Eötvös Loránd University

Faculty of Informatics

IFROS

Lidar-Based map-observation Localization Module for Autonomous Navigation

Author:
Eliyas Kidanemaraim Abraha
Bachelor of Science

Internal supervisor:
Zoltan Istenes
Assistant Professor

External supervisor:
Mohammad Aldibaja
Assistance Professor

Budapest, 2025

Contents

Chapter 1

Introduction

Accurate and consistent localization is a cornerstone of autonomous vehicle navigation. Whether operating on city streets, inside industrial warehouses, or across expansive campuses, autonomous systems rely on precise knowledge of their position to safely and efficiently perform tasks like obstacle avoidance, route following, and mission planning. As the demand for intelligent mobility grows, the need for localization systems that are both robust and real-time—especially in complex or signal-challenged environments—has become increasingly critic

1.1 Motivation

High-precision and reliable localization is crucial for autonomous vehicles and robots, as it directly affects tasks like path planning, obstacle avoidance, and other repetitive missions (e.g., warehouse deliveries or routine patrols)[1]. Typically, positioning relies on Global Navigation Satellite Systems (GNSS) and Inertial Navigation Systems (INS), which complement each other. GNSS can provide absolute position without drift—especially when using differential signals—while INS delivers frequent updates to track movement[2]. However, in urban areas with tall buildings or indoor environments, GNSS signals can be blocked or distorted, and INS alone may accumulate significant drift over time[3]. Moreover, these systems do not inherently provide information about the surrounding environment.

To address such limitations, researchers have adopted onboard sensors like LiDAR, monocular cameras, and stereo cameras. One widely used method is Simultaneous Localization and Mapping (SLAM), which enables a robot to estimate its position in a completely new environment without needing a prior map. While SLAM is flexible and powerful, it can drift if loop closures are infrequent or the vehicle cannot revisit known areas to correct its estimates.

An alternative that improves accuracy and stability is map-based localization, where a pre-built map (created beforehand by SLAM, GNSS-based surveys, or other methods) is used as a global reference [4]. In this approach, the vehicle's current sensor data—such as LiDAR scans—are matched against the detailed information in the map to determine its position. This matching process helps minimize drift and provides robust localization, even in environments where GNSS signals are unavailable or unreliable. Once the map is constructed, indoor and outdoor autonomous systems can repeatedly use it to perform their tasks more reliably, whether that involves delivering goods, patrolling a facility, or navigating urban streets. As a result, map-based localization is considered an indispensable component of future autonomous navigation, especially when consistent high accuracy and global reference information are required.

Despite its advantages, map-based localization introduces new challenges. Processing high-resolution 3D maps in real-time demands efficient data structures and fast scan-matching algorithms. Additionally, integrating local motion estimation (e.g., via LiDAR-Inertial Odometry) with global map alignment must be handled carefully to ensure both consistency and computational feasibility. Full graph-based optimization, while accurate, often becomes intractable over time due to the growth of the state space.

This thesis addresses these challenges by proposing a hybrid system that combines real-time LiDAR-Inertial Odometry (FAST-LIO2), efficient map-matching via multithreaded NDT (NDT-OMP), and a sliding-window factor graph fusion approach. The system dynamically loads submaps around the robot based on a configurable radius, allowing scalable scan-matching while maintaining global consistency. The goal is to develop a real-time, robust, and accurate localization framework suitable for large-scale and semi-dynamic environments.

1.2 Objective

The primary objective of this thesis is to develop a real-time, LiDAR-based localization framework that achieves both high-frequency motion estimation and drift-free global consistency in large-scale environments. This is accomplished by integrating LiDAR-Inertial Odometry with map-based scan matching, and fusing them through a computationally efficient sliding-window factor graph.

- To integrate a robust LiDAR-Inertial Odometry(LIO) module using FAST-LIO2, capable of providing real-time, low-drift pose estimates by tightly coupling LiDAR and IMU data.
- To develop a scan-to-map matching module by integrating a multithreaded Normal Distributions Transform (NDT-OMP) algorithm, enabling real-time alignment of incoming LiDAR scans with a pre-built 3D point cloud map.
- To design and implement a grid-based submap management system that loads only the relevant local tiles within a configurable radius of the robot's position and incrementally replaces out-of-range tiles without reloading the full map.
- To fuse odometry and map matching results using a sliding-window factor graph optimization approach, enabling efficient, bounded optimization that retains local consistency while discarding outdated states.
- To evaluate the proposed system on benchmark datasets and custom scenarios, measuring performance in terms of localization accuracy, computation time, memory usage, and robustness under varying environmental conditions.

1.3 Research Questions

The primary questions guiding this research are:

1. How does integrating a pre-built map improve the accuracy and robustness of LiDAR-based localization in GNSS-denied environments?
2. What is the impact of dynamic submap loading and grid-based map management on computational performance and scalability?
3. Can this approach maintain accurate localization in semi-dynamic or partially structured environments, such as urban or campus settings?
4. How can LiDAR-Inertial odometry and NDT-based map matching be effectively fused to enhance real-time localization accuracy and robustness?

1.4 Scope and limitations

1.4.1 Scope

The proposed localization system is designed for real-time operation in diverse environments, including indoor spaces, outdoor areas, and urban environments. It assumes the availability of a high-resolution 3D LIDAR map of the operation environment. The system provides accurate pose estimation by fusing LIDAR-Inertial Odometry with scan-to-map matching, and its particularly suited scenarios where GNSS is unreliable or unavailable.

1.4.2 Limitations

- The system does not perform global localization (i.e., it cannot determine an unknown initial pose within the map). It assumes the robot starts with a known or approximated initial position close to its true location.
- The approach relies entirely on a pre-built map; it does not perform simultaneous mapping (SLAM) or update the map during operation.
- Highly dynamic environments or areas with significant structural changes may affect scan matching accuracy.

1.5 Methodology overview

1.6 Overview of the thesis structure

- Chapter 2 – Literature Review: Reviews the state-of-the-art in LiDAR-based localization, including LiDAR–Inertial Odometry SLAM , map-based localization , scan matching and sensor fusion techniques.
- Chapter 3 – System Design and Implementation: Describes the design of the proposed system, including each module, the integration flow, and technical implementation details.
- Chapter 4 - Experimental setup and results: Presents experimental evaluations, including performance metrics and comparisons with baseline approaches.

- Chapter 5 – Discussion: Interprets the findings, discusses limitations, and addresses how the proposed system meets the research objectives.
- Chapter 6 - Conclusion and future work: Summarizes key contributions, discusses possible improvements, and proposes directions for further research.

Chapter 2

Litreature Review

2.1 Overview of Relevant Literature

This chapter presents an overview of relevant literature that underpins the design and development of the proposed localization system. Reviews key techniques in Lidar and LiDAR-Inertial Odometry, SLAM, and localization using a priori maps, with a focus on real-time performance, drift reduction, and global consistency.

2.1.1 LiDAR and LiDAR-Inertial Odometry Techniques

Localization is a crucial capability for mobile robots, referring to the process of determining the robot's position and orientation in a given environment. Early attempts at localization depended heavily on wheel encoders, commonly known as wheel odometry, but this method suffered from significant drift whenever the wheels slipped or the ground was uneven [5]. As robotics advanced, researchers explored range sensors and visual sensors to improve motion estimation [6]. At the same time, algorithms like Iterative Closest Point (ICP) emerged, enabling the alignment of consecutive scans and sparking a separate stream of research in range sensor-based odometry [7].

Building on these ideas, LiDAR-only odometry gained popularity due to LiDAR's immunity to lighting changes and ability to capture dense 3D data. One key approach uses straightforward scan matching via ICP [7], but such methods can struggle in environments with few unique geometric features. To address this, feature-based systems extract distinctive cues from each LiDAR scan, as demonstrated in LOAM [8], where sharp edges and planar surfaces guide the matching process. While LiDAR-

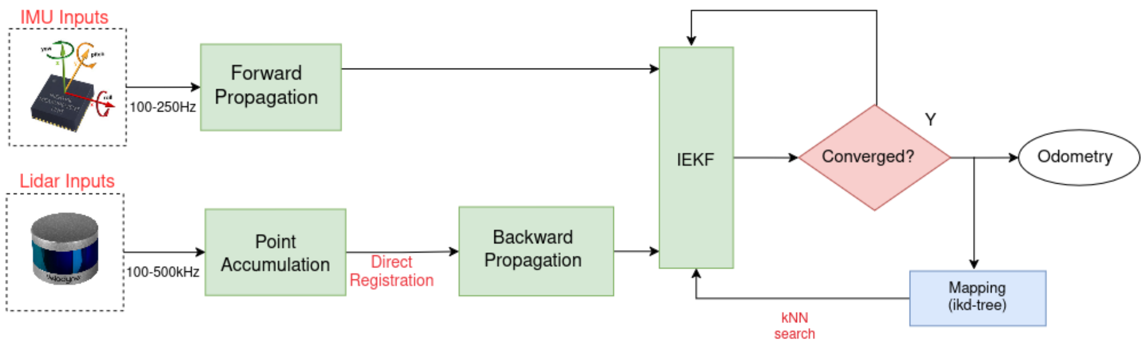

Figure 2.1: Architecture of FAST-LIO2: A tightly coupled LiDAR-Inertial Odometry [13]

only odometry can produce accurate short-term estimates, it may drift over time if the robot moves rapidly or the environment is mostly repetitive.

To reduce long-term drift, many researchers incorporate IMU (Inertial Measurement Unit) data, resulting in what is called LiDAR–Inertial Odometry (LIO). In a loosely coupled setup, LiDAR and IMU each compute their own pose, and then a higher-level filter (like an Extended Kalman Filter) merges these estimates [9]. This approach is easier to design because the LiDAR odometry module and the IMU integration module work somewhat independently. However, the separate modules might not fully exploit each other's measurements.

By contrast, tightly coupled designs feed raw IMU data directly into the LiDAR optimization. For instance, LIO-SAM employs a factor graph where both LiDAR and IMU "preintegration" factors are updated in the same framework, allowing more constraints to be used in each calculation [10]. As graph-based LiDAR–inertial odometry solutions grew more complex and demanding, researchers began exploring filter-based methods built around the Kalman filter. One notable example is LINS [11], which uses an iterated Error State Kalman filter (iESKF) to speed up pose estimation; however, it still faces performance bottlenecks when dealing with large numbers of LiDAR points, since computing the Kalman gain for every point can be costly. In response, FAST-LIO [12] introduced a more efficient Kalman gain calculation that significantly lowers this overhead. Later, As shown in Figure 2.1 FAST-LIO2 [13] expanded on that approach by completely removing the feature extraction step and matching raw LiDAR points directly to the map. To accelerate nearest-neighbor queries, it employs a specialized data structure called an ikd-Tree, allowing the system to achieve both high accuracy and even faster run.

Due to its high computational efficiency, accuracy, and real-time capability, FAST-LIO2 is adopted in this thesis as the LiDAR-Inertial Odometry (LIO) front-end. Its design aligns with the system's objective to maintain high-frequency odometry while minimizing drift, providing a solid foundation for downstream scan-to-map correction and graph-based fusion.

2.1.2 SLAM and Loop Closure-Based Localization

Moving beyond pure odometry, SLAM (Simultaneous Localization and Mapping) takes an extra step by adding loop closure mechanisms to maintain global consistency [14]. While LiDAR–Inertial Odometry can reduce local drift, it remains prone to cumulative error over long distances if no global corrections are made. By detecting revisits to previously mapped areas, a full SLAM pipeline—like Google Cartographer (for 2D LiDAR) [15] or other graph-based back ends [16]—refines the entire trajectory, resulting in notably higher accuracy. However, these systems carry greater computational cost and design complexity, since maintaining a global graph or submap structure calls for advanced data management. Some research also integrates GNSS (e.g., GPS) data for a global reference frame, further reducing drift when the robot is outdoors. Even then, GNSS signals often fail in heavily built-up city centers or fully indoor settings, limiting their usefulness for certain applications.

SLAM inherently relies on loop-closure events or prior constraints to limit its global drift; if these loop closures are sparse or absent over a long trajectory, the system can still accumulate significant positioning errors [16, 14]. That limitation underscores the motivation for map-based localization, in which an existing map is used to provide absolute constraints and reduce drift, even in lengthy or repetitive environments where loop closures are not guaranteed.

2.1.3 Localization Using Prior Maps

Estimation of the position of a sensor on a map is essential for navigation systems. While several kinds of map representations are used, depending on the use scenario 3D point cloud maps are among the most popular representations owing to their simplicity and expressiveness.[17]Because constructing a point cloud map is relatively easy with recent precise range sensors and mapping algorithms, point cloud maps are used for a wide range of applications, from indoor service robots to outdoor driving of autonomous vehicles.

Multiple recent works highlight the potential of map-aided localization for achieving robust and precise vehicle pose estimation.[18] propose building a 2D road-surface reflectivity map from lidar data, then using real-time lidar scan correlation against this map to localize within a few decimeters in busy urban areas. Its advantage is straightforward reflectivity matching, but it presupposes consistent road-surface intensity and requires dedicated prior mapping runs.[3] adopt a detailed segmentation approach: each incoming lidar frame is broken down into ground, curb, edge, and surface features, then aligned with a prior feature map; the refined estimates are further fused with a MEMS-IMU for centimeter-level accuracy at high frequency. Although this improves reliability even in dynamic traffic, the method can become computationally heavy due to multi-type feature extraction and large-scale map management.

By contrast,[19] utilize stereo-based “visual point cloud” maps, avoiding LiDAR hardware yet involving dense stereo matching for both the offline (map) and online (frame) steps. Their system then registers these point clouds via normal distribution transform (NDT). While it lowers sensor costs, it adds a heavy stereo workload and remains susceptible to lighting or texture issues. The authors also report that deviations can occur at sharp turns, albeit these are later corrected. Finally, [20] integrate LOAM odometry with a global segment-based prior map (via SegMap) for re-localization when drift accumulates. The system occasionally triggers place recognition in the global map, confirms geometry with RANSAC, then refines the pose using fine-grained ICP. Although segment-based matching proves efficient and effective, it still hinges on stable scene geometry and well-structured offline segmentation.

Overall, these methods demonstrate that leveraging a prior 3D or reflectivity map can curb drift and boost localization accuracy in challenging real-world environments. Still, each approach faces trade-offs in data collection overhead, dynamic scene handling, or computational costs.

2.1.4 Point Cloud Registration for Localization

Point cloud registration is a key technique in LiDAR-based localization, used to align a current LiDAR scan with a prior map to estimate the sensor's pose. The classic Iterative Closest Point (ICP) algorithm performs this by minimizing distances between matched point pairs across the two clouds. While ICP is conceptually simple, it suffers from sensitivity to noise, poor initial alignment, and local minima, making it unreliable for large-scale or noisy environments [7].

To address these limitations, the Normal Distributions Transform (NDT) was proposed by [21] and further elaborated in subsequent research[22]. NDT offers a compact representation of surfaces by converting point clouds into a set of local probability density functions (PDFs), each characterizing a surface section.Instead of relying on discrete point matches, NDT represents the map as a grid of voxels, each containing a Gaussian distribution that models the local geometry. Scan points are registered by maximizing their likelihood under these distributions. This makes NDT more robust to outliers, partial overlaps, and noisy measurements.

Modern implementations such as NDT-OMP further improve runtime performance through multi-threading, enabling real-time scan-to-map matching in large-scale environments [23]. Due to its robustness and efficiency, NDT is widely used in autonomous driving and is adopted in this thesis for scan-matching against a pre-built point cloud map.

2.2 Theoretical Background

2.2.1 Normal Distributions Transform (NDT)

The NDT algorithm begins by subdividing the space occupied by a scan into discrete grid cells—squares in 2D or cubes in 3D. A PDF is calculated for each cell based on the distribution of points within that cell. Each PDF describes how points within the cell are likely generated, assuming the points result from a multivariate normal random process. The probability density function for a cell is defined as:

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{D/2}|\mathbf{\Sigma}|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}-\mu)^T\mathbf{\Sigma}^{-1}(\mathbf{x}-\mu)\right) \tag{2.1}$$

where μ and Σ denote the mean vector and covariance matrix of the reference of the scan surface points within the cell where x lies. The mean and the covariance are computed as

$$\mu = \frac{1}{m}\sum_{k=1}^{m}\mathbf{y}_k \tag{2.2}$$

$$\mathbf{\Sigma} = \frac{1}{m-1}\sum_{k=1}^{m}(\mathbf{y}_k - \mu)(\mathbf{y}_k - \mu)^T \tag{2.3}$$

where $y_k = 1, ..., m \ldots$ are the positions of the reference scan points contained in the cell.

This probabilistic representation gives a piecewise smooth surface approximation with continuous derivatives, capturing the local surface position, orientation, and smoothness effectively.

When applying NDT for scan registration, the objective is to determine the pose of the current scan that maximizes the likelihood that scan points align with the reference map surface.The parameters to be optimised; that is, the rotation and translation of the pose estimate of the current scan, can be encoded in a vector $\vec{p}$. The current scan is represented as a point cloud $X = \{\vec{x}_1, \ldots, \vec{x}_n\}$. Assume that there is a spatial transformation function $T(\vec{p}, \vec{x})$ that moves a point $\vec{x}$ in space by the pose $\vec{p}$.Given some PDF $\vec{p}$ for scan points, the optimal pose maximizes the likelihood function:

$$\Psi = \prod_{k=1}^{n} p(T(\mathbf{p}, \mathbf{x}_k)) \tag{2.4}$$

or equivalently minimizes the negative log-likelihood of Ψ

$$-\log\Psi = -\sum_{k=1}^{n}\log p(T(\mathbf{p}, \mathbf{x}_k)) \tag{2.5}$$

The algorithm for NDT registration is shown in Algorithm 1, The primary distinction between 2D and 3D NDT algorithms resides in the spatial transformation function and its partial derivatives. In 2D, rotations are represented by a single angle, while in 3D, rotations require more complex representations (e.g., rotation matrices or quaternion representations).

Algorithm 1 Scan Registration using Normal Distributions Transform (NDT)

```
Require: Current scan point cloud $X = \{\vec{x}_1, \ldots, \vec{x}_n\}$, Reference map represented
    by PDFs in grid cells.
Ensure: Optimized pose vector $\vec{p}$ minimizing negative log-likelihood:
    $$-\log \Psi = -\sum_{k=1}^{n} \log p\left(T(\vec{p}, \vec{x}_k)\right)$$
1: Initialize pose estimate $\vec{p}$
2: repeat
3:     Compute objective function:
       $$score(\vec{p}) = -\sum_{k=1}^{n} \log p\left(T(\vec{p}, \vec{x}_k)\right)$$
4:     Compute gradient and Hessian:
       $$\vec{g}(\vec{p}) = \frac{\partial\, score(\vec{p})}{\partial\, \vec{p}}, \quad H(\vec{p}) = \frac{\partial^2\, score(\vec{p})}{\partial\, \vec{p}^2}$$
5:     Compute pose update using Newton's method:
       $$\Delta\vec{p} = -H(\vec{p})^{-1}\vec{g}(\vec{p})$$
6:     Determine step length $\alpha$ (line search):
       $$\alpha = \arg\min_{\alpha} score(\vec{p} + \alpha\Delta\vec{p})$$
7:     Update pose:
       $$\vec{p} \leftarrow \vec{p} + \alpha\Delta\vec{p}$$
8: until convergence criterion satisfied or maximum iterations reached
9: return final optimized pose $\vec{p}$
```

2.2.2 Sensor Fusion

Filtered Based Approaches

Filtering-based sensor fusion recursively estimates a system's state (e.g., a robot's position, orientation, velocity) using two main phases: a prediction step (where the state is propagated using a motion model) and an update step (where incoming sensor measurements refine that prediction) [24].This real-time, sequential estimation framework is widely employed in robotics to handle noisy sensors while maintaining tractable computational load. Nonetheless, large global corrections or loop closures can be challenging to incorporate without more advanced smoothing or factor-graph techniques[14].

1. **Extended Kalman Filter(EKF)** extends the linear Kalman Filter [25] to handle nonlinear systems by linearizing the motion and measurement models around the current estimate. Let $\mathbf{x}_k$ denote the state at time k, and $\hat{\mathbf{x}}_{k|k-1}$ be the predicted mean. The *prediction* step is

$$\hat{\mathbf{x}}_{k|k-1} = \mathbf{f}\big(\hat{\mathbf{x}}_{k-1|k-1}, \mathbf{u}_k\big), \quad \mathbf{P}_{k|k-1} = \mathbf{F}_k\,\mathbf{P}_{k-1|k-1}\,\mathbf{F}_k^T + \mathbf{Q}_k, \tag{2.6}$$

where $\mathbf{F}_k$ is the Jacobian of $\mathbf{f}$ w.r.t. the state, $\mathbf{Q}_k$ is the process noise covariance, and $\mathbf{u}_k$ is any control input. The *update* step uses a measurement model $\mathbf{h}$:

$$\mathbf{K}_k = \mathbf{P}_{k|k-1}\,\mathbf{H}_k^T\big(\mathbf{H}_k\,\mathbf{P}_{k|k-1}\,\mathbf{H}_k^T + \mathbf{R}_k\big)^{-1} \tag{2.7}$$

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k\big(\mathbf{z}_k - \mathbf{h}(\hat{\mathbf{x}}_{k|k-1})\big), \tag{2.8}$$

where $\mathbf{H}_k$ is the Jacobian of $\mathbf{h}$, $\mathbf{R}_k$ is measurement noise covariance, and $\mathbf{z}_k$ is the sensor measurement. The EKF is widely used due to its computational simplicity, although linearization can degrade accuracy in highly nonlinear settings [24].

2. **Error State Kalman Filter(ESKF)** focuses on estimating the *deviation* between a nominal trajectory and the true state [26]. A separate routine (e.g., IMU integration) generates a continuous nominal estimate, while the ESKF maintains small *error* states for position, orientation, and sensor biases:

$$\delta\mathbf{x}_k = \mathbf{x}_k - \hat{\mathbf{x}}_k^{\text{nominal}}. \tag{2.9}$$

By filtering only $\delta\mathbf{x}_k$, it improves numerical stability in rotation and bias terms, and is particularly effective for high-rate IMU fusion (e.g., in FAST-LIO [12] or LINS [11]). Major loop closures or global corrections remain difficult in a strictly forward-update framework [14].

3. **Unscented Kalman Filter(UKF)** avoids explicit Jacobian linearization by employing a deterministic sampling approach (sigma points) [27]. A set of points $\{\boldsymbol{\chi}_i\}$ is chosen around the mean and propagated through the nonlinear functions $\mathbf{f}$ and $\mathbf{h}$. The resulting transformed set describes the posterior distribution more accurately than a simple first-order expansion. While the UKF handles significant nonlinearities better than the EKF, it can be more computationally demanding [24].

Factor Graph Based Sensor Fusion

Factor graphs are a powerful framework for multi-sensor fusion in robotics, enabling the simultaneous estimation of a robot's trajectory over time. Unlike filtering approaches such as the Extended Kalman Filter (EKF), which operate recursively, factor graphs formulate state estimation as a global optimization problem, making them well-suited for handling nonlinearity, drift correction, and loop closures efficiently [14, 28].

A factor graph is an undirected bipartite graph that separates variable nodes (e.g., robot poses) from factor nodes (e.g., motion constraints or sensor measurements) [29]. As shown in Figure 2.2, green nodes represent state variables (e.g., the robot's pose at each time step), while red and blue nodes represent measurement and motion constraints, respectively. Each factor connects only to the variables it influences, forming a sparse, structured representation that is computationally efficient for optimization.

Mathematically, a factor graph models the joint posterior distribution over all states X, given observations Z, as a product of smaller, local functions:

$$P(X \mid Z) \propto \prod_{k} \phi_k(X_{\alpha(k)}), \tag{2.10}$$

where ϕ_k is a factor derived from a sensor model or prior, and $X_{\alpha(k)}$ is the subset of state variables relevant to that factor.

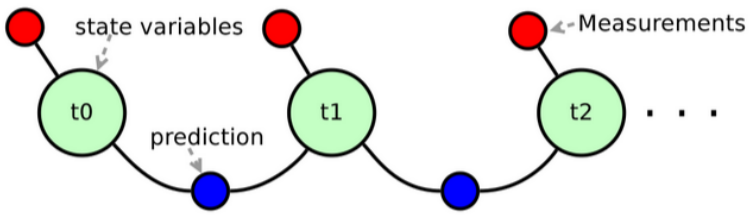

Figure 2.2: Factor graph representation: green nodes are state variables (poses), blue nodes are motion model factors, and red nodes represent sensor measurements.

In the context of localization, let $X = \{x_0, x_1, \ldots, x_n\}$ be the robot's pose trajectory and $Z = \{z_1, \ldots, z_M\}$ be the set of sensor measurements. The joint posterior can be factorized as:

$$P(X \mid Z) \propto P(x_0) \prod_{k=1}^{n} P(x_k \mid x_{k-1}) \prod_{m=1}^{M} P(z_m \mid X_{\alpha(m)}), \tag{2.11}$$

where:

- $P(x_0)$ is the prior on the initial state,
- $P(x_k \mid x_{k-1})$ is the motion model (e.g., from IMU preintegration),
- $P(z_m \mid X_{\alpha(m)})$ is the sensor measurement model (e.g., LiDAR scan matching).

Assuming Gaussian noise, the Maximum A Posteriori (MAP) estimation reduces to a nonlinear least-squares problem:

$$X^* = \arg\min_X \sum_k \|h_k(X_{\alpha(k)}) - z_k\|^2_{\Sigma_k}, \tag{2.12}$$

where $h_k(\cdot)$ is the measurement function, z_k is the observation, and Σ_k is the associated noise covariance matrix.

Incremental Solvers and Real-Time Performance

For real-time or large-scale applications, solving the entire problem from scratch at each step may be intractable [24]. Incremental smoothing algorithms like iSAM2 [30] maintain a factorization of the problem in a data structure (often a Bayes tree) that can be efficiently updated with new measurements. Rather than re-solving globally each time, iSAM2 carries out local re-linearization and partial variable reordering, providing near-constant-time updates for many practical cases.

2.3 Gaps in the literature and research questions

point cloud

Chapter 3

Methodology

3.1 System Overview

The architecture overview and the complete step-by-step algorithm flow of the proposed localization system are shown in Figure 3.1 and Algorithm **??**. The input of the system includes the raw online point-cloud, raw IMU data as well as the prior map , and output real-time accurate 6 DOF pose.

The system can be divided in to different modules:

1. Scan Pre-Processing: Performs a series of point cloud processing steps to filter and extract relevant features from the raw LiDAR data.
2. LIDAR-INERTIA Odometry - Estimates high-frequency odometry by fusing raw IMU data and LiDAR scans. The resulting odometry is added as a local constraint (odom factor) to the factor graph.
3. Dynamic Local Map Loader: Generates a local map around the robot's current pose within a specified radius from the prior map.
4. Scan-to-Map Matching: Aligns the current LiDAR scan with the local map to estimate the robot's pose in the prior map. The estimated pose is added as a prior constraint (map factor) to the factor graph.
5. Sliding Window Factor Graph Optimizer:Fuses odometry and map constraints within a sliding window to optimize and output a real-time, accurate 6-DoF pose.

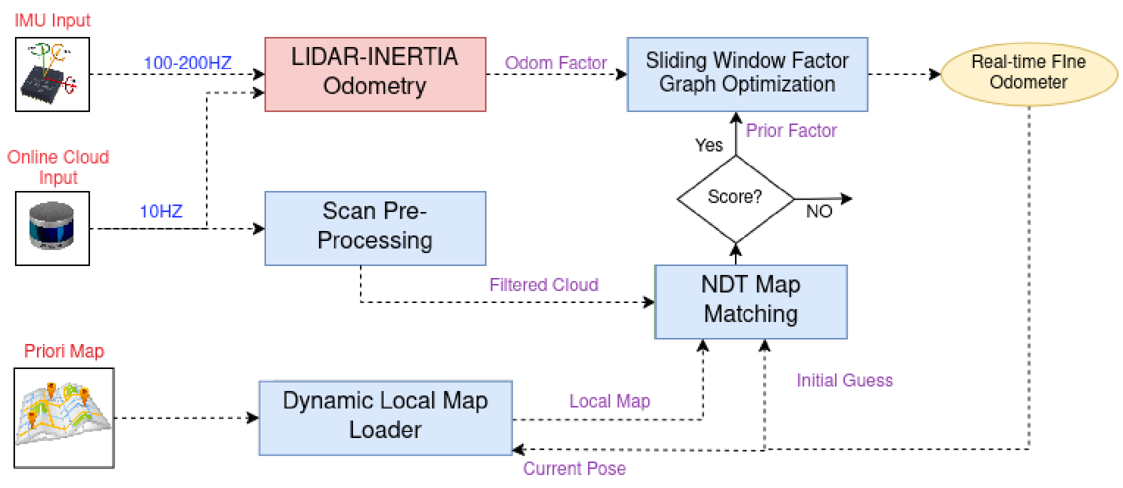

Figure 3.1: Complete Diagram of Map Based Localization

Algorithm 2 LiDAR-Inertial Localization with Prior Map

Input: $\mathcal{P}_t$ (LiDAR Point Cloud), $\mathcal{I}_t$ (IMU Data), $\mathcal{M}$ (Prior Map)
Output: $\mathbf{T}_t$ (Real-time 6-DoF Pose)

```
 1: while System is running do
 2:     Acquire LiDAR point cloud $\mathcal{P}_t$ and IMU data $\mathcal{I}_t$
 3:     Perform scan pre-processing on $\mathcal{P}_t$ to obtain filtered point cloud $\mathcal{P}_t^{filtered}$
 4:     Estimate LiDAR-Inertial Odometry using $\{\mathcal{P}_t^{filtered}, \mathcal{I}_t\}$ to obtain odometry pose $\mathbf{T}_t^{lio}$
 5:     if $||\mathbf{T}_t - \mathbf{T}_{last}^{map}|| > d_{threshold}$ then
 6:         Load dynamic local map $\mathcal{M}_t^{local}$ from $\mathcal{M}$ centered at $\mathbf{T}_t^{lio}$
 7:         Update $\mathbf{T}_{last}^{map} \leftarrow \mathbf{T}_t$
 8:     end if
 9:     Perform scan-to-map matching between $\mathcal{P}_t^{filtered}$ and $\mathcal{M}_t^{local}$ to obtain map-based pose $\mathbf{T}_t^{map}$
10:     if Matching score $> s_{threshold}$ then
11:         Add prior factor with $\mathbf{T}_t^{map}$ to factor graph
12:     end if
13:     Add odometry factor with $\mathbf{T}_t^{lio}$ to factor graph
14:     Optimize sliding window factor graph to obtain final pose $\mathbf{T}_t$
15: end while
```

3.1.1 LIDAR Scan Pre-Processing

Before aligning each incoming LiDAR scan to the global map, we first subject the raw point cloud to a series of preprocessing operations. These operations improve both efficiency and reliability by filtering out irrelevant regions and outliers, downsampling to reduce data volume, and transforming the data into a consistent coordinate frame.Our approach, illustrated in Figure3.2 consists of the following main steps:

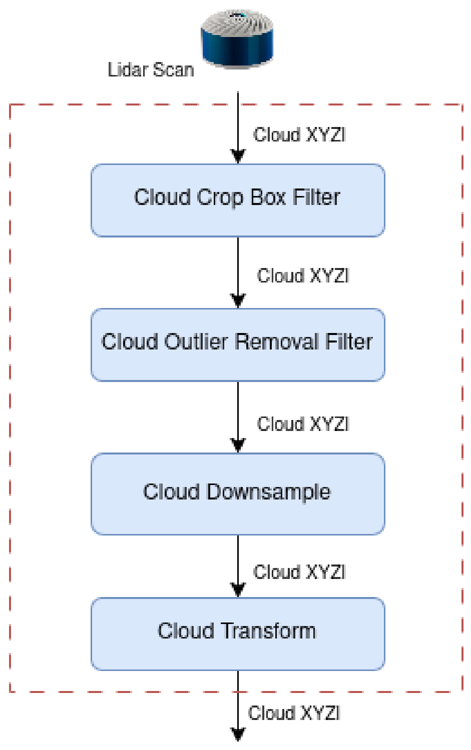

Figure 3.2: LIDAR scan pre-processing

- **Crop-Box Filter**: This step confines the point cloud to a user-defined three-dimensional region (a bounding "box") around the sensor or area of interest. By discarding all points lying outside this box, we focus only on the relevant portion of the scene, reducing both memory and computational overhead for subsequent processing.
- **Outlier Removal**: Once the point cloud is restricted to the area of interest, we remove random noise and spurious measurements using a statistical outlier filter. Concretely, we compute for each point the average distance to its k-nearest neighbors. Points whose mean distance exceeds the global mean plus a chosen multiplier of the standard deviation are marked as outliers and deleted. This practice is a commonly adopted heuristic that discards stray or floating points, which might result from occasional sensor glitches or moving objects in the environment.
- **Voxel-Grid Downsampling** To further ease computational load, the point cloud is then downsampled. A typical approach uses a voxel-grid filter: space is subdivided into small volumetric "voxels," and all points within each voxel are replaced by a single representative (often the centroid). This retains the

main geometry while reducing point density, thereby speeding up map-based matching algorithms.

- **Coordinate Transformation**: Finally, we apply a rigid transformation that aligns the processed point cloud to a consistent coordinate frame used throughout our localization pipeline—the vehicle's base link frame.This accounts for any known extrinsic offsets between the LiDAR sensor and the vehicle origin.

As summarized in Table 3.1, the LiDAR preprocessing parameters vary depending on the sensor type and application requirements. All filtering and scan processing operations are implemented using the Point Cloud Library (PCL).

Table 3.1: LiDAR Scan Processing Methods and Parametrs Across Different Sensor Setups

Processing Step	**KITTI(Velodyne HDL-64E)**	**MulRan(Ouster-OS1-64)**	**(Ouster-OS1-128)**
LiDAR Type	Velodyne HDL-64E	Ouster OS1-64	Ouster OS1-128
Max Range	∼120 m	∼120 m	∼240 m
Crop Box Filter	90-100m	90-100m	180-200m
Downsampling Method	Voxel-Grid(leaf-size: 0.5m)	Voxel-Grid(leaf-size: 0.4-0.6 m)	Voxel-Voxel(0.5-0.8 m)
Outlier Removal	Statistical (K=50, Std=1.0)	Statistical (K=40, Std=1.0)	Statistical(K=40 ,Std=1,0)

3.1.2 Map Pre-Processing

Grid-based Map Point Cloud Division

When using a high-resolution point cloud map for localization in large areas, loading the entire map can overwhelm memory and computation. To tackle this, *we divide the global map* into smaller "tiles," each containing only part of the overall data as shown in 3.3. Below is a brief outline:

- **Rationale**: Instead of loading the entire map, the localization system can dynamically fetch only the sub-map (tile) around the vehicle's estimated position, saving memory and speeding up matching.

- **Grid Definition**: A user-chosen tile size (e.g., 100 m × 100 m) defines a regular grid overlay on the map's bounding box. Each tile's data are saved as a separate file.

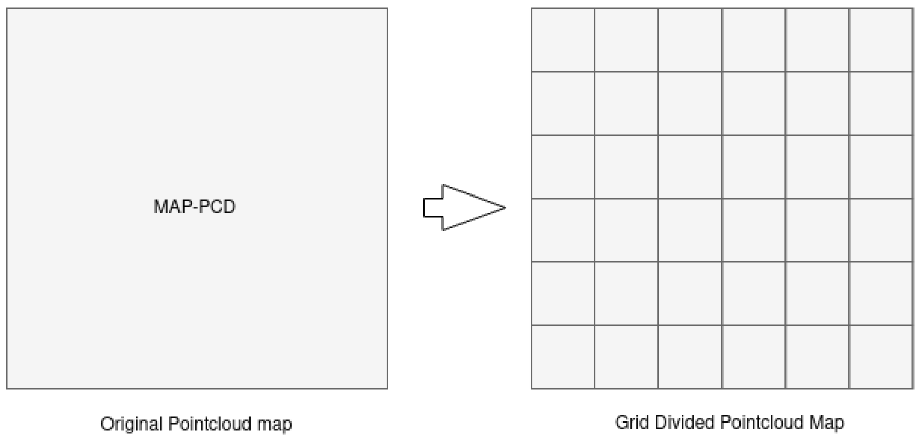

Figure 3.3: Grid Based Point-cloud division

- **Offline Preparation**: This division happens once, offline. The resulting sub-map files (e.g., `tile_0_0.pcd`, `tile_1_0.pcd`) are stored on disk.
- **Tile Indexing**: A simple coordinate scheme (e.g., (i, j) for each tile) allows quick lookup. As the vehicle moves, the system checks its approximate position and loads only the necessary tiles.

3.1.3 Dynamic Map Loading

Even with a divided map, loading all tiles at once is inefficient. Therefore, the system adopts a *dynamic loading* strategy, where only the sub-clouds near the robot's current position are fetched in real time. Concretely:

- **Robot Pose Check**: At each iteration (or after the robot travels a threshold distance), the system retrieves the robot's latest pose estimate.
- **Loading Radius**: Based on sensor range (e.g., LiDAR maximum range) and a user-defined margin, we determine a radius (e.g., 80 m) around the current pose within which map data is required.
- **Partial Tile Retrieval**: Only tiles intersecting this radius are loaded, reducing memory overhead. As the robot moves and crosses tile boundaries, newly relevant tiles are loaded and old ones are freed.
- **Update Frequency**: To avoid frequent reloads, the system triggers map updates only after the robot has moved a certain threshold (e.g., 20 m) since the last update.

This on-demand *dynamic loading* ensures that the system retains sufficient map detail for accurate localization while minimizing both I/O and memory consumption.

3.1.4 LIDAR Inertia Odometry

3.1.5 Scan-to-Map Matching

Bibliography

[1] Y. Kim, J. Jeong, and A. Kim. “Stereo camera localization in 3D LiDAR maps”. In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Madrid, Spain, Oct. 2018, pp. 1–9.

[2] H. Carvalho et al. “Optimal nonlinear filtering in GPS/INS integration”. In: *IEEE Transactions on Aerospace and Electronic Systems* 33 (1997), pp. 835–850. DOI: `10.1109/7.599911`.

[3] H. Liu et al. “A Precise and Robust Segmentation-Based Lidar Localization System for Automated Urban Driving”. In: *Remote Sensing* 11.12 (2019), p. 1348. DOI: `10.3390/rs11111348`.

[4] J. Levinson, M. Montemerlo, and S. Thrun. “Map-based precision vehicle localization in urban environments”. In: *Proceedings of Robotics: Science and Systems*. Vol. 4. Cambridge, MA, USA, 2007, pp. 1–8.

[5] A. Mohamed et al. “A Review on Visual and Non-visual Odometry Methods”. In: *Journal of Robotic Systems* (2020).

[6] X. Wang et al. “Comprehensive Survey of Visual Odometry: Trends and Evaluation”. In: *IEEE Access* (2019).

[7] Paul J. Besl and Neil D. McKay. “A Method for Registration of 3-D Shapes”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 1992.

[8] Ji Zhang and Sanjiv Singh. “LOAM: Lidar Odometry and Mapping in Real-time”. In: *Robotics: Science and Systems*. 2014.

[9] J. Tang et al. “A Loosely Coupled LiDAR-Inertial Fusion Method”. In: *Sensors* (2015).

[10] T. Shan et al. “LIO-SAM: Tightly-coupled Lidar Inertial Odometry via Smoothing and Mapping”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems* (2020).

[11] Tong Qin, Peiliang Li, and Shaojie Shen. “LINS: A LiDAR-Inertial State Estimator for Robust and Fast Localization”. In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 3867–3874.

[12] Wei Xu et al. “FAST-LIO: A Fast, Robust LiDAR-Inertial Odometry Package”. In: *IEEE Robotics and Automation Letters* 6.2 (2021), pp. 3317–3324.

[13] Wei Xu et al. “FAST-LIO2: Generalized Lightweight LIO system on ROS2”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems.* 2022, pp. 5958–5964.

[14] Cesar Cadena et al. “Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age”. In: *IEEE Transactions on Robotics* 32.6 (2016), pp. 1309–1332. DOI: 10.1109/TRO.2016.2624754.

[15] Wolfgang Hess, Stefan Kohlbrecher, et al. “Real-Time Loop Closure in 2D LIDAR SLAM”. In: *IEEE Intl. Conf. on Robotics and Automation (ICRA).* 2016, pp. 1271–1278.

[16] Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard. “Improved Techniques for Grid Mapping with Rao-Blackwellized Particle Filters”. In: *IEEE Transactions on Robotics.* 2007, pp. 34–46.

[17] Kenji Koide et al. “Tightly Coupled Range Inertial Localization on a 3D Prior Map Based on Sliding Window Factor Graph Optimization”. In: *arXiv preprint arXiv:2402.05540* (2024).

[18] Jesse Levinson, Michael Montemerlo, and Sebastian Thrun. “Map-Based Precision Vehicle Localization in Urban Environments”. In: *Proceedings of the Robotics: Science and Systems (RSS).* 2007.

[19] Xiaohu Lin et al. “Autonomous Vehicle Localization with Prior Visual Point Cloud Map Constraints in GNSS-Challenged Environments”. In: *Remote Sensing* 13.3 (2021), p. 506. DOI: 10.3390/rs13030506.

[20] Dávid Rozenberszki and András L. Majdik. “LOL: Lidar-only Odometry and Localization in 3D Point Cloud Maps”. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA).* 2020.

[21] 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).* Vol. 3. IEEE. 2003, pp. 2743–2748.

[22] Martin Magnusson et al. “Scan registration for autonomous mining vehicles using 3D-NDT”. In: *Journal of Field Robotics* 24.10 (2007), pp. 803–827.

[23] Kenji Koide, Jun Miura, and Emanuele Menegatti. “A Portable 3D LIDAR-based System for Long-term and Wide-area People Behavior Measurement”. In: *Advanced Robotic Systems* (2019).

[24] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics.* Cambridge, MA, USA: MIT Press, 2005.

[25] R. E. Kalman. “A New Approach to Linear Filtering and Prediction Problems”. In: *Journal of Basic Engineering* 82.1 (1960), pp. 35–45. DOI: 10.1115/1.3662552.

[26] Anastasios I. Mourikis and Stergios I. Roumeliotis. “A Multi-State Constraint Kalman Filter for Vision-aided Inertial Navigation”. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA).* 2007, pp. 3565–3572. DOI: 10.1109/ROBOT.2007.364024.

[27] Simon J. Julier and Jeffrey K. Uhlmann. “A New Extension of the Kalman Filter to Nonlinear Systems”. In: *Proceedings of AeroSense: The 11th International Symposium on Aerospace/Defense Sensing, Simulation and Controls.* 1997, pp. 182–193.

[28] Frank Dellaert. *Factor Graphs and GTSAM: A Hands-On Introduction.* https://doi.org/10.5281/zenodo.14117. Accessed: 2024-04-01. 2017.

[29] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger. “Factor Graphs and the Sum-Product Algorithm”. In: *IEEE Transactions on Information Theory* 47.2 (2001), pp. 498–519. DOI: 10.1109/18.910572.

[30] Michael Kaess et al. “iSAM2: Incremental Smoothing and Mapping Using the Bayes Tree”. In: *The International Journal of Robotics Research* 31.2 (2012), pp. 217–236. DOI: 10.1177/0278364911430419.

List of Figures

List of Tables

List of Algorithms

List of Codes