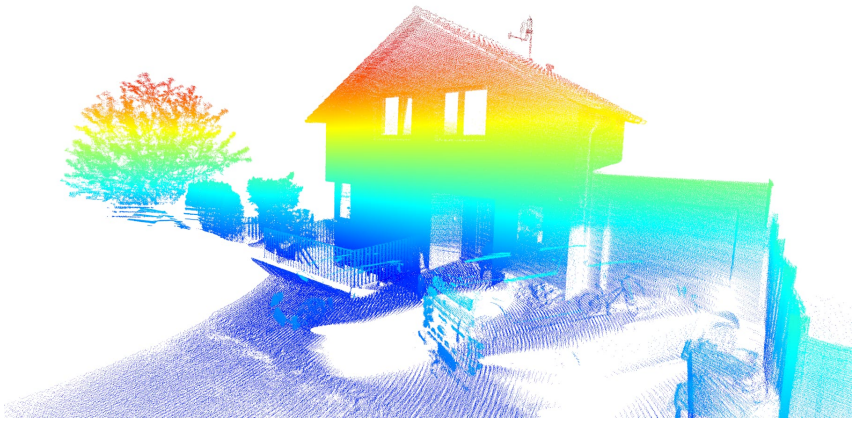


Figure 1.2: An example point cloud (points are colored by height), capturing the 3D representation of an outdoor scene. Multiple individual scans are placed into the same coordinate system to obtain a dense spatial representation.

1.2 Problem definition

The current work addresses a common and well-known problem in the area of Field Robotics, namely Simultaneous Localization and Mapping (SLAM), and combines the practicality of an industrial solution with a perception-based approach.

SDX-Compact (Fig. 3.1) is the main product of Sodex Innovations GmbH, consisting of multiple sensors that collect spatial and visual data. This module can be easily mounted on an arbitrary vehicle in order to expand its perception capabilities and convert it into a surveying device. While the vehicle is moving, the LiDAR sensor captures 3D scans of the surroundings, as well as related metadata (timestamps, localization information, signal intensity etc.). Because the rig includes a high-accuracy Inertial Navigation System (INS), the localization and orientation data can be used to join the collected point clouds and create a global 3D map of the traversed space.

Nonetheless, this suffers from two main limitations:

- Reliance on unstable signal: internally, the INS depends on information from a Global Navigation Satellite System (GNSS) receiver, which is limited to outdoor spaces and whose availability varies depending on weather conditions and surroundings (e.g. thick vegetation, tall buildings, bridges).
- Unsatisfactory accuracy: when merging point cloud data, positioning or localization errors introduce inconsistencies in the final 3D model, which hinders precise planning and construction.

Our work aims to address these limitations by introducing a component that utilizes the information collected by the LiDAR sensor in conjunction with the existing data. Three research questions have been formulated to guide this process:

1. What metrics exist for measuring the accuracy of point cloud registration? In this context, registration refers to placing a pair of related point sets in a common reference frame.
2. Can methods that use only visual information achieve higher quality point cloud registration (3D mapping) than merging based on Real-Time Kinematics (RTK)? Usually, GNSS systems provide meter-level accuracy. In the current scenario, however, the system is corrected using Real-Time Kinematics, such that the expected error is at centimeter-level.
3. To what extent is LiDAR-based odometry an alternative to GNSS localization? Previous research indicates that the spatial information present in 3D point clouds could be used to compute the relative displacement between consecutive scans, resulting in the ability to estimate odometry (an essential component of robotic localization) without dedicated sensors such as wheel encoders or accelerometers.

The main contribution of the project consists of developing an original framework for localization and mapping based on data collected with an industrial sensor rig. The results are more generic than if a particular physical robotic system were involved, and thus are relevant for virtually any robotic application with a similar setup.

The following chapters of this document will cover related research directions and efforts that our work builds upon (Chapter 2), a detailed description of the components and algorithms involved in developing the project (Chapter 3), an evaluation of the method based on its results (Chapter 4), as well as a series of conclusions that were drawn from the overall process (Chapter 5).

Chapter 2

Background and Literature Review

The goal of this chapter is to present the research context in which our project was conducted, by looking at common methods for each of the main components and highlighting those that inspired the current approach.

2.1 Simultaneous Localization and Mapping (SLAM)

SLAM constitutes a key research area in robotics, because it is a foundational building block for autonomous operation. This problem occurs when the robot does not have prior access to a map of the environment, so it must construct one while keeping track of its current position (*online* SLAM). If the goal is to optimize the entire sequence of poses along the robot path, this is known as the *full* SLAM problem. [11]

Usually, the problem is discretized along the time dimension, such that at time t we aim to compute the posterior function $p(x_t, M|z_{1:t}, u_{1:t})$, where x_t is the current state, M is the map, $z_{1:t}$ represents the set of measurements collected so far, and $u_{1:t}$ the control sequence. As each of these variables can have different concrete representations, depending on the task and the available sensors, a broad range of approaches have been proposed.

Aulinas et al. [12] note that the earliest methods rely on probabilistic models derived from the recursive Bayes rule, as such formulations can provide intuitive representations of the various noisy components involved in a robotic system. When employing the Kalman Filter (KF) [13] or its variations, the robot state, measurements and control inputs are modeled as multi-dimensional Gaussians, whose covariances describe the associated uncertainty. The algorithm alternates between *predictions*, when the state is modified based on u_t, and *updates*, when the prediction is evaluated against the latest observation z_t, and the

state hypothesis is updated accordingly. The Extended Kalman Filter (EKF) maintains this structure but can accommodate non-linearities in the displacement or measurement functions by using local linear approximations. Such methods have been successfully applied for indoor [14], aerial [15], and underwater [16] robots.

Particle Filter (PF), introduced by Del Moral [17], is a probabilistic approach that relies on the Monte Carlo method. Instead of an analytical form, the uncertainty is accounted for by considering a large set of samples (representing potential states) and weighting them based on measurement likelihood. This has a higher computational cost than the standard KF methods, but is not affected by any linearization limitations. Nie et al. [18] implemented a LiDAR SLAM algorithm based on PF localization, albeit for 2D mapping.

Another large group of methods is represented by *Visual SLAM*, when cameras are the main (and sometimes only) sensor used. As early as 1980, Moravec [19] developed a robot capable of estimating its motion by matching features in images captured at discrete poses, in a stop-and-go fashion. In 2007, SLAM was being performed using a single handheld camera [20, 21], triggering a separation from the popular, offline, Structure from Motion (SfM) techniques. The solution is even more robust when a stereo system is available [22], as this helps avoid the geometrical limitations of monocular vision.

2.2 Point cloud Registration

Before discussing LiDAR-based approaches in more detail, let us review the task of point cloud registration. Given two sets of points P, Q in arbitrary reference frames, representing (at least partially) the same scene, the goal is to find the transformation $\mathbf{T} \in \mathrm{SE}(3)$ (translation and rotation) that best aligns the points. This can be expressed as an optimization problem:

$$\underset{\mathbf{T}}{\operatorname{argmin}}\, J(\mathbf{T}P, Q), \tag{2.1}$$

where J is a custom cost function. The particular case where known point correspondences are provided has a closed-form solution [23], but the Iterative Closest Point (ICP) algorithm [24] removes this constraint by alternating between correspondence generation — each point in P is paired with its nearest neighbor from Q — and minimizing the point-pair distances. This approach is by far the most widespread, thanks to its simplic-

ity, computational efficiency (e.g., using a k-d tree for closest-point search) and many available variations [25, 26].

Chen and Medioni [27] proposed a "point-to-plane" minimization, while Segal et al. [28] formulated a generalized cost function by considering the probabilistic model underlying the point sets, and implemented a "plane-to-plane" minimization. Other variants modify the point selection strategy [29, 30], apply a weight to each pair [31], prune the set of the correspondences [32, 33] or employ a different minimization algorithm [34].

A different class of solutions operates by discretizing the space into voxels and estimating a normal distribution using the points in each voxel. Introduced in 2003, the Normal Distributions Transform (NDT) [35] algorithm is the main competitor to ICP, enabling registration to be performed without the need for point-to-point correspondences. Magnusson et al. [36] extended this work to 3D scans, and performed a thorough comparison between this and ICP [37] for the challenging task of registering point clouds captured in underground mines, where few usable features are present. More recently, the method has been applied for outdoor mapping [38], and as the basis for a Lidar Odometry and Mapping solution [39].

Neural Networks can also be used for point cloud registration, by creating correspondences based on features extracted by a descriptor architecture [40, 41]. This is particularly useful when no initial estimate of the transformation between the two point sets is available — a scenario that the other approaches cannot directly handle.

2.3 LiDAR-based Odometry and Mapping

Despite their high price range, LiDAR sensors have gained significant popularity in the last decade, becoming the go-to solution for autonomous mobility. Unlike cameras, LiDARs are not dependent on ambient lighting, so they can operate in total darkness, and they provide high-frequency, reliable 3D information without the need for additional processing. Basic applications include obstacle detection [42, 43], but the amount of information they provide makes them excellent for localization and mapping purposes.

A key reference is LiDAR Odometry and Mapping (LOAM), the work of Zhang and Singh [44], who used a motorized, 2-axis LiDAR scanner to generate 3D point clouds. To register consecutive scans, they perform feature extraction (sharp edges or planar patches), identify correspondences to the previous scan, and find the optimal transformation using the Levenberg-Marquardt method [45, 46]. To account for displacement during the beam

sweep, points are reprojected by linear interpolation, assuming constant velocity — this is known as *motion/distortion compensation*. Mapping takes place in parallel, at a slower rate, and the current scan is registered to the map created so far. At that time, this implementation achieved the best results[1] on the KITTI Odometry Benchmark [47]. V-LOAM [48] improved this solution by introducing a visual odometry component based on a fish-eye camera.

A few years later, LeGO-LOAM [49] extended the feature extraction process by performing segmentation on a range image generated from the 3D point cloud. Clusters are filtered by size, in order to discard features belonging to potentially noisy points. Another modification is that the parameters of the transformation between consecutive scans are optimized separately: $t_z, \theta_{roll}, \theta_{pitch}$ are computed using planar feature correspondences, then fixed during the optimization of t_x, t_y, θ_{yaw}, which only uses edge features. The method is compared to LOAM and achieves higher accuracy in outdoor scenarios, while being an order of magnitude faster.

F-LOAM [50] proposes a two-step motion compensation process. For odometry computation, the constant velocity model is used, but the points are re-corrected using the optimized pose before being registered to the map. Correspondences are weighted based on the "local smoothness" of the feature, and the non-linear optimization is solved using the Gauss-Newton method. The map is not updated at every scan, but only when the translational change reaches a predefined threshold. This is a common keyframe selection technique inherited from Visual Odometry.

A slightly different approach introduces the use of inertial sensors, leading to LiDAR-Inertial Odometry (LIO). This aims to compensate for the lack of reliable 3D features in specific environments, as accelerometers can provide satisfactory motion estimates for short displacements. FAST-LIO [51] applies such a technique for a drone equipped with a high-frequency solid-state LiDAR, by performing tightly-coupled fusion: instead of using the IMU output to correct the scan registration, it is applied on the features extracted from the point cloud. COIN-LIO [52] introduces a photometric error component, based on the beam intensity values returned by the LiDAR. A monochrome "intensity image" is constructed and filtered such that matching can occur between consecutive frames, and these new correspondences extend the residual vector that is minimized for odometry computation. The approach achieves state-of-the-art performance on a dataset of

[1] `https://www.cvlibs.net/datasets/kitti/eval_odometry.php`

geometrically-degenerate scenes (ENWIDE [2]).

In the class of deep-learning techniques, we highlight Deep Matching LiDAR Odometry (DMLO) [53], which translates the registration problem into a supervised machine learning task. Point clouds are projected into a 2D map using cylinder encoding, with range and intensity values as channels, and a Convolutional Neural Network (CNN) architecture is trained to predict correspondences from pairs of projections. Training samples are constructed from a subset of the target dataset, and the method proves robust across different LiDAR hardware, but does not surpass LOAM on the KITTI sequences. In comparison to [54], this solution does not leave the geometric problem to the inner workings of the CNN.

The approach that our work draws most inspiration from is KISS-ICP [55], a LiDAR-only odometry framework built around point-to-point ICP. This method proposes a few modifications that cooperate towards a hardware-agnostic solution, with a small number of adjustable parameters. The first contribution is a constant velocity motion estimation, which provides the optimization step with an initial guess. The same motion estimation is used for distortion compensation. Secondly, feature extraction is replaced by two stages of voxel-based down-sampling: the first down-sampled point cloud is used to extend the map, while the even lower-resolution set is registered against the existing map to compute the pose estimate. Perhaps the key component is an adaptive distance threshold for correspondence outlier removal. This threshold is updated based on the deviation between the predicted and optimized pose, acting as a form of uncertainty estimation. Additionally, the optimization problem employs a robust kernel, whose scale parameter is related to the adaptive threshold.

[2] `https://projects.asl.ethz.ch/datasets/enwide`

Chapter 3

Methodology

This chapter will address the core contributions of our project. We present the hardware involved in sensor fusion, describe the data acquisition procedure, then introduce our final solution, by detailing the research and implementation process, as well as design decisions that had to be made along the way.

3.1 Prerequisites

Let us first introduce the mathematical notions used throughout this work. The group of rotations in 3D space is $\mathrm{SO}(3) = \left\{\mathbf{R} \in \mathbb{R}^{3\times 3} : \det \mathbf{R} = 1, \mathbf{R}^{\mathrm{T}}\mathbf{R} = \mathbf{I}_3\right\}$. For any vector $\mathbf{w} \in \mathbb{R}^3$ we can define a *skew-symmetric* matrix:

$$\widehat{\mathbf{w}} = [\mathbf{w}]_\times = \begin{bmatrix} 0 & -w_3 & w_2 \\ w_3 & 0 & -w_1 \\ -w_2 & w_1 & 0 \end{bmatrix} \tag{3.1}$$

and $\mathfrak{so}(3) = \{\widehat{\mathbf{w}} : \mathbf{w} \in \mathbb{R}^3\}$ is the set of all such matrices. The exponential maps $\exp : \mathfrak{so}(3) \to \mathrm{SO}(3)$ and $\mathrm{Exp} : \mathbb{R}^3 \to \mathrm{SO}(3)$ are defined as $\exp(\widehat{\mathbf{w}}) = \mathrm{Exp}(\mathbf{w}) = e^{\widehat{\mathbf{w}}}$, and Rodrigues' formula [56] provides a closed formulation of the matrix exponential:

$$e^{\widehat{\mathbf{w}}} = \mathbf{I}_3 + \frac{\widehat{\mathbf{w}}}{|\mathbf{w}|}\sin|\mathbf{w}| + \frac{\widehat{\mathbf{w}}^2}{|\mathbf{w}|^2}\left(1 - \cos|\mathbf{w}|\right). \tag{3.2}$$

The inverse operations are $\log : \mathrm{SO}(3) \to \mathfrak{so}(3)$ and $\mathrm{Log} : \mathrm{SO}(3) \to \mathbb{R}^3$, respectively. If $\widehat{\mathbf{w}} = \log(\mathbf{R})$ and $\mathbf{R} \neq \mathbf{I}_3$, then:

$$|\mathbf{w}| = \cos^{-1}\left(\frac{\mathrm{tr}(\mathbf{R}) - 1}{2}\right) \text{ and } \frac{\mathbf{w}}{|\mathbf{w}|} = \frac{1}{2\sin|\mathbf{w}|}\begin{bmatrix} r_{32} - r_{23} \\ r_{13} - r_{31} \\ r_{21} - r_{12} \end{bmatrix}, \tag{3.3}$$

and if $\mathbf{R} = \mathbf{I}_3$, $|\mathbf{w}| = 0$.

Furthermore, we consider the group of Euclidean transformations in 3D space (rotation and translation):

$$\mathrm{SE}(3) = \left\{ \mathbf{M} \in \mathbb{R}^{4\times4} : \mathbf{M} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^{\mathrm{T}} & 1 \end{bmatrix}, \mathbf{R} \in \mathrm{SO}(3), \mathbf{t} \in \mathbb{R}^3 \right\}. \tag{3.4}$$

The corresponding tangent space is:

$$\mathfrak{se}(3) = \left\{ \widehat{\xi} \in \mathbb{R}^{4\times4} : \widehat{\xi} = \widehat{\begin{bmatrix} \mathbf{v} \\ \mathbf{w} \end{bmatrix}} = \begin{bmatrix} \widehat{\mathbf{w}} & \mathbf{v} \\ \mathbf{0}^{\mathrm{T}} & 1 \end{bmatrix}, \mathbf{w} \in \mathbb{R}^3, \mathbf{v} \in \mathbb{R}^3 \right\}. \tag{3.5}$$

The exponential maps $\exp : \mathfrak{se}(3) \to \mathrm{SE}(3)$ and $\mathrm{Exp} : \mathbb{R}^6 \to \mathrm{SE}(3)$ are defined as:

$$\exp\left(\widehat{\xi}\right) = \mathrm{Exp}(\xi) = \begin{bmatrix} \mathrm{Exp}(\mathbf{w}) & \mathbf{V}(\mathbf{w})\mathbf{v} \\ \mathbf{0}^{\mathrm{T}} & 1 \end{bmatrix}, \tag{3.6}$$

where:

$$\mathbf{V}(\mathbf{w}) = \mathbf{V}(\theta\mathbf{u}) = \mathbf{I}_3 + \frac{1 - \cos\theta}{\theta}\widehat{\mathbf{u}} + \frac{\theta - \sin\theta}{\theta}\widehat{\mathbf{u}}^2. \tag{3.7}$$

The logarithmic map $\mathrm{Log} : \mathrm{SE}(3) \to \mathbb{R}^6$ is:

$$\xi = \mathrm{Log}\left(\begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^{\mathrm{T}} & 1 \end{bmatrix}\right) = \begin{bmatrix} \mathbf{V}^{-1}(\mathbf{w})\mathbf{t} \\ \mathrm{Log}(\mathbf{R}) \end{bmatrix}, \tag{3.8}$$

and we express $\log : \mathrm{SE}(3) \to \mathfrak{se}(3)$ as $\widehat{\xi} = \log(\mathbf{M}) = \widehat{\mathrm{Log}(\mathbf{M})}$. This allows introducing an interpolation function, with a parameter $\alpha \in [0, 1]$:

$$\varphi(\mathbf{M}_1, \mathbf{M}_2, \alpha) := \mathbf{M}_1 \exp\left(\alpha \cdot \log\left(\mathbf{M}_1^{-1}\mathbf{M}_2\right)\right). \tag{3.9}$$

3.2 Hardware

In this context, hardware refers to the set of sensors used for data collection, determined by the existing industrial setup. The data capturing system is controlled by an

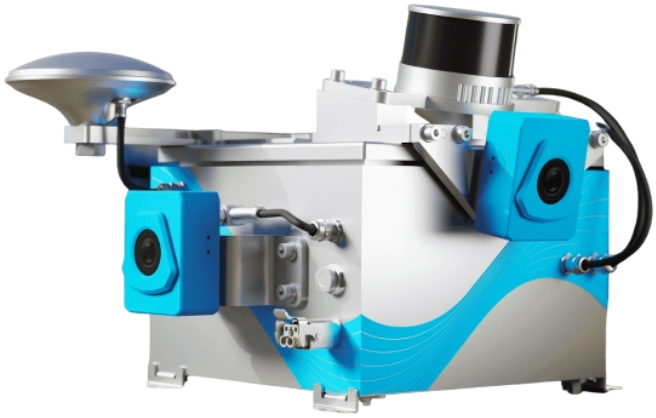


Figure 3.1: The SDX-Compact manufactured by Sodex Innovations GmbH. The set of sensors consists of a 3D LiDAR scanner, three RGB cameras and a high-accuracy positioning system. Image source: GeoLanes

Nvidia Jetson board, which operates as a middle-man for time synchronisation between the sensors, and coordinates the various data streams. Even though the solution was designed such that it does not inherently rely on any particular device, the relationship between hardware capabilities, data quality and final output makes it crucial to understand the sensors involved in the process. Beyond the components described below, the setup includes three industrial grade ArkCam Basic+ wide angle cameras which provide a 1920x1080 RGB stream over Ethernet. These are not utilised within this project, but represent a noticeable motivation for future work directions.

3.2.1 LiDAR Sensor

The SDX-Compact (Fig. 3.1) is equipped with a Pandar XT32 [57] LiDAR sensor, manufactured by Hesai Technology. This is a mechanical rotating LiDAR with a full 360° horizontal field of view and 32 beams distributed vertically, at 1° resolution. With our settings, the sensor produces 10 complete scans per second, resulting in a horizontal resolution of 0.18°. The maximum operational range is 120m, decreasing to 50m for low-reflectivity targets. The official specifications state a typical accuracy of ±1cm, with precision ±0.5cm, in a static environment. For each beam, the strongest return is processed, leading to 640,000 points being generated per second. The high output bandwidth is handled by an Ethernet connection, over which points are sent as UDP packets. The sensor also supports PTP synchronisation, essential for high-quality sensor fusion.

The same device has been used for LiDAR odometry applications in the past [58], and

has similar specifications to other popular scanners, such as Velodyne VLP-32C, Ouster OS1-32 or Robosense Helios 32.

3.2.2 GNSS/INS receiver

Another component of the sensor stack is the Septentrio AsteRx SBi3 Pro+ GNSS/INS receiver [59], which provides global positioning and orientation data at a rate of 100Hz. Internally, this relies on two distinct mechanisms.

The localization information comes from a dual antenna GNSS module compatible with several GNSS consellations (e.g., GPS[3], GLONASS, Galileo), to ensure optimal worldwide coverage. In standalone mode, the advertised typical accuracy is 1m, but the receiver also acts as an NTRIP (a protocol for differential GPS) client, gathering correction information, in order to achieve centimeter-level accuracy.

An Intertial Measurement Unit (IMU) module records acceleration data and provides the remaining orientation angles (roll, pitch, yaw) to compute the complete pose, in 6 Degrees of Freedom (DoF). This is integrated with the absolute GNSS measurements using the patented FUSE+ technology [60], resulting in an orientation error below 5-10°.

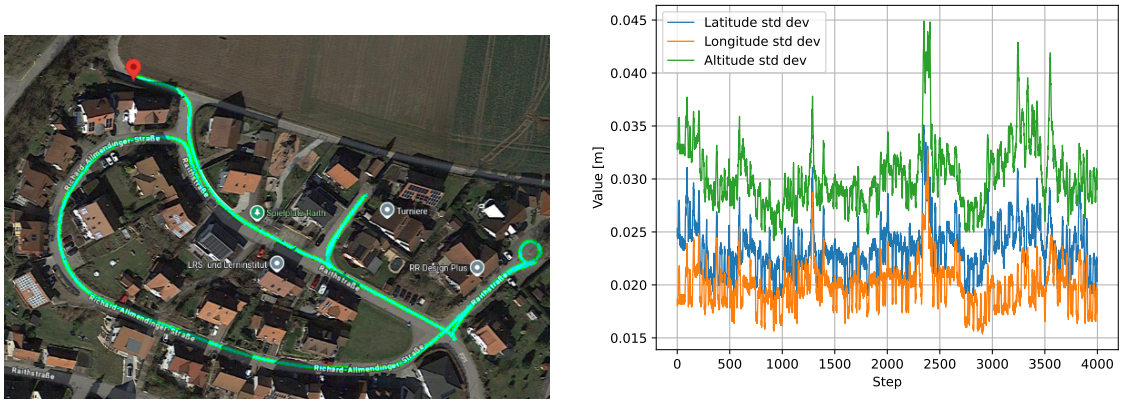
Like any system reliant on satellite communication, this will suffer significantly in situations where the signal propagation is disturbed (heavy clouds, "urban canyons", thick vegetation, spoofing), even leading to loss of *GNSS fix*.

To conclude this section, we recognize and underline the importance of accurate extrinsic calibration between the LiDAR and the local INS coordinate frame, which has to be performed prior to any reliable data collection procedure. Given the radically different modalities of these two sensors, this is not a trivial task [61] and lies outside the scope of the current work.

3.3 Data acquisition and pre-processing

Data is collected by mounting the sensor rig on the roof of a vehicle, and driving around a target area at a relatively low speed (e.g., up to 40km/h), see Fig. 3.2. In comparison with other public datasets [62, 63], capturing data in non-urban environments introduces some challenges — scenes dominated by vegetation, with few identifiable fea-

[3]Throughout this paper, GNSS and GPS are used interchangeably to refer to the global localization sensor, regardless of the underlying satellite system used during data collection.



(a) The trajectory, overlaid on a satellite image of the region.

(b) Standard deviation of GNSS readings along the trajectory.

Figure 3.2: An example dataset collected in rural Germany (Lienziegen).

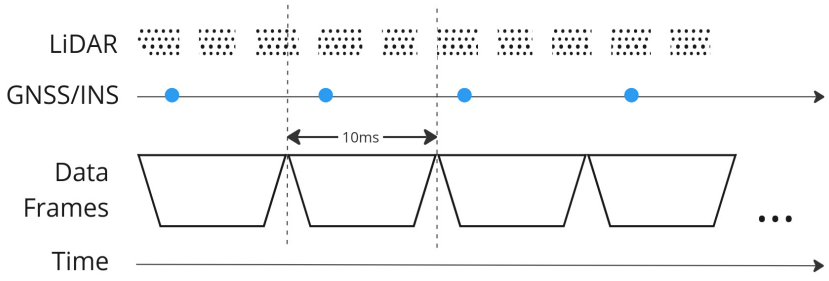


Figure 3.3: The data synchronisation process.
Time is discretized into fixed-size intervals, resulting in data frames of custom resolution. These are populated with information from the two sensors: LiDAR points arrive in groups, as UDP packets, while GNSS readings have a frequency of approx. 100Hz. We employ a data frame size of 10ms to ensure that each interval has an associated GNSS measurement, and place all corresponding packets in the same data frame.

tures, uneven terrain — while minimizing others — negligible amount of dynamic objects, no tall buildings that could affect GNSS signal propagation.

The raw sensor output is uploaded to a cloud storage facility, and is later processed into *data frames* that fuse the available information (Fig. 3.3). This can occur because the sensors are PTP-synchronized on initialization, and all recorded data is timestamped.

The next pre-processing step consists of dividing the sequence of data frames such that we operate on individual scans (also known as sweeps) produced by the LiDAR. A scan corresponds to a complete 360° rotation, which takes 100ms, so we join the points in 10 consecutive data frames to obtain a single scan.

Every INS reading undergoes a map projection, to obtain x, y, z coordinates in the East-North-Up frame. We consider the frame of the GNSS receiver as the *ego* coordinate system. The roll ϕ, pitch θ and yaw angles ψ determine the absolute orientation, so we

compute a global pose ${}^{W}\mathbf{T}_{ego} \in \mathrm{SE}(3)$ as:

$$ {}^{W}\mathbf{T}_{ego} = \mathrm{Translation}(x, y, z) \cdot \mathrm{Rot}_z(\psi) \cdot \mathrm{Rot}_x(\theta) \cdot \mathrm{Rot}_y(\phi), \tag{3.10} $$

where $\mathrm{Rot}_k(\alpha)$ is the transformation matrix corresponding to a rotation of α around axis k. Because the sensor provides error estimates in the form of global one-sigma values $\{\sigma_x, \sigma_y, \sigma_z, \sigma_\phi, \sigma_\theta, \sigma_\psi\}$, we construct the covariance matrix:

$$ \mathbf{\Sigma} = \mathrm{diag}\left(\sigma_x^2, \sigma_y^2, \sigma_z^2, \sigma_\phi^2, \sigma_\theta^2, \sigma_\psi^2\right) \tag{3.11} $$

and transform it using the adjoint map of the rotation component:

$$ \mathbf{\Sigma}_{ego} = \mathrm{Adj}\left(\mathbf{R}, \mathbf{0}\right) \cdot \mathbf{\Sigma} \cdot \mathrm{Adj}\left(\mathbf{R}, \mathbf{0}\right)^{\mathrm{T}}, \tag{3.12} $$

where:

$$ \mathrm{Adj}\left(\mathbf{R}, \mathbf{t}\right) = \begin{bmatrix} \mathbf{R} & \mathbf{0}_{3\times3} \\ [\mathbf{t}]_\times \mathbf{R} & \mathbf{R} \end{bmatrix}. \tag{3.13} $$

A LiDAR range reading r, captured at azimuth α with an elevation angle of ϕ, can be converted to a 3D location in the LiDAR frame:

$$ {}^{\mathrm{LiDAR}}\mathbf{p} = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} = \begin{bmatrix} r\cos\phi\sin\alpha \\ r\cos\phi\cos\alpha \\ r\sin\phi \end{bmatrix}. \tag{3.14} $$

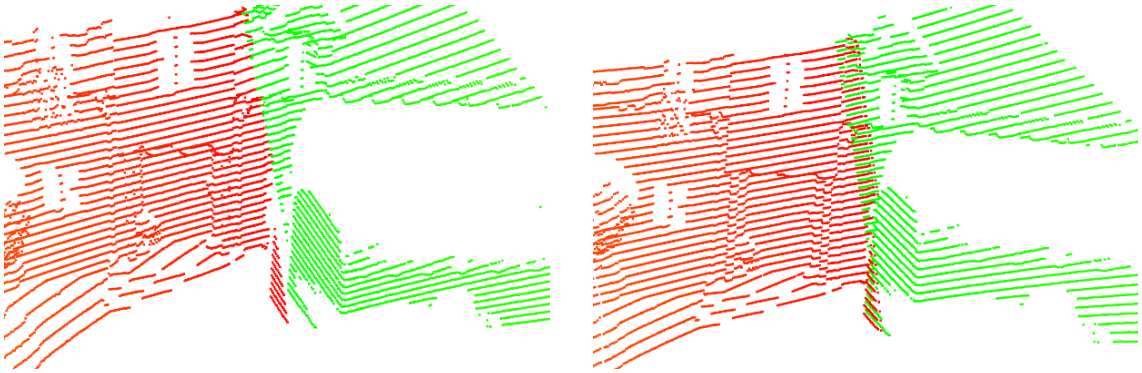
If ${}^{ego}\mathbf{T}_{\mathrm{LiDAR}}$ is the pose of the LiDAR in the ego frame (from extrinsic calibration), we can compute the location of a point in the ego frame:

$$ \begin{bmatrix} {}^{ego}\mathbf{p}_{\mathrm{LiDAR}} \\ 1 \end{bmatrix} = {}^{ego}\mathbf{T}_{\mathrm{LiDAR}} \begin{bmatrix} {}^{\mathrm{LiDAR}}\mathbf{p} \\ 1 \end{bmatrix}. \tag{3.15} $$

3.3.1 Motion compensation

At this stage, it is worth discussing the distortion effect that occurs when a rotating LiDAR sensor is moved at a relatively high velocity. Because it operates in a relative coordinate frame and different beams of the sweep are fired at different times, the beams corresponding to azimuth 0° will fire approximately 100ms earlier than the beams corresponding to azimuth 359°. Placing the resulting points in the same coordinate frame would lead to undesired artifacts, such as duplicate or warped structures. (Fig. 3.4a)

This open research problem [64] is commonly addressed by estimating the sensor pose change during a sweep [65, 55], and IMU integration proves satisfactory [66], given



(a) Before: an artificial duplicate surface (b) After: surface is corrected

Figure 3.4: Motion artifacts and the result of motion compensation.
Green: points from the beginning of the sweep. Red: points from the end of the sweep. Without motion compensation, the vertical surface yields two conflicting clusters, so the scan cannot be used for accurate mapping in the global frame.

the short time interval involved. In our case, the ego poses $\left\{{}^{W}\mathbf{T}_{i,0}, {}^{W}\mathbf{T}_{i,1}, \ldots, {}^{W}\mathbf{T}_{i,m}\right\}$ computed from INS measurements during sweep i are used to bring all points into the frame defined by ${}^{W}\mathbf{T}_{i,0}$. A point ${}^{i,k}\mathbf{p}$ belongs to data frame k, so it will be replaced by:

$$\begin{bmatrix} {}^{i,0}\mathbf{p} \\ 1 \end{bmatrix} = {}^{i,0}\mathbf{T}_{i,k} \begin{bmatrix} {}^{i,k}\mathbf{p} \\ 1 \end{bmatrix} = \left({}^{W}\mathbf{T}_{i,0}\right)^{-1} {}^{W}\mathbf{T}_{i,k} \begin{bmatrix} {}^{i,k}\mathbf{p} \\ 1 \end{bmatrix}. \tag{3.16}$$

We also experimented with approaches that do not rely on the absolute pose measurements for subsections of the sweep, such as interpolation using point timestamps or point indices (based on the order in which the points are returned), but these did not yield better results. A potential drawback of our method is that it disregards the localization noise, which could prove counterproductive if the GNSS receiver has low accuracy.

Through this step we are effectively removing the need for sub-scan information and creating a simpler data structure in which each scan is associated a single ${}^{W}\mathbf{T}_{ego}$ pose (from the first data frame), and the points in a sweep can be treated as a unified set.

Without loss of generality, we transform the sequence of global poses $\left\{{}^{W}\mathbf{T}_{0}, {}^{W}\mathbf{T}_{1}, \ldots, {}^{W}\mathbf{T}_{n}\right\}$ into the frame of the first pose, by multiplying each pose with $\left({}^{W}\mathbf{T}_{0}\right)^{-1}$, to obtain $\{\mathbf{T}_0, \mathbf{T}_1, \ldots, \mathbf{T}_n\}$. Naturally, $\mathbf{T}_0$ will always be $\mathbf{I}_4$, which simplifies the initialization of the odometry estimation. The covariance of each pose is adjusted with the adjoint of $\left({}^{W}\mathbf{T}_{0}\right)^{-1}$.

3.4 Solution architecture

At its core, the odometry and mapping solution that we propose requires minimal input: a sequence of point clouds captured by a moving LiDAR scanner, and a timestamp for each scan. If available, absolute localization/INS measurements can be used as additional input.

Let $\mathfrak{P} = \{P_0, P_1, \dots P_n\}$ represent the set of point clouds that we operate on, with point coordinates $P_k = \{\mathbf{p}_{k,i} : \mathbf{p}_{k,i} \in \mathbb{R}^3\}$ expressed in the local frame, $\mathfrak{s} = \{s_0, s_1, \dots s_n\}$ the corresponding timestamps, and $\widehat{\mathfrak{T}} = \left\{\widehat{\mathbf{T}}_0, \widehat{\mathbf{T}}_1, \dots \widehat{\mathbf{T}}_n\right\}$ the ground truth poses at which each scan was captured. In an ideal setting, a point cloud registration algorithm would take P_i and P_{i+1} as input and return the transformation $\Delta\widehat{\mathbf{T}}_i = \widehat{\mathbf{T}}_i^{-1}\widehat{\mathbf{T}}_{i+1}$, i.e., the ground truth displacement between consecutive poses. This would correspond to a perfect odometry model, and the complete trajectory could be reconstructed by accumulating the computed displacement. Unfortunately, with the exception of some simulation environments, such an approach is not actually feasbile. As we have already seen, LiDAR output is not perfect, especially in dynamic scenes, and some scenarios are simply unsuitable for odometry estimation based on 3D features [67].

The output of our solution can be formulated as $\mathfrak{T} = \{\mathbf{T}_0, \mathbf{T}_1, \dots \mathbf{T}_n\}$. Pose $\mathbf{T}_k \in \mathrm{SE}(3)$ represents the estimated location and orientation of the *ego* frame from which the points P_k were observed. The complete *map* estimate, represented as a larger point cloud in global coordinates, is the set of all points, each transformed according to their respective pose:

$$M(\mathfrak{T}, \mathfrak{P}) = \left\{ \mathbf{p}^*_{k,i} \in \mathbb{R}^3 : \begin{bmatrix} \mathbf{p}^*_{k,i} \\ 1 \end{bmatrix} = \mathbf{T}_k \begin{bmatrix} \mathbf{p}_{k,i} \\ 1 \end{bmatrix}, \mathbf{p}_{k,i} \in P_k, k \in \overline{0 \dots n} \right\}. \tag{3.17}$$

This uncovers two possible evaluation directions, which also constitute the high-level aims of our method. On one hand, the output trajectory $\mathfrak{T}$ should match the actual path of the system as accurately as possible. On the other, the final map $M(\mathfrak{T}, \mathfrak{P})$ should not hide or damage small details in the scene, to obtain a high-quality 3D reconstruction. Although not immediately obvious, these can result in conflicting requirements, but they anticipate an underlying optimization problem.

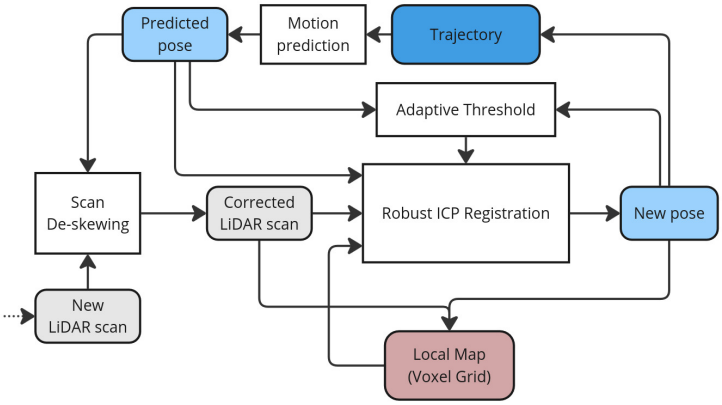


Figure 3.5: The KISS ICP pipeline.
An incoming scan is motion-compensated using the displacement between the previous estimated poses, then it is downsampled and registered against the local map. The pose estimated by the ICP algorithm is added to the trajectory, and the adaptive threshold is updated by evaluating the accuracy of the motion estimation. This determines the ICP outlier removal threshold, improving the system's ability to react to unexpected motions.

3.4.1 Overview

The final architecture is the result of a series of experiments. The process began with a LiDAR-only framework, immitating the KISS-ICP [55] implementation, shown in Fig. 3.5. In this approach, scans are processed one-by-one, attempting to compute the optimal pose at each step, and the map is built incrementally. This is computationally efficient and becomes essential when a robot operates in an unknown environment, as it provides a localization estimate at any given time. In the absence of absolute positioning information, however, this diverges from the ground-truth trajectory (Fig. 3.6), creating a distorted map.

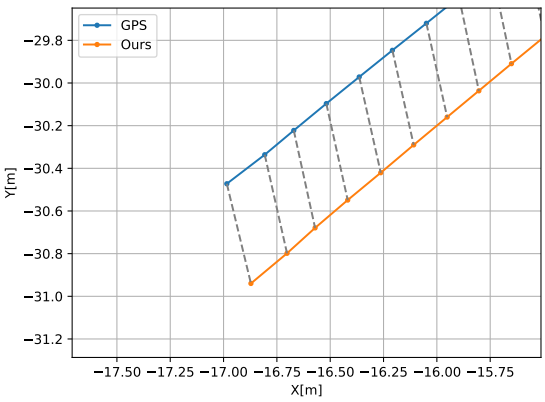


Figure 3.6: When the GPS information is not used, the trajectory computed based on LiDAR data diverges. Here, after 300 scans (approx. 30 seconds), the last location is approx. 0.5m away from the corresponding GPS position.

Early efforts focused on introducing the GPS readings as a constraint at each step, alongside motion prediction and registration results. This led to experimenting with each component, as well as methods for combining the respective poses, while regularly monitoring the quality of the output trajectories and maps. As the solution was evolving, it was tested on more challenging scenarios, by adding artificial noise to the GPS readings, or skipping some of them altogether.

Later on, we also considered addressing the pose estimation problem holistically, instead of in an incremental fashion, and experimented with architectures based on factor graph optimization [68], where nodes are linked using registration and GPS constraints. The same building blocks are needed, but the relationship between them is guided by the graph paradigm. Eventually, this was adapted to obtain a hybrid solution (Fig. 3.13), which combines the advantages of the two frameworks in a very intuitive manner.

3.4.2 Reconciling motion prediction, GNSS and registration

Before discussing the components in detail, we would like to provide a general picture of the rationale behind combining the pose estimates that we operate on. We assume that for every scan P_k we can compute three poses in the world frame, and aim to generate $\mathbf{T}_k$ as a function of these:

1. A registration-predicted pose $\mathbf{T}_k^{\text{ICP}}$, from a point cloud registration method that optimizes map alignment between scans in the global frame.
2. An absolute localization pose $\mathbf{T}_k^{\text{GPS}}$, generated by the INS sensor.
3. A motion-predicted pose $\mathbf{T}'_k$, output by the algorithm described in Section 3.4.3.

Each of these serves a different purpose: using only $\mathbf{T}_k^{\text{ICP}}$ would lead to a high-quality 3D map, while diverging from the actual locations; $\mathbf{T}_k^{\text{GPS}}$ estimates the absolute position and orientation, but its accuracy varies depending on the environment; the $\mathbf{T}'_k$ pose can help filter out noise in the GNSS data and smoothen the trajectory, but comes from a motion prediction mechanism that cannot operate in the absence of other sensors.

In an initial trial, we observed what happens when the GPS pose is used as the initial state of the registration algorithm, with the intuition that this could prevent trajectory divergence without significantly affecting map quality. In practice, this leads to an unwanted behavior. At first, the registration is not affected by the input pose, converging to the pure $\mathbf{T}_k^{\text{ICP}}$. As the map grows, the divergence is not constrained, so it reaches a stage where the $\mathbf{T}_k^{\text{GPS}}$ pose is a very poor initial guess for registration, which gets stuck in a

local minimum. Depending on the displacement between scans and the nature of the 3D scenes, this can even happen after just a few scans.

A subsequent idea was to treat $\mathbf{T}_k^{\text{ICP}}$ as local refinement for $\mathbf{T}_k^{\text{GPS}}$, by applying an interpolation, as defined in Eq. 3.9. If $\mathbf{T}_k = \varphi\left(\mathbf{T}_k^{\text{GPS}}, \mathbf{T}_k^{\text{ICP}}, \alpha\right)$, the interpolation parameter α controls how much the registration pose is allowed to alter the GPS prior — when $\alpha = 1$, we obtain the pure registration-based trajectory. This approach can prevent trajectory divergence while improving map quality, for small values of α (e.g., 0.1). However, since α is fixed, it does not address the problem of varying GPS noise or faulty registration results.

This can also be formulated as a generic optimization problem. First, we define a distance measure between an arbitrary pose $\mathbf{T}$ and a target pose $\mathbf{M}$:

$$\mathcal{D}\left(\mathbf{T}, \mathbf{M}, \mathbf{\Sigma}_{\mathbf{M}}\right) = \left(\text{Log}\left(\mathbf{T}^{-1}\mathbf{M}\right)\right)^{\text{T}} \cdot \mathbf{\Sigma}_{\mathbf{M}}^{-1} \cdot \text{Log}\left(\mathbf{T}^{-1}\mathbf{M}\right), \tag{3.18}$$

where $\mathbf{\Sigma}_{\mathbf{M}}$ is the covariance of $\mathbf{M}$, and $\text{Log}\left(\mathbf{T}^{-1}\mathbf{M}\right) \in \mathbb{R}^6$ is the twist vector residual. What we aim to optimize is:

$$\mathcal{J}\left(\mathbf{T}\right) = \beta_{\text{Reg}}\mathcal{J}_{\text{Reg}}(\mathbf{T}) + \beta_{GPS}\mathcal{D}\left(\mathbf{T}, \mathbf{T}_k^{\text{GPS}}, \mathbf{\Sigma}_{\text{GPS}}\right) + \beta_{Motion}\mathcal{D}\left(\mathbf{T}, \mathbf{T}_k', \mathbf{\Sigma}'\right), \tag{3.19}$$

where $\mathcal{J}_{\text{Reg}}$ defines the cost associated with the registration, and the β weights balance the influence of the components. If we assume that $\mathbf{T}_k'$ already includes the information from $\mathbf{T}_k^{\text{GPS}}$, the middle term can be discarded.

In one of our experiments, we expressed $\mathcal{J}_{\text{Reg}}$ as the L2 norm of the vector of point cloud registration residuals. In classical ICP, these are pairwise distances between points that have been matched using a nearest-neighbor search. However, optimizing this jointly with the other pose-based residuals (from $\mathcal{D}$) leads to numerical complications, because of the different scales of the errors involved. We note that this could represent a future research direction.

Another approach is to see $\mathbf{T}_k^{\text{ICP}}$ as the optimal registration pose and have

$$\mathcal{J}_{\text{Reg}}(\mathbf{T}) = \mathcal{D}\left(\mathbf{T}, \mathbf{T}_k^{\text{ICP}}, \mathbf{\Sigma}_{\text{ICP}}\right), \tag{3.20}$$

where $\mathbf{\Sigma}_{\text{ICP}}$ is a covariance matrix that estimates the registration uncertainty — this is explained in more detail in Section 3.4.4. The cost function becomes a weighted sum of

$\mathcal{D}$ functions, so the optimization problem at time step k is:

$$\mathbf{T}_k = \underset{\mathbf{T}}{\operatorname{argmin}} \sum_{\mathbf{M}} \beta_{\mathbf{M}} \mathcal{D}\left(\mathbf{T}, \mathbf{M}, \mathbf{\Sigma}_{\mathbf{M}}\right), \tag{3.21}$$

where $\mathbf{M} \in \left\{\mathbf{T}_k^{\text{ICP}}, \mathbf{T}_k^{\text{GPS}}, \mathbf{T}_k'\right\}$. As this is intractable on the non-linear manifold SE (3), an approximation of the solution can be computed using the Weiszfeld algorithm (Alg. 1). We disregard the scalar weights $\beta_{\mathbf{M}}$, as they can be accounted for, numerically, in the covariances $\mathbf{\Sigma}_{\mathbf{M}}$.

Algorithm 1 Weiszfeld algorithm for minimizing a custom pose distance metric.

Input:

- $\mathcal{M}$: list of poses $\mathbf{M}$ with associated covariance $\mathbf{\Sigma}_{\mathbf{M}}$
- $\mathbf{T}_0$: initial pose estimate
- $i_{\max}$: upper iteration threshold for the algorithm

Output:

- $\mathbf{T}$: approximation of the pose that minimizes $\sum_{\mathbf{M}} \mathcal{D}\left(\mathbf{T}, \mathbf{M}, \mathbf{\Sigma}_{\mathbf{M}}\right)$

Funct Weiszfeld($\mathcal{M}, \mathbf{T}_0, i_{\max}$)

$\mathbf{T} \leftarrow \mathbf{T}_0$
for $i = 1, \ldots, i_{\max}$ **do**
  $w_\xi \leftarrow \mathbf{0}_{6\times 1}$
  $w \leftarrow 0$
  for each $\mathbf{M} \in \mathcal{M}$ **do**
    $d_{\mathbf{M}} \leftarrow \max\left(\mathcal{D}\left(\mathbf{T}, \mathbf{M}, \mathbf{\Sigma}_{\mathbf{M}}\right), 10^{-7}\right)$ ▷ Apply threshold to avoid singularities
    $w_\xi \leftarrow w_\xi + \operatorname{Log}\left(\mathbf{T}^{-1}\mathbf{M}\right) d_{\mathbf{M}}^{-1}$
    $w \leftarrow w + d_{\mathbf{M}}^{-1}$
  end for
  if $w < 10^{-7}$ **then**
    break ▷ Early stop, degenerate weight sum
  end if
  $\xi = w_\xi / w$
  if $||\xi||_2 < 10^{-7}$ **then**
    break ▷ Early stop, no significant update occurred
  end if
  $\mathbf{T} \leftarrow \mathbf{T}\operatorname{Exp}\left(\xi\right)$
end for
return $\mathbf{T}$

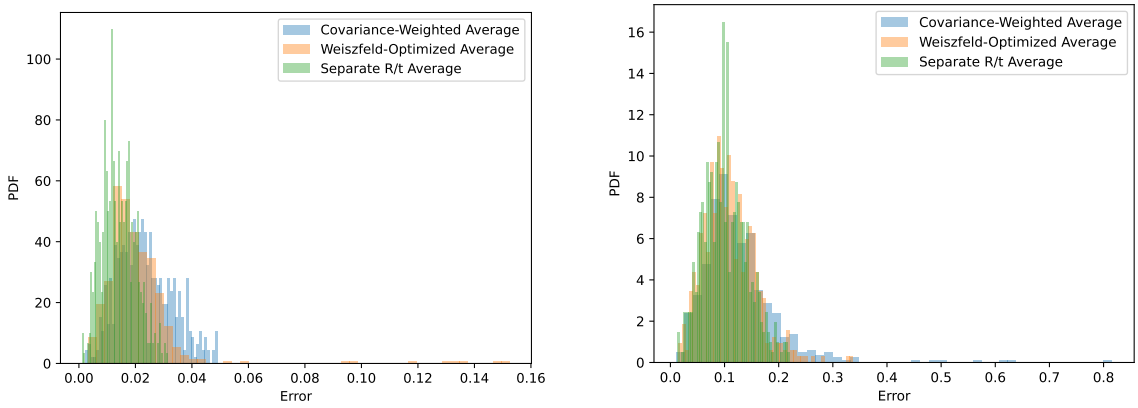
Alternatively, we observe that Eq. 3.21 involves a covariance-weighted sum of residuals, so another approximation of the solution is obtained from the covariance-weighted average in the twist vector space:

$$\xi = \left(\sum_{\mathbf{M}} \mathbf{\Sigma}_{\mathbf{M}}^{-1}\right)^{-1} \sum_{\mathbf{M}} \mathbf{\Sigma}_{\mathbf{M}}^{-1} \operatorname{Log}\left(\mathbf{T}_0^{-1}\mathbf{M}\right), \tag{3.22}$$

where $\mathbf{T} = \mathbf{T}_0 \mathrm{Exp}(\xi)$ and $\mathbf{T}_0$ is a linearization pose.

In Structure from Motion (SfM) tasks [69], pose averaging is split into rotation and translation averaging. For a set of matrices $\mathcal{R} = \{\mathbf{R}_i \in \mathrm{SO}(3)\}$ with weights $\mathcal{W} = \{w_i \in \mathbb{R}\}$, we can first perform Singular Value Decomposition (SVD), $\mathbf{U}\mathbf{D}\mathbf{V}^{\mathrm{T}} = \sum_i w_i \mathbf{R}_i$. The *chordal mean* [70] is then obtained as $\bar{\mathbf{R}} = \mathbf{U}\mathbf{V}^{\mathrm{T}}$. The scalar weights must be approximated from the covariance matrix, e.g., $w_i = \mathrm{tr}(\mathbf{\Sigma}_{\mathbf{R}_i})$. For translations, we exploit the linearity properties and compute $\bar{\mathbf{t}} = \left(\sum_{\mathbf{t}} \mathbf{\Sigma}_{\mathbf{t}}^{-1}\right)^{-1} \sum_{\mathbf{t}} \mathbf{\Sigma}_{\mathbf{t}}^{-1} \mathbf{t}$.

These three approaches have been compared by generating large sets of poses with known covariance around an arbitrary reference pose (treated as the ground truth mean), and assessing how the predicted mean differs from it. The results of this experiment are presented in Fig. 3.7. We observed no significant effect of either of these methods on the final trajectory, so the covariance-weighted approximation was used, thanks to its relatively simple formulation.



(a) Distribution of average translation error. (b) Distribution of average rotation error.

Figure 3.7: Comparing the behaviour of three different pose averaging techniques. We generate a random pose (expected mean), then sample 100 poses around it, by sampling a normal distribution of known σ for each parameter. The sample poses are averaged using the target algorithm, and we compute a translation and a rotation error: $e(\mathbf{t}_1, \mathbf{t}_2) = ||\mathbf{t}_1 - \mathbf{t}_2||_2$ and $e(\mathbf{R}_1, \mathbf{R}_2) = \left|\left|\mathrm{Log}\left(\mathbf{R}_1^{-1}\mathbf{R}_2\right)\right|\right|_2$, respectively.

3.4.3 Motion prediction

The first component that we analyse addresses the problem of predicting the next pose in the trajectory. Given the trajectory computed so far $\{\mathbf{T}_0, \mathbf{T}_1, \dots \mathbf{T}_t\}$, we are looking for $\mathbf{T}'_{t+1}$ that estimates the location and orientation of scan P_{t+1} in the world frame.

This influences two other elements of the framework: motion compensation and point cloud registration. In the absence of IMU data, the displacement between consecutive

scans can be interpolated to correct individual points. Point cloud registration uses this predicted pose as an initial guess, for faster convergence, as we will see in a later section.

If the scans are captured at small, equal intervals, as is usually the case for LiDAR sensors, the movement can be approximated by a constant velocity model. The displacement between the previous two scans is propagated using $\Delta\mathbf{T}'_t \approx \Delta\mathbf{T}_{t-1} = \mathbf{T}_{t-1}^{-1}\mathbf{T}_t$, which leads to $\mathbf{T}'_{t+1} = \mathbf{T}_t\Delta\mathbf{T}_{t-1}$. In our case, the time steps are not necessarily equal, and we would like to correct this pose estimate using the GPS measurements, when available, so we experimented with a Kalman Filter for computing the location component of $\mathbf{T}'_{t+1}$. The state $\boldsymbol{x}_k \sim \mathcal{N}(\overline{\mathbf{x}}_k, \mathbf{P}_k)$ is formulated as:

$$\overline{\mathbf{x}}_k = \left[x_k, y_k, z_k, v_{xk}, v_{y_k}, v_{zk}\right]^{\mathrm{T}} \tag{3.23}$$

$$\mathbf{P}_k = \operatorname{diag}\left(\sigma_{x_k}^2, \sigma_{y_k}^2, \sigma_{z_k}^2, \sigma_{v_{xk}}^2, \sigma_{v_{y_k}}^2, \sigma_{v_{z\,k}}^2\right). \tag{3.24}$$

Over a time step Δt, this changes according to a linear model:

$$\widehat{\overline{\mathbf{x}}}_{k+1} = \mathbf{F}(\Delta t) \cdot \overline{\mathbf{x}}_k \tag{3.25}$$

$$\widehat{\mathbf{P}}_{k+1} = \mathbf{F}(\Delta t)^{\mathrm{T}} \cdot \mathbf{P}_k \cdot \mathbf{F}(\Delta t) + \mathbf{Q}(\Delta t), \tag{3.26}$$

where $\mathbf{F}(\Delta t) = \begin{bmatrix} \mathbf{I}_3 & \mathbf{I}_3\Delta t \\ \mathbf{0}_3 & \mathbf{I}_3 \end{bmatrix}$ and $\mathbf{Q}(\Delta t)$ is the estimated process noise, which scales with the time interval. In the update step, we compute the new mean and covariance using a measurement $\boldsymbol{z}_k \sim \mathcal{N}(\mathbf{z}_k, \mathbf{R})$, by applying a Kalman gain:

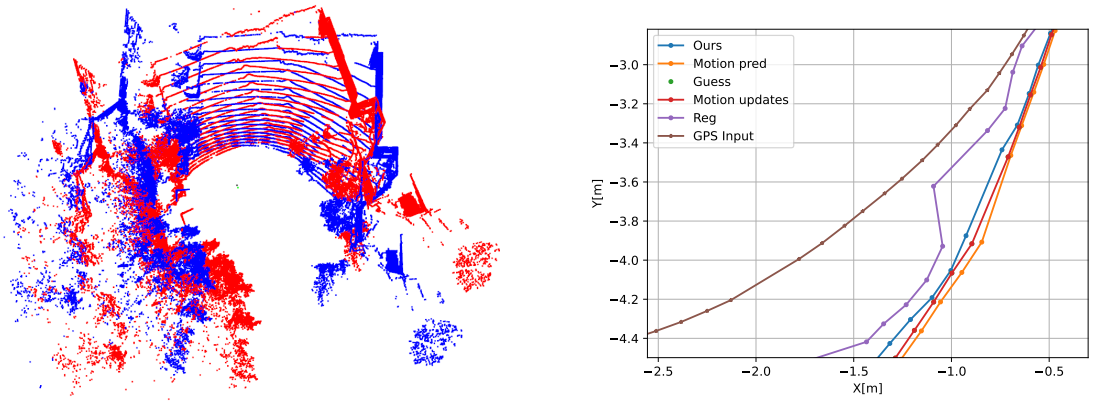
$$\mathbf{K} = \widehat{\mathbf{P}}_{k+1}\mathbf{H}\left(\mathbf{H}\widehat{\mathbf{P}}_{k+1}\mathbf{H}^{\mathrm{T}} + \mathbf{R}\right)^{-1} \tag{3.27}$$

$$\overline{\mathbf{x}}_{k+1} = \widehat{\overline{\mathbf{x}}}_{k+1} + \mathbf{K}\left(\mathbf{z}_k - \mathbf{H}\widehat{\overline{\mathbf{x}}}_k\right) \tag{3.28}$$

$$\mathbf{P}_{k+1} = (\mathbf{I} - \mathbf{K}\mathbf{H})\,\widehat{\mathbf{P}}_{k+1}. \tag{3.29}$$

The $\mathbf{z}_k$ vector consists of a location $[x_{\mathrm{GPS}}, y_{\mathrm{GPS}}, z_{\mathrm{GPS}}]^{\mathrm{T}}$ and velocity values $\frac{1}{\Delta t}[x_{\mathrm{GPS}} - x_k, y_{\mathrm{GPS}} - y_k, z_{\mathrm{GPS}} - z_k]^{\mathrm{T}}$, assuming linear displacement. The Jacobian $\mathbf{H}$ with respect to the state vector is $\mathbf{I}_6$, and the covariance of the measurement noise $\mathbf{R}$ can be estimated using the covariance of the GNSS reading. If no GNSS reading is associated with the current scan, the pose estimate is retrieved from the predicted position $\widehat{\overline{\mathbf{x}}}_{k+1}$, and the update step is performed based on the pose computed by the registration procedure.

This type of motion prediction method suffers from several disadvantages which outweigh its benefits for our use case. Firstly, it does not address the rotation component



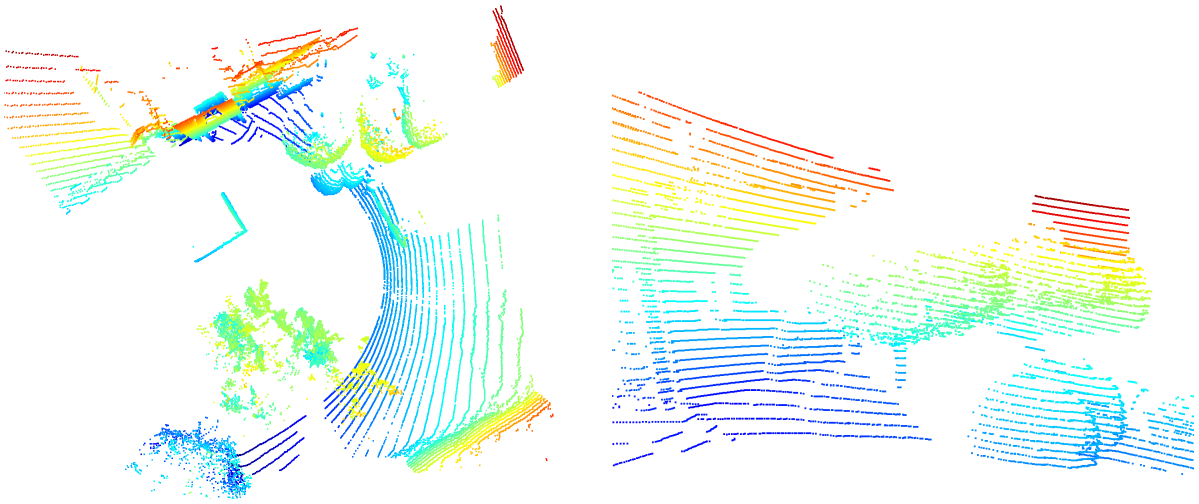
(a) Registration failure case, when the rotation between consecutive scans (red and blue) is estimated incorrectly. The ICP algorithm converges to an erroneous local minimum.

(b) If the motion model is too confident in its velocity estimate, or uneven time intervals are not handled correctly, the motion prediction will overshoot. Trajectory starts from top right.

Figure 3.8: Challenges of using a Kalman Filter motion model.

of the poses. Some possible workarounds include copying the orientation from the previous pose and letting the registration component correct that — this has been observed to fail when a larger time gap occurs, especially for curved trajectories (Fig. 3.8a) —, using the rotation returned by the INS, or having a separate method for computing the new rotation. Secondly, it introduces the need for reliable noise estimation (Fig. 3.8b), such that the model is reactive to unexpected movement, while filtering out measurement noise. Process noise covariance $\mathbf{Q}(\boldsymbol{\Delta t})$ and the measurement input covariance $\mathbf{R}$ require approximations which are difficult to balance for a general solution. Therefore, using the estimated uncertainty of the predicted pose in further computations would only propagate potential sources of error.

The approach that we propose relies on the interpolation properties of the SE (3) manifold (Eq. 3.9). In our case, using $\mathbf{T}_{t-1}$ and $\mathbf{T}_t$ to compute $\mathbf{T}'_{t+1}$ is actually a matter of extrapolation, because we expect α to be approximately 2 at most steps, which would lead to unstable behaviour in the Lie group transformation. To avoid this, we first compute $\mathbf{T}''_{t+1} = \mathbf{T}_t \mathbf{T}_{t-1}^{-1} \mathbf{T}_t$, and then obtain $\mathbf{T}'_{t+1} = \varphi\left(\mathbf{T}_t, \mathbf{T}''_{t+1}, \alpha_t - 1\right)$, where $\alpha_t = \frac{s_{t+1} - s_{t-1}}{s_t - s_{t-1}}$. For the scenarios that we have tested on, this provides a very good approximation of the correct displacement, which can be further refined using the information in the point cloud.



(a) The circular pattern on the ground section is typical for rotating LiDARs. Point density decreases with distance and incidence angle.

(b) Laser rays cannot penetrate opaque surfaces. This results in a "shadow" effect — no points are generated for objects that are occluded from the sensor viewpoint.

Figure 3.9: Particularities of LiDAR output.

3.4.4 Point Cloud Registration

The LiDAR sensor perceives the environment in a digital, discrete way, and outputs a set of 3D coordinates in the local frame. In a static scene, or after accurate motion compensation (Section 3.3.1), this represents a high-quality 3D snapshot of the environment from the current viewpoint. The mechanical design of the sensor introduces some particularities in the data (Fig. 3.9), but the measurement accuracy is excellent. As described earlier, applying a noise-resilient registration algorithm between consecutive scans constitutes a sensible odometry estimate.

The ICP algorithm

A baseline for *fine* registration methods, Iterative Closest Point (Algorithm 2) is naturally applicable to our scenario, albeit with some modifications. As opposed to *coarse* registration, where the aim is to find a rough alignment between arbitrary point clouds, we can exploit the motion prediction and/or the GPS readings as prior information about their global pose.

At every iteration step, the algorithm generates a set of correspondence hypotheses — a heuristic approximation of true correspondences — by finding the closest point from Q to each point in P'. The assumption is that the alignment of the two point clouds is improved by minimizing the distance associated with each correspondence. Depending on

Algorithm 2 Pseudocode of the point-to-point ICP algorithm.

Input:

- P, Q: two point clouds, in local coordinate frames
- $\mathbf{T}_{\text{init}}$: initial transformation guess between P and Q
- $d_{\max}$: upper distance threshold for point neighbor search
- $i_{\max}$: upper iteration threshold for the algorithm

Output:

- $\mathbf{T}$: registration pose between P and Q

Funct ICP$(P, Q, \mathbf{T}_{\text{init}}, d_{\max}, i_{\max})$

$\mathbf{T} \leftarrow \mathbf{T}_{\text{init}}$
$P' \leftarrow P$ transformed with $\mathbf{T}$
for $i = 1, \ldots, i_{\max}$ **do**
    Compute nearest-neighbor correspondences between P' and Q
    Discard correspondences where $||p' - q||_2 > d_{\max}$
    Compute $\mathbf{T}' \leftarrow \underset{\mathbf{T}}{\operatorname{argmin}} \sum_{p',q} \left|\left| \mathbf{T} \begin{bmatrix} p' \\ 1 \end{bmatrix} - \begin{bmatrix} q \\ 1 \end{bmatrix} \right|\right|_2$
    $\mathbf{T} \leftarrow \mathbf{T}'\mathbf{T}$ ▷ Update the transformation estimate
    $P' \leftarrow P'$ transformed with $\mathbf{T}'$ ▷ Update the point cloud
end for
return T

how the distance metric is defined, the algorithm can focus on different characteristics of the point clouds. The simplest formulation uses a point-to-point distance metric. Given a transformation estimate $\mathbf{T} = (\mathbf{R}, \mathbf{t})$, we have:

$$d_1(\mathbf{p}, \mathbf{q}) = ||\mathbf{R}\mathbf{p} + \mathbf{t} - \mathbf{q}||_2 . \tag{3.30}$$

This is a computationally-efficient approach, but is more sensitive to noise and outliers compared to the alternatives, and it does not take into account any geometrical features of the point clouds. To improve this, a point-to-plane metric [27] can be used, which considers the unit surface normal $\mathbf{n}_q$ computed at $\mathbf{q}$:

$$d_2(\mathbf{p}, \mathbf{q}) = \left(\mathbf{n}_q{}^{\mathrm{T}} \left(\mathbf{R}\mathbf{p} + \mathbf{t} - \mathbf{q} \right) \right)^2 . \tag{3.31}$$

Because $d_2(\mathbf{p}, \mathbf{q})$ is null when $\mathbf{p}$ lies on the plane defined by $\mathbf{q}$ and $\mathbf{n}_q$, this approach improves surface registration, but will struggle in unstructured environments where the normals cannot be reliably estimated. In general, this is achieved by considering the neighborhood of each point in Q, fitting a plane to it, and computing its normal.

In a more general formulation, the local surface properties are captured by the covariance matrices $\mathbf{C}_p, \mathbf{C}_q$ computed at $\mathbf{p}$ and $\mathbf{q}$ respectively. The resulting distance metric [28] is:

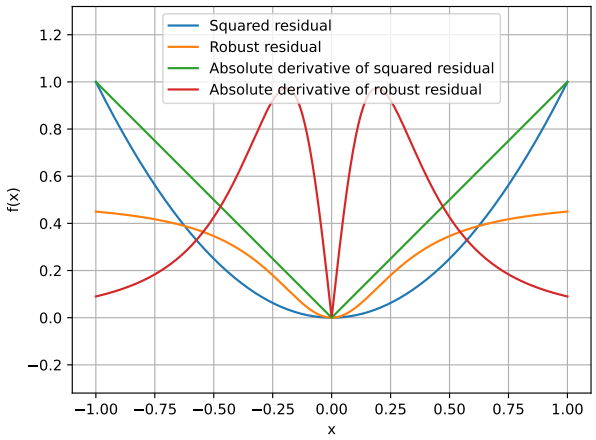


Figure 3.10: Applying the Geman-McClure function ($\kappa = 0.11$) to the correspondence residuals modifies their relative contribution during least-squares optimization, potentially improving convergence.

$$d_3(\mathbf{p}, \mathbf{q}) = (\mathbf{R}\mathbf{p} + \mathbf{t} - \mathbf{q})^{\mathrm{T}} \left(\mathbf{R}\mathbf{C}_p\mathbf{R}^{\mathrm{T}} + \mathbf{C}_q\right)^{-1} (\mathbf{R}\mathbf{p} + \mathbf{t} - \mathbf{q}) \,. \tag{3.32}$$

All of the above can constitute the residual terms when computing the optimal transformation using a Least-Squares solver. To improve noise robustness, several additional techniques are employed. We prevent the creation of correspondences beyond a given distance threshold, to avoid introducing incorrect point pairs. Using a small threshold, however, causes the algorithm to converge to a local minimum, as it is unable to correct for larger displacements (e.g., when the initial transformation guess is far from the optimum). Inspired by *Trimmed ICP* [71], we sort correspondences by distance and discard those that are above the 70th percentile, under the assumption that the majority of pairs will provide sufficient information for a satisfactory alignment.

Moreover, we apply the Geman-McClure function (Fig. 3.10) on the point-to-point distance metric, like in [55], which diminishes the influence of large residuals during minimization. For a residual r, this is defined as:

$$\rho(r) = \frac{r^2/2}{\kappa + r^2}, \tag{3.33}$$

where κ is a custom scaling factor, set to 0.11 in our experiments. The optimization problem becomes:

$$\mathbf{T}^* = \underset{\mathbf{T}}{\operatorname{argmin}} \sum_i \rho\left(||\mathbf{R}\mathbf{p}_i + \mathbf{t} - \mathbf{q}_i||_2\right), \tag{3.34}$$

which can be solved using the Gauss-Newton algorithm. We parametrize $\mathbf{T}$ as a twist

vector $\xi \in \mathbb{R}^6$ and we linearize around $\mathbf{I}_4$, to obtain $\mathbf{T} \approx \mathbf{I}_4 + \widehat{\xi}$. The Jacobian of the residual $\mathbf{r}_i = \mathbf{R}\mathbf{p}_i + \mathbf{t} - \mathbf{q}_i$ with respect to ξ is:

$$\mathbf{J}_{\mathbf{r}_i} = \begin{bmatrix} \mathbf{I}_3 & -\widehat{\mathbf{t}} \end{bmatrix} \in \mathbb{R}^{3\times 6}, \tag{3.35}$$

and each correspondence is associated a weight, according to the robust kernel:

$$w(\mathbf{r}_i) = \frac{\kappa^2}{(\kappa + ||\mathbf{r}_i||_2^2)^2}. \tag{3.36}$$

The resulting normal equation is:

$$\sum_i w(\mathbf{r}_i) {\mathbf{J}_{\mathbf{r}_i}}^{\mathrm{T}} \mathbf{J}_{\mathbf{r}_i} \xi = \sum_i w(\mathbf{r}_i) {\mathbf{J}_{\mathbf{r}_i}}^{\mathrm{T}} \mathbf{r}_i, \tag{3.37}$$

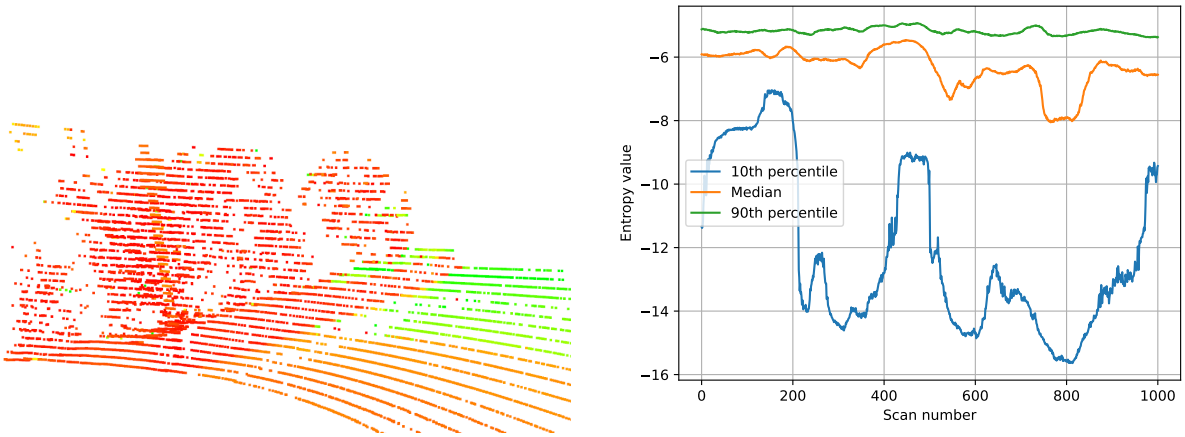
which can be rewritten as a conventional matrix equation $\mathbf{A}\xi = \mathbf{b}$. This can be solved in several ways, but we selected the Cholesky factorization method, thanks to its numerical stability. The wanted transformation is then retrieved as $\mathbf{T} = \mathrm{Exp}\,(\xi)$.

Point Cloud Filtering

We have also experimented with approaches that decrease the size of the point clouds used for registration. This is desirable because it reduces processing time and has the potential to remove points that skew registration results, such as those originating from regions occupied by trees or bushes. These regions are usually not sparse, so filtering based on the distance to the nearest neighbor or a related statistical measure was not very effective. Similarly, random downsampling is not effective, because it does not address the noise issues.

A promising method was based on the formulation described in [72], which computes a per-point entropy measure using the covariance of its neighborhood. A low entropy value indicates a structured, well-formed region (e.g., points on a planar surface), while high entropy is usually associated with noisy regions. (Fig. 3.11a) The size of the neighborhood, defined as a radius around each point, was set to 0.3m. Filtering discards points with entropy above a certain percentile, to avoid setting a fixed threshold that would not be able to adjust to variable scene features. (Fig. 3.11b) Unfortunately, this is computationally expensive and does not visibly improve registration results.

More advanced techniques involve finding salient points or structures that can be identified in multiple scans (also referred to as *landmarks*, in the SLAM paradigm), either



(a) Entropy values in a scene with plenty of vegetation. Red - high entropy, green - low entropy. For covariance estimation, the neighborhood radius is 0.3m.

(b) Per-scan entropy across a section of the dataset. Entropy values vary depending on the scene features, making it difficult to find a satisfactory global threshold.

Figure 3.11: Point Cloud entropy analysis.

through feature extraction [73] or 3D segmentation, but this introduces a map representation that is not really applicable to our use case.

We employ a voxelization mechanism similar to [55], to balance computation time and noise robustness. The 3D space is divided into equal-size cells and the first point in each cell is kept. The voxel size controls the resolution of the filtered point cloud, and should be smaller than the distance threshold used for correspondence generation, to avoid degenerate scenarios.

Estimating Registration Covariance

A particular challenge is raised by the need to estimate the covariance of the pose obtained as the registration result. This serves as a confidence measure for the optimization framework, where the residuals are scaled with the inverse of $\mathbf{\Sigma}_{\text{ICP}} \in \mathbb{R}^{6\times6}$. Brossard et al. [74] provided an in-depth analysis of this problem. Previous works have computed covariance based on the Jacobian involved in the correspondence distance minimization [75] or by employing a Kalman Filter that is updated for each point, using the mean squared correspondence distances as a measurement noise estimate [76].

We propose a heuristic approach based on the Root Mean Squared Error (RMSE) e of the inlier correspondences in the registration result — those for which the distance is below the provided threshold. Assuming that a perfect alignment corresponds to a

covariance $\mathbf{\Sigma}^*_{\text{ICP}}$ with $\sigma_{x,y,z} = 0.01$ and $\sigma_{\phi,\theta,\psi} = 2°$, we compute a factor:

$$r = 1 + \max\left(0, \frac{e - e_{\min}}{\sigma_e}\right) \tag{3.38}$$

and set $\mathbf{\Sigma}_{\text{ICP}} = r \cdot \mathbf{\Sigma}^*_{\text{ICP}}$. Here, σ_e represents the standard deviation of the RMSE values computed so far, so r is an indicator of how accurate the current registration is, relative to $e_{\min}$, the smallest RMSE observed up to that point. We note that RMSE can sometimes be misleading (e.g., when registration reaches a local minimum), but it provides a reasonable estimate of alignment quality in this context.

Summary

Following multiple experiments, we arrived at a procedure that can account for poor initialization and scene noise, while ensuring high-quality alignment between the current scan and the existing map. This is a two-step process, mimicking a coarse-to-fine registration procedure. Let M, S be the map and current scan point clouds, respectively, and $v(P, \alpha)$ a voxelization function that downsamples point cloud P using voxel size α (in meters). First, we perform robust point-to-point ICP between $v(S, 0.2)$ and $v(M, 0.4)$, using the motion prediction pose for initialization. The correspondence set is trimmed at the 70th percentile, and the number of iterations in the ICP procedure is not constrained, but we halt when the norm of the estimated ξ is below a small threshold (10^{-4}).

The resulting pose is used to intialize a dense registration stage: $v(S, 0.1)$ is aligned with $v(M, 0.1)$ using Generalized ICP (GICP), for at most 100 iterations. In both cases, we use a distance threshold of 0.5m, to account for corrections even in sparser areas of the point cloud (e.g., at the edges).

3.4.5 Map Handling

Because high-quality mapping constitutes one of our primary goals, having a sensible map representation is a key component of the solution. We treat the map as a point cloud in the global frame, built by concatenating a sequence of scans after transforming them with the best pose estimate computed so far. We do not attempt to store a set of key points, as this does not contribute to fine registration, but we constrain the size of the point cloud that we operate on using a sliding window approach. At every step, only the last $k = 10$ scans are considered. This straightforward method ensures temporal and

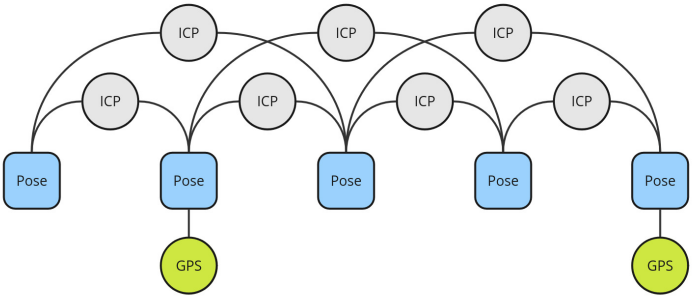


Figure 3.12: The factor graph structure.
If a GPS reading is available, a unary factor is created. The displacement predicted with point cloud registration (ICP) is used to create binary factors between consecutive poses, like a classical odometry factor. To achieve higher map quality, we also add "skip connections" based on the registration result.

spatial relevance: we do not store points that have been observed a long time in the past or points that are far away from the current location.

One drawback is that it hinders the implementation of a loop-closure mechanism, as there is no possibility to perform registration against older scans, but this is not a stringent problem, as the usage of GNSS readings prevents large localization drifts.

3.4.6 Graph Optimization

As stated earlier, we also experimented with solutions that employ factor graph optimization, thanks to its natural affinity to the SLAM problem. A factor graph [77] is a bipartite graph where nodes represent either:

- variables — these refer to variables in the problem domain; in our case, they are the estimated poses $\mathbf{T}_i$ associated with each LiDAR scan, or
- factors — probabilistic constraints, derived from measurements or prior information, whose residuals are to be minimized.

In our problem, two types of factors are employed. The GPS readings $\left(\mathbf{T}_i^{\mathrm{GPS}}, \mathbf{\Sigma}_i^{\mathrm{GPS}}\right)$ introduce unary factors that constrain a single pose variable. The twist vector residual is:

$$\mathbf{r}_i^{\mathrm{GPS}}\left(\mathbf{T}_i\right) = \mathrm{Log}\left(\left(\mathbf{T}_i^{\mathrm{GPS}}\right)^{-1}\mathbf{T}_i\right). \tag{3.39}$$

Odometry measurements $\left(\mathbf{T}_{ij}^{\mathrm{ICP}}, \mathbf{\Sigma}_{ij}^{\mathrm{ICP}}\right)$ obtained from the point cloud registration create binary factors that are connected to the two pose variables between which the displace-

ment occurred. The associated residual is:

$$\mathbf{r}_{ij}^{\mathrm{ICP}}\left(\mathbf{T}_i, \mathbf{T}_j\right) = \mathrm{Log}\left(\left(\mathbf{T}_{ij}^{\mathrm{ICP}}\right)^{-1}\left(\mathbf{T}_i^{-1}\mathbf{T}_j\right)\right). \tag{3.40}$$

If we define a covariance-scaled norm $||\mathbf{r}||_{\mathbf{\Sigma}}^2 = \mathbf{r}^{\mathrm{T}}\mathbf{\Sigma}^{-1}\mathbf{r}$ (note the similarity with Eq. 3.18), we can formulate the joint optimization problem:

$$\mathfrak{T}^* = \underset{\mathfrak{T}}{\mathrm{argmin}}\left(\sum_i \left|\left|\mathbf{r}_i^{\mathrm{GPS}}\left(\mathbf{T}_i\right)\right|\right|_{\mathbf{\Sigma}_i^{\mathrm{GPS}}}^2 + \sum_i \left|\left|\mathbf{r}_{ij}^{\mathrm{ICP}}\left(\mathbf{T}_i, \mathbf{T}_j\right)\right|\right|_{\mathbf{\Sigma}_{ij}^{\mathrm{ICP}}}^2\right). \tag{3.41}$$

In contrast with Section 3.4.2, this addresses the entire set of pose estimates, which is essential if trajectory corrections are to be applied retroactively and not just for the current pose.

The generic formulation of the registration pose residual $\mathbf{r}_{ij}^{\mathrm{ICP}}$ allows for a modification that improves overall map quality. At time step k, after aligning the current scan P_k to the existing local map built from scans $\{P_{k-5}, \dots P_{k-1}\}$, we create two odometry factors, for $(\mathbf{T}_{k-1}, \mathbf{T}_k)$ and $(\mathbf{T}_{k-2}, \mathbf{T}_k)$. The latter is motivated by the early observation that a registration-only trajectory yields a high-quality 3D map, so we use an odometry factor to enforce the registration result not only between consecutive poses. The final solution architecture, including the Factor Graph component, is presented in Fig. 3.13.

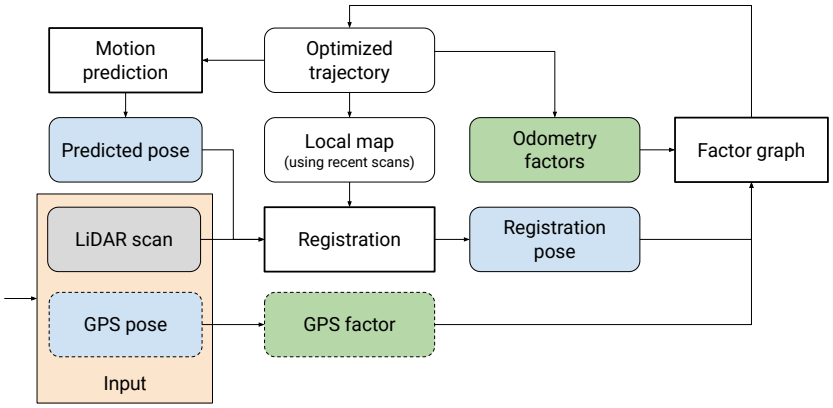


Figure 3.13: Solution architecture and workflow.
At each time step, the Factor Graph provides the best trajectory estimate using the information available so far. Registration is performed against a local map built from recent scans, using a motion-predicted pose as initial estimate. When a new pose is added to the graph, the registration result is used to create odometry factors that link it to previous poses. If a GPS pose is available, it introduces an additional factor.

3.5 Implementation details

The experiments and framework described above have been implemented in Python 3.8.10 [78], a high-level, interpreted programming language, which enabled fast development iterations. We make intensive use of existing libraries for linear algebra operations (NumPy [79], pytransform3d [80]) and point cloud processing (Open3D [81]). We treat poses and point clouds as NumPy arrays of size $(4, 4)$ and $(n, 3)$, respectively. During the registration process, the nearest-neighbor search required for correspondence generation relies on the KD-tree data structure implemented in SciPy [82]. As we do not perform any modifications to the algorithm, we use the Generalized ICP implementation provided by Open3D. For factor graph operations, we chose the popular GTSAM [83] library, despite the limited documentation available for its Python bindings.

An important mention is reserved to the Numba [84] compiler library, which enabled significant performance improvements for operations on large NumPy arrays. Performing motion compensation, for example, would not be possible in real-time (i.e., processing several LiDAR scans per second) in pure Python, but the just-in-time compilation provided by the Numba decorators, combined with parallel processing, alleviate this problem.

The code is divided into modules that group similar functionality, and we followed Object-Oriented Programming principles loosely, by having classes for the main components (e.g., Registration, Motion model, Graph optimization). For faster experimentation, the data structure presented in Section 3.3 is transformed into raw numerical form, stored on disk as `.npy` files, and re-loaded at runtime. Naturally, some custom interfacing would have to occur in order to apply the current implementation for a live robotic system, but our particular use case did not raise the need for integration with the ROS [85] framework.

Chapter 4

Results

This chapter provides a quantitative and qualitative assessment of our method, by applying it to various scenarios and analysing its output. We begin by formulating a set of odometry and mapping metrics, then observe how our solution behaves if some of its components are altered, and test it on multiple datasets.

4.1 Evaluation metrics

Let us first define the main metrics used for evaluation. In terms of trajectory, Zhang and Scaramuzza [86] provide a description of typical metrics for visual odometry, which also apply to LiDAR odometry. Given a ground-truth pose $\mathbf{T}_i = (\mathbf{R}_i, \mathbf{t}_i)$ and a corresponding pose $\widehat{\mathbf{T}}_i = \left(\widehat{\mathbf{R}}_i, \widehat{\mathbf{t}}_i\right)$, we can define translation and rotation errors as:

$$\begin{aligned} e_{\text{trans}}(\mathbf{t}_i, \widehat{\mathbf{t}}_i) &= \left|\left|\mathbf{t}_i - \widehat{\mathbf{t}}_i\right|\right|_2 \\ e_{\text{rot}}(\mathbf{R}_i, \widehat{\mathbf{R}}_i) &= \left|\left|\text{Log}\left(\widehat{\mathbf{R}}_i^{\text{T}} \mathbf{R}_i\right)\right|\right|_2 . \end{aligned} \tag{4.1}$$

A simple way to compare two pose sequences that start at the same pose is to compute $e_{\text{trans}}(\mathbf{t}_N, \widehat{\mathbf{t}}_N)$ and $e_{\text{rot}}(\mathbf{R}_N, \widehat{\mathbf{R}}_N)$, the errors at the final state N. A more informative measure is given by the Absolute Trajectory Error (ATE) [87]:

$$\begin{aligned} \text{ATE}_{\text{trans}} &= \sqrt{\frac{1}{N}\sum_{i=1}^{N} e_{\text{trans}}^2(\mathbf{t}_i, \widehat{\mathbf{t}}_i)} \\ \text{ATE}_{\text{rot}} &= \sqrt{\frac{1}{N}\sum_{i=1}^{N} e_{\text{rot}}^2(\mathbf{R}_i, \widehat{\mathbf{R}}_i)}. \end{aligned} \tag{4.2}$$

We compute this after transforming the ground truth and predicted trajectories such that they begin with $\mathbf{T}_0 = \mathbf{I}_4$. In the KITTI Benchmark [47], this was extended to Relative Trajectory Error (RTE), a metric that is applied on sub-sequences of a certain length or velocity, for a detailed evaluation. Given a sub-sequence (i, j), we compute $\mathbf{t}_{i,j} = \mathbf{R}_i{}^{\mathrm{T}}(\mathbf{t}_j - \mathbf{t}_i)$ and $\mathbf{R}_{i,j} = \mathbf{R}_i{}^{\mathrm{T}}\mathbf{R}_j$, then derive the relative error, as a percentage, using:

$$\begin{aligned} \mathrm{RTE}_{\mathrm{trans}} &= \frac{e_{\mathrm{trans}}(\mathbf{t}_{i,j}, \widehat{\mathbf{t}}_{i,j})}{||\mathbf{t}_{i,j}||_2} \cdot 100 \\ \mathrm{RTE}_{\mathrm{rot}} &= \frac{e_{\mathrm{rot}}(\mathbf{R}_{i,j}, \widehat{\mathbf{R}}_{i,j})}{||\mathrm{Log}\,(\mathbf{R}_{i,j})||_2} \cdot 100. \end{aligned} \tag{4.3}$$

Addressing one of the research questions that motivated this work, we also aim to quantify the quality of the resulting maps, to observe the improvements we bring to the GNSS-merged point cloud. Surprisingly, existing SLAM solutions rarely include rigorous map evaluation, because the map is seen as a localization-enabling tool, rather than a by-product of the system. The metrics here are closely related to the registration task, but differ in that they are applied on a complete 3D map, instead of a pair of point sets. We note that computing a single value for an entire point cloud is rather misleading, due to variations that appear between its regions, so we aim for point-based metrics that we analyse through a histogram or Cumulative Distribution Function (CDF).

A high-quality 3D map is expected to be dense, consistent, and provide high detail for observed features. A very trivial metric is the nearest-neighbor distance. Without considering degenerate cases, having a small nearest-neighbor distance is desirable, but this is constrained by the resolution of the sensor — the distance would be small even when point clouds do not align well, if the LiDAR has high resolution. Alternatively, we can compute the number of points within a given radius from each point. This is a reliable measure of local density, but it is unable to distinguish between randomly-scattered points and a structured region. Inspired by the point-to-plane distance (Eq. 3.31), we can compute a metric that measures the deviation of each point from the surface determined by its neighborhood. To avoid noise, we evaluate this for regions that have a high chance of representing planar surfaces, indicated by the normal consistency c. If $\mathbf{n}$ and $\{\mathbf{n}_1, \dots \mathbf{n}_k\}$ are the normals of a point and its neighbors, then $c = \left|\mathbf{n}^{\mathrm{T}} \frac{1}{k} \sum_{i \in (1,k)} \mathbf{n}_i\right|$.

Another useful metric is given by point entropy, as described in [72]. Considering all points within a radius r around point $\mathbf{p}_k$, we compute the covariance $\mathbf{\Sigma}(\mathbf{p}_k)$, then derive

the differential entropy as:

$$h(\mathbf{p}_k) = \frac{1}{2}\ln\left(2\pi e \det\left(\mathbf{\Sigma}(\mathbf{p}_k)\right)\right) \tag{4.4}$$

Adjusting the neighborhood radius controls the granularity of the metric, with a value around 0.2m providing a fair trade-off between detail level and computational cost.

To assess computational performance, we provide the processing duration (s) indicating the real time taken for a single input scan, without considering pre-processing operations, when executed on an Intel Core i7-10750H CPU.

4.2 Parameter analysis

In this section we discuss some of the main parameters of our method, in order to better understand their effect on performance, trajectory estimation and final map quality.

4.2.1 Point cloud voxelization

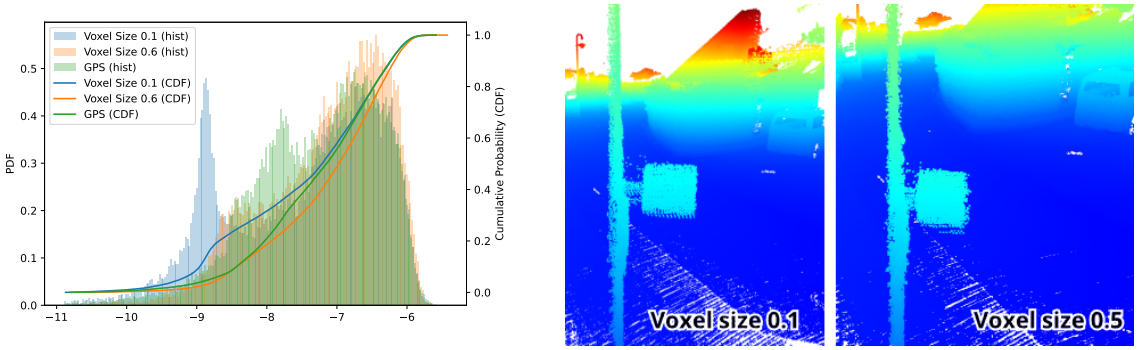
Following the motion compensation applied during dataset pre-processing, every input point cloud undergoes a voxelization step (as described in Section 3.4.4) designed to reduce sensor noise while keeping a sufficient amount of information for registration. We select a subsection of the trajectory consisting of 500 LiDAR scans captured over approx. 114m at 8km/h, and execute the pipeline in LiDAR-only mode (disregarding the GNSS information), with multiple voxel size values. As there are no GPS constraints, the graph optimization step does not modify the pose priors computed by LiDAR odometry. The corresponding GPS trajectory is treated as ground-truth, due to its low uncertainty. The results of this evaluation are presented in Table 4.1. Using a larger voxel size decreases the computational cost, as the point clouds involved have smaller resolution, but introduces considerable problems for odometry estimation, as indicated by the ATE values. We also look at the resulting maps, to confirm that a small voxel size leads to higher-quality mapping, as shown in Fig. 4.1.

4.2.2 Local map size

Next, we observe the behavior of the algorithm in relation to the size of the local map, i.e., the number of previous scans that the registration is performed against. Given the high

Voxel Size	Median Duration	ATE Trans.	ATE Rot.	Final Error Trans.	Final Error Rot.	Avg. RMSE	Avg. Corr. Trans.	Avg. Corr. Rot.
0.1	0.2984	1.2814	0.0294	2.5337	0.0467	0.0555	0.0080	0.0010
0.2	0.1684	1.3858	0.0315	2.7607	0.0511	0.0868	0.0210	0.0024
0.3	0.1586	1.5499	0.0347	3.0500	0.0542	0.1206	0.0351	0.0040
0.4	0.1589	1.7736	0.0418	3.5078	0.0659	0.1545	0.0519	0.0055
0.5	0.1603	1.4667	0.0358	2.9164	0.0568	0.1876	0.0819	0.0077
0.6	0.1635	1.9237	0.0475	3.7190	0.0677	0.2186	0.1069	0.0116

Table 4.1: Metrics for varying voxel sizes. We also report average RMSE values (after registration), alongside the average correction computed by the GICP registration, following the ICP alignment. Translation values are in meters, and rotation values in radians. The duration does not directly depend on the voxel size, because more ICP iterations are required for registration, when a larger voxel size is used.



(a) Distribution of point entropies.

(b) Visual comparison.

Figure 4.1: Voxel size effect on map quality.

frequency and operational range of the sensor, consecutive scans have large overlapping areas which represent useful cues for point cloud alignment. This is proven by the low odometry errors obtained for map size ≥ 5 in Table 4.2. When larger maps are used, the computational time increases, without significant improvements for trajectory estimation.

4.2.3 Registration strategy

We also assess the influence of the Generalized ICP registration step. During the experimental phase, we observed that this has a very positive effect on accurately matching the ground plane, and this is confirmed by the metrics in Table 4.3. For a more thorough analysis, we show the results corresponding to several distance threshold values, and compare against the case where only point-to-point ICP is used. The no-GICP version leads to higher trajectory errors overall, but we note that the Z component (elevation) has the highest contribution to this error. The XY error is smaller when GICP is not used

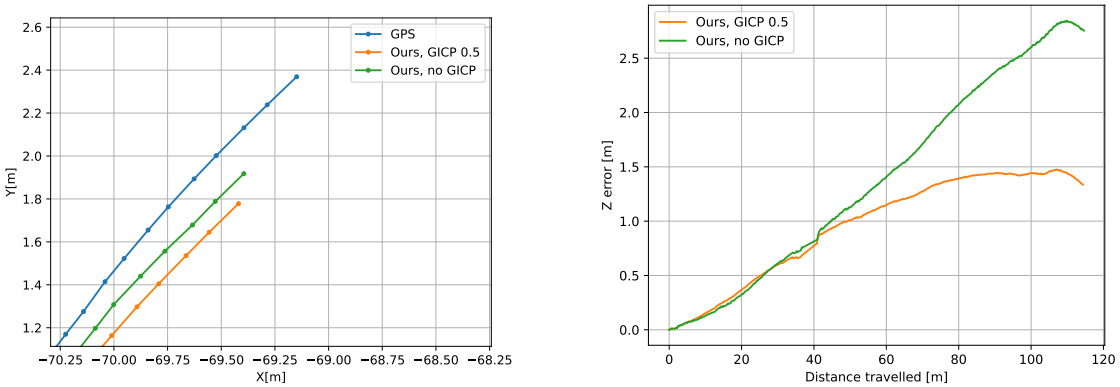
Map Size	Median Duration	ATE Trans.	ATE Rot.	Final Error Trans.	Final Error Rot.	Avg. RMSE	Avg. Corr. Trans.	Avg. Corr. Rot.
1	0.2349	5.0058	0.1564	10.2891	0.2603	0.0940	0.0366	0.0021
2	0.2308	4.3812	0.1355	9.1949	0.2318	0.0818	0.0171	0.0016
5	0.2632	1.8325	0.0458	3.6849	0.0734	0.0615	0.0094	0.0011
10	0.2962	1.2814	0.0294	2.5337	0.0467	0.0555	0.0080	0.0010
15	0.3242	1.1645	0.0263	2.2848	0.0415	0.0540	0.0076	0.0010
25	0.3984	1.1040	0.0245	2.1504	0.0384	0.0531	0.0079	0.0011

Table 4.2: Metrics for varying map sizes.

(Fig. 4.2a), but the Z error is almost doubled (Fig. 4.2b). In terms of map quality, using GICP clearly helps reduce point entropies and point-to-plane errors, so the surfaces of the output map capture more details, but additional constraints (i.e., GPS) will help prevent drifts.

GICP Dist. Thresh.	Median Duration	ATE Trans.	ATE Rot.	Final Error Trans.	Final Error Rot.	Avg. RMSE	Avg. Corr. Trans.	Avg. Corr. Rot.
-	0.2111	1.8036	0.0371	2.8009	0.0571	0.0601	-	-
0.1	0.2949	1.1686	0.0275	1.5486	0.0357	0.0457	0.0070	0.0009
0.2	0.3020	1.1167	0.0280	1.4442	0.0340	0.0520	0.0070	0.0009
0.5	0.2860	1.1958	0.0326	1.4858	0.0390	0.0599	0.0073	0.0010

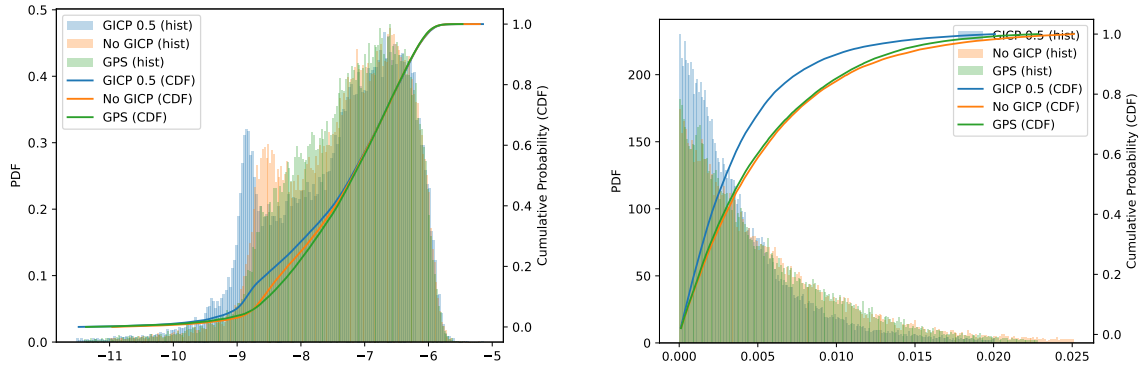
Table 4.3: Metrics for GICP variations.



(a) Trajectory end point location.

(b) Z error evolution.

Figure 4.2: Trajectory evaluation with and without GICP.



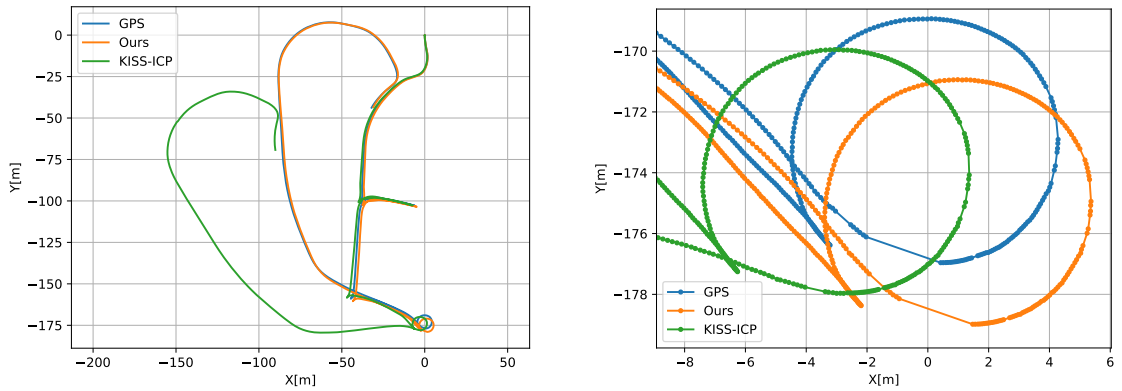
(a) Distribution of point entropies.

(b) Point-to-plane errors (first 20 scans)

Figure 4.3: Map evaluation with and without GICP.

4.3 Odometry evaluation

The first odometry evaluation we perform is on the trajectory recorded with the SDX-Compact system, depicted in Fig. 4.4. We do not use the GPS data during execution, but treat it as ground truth to obtain the trajectory error metrics in Table 4.4. Because we consider scan timestamps during motion prediction, we are able to handle large "jumps" in the scan sequence (Fig. 4.4b), something that other methods struggle with. However, elevation estimation is still an issue, as seen in Fig. 4.5b.



(a) Full trajectory comparison.

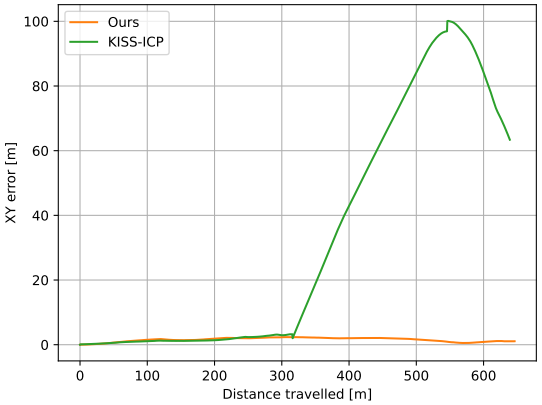
(b) A troublesome region with time jumps.

Figure 4.4: Odometry evaluation on custom trajectory: our method and KISS-ICP [55] against GPS data.

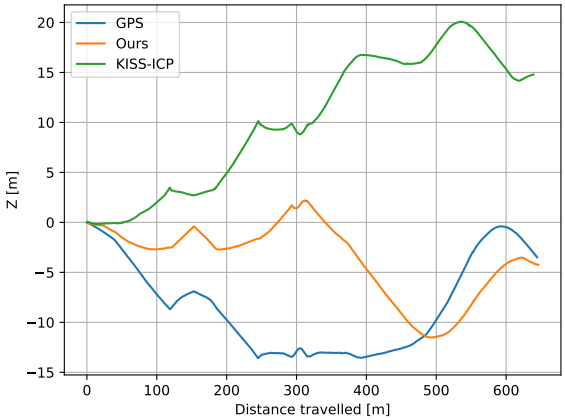
We also evaluate on the sequences of the KITTI dataset [88] which include ground-truth pose values from a GPS + IMU sensor system. This is a popular benchmark for urban SLAM problems, and it represents an interesting challenge for our solution because it uses a different LiDAR sensor (Velodyne HDL-64E), the environment is different, and

	ATE		Final		Avg. RTE					
					$k = 1$		$k = 10$		$k = 100$	
	tra.	**rot.**	**tra.**	**rot.**	**tra.**	**rot.**	**tra.**	**rot.**	**tra.**	**rot.**
K-ICP	46.879	0.407	65.965	0.688	3.72	28.89	1.95	17.31	3.05	17.49
Ours	8.232	0.102	1.311	0.095	2.81	16.67	1.36	10.88	1.15	11.28

Table 4.4: Comparison with KISS-ICP [55] on custom trajectory. We report RTE (%) for sub-sequences of various lengths (k is the number of LiDAR scans).



(a) XY errors. The KISS-ICP method diverges after a jump in the sequence.



(b) Z values. Both methods struggle to correctly estimate the Z component of the location.

Figure 4.5: Position analysis on custom trajectory.

moving vehicles are present. To operate on this dataset, we adjust the solution as follows:

- Discard LiDAR points that are farther than 50m or closer than 1m, to avoid high distortion and points belonging to the sensor rig
- Use a larger base voxel size: as the scanned environment is larger, we use a default voxel size of 0.2m instead of 0.1m

The results of this evaluation are presented in Table 4.5.

4.4 GPS-integrated evaluation

4.5 Mapping evaluation

Seq.	Length	ATE XY	ATE Tra.	ATE Rot.	Final XY	Final Tra.	Final Rot.	Avg. RTE (100m) XY	Avg. RTE (100m) Tra.	Avg. RTE (100m) Rot.
00	3723.24	7.29	16.28	0.07	10.59	11.58	0.06	0.81	1.23	12.88
01	2453.26	24.53	190.1	0.21	39.9	291.51	0.3	0.98	1.31	45.7
02	5067.02	22.2	50.39	0.13	49.72	99.53	0.22	0.7	1.22	11.99
03	560.85	2.92	8.84	0.05	6.35	18.15	0.08	0.57	0.67	7.69
04	393.65	0.72	4.98	0.03	1.16	11.76	0.06	0.39	0.89	125.81
05	2205.2	4.62	6.92	0.04	9.2	13.79	0.07	0.5	1.02	20.44
06	1232.69	2.03	3.78	0.03	4.88	8.79	0.05	0.59	0.91	54.07
07	694.39	0.5	1.8	0.02	0.96	1.36	0.02	0.58	1.0	5.12
08	3222.02	16.73	28.33	0.08	20.9	33.15	0.11	1.04	1.52	24.66
09	1704.96	6.12	14.12	0.06	9.28	9.78	0.04	0.68	1.03	8.1
10	919.4	3.94	18.8	0.07	4.61	21.78	0.09	0.65	1.04	8.79

Table 4.5: Odometry evaluation on KITTI sequences. We report the XY-only error values to emphasize localization performance. RTE values are reported in %, averaged over all 100m sub-sequences.

Chapter 5

Conclusion

Bibliography

[1] Eric Roberts. *Robotics: A Brief History*. https://cs.stanford.edu/people/eroberts/courses/soco/projects/1998-99/robotics/history.html. Accessed: 2025-02-01. 1999.

[2] Mario Taddei. *Leonardo's Robots*. https://tinyurl.com/mt2a4s9x. Archived version accessed: 2025-02-01. 2008.

[3] Boston Dynamics. *Leaps, Bounds, and Backflips*. https://bostondynamics.com/blog/leaps-bounds-and-backflips/. Accessed: 2025-02-01. 2023.

[4] Subhodeep Mukherjee et al. "Humanoid robot in healthcare: A Systematic Review and Future Research Directions". In: *2022 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COM-IT-CON)*. Vol. 1. 2022, pp. 822–826. DOI: 10.1109/COM-IT-CON54601.2022.9850577.

[5] Kai-Yi Chin, Chin-Hsien Wu, and Zeng-Wei Hong. "A Humanoid Robot as a Teaching Assistant for Primary Education". In: *2011 Fifth International Conference on Genetic and Evolutionary Computing*. 2011, pp. 21–24. DOI: 10.1109/ICGEC.2011.13.

[6] Uvais Qidwai, Saad Bin Abul Kashem, and Olcay Conor. "Humanoid Robot as a Teacher's Assistant: Helping Children with Autism to Learn Social and Academic Skills". In: *Journal of Intelligent & Robotic Systems* 98.3 (2020), pp. 759–770. ISSN: 1573-0409. DOI: 10.1007/s10846-019-01075-1. URL: https://doi.org/10.1007/s10846-019-01075-1.

[7] Do Hyoung Kim et al. "Development of a Facial Expression Imitation System". In: *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2006, pp. 3107–3112. DOI: 10.1109/IROS.2006.282329.

[8] Francis B Colavita. "Human sensory dominance". In: *Perception & Psychophysics* 16.2 (1974), pp. 409–412.

[9] Fabian Hutmacher. "Why Is There So Much More Research on Vision Than on Any Other Sensory Modality?" In: *Frontiers in Psychology* 10 (2019), p. 2246. DOI: 10.3389/fpsyg.2019.02246. URL: https://doi.org/10.3389/fpsyg.2019.02246.

[10] David Hecht and Miriam Reiner. "Sensory dominance in combinations of audio, visual and haptic stimuli". In: *Experimental brain research* 193 (2009), pp. 307–314.

[11] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005. ISBN: 0262201623.

[12] Josep Aulinas et al. "The SLAM problem: a survey". In: *Artificial Intelligence Research and Development* (2008), pp. 363–371.

[13] Rudolph Emil Kalman. "A New Approach to Linear Filtering and Prediction Problems". In: *Transactions of the ASME–Journal of Basic Engineering* 82.Series D (1960), pp. 35–45.

[14] A.J. Davison and D.W. Murray. "Simultaneous localization and map-building using active vision". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24.7 (2002), pp. 865–880. DOI: 10.1109/TPAMI.2002.1017615.

[15] Chunbo Luo et al. "UAV position estimation and collision avoidance using the extended Kalman filter". In: *IEEE Transactions on vehicular technology* 62.6 (2013), pp. 2749–2762.

[16] Albert Palomer, Pere Ridao, and David Ribas. "Inspection of an underwater structure using point-cloud SLAM with an AUV and a laser scanner". In: *Journal of field robotics* 36.8 (2019), pp. 1333–1344.

[17] Pierre Del Moral. "Nonlinear filtering: Interacting particle resolution". In: *Comptes Rendus de l'Académie des Sciences-Series I-Mathematics* 325.6 (1997), pp. 653–658.

[18] Fuyu Nie et al. "LCPF: A Particle Filter Lidar SLAM System With Loop Detection and Correction". In: *IEEE Access* 8 (2020), pp. 20401–20412. DOI: 10.1109/ACCESS.2020.2968353.

[19] Hans Peter Moravec. *Obstacle avoidance and navigation in the real world by a seeing robot rover*. Stanford University, 1980.

[20] Andrew J. Davison et al. "MonoSLAM: Real-Time Single Camera SLAM". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 29.6 (2007), pp. 1052–1067. DOI: 10.1109/TPAMI.2007.1049.

[21] Georg Klein and David Murray. "Parallel tracking and mapping for small AR workspaces". In: *2007 6th IEEE and ACM international symposium on mixed and augmented reality*. IEEE. 2007, pp. 225–234.

[22] Christopher Mei et al. "RSLAM: A system for large-scale mapping in constant-time using stereo". In: *International journal of computer vision* 94 (2011), pp. 198–214.

[23] K. S. Arun, T. S. Huang, and S. D. Blostein. "Least-Squares Fitting of Two 3-D Point Sets". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* PAMI-9.5 (1987), pp. 698–700. DOI: 10.1109/TPAMI.1987.4767965.

[24] Paul J Besl and Neil D McKay. "Method for registration of 3-D shapes". In: *Sensor fusion IV: control paradigms and data structures*. Vol. 1611. Spie. 1992, pp. 586–606.

[25] Szymon Rusinkiewicz and Marc Levoy. "Efficient variants of the ICP algorithm". In: *Proceedings third international conference on 3-D digital imaging and modeling*. IEEE. 2001, pp. 145–152.

[26] Xiaoshui Huang et al. *A comprehensive survey on point cloud registration*. 2021. arXiv: 2103.02690 [cs.CV]. URL: https://arxiv.org/abs/2103.02690.

[27] Y. Chen and G. Medioni. "Object modeling by registration of multiple range images". In: *Proceedings. 1991 IEEE International Conference on Robotics and Automation*. 1991, 2724–2729 vol.3. DOI: 10.1109/ROBOT.1991.132043.

[28] Aleksandr Segal, Dirk Haehnel, and Sebastian Thrun. "Generalized-icp." In: *Robotics: science and systems*. Vol. 2. 4. Seattle, WA. 2009, p. 435.

[29] T. Masuda, K. Sakaue, and N. Yokoya. "Registration and integration of multiple range images for 3-D model construction". In: *Proceedings of 13th International Conference on Pattern Recognition*. Vol. 1. 1996, 879–883 vol.1. DOI: 10.1109/ICPR.1996.546150.

[30] Greg Turk and Marc Levoy. "Zippered polygon meshes from range images". In: *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*. 1994, pp. 311–318.

[31] Guy Godin, Marc Rioux, and Rejean Baribeau. "Three-dimensional registration using range and intensity information". In: *Videometrics III*. Vol. 2350. SPIE. 1994, pp. 279–290.

[32] Kari Pulli. "Multiview registration for large data sets". In: *Second international conference on 3-d digital imaging and modeling (cat. no. pr00062)*. IEEE. 1999, pp. 160–168.

[33] Sofien Bouaziz, Andrea Tagliasacchi, and Mark Pauly. "Sparse iterative closest point". In: *Computer graphics forum*. Vol. 32. 5. Wiley Online Library. 2013, pp. 113–123.

[34] Gérard Blais and Martin D. Levine. "Registering multiview range data to create 3D computer objects". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 17.8 (1995), pp. 820–824.

[35] Peter Biber and Wolfgang Straßer. "The normal distributions transform: A new approach to laser scan matching". In: *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453)*. Vol. 3. IEEE. 2003, pp. 2743–2748.

[36] Martin Magnusson, Achim Lilienthal, and Tom Duckett. "Scan registration for autonomous mining vehicles using 3D-NDT". In: *Journal of Field Robotics* 24.10 (2007), pp. 803–827.

[37] Martin Magnusson et al. "Evaluation of 3D registration reliability and speed-A comparison of ICP and NDT". In: *2009 IEEE International Conference on Robotics and Automation*. IEEE. 2009, pp. 3907–3912.

[38] Yueqian Shen et al. "MI-NDT: Multiscale Iterative Normal Distribution Transform for Registering Large-Scale Outdoor Scans". In: *IEEE Transactions on Geoscience and Remote Sensing* 62 (2024), pp. 1–13. DOI: 10.1109/TGRS.2024.3437162.

[39] Shoubin Chen et al. "NDT-LOAM: A real-time LiDAR odometry and mapping with weighted NDT and LFA". In: *IEEE Sensors Journal* 22.4 (2021), pp. 3660–3671.

[40] Zan Gojcic et al. "The perfect match: 3d point cloud matching with smoothed densities". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 5545–5554.

[41] Haowen Deng, Tolga Birdal, and Slobodan Ilic. "Ppfnet: Global context aware local features for robust 3d point matching". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 195–205.

[42] Alireza Asvadi et al. “3D Lidar-based static and moving obstacle detection in driving environments: An approach based on voxels and multi-region ground planes”. In: *Robotics and Autonomous Systems* 83 (2016), pp. 299–311.

[43] Liang Chen, Jian Yang, and Hui Kong. “Lidar-histogram for fast road and obstacle detection”. In: *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2017, pp. 1343–1348.

[44] Ji Zhang, Sanjiv Singh, et al. “LOAM: Lidar odometry and mapping in real-time.” In: *Robotics: Science and systems*. Vol. 2. 9. Berkeley, CA. 2014, pp. 1–9.

[45] Kenneth Levenberg. “A method for the solution of certain non-linear problems in least squares”. In: *Quarterly of applied mathematics* 2.2 (1944), pp. 164–168.

[46] Donald W. Marquardt. “An Algorithm for Least-Squares Estimation of Nonlinear Parameters”. In: *Journal of the Society for Industrial and Applied Mathematics* 11.2 (1963), pp. 431–441. DOI: 10.1137/0111030. eprint: https://doi.org/10.1137/0111030. URL: https://doi.org/10.1137/0111030.

[47] Andreas Geiger, Philip Lenz, and Raquel Urtasun. “Are we ready for autonomous driving? the kitti vision benchmark suite”. In: *2012 IEEE conference on computer vision and pattern recognition*. IEEE. 2012, pp. 3354–3361.

[48] Ji Zhang and Sanjiv Singh. “Visual-lidar odometry and mapping: Low-drift, robust, and fast”. In: *2015 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2015, pp. 2174–2181.

[49] Tixiao Shan and Brendan Englot. “LeGO-LOAM: Lightweight and Ground-Optimized Lidar Odometry and Mapping on Variable Terrain”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2018, pp. 4758–4765.

[50] Han Wang et al. “F-loam: Fast lidar odometry and mapping”. In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2021, pp. 4390–4396.

[51] Wei Xu and Fu Zhang. “FAST-LIO: A Fast, Robust LiDAR-Inertial Odometry Package by Tightly-Coupled Iterated Kalman Filter”. In: *IEEE Robotics and Automation Letters* 6.2 (2021), pp. 3317–3324. DOI: 10.1109/LRA.2021.3064227.

[52] Patrick Pfreundschuh et al. "COIN-LIO: Complementary Intensity-Augmented LiDAR Inertial Odometry". In: *2024 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2024, pp. 1730–1737.

[53] Zhichao Li and Naiyan Wang. "Dmlo: Deep matching lidar odometry". In: *2020 ieee/rsj international conference on intelligent robots and systems (iros)*. IEEE. 2020, pp. 6010–6017.

[54] Qing Li et al. "Lo-net: Deep real-time lidar odometry". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2019, pp. 8473–8482.

[55] Ignacio Vizzo et al. "KISS-ICP: In Defense of Point-to-Point ICP – Simple, Accurate, and Robust Registration If Done the Right Way". In: *IEEE Robotics and Automation Letters (RA-L)* 8.2 (2023), pp. 1029–1036. DOI: 10.1109/LRA.2023.3236571.

[56] Richard M Murray, Zexiang Li, and S Shankar Sastry. *A mathematical introduction to robotic manipulation*. CRC press, 2017.

[57] Hesai Technology. *XT16/32/32M High-Precision 360° Mid-Range Lidar*. https://www.hesaitech.com/product/xt16-32-32m/. Accessed: 2025-02-28.

[58] Tiziano Guadagnino et al. *Kinematic-ICP: Enhancing LiDAR Odometry with Kinematic Constraints for Wheeled Mobile Robots Moving on Planar Surfaces*. 2025. arXiv: 2410.10277 [cs.RO]. URL: https://arxiv.org/abs/2410.10277.

[59] Septentrio. *AsteRx SBi3 Pro+*. Accessed: 2025-02-28. 2025. URL: https://www.septentrio.com/en/products/gnss-receivers/gnss-ins-ruggedized-boxes/asterx-sbi3-pro-plus.

[60] Septentrio. *FUSE+ Sensor Fusion*. Accessed: 2025-02-28. 2025. URL: https://www.septentrio.com/en/learn-more/advanced-positioning-technology/fuse-sensor-fusion.

[61] Xin Nie et al. "Two-Step Self-Calibration of LiDAR-GPS/IMU Based on Hand-Eye Method". In: *Symmetry* 15.2 (2023). ISSN: 2073-8994. DOI: 10.3390/sym15020254. URL: https://www.mdpi.com/2073-8994/15/2/254.

[62] Holger Caesar et al. "nuScenes: A multimodal dataset for autonomous driving". In: *CoRR* abs/1903.11027 (2019). arXiv: 1903.11027. URL: http://arxiv.org/abs/1903.11027.

[63] Jean-Luc Déziel et al. “PixSet : An Opportunity for 3D Computer Vision to Go Beyond Point Clouds With a Full-Waveform LiDAR Dataset”. In: *CoRR* abs/2102.12010 (2021). arXiv: 2102.12010. URL: https://arxiv.org/abs/2102.12010.

[64] Matthew McDermott and Jason Rife. “Correcting Motion Distortion for LIDAR Scan-to-Map Registration”. In: *IEEE Robotics and Automation Letters* 9.2 (2024), pp. 1516–1523. DOI: 10.1109/LRA.2023.3346757.

[65] Seungpyo Hong, Heedong Ko, and Jinwook Kim. “VICP: Velocity updating iterative closest point algorithm”. In: *2010 IEEE International Conference on Robotics and Automation*. 2010, pp. 1893–1898. DOI: 10.1109/ROBOT.2010.5509312.

[66] Lei He, Zhe Jin, and Zhenhai Gao. “De-Skewing LiDAR Scan for Refinement of Local Mapping”. In: *Sensors* 20.7 (2020). ISSN: 1424-8220. DOI: 10.3390/s20071846. URL: https://www.mdpi.com/1424-8220/20/7/1846.

[67] Iulian Filip et al. “Lidar SLAM Comparison in a Featureless Tunnel Environment”. In: *2022 22nd International Conference on Control, Automation and Systems (ICCAS)*. 2022, pp. 1648–1653. DOI: 10.23919/ICCAS55662.2022.10003820.

[68] Frank Dellaert, Michael Kaess, et al. “Factor graphs for robot perception”. In: *Foundations and Trends® in Robotics* 6.1-2 (2017), pp. 1–139.

[69] Linfei Pan et al. “Global structure-from-motion revisited”. In: *European Conference on Computer Vision*. Springer. 2024, pp. 58–77.

[70] Richard Hartley et al. “Rotation averaging”. In: *International journal of computer vision* 103 (2013), pp. 267–305.

[71] D. Chetverikov et al. “The Trimmed Iterative Closest Point algorithm”. In: *2002 International Conference on Pattern Recognition*. Vol. 3. 2002, 545–548 vol.3. DOI: 10.1109/ICPR.2002.1047997.

[72] Daniel Adolfsson et al. “Coral–are the point clouds correctly aligned?” In: *2021 European conference on mobile robots (ECMR)*. IEEE. 2021, pp. 1–7.

[73] Radu Bogdan Rusu, Nico Blodow, and Michael Beetz. “Fast Point Feature Histograms (FPFH) for 3D registration”. In: *2009 IEEE International Conference on Robotics and Automation*. 2009, pp. 3212–3217. DOI: 10.1109/ROBOT.2009.5152473.

[74] Martin Brossard, Silvere Bonnabel, and Axel Barrau. "A New Approach to 3D ICP Covariance Estimation". In: *IEEE Robotics and Automation Letters* 5.2 (Apr. 2020), pp. 744–751. ISSN: 2377-3774. DOI: 10.1109/lra.2020.2965391. URL: http://dx.doi.org/10.1109/LRA.2020.2965391.

[75] Sai Manoj Prakhya et al. "A closed-form estimate of 3D ICP covariance". In: *2015 14th IAPR International Conference on Machine Vision Applications (MVA)*. 2015, pp. 526–529. DOI: 10.1109/MVA.2015.7153246.

[76] Rick H. Yuan, Clark N. Taylor, and Scott L. Nykl. "Accurate Covariance Estimation for Pose Data From Iterative Closest Point Algorithm". In: *NAVIGATION: Journal of the Institute of Navigation* 70.2 (2023), navi.562. ISSN: 0028-1522. DOI: 10.33012/navi.562. eprint: https://navi.ion.org/content/70/2/navi.562.full.pdf. URL: https://navi.ion.org/content/70/2/navi.562.

[77] Frank Dellaert. "Factor graphs and GTSAM: A hands-on introduction". In: *Georgia Institute of Technology, Tech. Rep* 2.4 (2012).

[78] Guido van Rossum and Python Software Foundation. *Python Programming Language*. Version 3.8.10. 2025. URL: https://www.python.org.

[79] Charles R. Harris et al. "Array programming with NumPy". In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

[80] Alexander Fabisch. "pytransform3d: 3D Transformations for Python". In: *Journal of Open Source Software* 4.33 (2019), p. 1159. DOI: 10.21105/joss.01159. URL: https://doi.org/10.21105/joss.01159.

[81] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. "Open3D: A Modern Library for 3D Data Processing". In: *arXiv:1801.09847* (2018).

[82] Pauli Virtanen et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

[83] Frank Dellaert and GTSAM Contributors. *borglab/gtsam*. Version 4.2a8. May 2022. DOI: 10.5281/zenodo.5794541. URL: https://github.com/borglab/gtsam.

[84] Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. "Numba: a LLVM-based Python JIT compiler". In: *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*. LLVM '15. Austin, Texas: Association for Computing Machinery, 2015. ISBN: 9781450340052. DOI: 10.1145/2833157.2833162. URL: https://doi.org/10.1145/2833157.2833162.

[85] Morgan Quigley et al. "ROS: an open-source Robot Operating System". In: vol. 3. Jan. 2009.

[86] Zichao Zhang and Davide Scaramuzza. "A tutorial on quantitative trajectory evaluation for visual (-inertial) odometry". In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2018, pp. 7244–7251.

[87] Rainer Kümmerle et al. "On measuring the accuracy of SLAM algorithms". In: *Autonomous Robots* 27 (2009), pp. 387–407.

[88] Andreas Geiger et al. "Vision meets robotics: The kitti dataset". In: *The international journal of robotics research* 32.11 (2013), pp. 1231–1237.

List of Figures

Acronyms

ATE Absolute Trajectory Error.

CDF Cumulative Distribution Function.

CNN Convolutional Neural Network.

DoF Degrees of Freedom.

EKF Extended Kalman Filter.

GICP Generalized ICP.

GLONASS Global Navigation Satellite System (Russian).

GNSS Global Navigation Satellite System.

GPS Global Positioning System.

ICP Iterative Closest Point.

IMU Intertial Measurement Unit.

INS Inertial Navigation System.

KF Kalman Filter.

LiDAR Light Detection and Ranging.

LIO LiDAR-Inertial Odometry.

LOAM LiDAR Odometry and Mapping.

NDT Normal Distributions Transform.

PF Particle Filter.

PTP Precision Time Protocol.

RMSE Root Mean Squared Error.

RTE Relative Trajectory Error.

RTK Real-Time Kinematics.

SfM Structure from Motion.

SLAM Simultaneous Localization and Mapping.

SVD Singular Value Decomposition.

UDP Universal Datagram Protocol.

Glossary

Field Robotics The area of robotics that focuses on robots operating in unstructured outdoor environments for tasks like agriculture or exploration.

GNSS fix The desirable situation in which a connection to the satellite system is established, such that live localization information can be transmitted.

k-d tree A data structure for organizing entries in a k-dimensional space using a tree structure. When k is not very large, it provides logarithmic look-up time.

keyframe An image frame or 3D scan that is used for odometry estimation. Because modern sensors usually operate at higher frequencies than needed for most algorithms, it is a reasonable decision to skip frames based on a predefined pattern or condition, without sacrificing accuracy.

odometry The process of computing relative displacement of a robot using on-board sensors.

point cloud A set of point coordinates obtained as the output of a scanning sensor.

registration The process of aligning multiple point clouds in a single coordinate system such that matching features are as close as possible.

sensor fusion The process of combining information from multiple sensors, to augment the knowledge about the environment and increase the level of autonomy.

Structure from Motion (SfM) The task of reconstructing a 3D scene and a sequence of camera poses from a set of images.

surveying The work of examining and recording the area and features of a piece of land in order to construct a map, plan, or detailed description thereof.

voxel A cuboid-shaped region in 3D space; a 3D cell.