Technical Notes on LiDAR–Inertial SLAM

Naoki Akai[1]

October 4, 2025

[1]Nagoya University, LOCT Co., Ltd.

Preface

I have been engaged in research on autonomous navigation of mobile robots since joining my laboratory in 2011. At that time, 3D LiDAR was not yet common; when people said "LiDAR," they generally meant 2D LiDAR. For localization using 2D LiDAR, Monte Carlo Localization (MCL) [1] was the mainstream method. MCL is a particle filter-based approach and belongs to the family of probabilistic methods. Because it is relatively easy to implement and highly extensible, I have mainly focused my research on probabilistic localization methods using MCL. In particular, I have worked on proposing new frameworks centered around 2D LiDAR. If I may be a little self-promotional, one of my representative works is the following paper [2].

However, in the broader robotics community, 3D LiDAR was becoming the standard. Around 2023, I was approached by a company asking whether it would be possible to develop an LIO system using 3D LiDAR. Although I was familiar with the term LIO, I had not been closely following its research. To start, I read the well-known FAST LIO paper [3]. My first impression after reading it was simply, "I don't understand." The paper assumed knowledge of Lie groups and Lie algebras as if it were common sense, and for someone without that background, it was incomprehensible. However, I also realized that what FAST LIO achieved was remarkable, so I decided to work through it. After about six months of study, I managed to understand the minimum essentials and became able to implement LIO.

Still, implementing LIO alone did not result in clean maps. From there, I also implemented graph-based SLAM using Lie groups. After more than a year, I felt I had finally acquired at least the foundational understanding of LIO and SLAM with Lie groups. Based on this experience, I decided to reorganize and reimplement the functionality of LIO and SLAM, which led to the development of *plain_slam_ros2*[1]. SLAM software–especially LIO and related systems–tends to become very large and complex, making it difficult for newcomers to explore the code. In contrast, *plain_slam_ros2* was designed to be relatively compact and well-organized. After creating *plain_slam_ros2*, I decided to compile all the knowledge I had used into a single reference, in the hope that it would serve as a resource for those studying LiDAR SLAM in the future. That was the motivation behind writing this book[2].

To understand this book, the following mathematical knowledge is assumed as prerequisites:

- Linear algebra (matrix operations, rotation matrices, rigid transformations)
- Calculus (partial derivatives of multivariable functions, Jacobians)

Additionally, the following knowledge will deepen your understanding:

- Probability and statistics (Gaussian distribution, maximum likelihood estimation)
- Numerical optimization (nonlinear least squares)
- Lie groups and Lie algebras (SO(3), SE(3) and their exponential/logarithmic maps)

That said, the minimum necessary mathematical background required to follow this book is summarized in Chapter 2.

[1]`https://github.com/NaokiAkai/plain_slam_ros2`

[2]I refer to it as a "book," but please note that it has not undergone formal external review, so there may be errors.

Chapter 1

Introduction

1.1 Background

In the context of autonomous navigation for robots and self-driving vehicles, technologies for building maps and recognizing one's position on those maps–commonly referred to as **localization** and **Simultaneous Localization and Mapping** (SLAM)–are considered essential [4]. Traditionally, these technologies have relied on a framework known as **odometry**, which estimates motion. Put simply, localization and SLAM determine a robot's position on a map (in the case of SLAM, the map is built online) by aligning sensor observations with the map. By incorporating motion predictions from odometry, it is possible to anticipate how far the system has moved and thus restrict the search space during the map-observation matching process. Consequently, using odometry can improve both the accuracy and robustness of localization and SLAM.

However, using odometry requires sensors in addition to the external perception sensors (such as LiDAR or cameras) typically employed in localization and SLAM [5]. The simplest way to build an odometry system–in terms of implementation effort–is to use an **Inertial Measurement Unit** (IMU). An IMU measures accelerations and angular velocities with respect to the sensor's origin, and integrating these values yields an estimate of motion. Unfortunately, IMU measurements contain significant errors, and straightforward integration results in extremely poor accuracy in position and orientation estimates, rendering them unsuitable for localization or SLAM in most cases. Thus, constructing an odometry system solely with an IMU is nearly impossible.

The most widely used odometry system in mobile robotics and autonomous driving relies on measuring wheel rotations with encoders and integrating the results to estimate motion. This is known as **wheel odometry**. Provided that wheel slip does not occur, wheel odometry can estimate motion very accurately over short distances. However, to employ wheel odometry, significant modifications must be made to the robot's hardware in addition to adding external sensors. For this reason, wheel odometry cannot be considered a plug-and-play solution. Moreover, since wheel odometry depends on measuring wheel rotations, it can only be applied to wheeled platforms.

As an alternative to wheel odometry, **visual odometry** was proposed [6]. Visual odometry estimates motion by tracking features extracted from images. Therefore, it can be applied to non-wheeled platforms as well. However, it is generally known that the accuracy of visual odometry is not as high as that of wheel odometry.

Similarly, various methods were explored to estimate motion using LiDAR. Among these, the representative work that popularized the term **LiDAR odometry** (LO) is LiDAR Odometry and Mapping (LOAM) [7]. LO estimates motion by sequentially registering LiDAR point clouds. Because LiDAR provides highly accurate range measurements, LO achieves high-accuracy motion estimation and thus has been widely adopted in various applications.

Nevertheless, LO also has limitations. The scan rate of LiDAR–especially 3D LiDAR–is relatively slow (typically around 10 $\sim$ 20 Hz), making it difficult to estimate fast motions, particularly those involving rotation. To overcome this problem, **LiDAR-Inertial Odometry** (LIO) was proposed, which fuses LiDAR with IMU measurements for motion estimation. IMUs can measure accelerations and angular velocities at high frequencies (typically above 100 Hz), allowing interpolation of motion between LiDAR scans. This interpolation enables the correction of LiDAR point clouds distorted by rapid motion. Furthermore, LIO estimates additional states not considered in LO, such as velocities and IMU measurement biases. As a result, integration of IMU measurements becomes more accurate, enabling higher-precision motion estimation. It is worth noting that **Visual-Inertial Odometry** (VIO), which fuses vision and IMU, was proposed slightly earlier than LIO (see, for example, [8]).

1.2 Performance and Limitation of LIO

The evolution of LIO algorithms has been remarkable, but in recent years the performance of LiDAR sensors themselves has also advanced significantly. A decade ago (when the author began research in 2011), it was common to see papers include the phrase, "Since LiDAR is too expensive, we propose a method using cameras instead." Nowadays, however, 3D LiDARs are available for around 100,000 JPY, and surprisingly, even devices in this price range are capable of measuring nearly 360 degrees up to ranges of about 100 meters. As a result, performing highly accurate motion estimation using LIO has become fairly common. Moreover, with LIO alone, it is now possible to construct sufficiently accurate point cloud maps in small-scale environments. Consequently, it has become feasible to mount compact LiDARs on aerial platforms such as drones and generate high-precision point cloud maps with relative ease.

That said, LIO is fundamentally an odometry system, meaning that it only estimates motion. No matter how accurately motion is estimated, the results inevitably contain errors (drift). Therefore, relying solely on LIO to build maps of large-scale environments remains challenging. In particular, in large-scale scenarios that involve loops (revisiting previously traversed places), map consistency cannot be maintained–the same objects will no longer be correctly mapped to the same locations. By contrast, SLAM is designed to ensure consistency even in environments containing loops. In other words, if one wishes to build highly accurate maps, the use of SLAM is indispensable.

Furthermore, LIO is merely a motion estimation system. In practical applications, simply knowing the motion is often of limited value; the greater benefit comes from knowing one's location on a map constructed with SLAM. For instance, in factory settings where one wishes to manage the positions of AGVs or forklifts, LIO alone is insufficient, and localization is required. Thus, merely having access to high-accuracy LIO does not by itself enable the proposal of new application systems. Rather, it is essential to properly understand the algorithms underlying LIO and extend them to SLAM and localization.

1.3 This Book in the Context of Existing Methods

There already exist numerous open-source implementations of LIO and SLAM. For example, well-known open-source systems for LIO include LIO-SAM [9] and FAST-LIO [10]. These methods deliver extremely high performance, and simply downloading and running them is sufficient to achieve accurate motion estimation. Moreover, LIO-SAM also incorporates SLAM functionality, making it possible to perform map building as well. As for LiDAR-based SLAM, a prominent open-source system is GLIM [11], which also achieves very high accuracy and can construct highly precise point cloud maps across a variety of environments.

However, since many of these open-source codes implement a wide range of functions, they inevitably become large in scale. For beginners trying to learn SLAM, it is often difficult to determine where to start or what to focus on, which frequently results in using the code only as a black box rather than as a learning resource. This book, along with its accompanying source code, emphasizes keeping the software structure as simple as possible. The developed source code provides functionality for LIO, SLAM, and localization, with the entire implementation contained in fewer than 2,000 lines of C++ (excluding blank lines and comments). Within this code, all the essential components are implemented, including scan matching, LiDAR-IMU fusion (both loosely coupled and tightly coupled), loop detection, and pose graph optimization (terminology will be explained in later sections). In addition, external dependencies are kept to a minimum so that LIO and SLAM can be implemented almost entirely from scratch. The main dependencies are limited to *Sophus* (built on *Eigen*) and *nanoflann* (with *YAML* used only for parameter configuration). Sophus is a library for linear algebra and Lie groups, while nanoflann is for nearest-neighbor search. Because the core parts of LIO and SLAM are implemented from scratch, the codebase is structured in a way that makes it easier for beginners to understand how optimization and other processes are actually realized.

Chapter 2

Mathematical Background

2.1 Notation

In this book, we primarily deal with real numbers. Unless otherwise specified, all numbers are assumed to be real. The notation is as follows: a scalar is denoted by $a \in \mathbb{R}$, an N-dimensional vector by $\mathbf{a} \in \mathbb{R}^N$, and an $N \times M$ matrix by $A \in \mathbb{R}^{N \times M}$. We also frequently use the identity matrix. To make the dimension explicit, we attach a subscript; for example, the N-dimensional identity matrix is denoted by I_N.

2.2 Jacobian

In this book, we primarily make use of **optimization**. Optimization refers to the process of minimizing (or maximizing) a function $f(\mathbf{x})$ defined over a variable $\mathbf{x} = (x_1, \cdots, x_N)^\top \in \mathbb{R}^N$. Such a function is often referred to as a cost function in the context of optimization. The goal is to determine the variable $\mathbf{x}$ that minimizes (or maximizes) the value of f. There exist various approaches to perform optimization, but a fundamental step is to compute the gradient of the function, which is defined in equation (2.1).

$$\frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} = \begin{pmatrix} \frac{\partial f(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial f(\mathbf{x})}{\partial x_N} \end{pmatrix}. \tag{2.1}$$

The derivative of a function with respect to its vector argument is referred to as the **Jacobian**. For a vector-valued function $\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), \ldots, f_M(\mathbf{x}))^\top \in \mathbb{R}^M$, the Jacobian can also be defined, as expressed in equation (2.2).

$$\frac{\partial \mathbf{f}\left(\mathbf{x}\right)}{\partial \mathbf{x}} = \begin{pmatrix} \frac{\partial f_1(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial f_1(\mathbf{x})}{\partial x_N} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_M(\mathbf{x})}{\partial x_1} & \cdots & \frac{\partial f_M(\mathbf{x})}{\partial x_N} \end{pmatrix}. \tag{2.2}$$

In this book, the Jacobian is denoted by J, unless otherwise specified.

The computation of the Jacobian is generally carried out according to the definitions given in equations (2.1) and (2.2). However, it can also be derived in a slightly different manner, and in this book we primarily adopt this approach. Specifically, we approximate the change in the function $f(\mathbf{x})$ resulting from a perturbation $\delta\mathbf{x}$ using a Taylor expansion.

$$f\left(\mathbf{x} + \delta\mathbf{x}\right) \simeq f\left(\mathbf{x}\right) + J\delta\mathbf{x} + \frac{1}{2}\delta\mathbf{x}^\top H \delta\mathbf{x}, \tag{2.3}$$

where $H = \partial^2 f(\mathbf{x})/\partial \mathbf{x}^2$, which is referred to as the **Hessian**. By neglecting the second-order infinitesimal terms and assuming that both sides of equation (2.3) are equal, we obtain the following expression.

$$f\left(\mathbf{x}+\delta\mathbf{x}\right) - f\left(\mathbf{x}\right) = J\delta\mathbf{x} \tag{2.4}$$

In other words, if the difference between $f(\mathbf{x}+\delta\mathbf{x})$ and $f(\mathbf{x})$ can be expressed in the form $J\delta\mathbf{x}$, then the Jacobian can be obtained.

As a simple example, consider $f(x) = x^2$. The Jacobian[1] of this function is $\partial x^2/\partial x = 2x$. Nevertheless, let us derive the Jacobian using the method shown in equation (2.4).

$$\begin{aligned}(x+\delta x)^2 - x^2 =& x^2 + 2x\delta x + \delta x^2 - x^2, \\ =& 2x\delta x,\end{aligned} \tag{2.5}$$

where the second-order infinitesimal term was neglected by assuming $\delta x^2 \simeq 0$. As shown in equation (2.5), we correctly obtained the Jacobian of $f(x) = x^2$. Using this approach, the Jacobian can be computed without directly differentiating the function[2].

2.3 Gauss-Newton Method

The problems addressed in this book are often reduced to optimization problems. While there exist various methods for solving such problems, in this book we primarily employ the **Gauss-Newton method**.

To introduce the Gauss-Newton method, we first define the state variable $\mathbf{x} \in \mathbb{R}^N$ and the residual (or residual vector) $\mathbf{r}(\mathbf{x}) \in \mathbb{R}^M$, which depends on this state. Using multiple residual vectors, we then define the following cost function.

$$E\left(\mathbf{x}\right) = \sum_i \|\mathbf{r}_i\left(\mathbf{x}\right)\|_2^2 \in \mathbb{R}, \tag{2.6}$$

where $\|\cdot\|_2^2$ denotes the squared Euclidean norm of a vector. The state that minimizes the cost function is then defined as follows.

$$\mathbf{x}^* = \underset{\mathbf{x}}{\operatorname{argmin}}\, E\left(\mathbf{x}\right). \tag{2.7}$$

This means that we seek a state $\mathbf{x}^*$ within the domain of $\mathbf{x}$ that minimizes the cost function $E(\mathbf{x})$. Such a solution is referred to as the **optimal solution**.

To consider optimization using the Gauss-Newton method, we first approximate the residual vector by a first-order Taylor expansion.

$$\mathbf{r}\left(\mathbf{x}+\delta\mathbf{x}\right) \simeq \mathbf{r}\left(\mathbf{x}\right) + J\delta\mathbf{x}, \tag{2.8}$$

where J denotes the Jacobian of the residual vector $\mathbf{r}$ with respect to the state vector $\mathbf{x}$, i.e., $J = \partial\mathbf{r}/\partial\mathbf{x} \in \mathbb{R}^{M\times N}$. Next, we consider the squared norm of the approximated residual vector given in equation (2.8).

$$\begin{aligned}\left(\mathbf{r}+J\delta\mathbf{x}\right)^\top\left(\mathbf{r}+J\delta\mathbf{x}\right) =& \left(\mathbf{r}^\top + \delta\mathbf{x}^\top J^\top\right)\left(\mathbf{r}+J\delta\mathbf{x}\right), \\ =& \mathbf{r}^\top\mathbf{r} + \mathbf{r}^\top J\delta\mathbf{x} + \delta\mathbf{x}^\top J^\top\mathbf{r} + \delta\mathbf{x}^\top J^\top J\delta\mathbf{x}, \\ =& \mathbf{r}^\top\mathbf{r} + 2\delta\mathbf{x}^\top J^\top\mathbf{r} + \delta\mathbf{x}^\top J^\top J\delta\mathbf{x},\end{aligned} \tag{2.9}$$

[1]Strictly speaking, since the argument is a scalar, it is usually referred to as the derivative; however, in the one-dimensional case, it can formally be regarded as a Jacobian.

[2]The Jacobian obtained here is derived as a first-order approximation by applying a Taylor expansion and neglecting higher-order infinitesimal terms. In the limit as $\delta x \to 0$, it coincides with the theoretical derivative, whereas for finite differences it provides a numerical approximation.

where we use the relation $\mathbf{r}^\top J\delta\mathbf{x} = \delta\mathbf{x}^\top J^\top \mathbf{r}$.

Next, we regard equation (2.9) as a function of $\delta\mathbf{x}$, denoted by $f(\delta\mathbf{x})$, and take its derivative with respect to $\delta\mathbf{x}$.

$$\frac{\partial f(\delta\mathbf{x})}{\partial\delta\mathbf{x}} = 2J^\top\mathbf{r} + 2J^\top J\delta\mathbf{x}. \tag{2.10}$$

Then, by assuming that equation (2.10) equals $\mathbf{0}$, the following expression holds.

$$J^\top J\delta\mathbf{x} = -J^\top\mathbf{r}. \tag{2.11}$$

Here, let us consider $\delta\mathbf{x}$ that satisfies equation (2.11). The function $f(\delta\mathbf{x})$ represents an approximation of the residual vector when the state is perturbed by $\delta\mathbf{x}$, and its squared norm is computed. By differentiating this with respect to $\delta\mathbf{x}$ and setting the result equal to $\mathbf{0}$, we can obtain the value of $\delta\mathbf{x}$ that minimizes the squared norm of the approximated residual vector. In other words, updating $\mathbf{x}$ by this $\delta\mathbf{x}$ decreases the cost function.

In the derivation of equation (2.9), we considered the approximation of a single residual vector. However, the actual cost function involves the sum of squared norms of multiple residual vectors. Therefore, it is necessary to approximate the cost function when the state $\mathbf{x}$ is perturbed by $\delta\mathbf{x}$, which leads to equation (2.12).

$$E(\mathbf{x}+\delta\mathbf{x}) \simeq \sum_i \left(\mathbf{r}_i^\top\mathbf{r}_i + 2\delta\mathbf{x}^\top J_i^\top\mathbf{r}_i + \delta\mathbf{x}^\top J_i^\top J_i\delta\mathbf{x}\right). \tag{2.12}$$

Similarly, by differentiating equation (2.12) with respect to $\delta\mathbf{x}$ and setting the result to $\mathbf{0}$, we obtain the following expression.

$$\sum_i J_i^\top J_i\delta\mathbf{x} = -\sum_i J_i^\top\mathbf{r}_i. \tag{2.13}$$

Here, for simplicity, we introduce the following variables.

$$\begin{aligned} H &= \sum_i J_i^\top J_i, \\ \mathbf{b} &= \sum_i J_i^\top\mathbf{r}_i. \end{aligned} \tag{2.14}$$

With this notation, equation (2.13) can be written as $H\delta\mathbf{x} = -\mathbf{b}$. Here, H and $\mathbf{b}$ are often referred to as the Hessian and the gradient, respectively. In optimization using the Gauss-Newton method, once $\delta\mathbf{x}$ satisfying $H\delta\mathbf{x} = -\mathbf{b}$ is obtained, the state is updated as follows.

$$\mathbf{x} \leftarrow \mathbf{x} + \delta\mathbf{x}. \tag{2.15}$$

Note that from $H\delta\mathbf{x} = -\mathbf{b}$, one can naturally derive $\delta\mathbf{x} = -H^{-1}\mathbf{b}$. However, since $H \in \mathbb{R}^{N\times N}$, the direct computation of the inverse becomes intractable when N is large. For this reason, it is uncommon to compute the inverse explicitly, and we instead refer to $\delta\mathbf{x}$ as "the solution satisfying $H\delta\mathbf{x} = -\mathbf{b}$." When N is not large, computing H^{-1} directly does not pose a practical problem.

2.4 Robust Kernel

In performing optimization, multiple residual vectors are defined. However, if incorrect correspondences (i.e., mismatches) are used to construct residual vectors, they can adversely affect the optimization. Typically, the norm of residual vectors arising from mismatches tends to be larger than that of residuals from correct correspondences. Therefore, it is effective to suppress the influence on optimization according to the magnitude of the residual norm. To achieve this effect, a **robust kernel** can be introduced.

In general, a cost function with a robust kernel is expressed as follows.

$$E(\mathbf{x}) = \sum_i \rho\left(\|\mathbf{r}_i(\mathbf{x})\|_2^2\right), \tag{2.16}$$

where $\rho(\cdot)$ represents the robust kernel. Various types of robust kernels exist, but in this book we employ the **Huber loss**. The Huber loss is defined as follows.

$$\rho(s) = \begin{cases} s & \text{if } s \leq \delta^2 \\ 2\delta\sqrt{s} - \delta^2 & \text{otherwise} \end{cases}, \tag{2.17}$$

where δ is an arbitrary positive real number.

To consider the Gauss-Newton method with the Huber loss, we evaluate the Huber loss applied to the linear approximation of the residual vector, as shown in equation (2.9); that is, $\mathbf{r}(\mathbf{x}+\delta\mathbf{x})$ is approximated by $\mathbf{r}(\mathbf{x}) + J\delta\mathbf{x}$.

$$\rho\left((\mathbf{r} + J\delta\mathbf{x})^\top (\mathbf{r} + J\delta\mathbf{x})\right) = \rho\left(\mathbf{r}^\top\mathbf{r} + 2\delta\mathbf{x}^\top J^\top \mathbf{r} + \delta\mathbf{x}^\top J^\top J\delta\mathbf{x}\right). \tag{2.18}$$

Here, let $s = \mathbf{r}^\top\mathbf{r} + 2\delta\mathbf{x}^\top J^\top\mathbf{r} + \delta\mathbf{x}^\top J^\top J\delta\mathbf{x}$. From equation (2.17), when $s \leq \delta^2$, the value of s is directly used. Therefore, we focus on the case where s exceeds δ^2, and differentiate the result of substituting the given s into the Huber loss with respect to $\delta\mathbf{x}$.

$$\begin{aligned} \frac{\partial\rho(s)}{\partial\delta\mathbf{x}} &= \frac{\partial\rho(s)}{\partial s}\frac{\partial s}{\partial\delta\mathbf{x}}, \\ &= \frac{\delta}{\sqrt{s}}\left(2J^T\mathbf{r} + 2J^\top J\delta\mathbf{x}\right). \end{aligned} \tag{2.19}$$

Then, by assuming that equation (2.19) equals $\mathbf{0}$, we obtain the following result.

$$\frac{\delta}{\sqrt{s}}J^\top J\delta\mathbf{x} = -\frac{\delta}{\sqrt{s}}J^T\mathbf{r}. \tag{2.20}$$

Compared with equation (2.11), equation (2.20) contains the factor $\delta/\sqrt{s}$ on both sides. Note that this factor can, in principle, be eliminated. However, since in practice we consider the sum of multiple residual vectors, and the value of $\delta/\sqrt{s}$ differs for each residual vector, we retain it in the formulation.

Then, as shown in equation (2.14), by defining the Hessian and the gradient while taking all residual vectors into account, we obtain the following.

$$\begin{aligned} H &= \sum_i \rho'(s_i) J_i^\top J_i, \\ \mathbf{b} &= \sum_i \rho'(s_i) J_i^\top \mathbf{r}_i, \end{aligned} \tag{2.21}$$

where $\rho'(s) = \partial\rho(s)/\partial s$, which, from equation (2.17), is given as follows.

$$\rho'(s) = \begin{cases} 1 & \text{if } s \leq \delta^2 \\ \frac{\delta}{\sqrt{s}} & \text{otherwise} \end{cases}. \tag{2.22}$$

However, since $s = \mathbf{r}^\top\mathbf{r} + 2\delta\mathbf{x}^\top J^\top\mathbf{r} + \delta\mathbf{x}^\top J^\top J\delta\mathbf{x}$, $\rho'(\cdot)$ directly depends on $\delta\mathbf{x}$. As a result, the linear structure of the Gauss-Newton method breaks down, and the update cannot be obtained in the form shown in equation (2.13). To address this, we let $s_0 = \mathbf{r}^\top\mathbf{r}$ and $\delta s = 2\delta\mathbf{x}^\top J^\top\mathbf{r} + \delta\mathbf{x}^\top J^\top J\delta\mathbf{x}$, and approximate $\rho'(s) \simeq \rho'(s_0) + \rho''(s_0)\delta s$. Then, by regarding $\rho''(s_0)\delta s$ as a higher-order infinitesimal term and neglecting it, we approximate $\rho'(s) \simeq \rho'(s_0)$. Consequently, $\rho'(s) \simeq$

$\rho'(s_0)$ no longer depends on $\delta\mathbf{x}$, thereby preserving the linear structure of the Gauss-Newton method.

Finally, we define the weight $w = \rho'(\mathbf{r}^\top \mathbf{r})$ and specify the Hessian and gradient as follows.

$$\begin{aligned} H &= \sum_i w_i J_i^\top J_i, \\ \mathbf{b} &= \sum_i w_i J_i^\top \mathbf{r}_i. \end{aligned} \tag{2.23}$$

By using these H and $\mathbf{b}$ to solve for $\delta\mathbf{x}$ satisfying $H\delta\mathbf{x} = -\mathbf{b}$, and updating the state according to equation (2.15), optimization with the Gauss-Newton method incorporating the Huber loss can be performed. Although the derivation of the state update for the Gauss-Newton method with the Huber loss is somewhat more involved, the actual implementation is straightforward: it simply requires computing the function given in equation (2.22) as a weight and applying it to the corresponding Hessian and gradient. In many cases, employing the Huber loss makes the optimization more robust.

2.5 Lie Group and Lie Algebra

In this book, we frequently make use of **Lie groups** and **Lie algebras**. While a rigorous explanation of Lie groups and Lie algebras is beyond the scope of this text, throughout this book the term Lie group will specifically refer to **rotation matrices** and **rigid transformation matrices**.

A rotation matrix is defined as follows.

$$\{R \in \mathbb{R}^{3\times 3} | R^\top R = I, \det(R) = 1\}. \tag{2.24}$$

A rotation matrix, also referred to as the Special Orthogonal group in three dimensions (SO(3)), represents rotations in three-dimensional space. A rigid transformation matrix is defined as follows.

$$\left\{ \begin{pmatrix} R & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{pmatrix} \in \mathbb{R}^{4\times 4} | R \in \mathrm{SO}(3), \mathbf{t} \in \mathbb{R}^3 \right\}. \tag{2.25}$$

A rigid transformation matrix, also referred to as the Special Euclidean group in three dimensions (SE(3)), represents a pose in three-dimensional space, consisting of both rotation and translation. In LIO and SLAM, the state is typically represented using SO(3) and SE(3). As is evident from equation (2.25), the variables that constitute T are $\mathbf{t}$ and R. Therefore, in this book we often adopt the simplified notation $T = (R \mid \mathbf{t})$.

The advantage of using Lie groups is that rotations can be handled smoothly and in a mathematically natural way, without sudden jumps or discontinuities in their representation. Although the details are omitted here, such spaces are referred to as **manifolds**. While the concept may feel somewhat abstract at first, let us consider the simple example of planar rotation, namely an angle θ in the xy-plane. The angle θ is usually defined in the range $0 \leq \theta < 2\pi$ (or $-\pi \leq \theta < \pi$). Although $\theta = 0$ and $\theta = 2\pi$ are numerically different, they represent the same rotational state. Consequently, using θ directly as a representation may give the appearance of discontinuity. By contrast, Lie groups allow changes in rotation to be represented on a "seamless space," enabling continuous treatment from start to finish. Moreover, operations such as addition and differentiation can be performed under a unified mathematical framework, ensuring consistency and smoothness.

On the other hand, representing states on Lie groups makes the treatment somewhat less intuitive. For example, suppose the robot state is represented by a vector $\mathbf{x}$ in some vector space. If the robot state changes by $\Delta\mathbf{x}$, one might intuitively expect the new state to be given by the simple addition $\mathbf{x} + \Delta\mathbf{x}$ (noting that, as mentioned earlier, special care must be taken when angles such as θ are involved due to their defined ranges). However, as shown in equations (2.24) and (2.25), SO(3) and SE(3) are subject to specific constraints, and performing such addition would violate these constraints.

For instance, let the change in pose, including both translation and rotation in three-dimensional space, be defined as $\Delta T \in \mathrm{SE}(3)$ as follows.

$$\Delta T = \begin{pmatrix} \Delta R & \Delta \mathbf{t} \\ \mathbf{0}^\top & 1 \end{pmatrix}. \tag{2.26}$$

If we consider the simple addition of T and ΔT, it can be written as follows.

$$T + \Delta T = \begin{pmatrix} R + \Delta R & \mathbf{t} + \Delta \mathbf{t} \\ \mathbf{0}^\top & 2 \end{pmatrix} \notin \mathrm{SE}(3). \tag{2.27}$$

It is clear that $T + \Delta T$ does not satisfy equation (2.25), and therefore it does not represent a rigid transformation matrix. To reflect the change while preserving the constraints of SE(3), we compute $T\Delta T$[3].

$$T\Delta T = \begin{pmatrix} R\Delta R & R\Delta \mathbf{t} + \mathbf{t} \\ \mathbf{0}^\top & 1 \end{pmatrix} \in \mathrm{SE}(3). \tag{2.28}$$

Moreover, when the difference between $T_1, T_2 \in \mathrm{SE}(3)$ is denoted as ΔT, the relation $T_1 \Delta T = T_2$ holds. Thus, the difference between T_1 and T_2 is defined as follows.

$$\begin{aligned} \Delta T &= T_1^{-1} T_2, \\ &= \begin{pmatrix} R_1^\top R_2 & R_2(\mathbf{t}_2 - \mathbf{t}_1) \\ \mathbf{0}^\top & 1 \end{pmatrix} \in \mathrm{SE}(3). \end{aligned} \tag{2.29}$$

By using equations (2.28) and (2.29), state changes in the space of SE(3) (or SO(3)) can be expressed. However, compared to addition and subtraction in vector spaces, these operations are less intuitive and are not in a form to which the Gauss-Newton method, as described in the previous sections, can be directly applied. To address this, we consider whether states represented on Lie groups can be associated with vectors. This is made possible by employing the Lie algebra, which is the vector space corresponding to a Lie group. Specifically, the Lie algebras associated with SO(3) and SE(3) are denoted by $\mathfrak{so}(3)$ and $\mathfrak{se}(3)$, respectively. Strictly speaking, $\mathfrak{so}(3)$ and $\mathfrak{se}(3)$ are sets of matrices in $\mathbb{R}^{3\times3}$ and $\mathbb{R}^{4\times4}$, but since they can be parameterized by three and six independent real numbers, respectively, they can be represented as three- and six-dimensional vectors. Furthermore, the mappings between SO(3) and $\mathfrak{so}(3)$, and between SE(3) and $\mathfrak{se}(3)$, are referred to as the **exponential map** and the **logarithm map**, respectively.

$$\begin{aligned} \exp &: \mathfrak{so}(3) \to \mathrm{SO}(3), \\ \log &: \mathrm{SO}(3) \to \mathfrak{so}(3), \end{aligned} \tag{2.30}$$

$$\begin{aligned} \exp &: \mathfrak{se}(3) \to \mathrm{SO}(3), \\ \log &: \mathrm{SO}(3) \to \mathfrak{se}(3). \end{aligned} \tag{2.31}$$

Note that the exponential and logarithm maps shown in equations (2.30) and (2.31) are distinct. However, in this book we do not explicitly specify whether SO(3) or SE(3) is used in each case; instead, the type of map is inferred from the variable provided as the argument.

By employing Lie algebras, optimization methods such as the Gauss-Newton method can be applied even when the states are represented on Lie groups. Before discussing optimization using Lie algebras, however, we first provide an explanation of the exponential and logarithm maps. The derivations of these maps are omitted, and only their computational procedures are presented.

[3]Since the result of matrix multiplication depends on whether it is applied from the left or the right, the choice of how ΔT acts is important. In this book, changes associated with motion are generally modeled using right multiplication.

2.5.1 Exponential and Logarithm Maps

For a three-dimensional vector $\boldsymbol{\phi}$, the exponential map shown in equation (2.30) is defined as follows.

$$\begin{aligned} \theta &= \|\boldsymbol{\phi}\|_2, \\ \exp\left(\boldsymbol{\phi}^{\wedge}\right) &= I_3 + \frac{\sin\theta}{\theta}\boldsymbol{\phi}^{\wedge} + \frac{1-\cos\theta}{\theta^2}\left(\boldsymbol{\phi}^{\wedge}\right)^2, \end{aligned} \tag{2.32}$$

where $(\cdot)^{\wedge}$ denotes the operation that maps a three-dimensional vector to $\mathfrak{so}(3)$ (or a six-dimensional vector to $\mathfrak{se}(3)$). In the case of a three-dimensional vector, it can also be regarded as the operation that generates a **skew-symmetric matrix**.

$$\boldsymbol{\phi}^{\wedge} = \begin{pmatrix} 0 & -\phi_z & \phi_y \\ \phi_z & 0 & -\phi_x \\ -\phi_y & \phi_x & 0 \end{pmatrix} \in \mathfrak{so}(3), \tag{2.33}$$

As a brief aside, one property of skew-symmetric matrices is that $\mathbf{a}^{\wedge}\mathbf{b} = -\mathbf{b}^{\wedge}\mathbf{a}$, where $\mathbf{a}, \mathbf{b} \in \mathbb{R}^3$.

$$\begin{aligned} \mathbf{a}^{\wedge}\mathbf{b} &= \begin{pmatrix} -a_z b_y + a_y b_z \\ a_z b_x - a_x b_z \\ -a_y b_x + a_x b_y \end{pmatrix}, \\ &= \begin{pmatrix} 0 & b_z & -b_y \\ -b_z & 0 & b_x \\ b_y & -b_x & 0 \end{pmatrix}, \begin{pmatrix} a_x \\ a_y \\ a_z \end{pmatrix} \\ &= -\mathbf{b}^{\wedge}\mathbf{a}. \end{aligned} \tag{2.34}$$

This property will be used later when computing Jacobians in the implementation.

Furthermore, the logarithm map shown in equation (2.30) is defined as follows.

$$\begin{aligned} \theta &= \arccos\left(\frac{\mathrm{tr}(R)-1}{2}\right), \\ \log(R) &= \frac{\theta}{2\sin\theta}\left(R - R^{\top}\right). \end{aligned} \tag{2.35}$$

Note that since $\log(R) = \boldsymbol{\phi}^{\wedge} \in \mathfrak{so}(3)$, we define the operation of extracting the three-dimensional vector $\boldsymbol{\phi} = (\phi_x \ \phi_y \ \phi_z)^{\top}$ from this skew-symmetric matrix as follows.

$$\boldsymbol{\phi} = \left(\boldsymbol{\phi}^{\wedge}\right)^{\vee}. \tag{2.36}$$

As will be shown later, the operation of extracting a six-dimensional vector from $\mathfrak{se}(3)$ is similarly denoted by $(\cdot)^{\vee}$.

In equations (2.32) and (2.35), the parameter θ appears in the denominator, which makes the computation unstable when $\theta \ll 1$. Therefore, in the case of $\theta \ll 1$, the computation is approximated as follows.

$$\begin{aligned} \exp\left(\boldsymbol{\phi}^{\wedge}\right) &\simeq I_3 + \boldsymbol{\phi}^{\wedge} + \frac{1}{2}\left(\boldsymbol{\phi}^{\wedge}\right)^2, \\ \log(R) &\simeq \frac{1}{2}\left(R - R^{\top}\right). \end{aligned} \tag{2.37}$$

Next, in order to consider the exponential and logarithm maps for SE(3), we define the six-dimensional vector $\boldsymbol{\xi} = \left(\mathbf{v}^{\top} \ \boldsymbol{\phi}^{\top}\right)^{\top}$. We then define the operation that maps this six-dimensional vector to $\mathfrak{se}(3)$ as follows[4].

$$\boldsymbol{\xi}^{\wedge} = \begin{pmatrix} \boldsymbol{\phi}^{\wedge} & \mathbf{v} \\ \mathbf{0}^{\top} & 0 \end{pmatrix} \in \mathfrak{se}(3), \tag{2.38}$$

where $\boldsymbol{\phi}^{\wedge}$ denotes the operation that returns the skew-symmetric matrix defined in equation (2.33). Using equation (2.38), the exponential map shown in equation (2.31) is defined as follows.

$$\exp\left(\boldsymbol{\xi}^{\wedge}\right) = \begin{pmatrix} \exp\left(\boldsymbol{\phi}^{\wedge}\right) & J_l\left(\boldsymbol{\phi}^{\wedge}\right)\mathbf{v} \\ \mathbf{0}^{\top} & 1 \end{pmatrix}, \tag{2.39}$$

where $\exp\left(\boldsymbol{\phi}^{\wedge}\right)$ corresponds to the expression given in equation (2.32), and $J_l(\cdot)$ denotes the left Jacobian, which is defined as follows.

$$J_l\left(\boldsymbol{\phi}^{\wedge}\right) = I_3 + \frac{1-\cos\theta}{\theta^2}\boldsymbol{\phi}^{\wedge} + \frac{\theta - \sin\theta}{\theta^3}\left(\boldsymbol{\phi}^{\wedge}\right)^2, \tag{2.40}$$

where $\theta = \|\boldsymbol{\phi}\|_2$.

The logarithm map shown in equation (2.31) is defined as follows.

$$\log\left(T\right) = \begin{pmatrix} \boldsymbol{\phi}^{\wedge} & J_l^{-1}\left(\boldsymbol{\phi}\right)\mathbf{t} \\ \mathbf{0}^{\top} & 0 \end{pmatrix}, \tag{2.41}$$

where $J_l^{-1}(\cdot)$ denotes the inverse of the left Jacobian, which is defined as follows.

$$J_l^{-1}\left(\boldsymbol{\phi}\right) = I_3 - \frac{1}{2}\boldsymbol{\phi}^{\wedge} + \left(\frac{1}{\theta^2} - \frac{1+\cos\theta}{2\theta\sin\theta}\right)\left(\boldsymbol{\phi}^{\wedge}\right)^2. \tag{2.42}$$

Similarly, $\theta = \|\boldsymbol{\phi}\|_2$. Note that since $\log(T) \in \mathfrak{se}(3)$ is a matrix in $\mathbb{R}^{4\times 4}$, we define the operation of extracting the six-dimensional vector $\boldsymbol{\xi}$ from it as follows.

$$\begin{aligned} \boldsymbol{\xi} &= \begin{pmatrix} J_l^{-1}\left(\boldsymbol{\phi}\right)\mathbf{t} \\ \boldsymbol{\phi} \end{pmatrix}, \\ &= \left(\log\left(T\right)\right)^{\vee}, \end{aligned} \tag{2.43}$$

where $\boldsymbol{\phi} = \left(\log(R)\right)^{\vee}$.

Note that in equations (2.40) and (2.42), the parameter θ also appears in the denominator. Therefore, when $\theta \ll 1$, it is necessary to perform approximate computations, as shown in equation (2.37), in order to avoid numerical instability.

2.5.2 State Update via Lie Algebra

As described above, the logarithm map enables mapping elements of a Lie group to elements of its Lie algebra, thereby allowing optimization to be applied in a vector space. This makes it possible, for example, to apply nonlinear optimization algorithms such as the Gauss-Newton method to Lie groups. However, since the final state must be represented as an element of the Lie group, we summarize here the procedure for state update via Lie algebra. In the following, we focus on SE(3), but the same idea can also be applied to SO(3).

For instance, suppose that by using the Gauss-Newton method or a similar approach, an update $\delta\boldsymbol{\xi} \in \mathbb{R}^6$ on the Lie algebra corresponding to the state $T \in \mathrm{SE}(3)$ is obtained. This update is then

[4]In equation (2.33), $(\cdot)^{\wedge}$ is defined as the operation that generates a skew-symmetric matrix corresponding to a three-dimensional vector. In the case of a six-dimensional vector, however, it is treated as the operation that generates an element of $\mathfrak{se}(3)$, as shown in equation (2.38).

mapped back to the Lie group via the exponential map, and the actual state update is performed as follows.

$$T \leftarrow \exp\left(\delta\boldsymbol{\xi}^{\wedge}\right) T. \tag{2.44}$$

This operation can also be regarded as adding the Lie algebra element $\delta\boldsymbol{\xi}$ to the Lie group element T, and in correspondence with equation (2.15), it is sometimes written as follows.

$$T \leftarrow T \boxplus \delta\boldsymbol{\xi}. \tag{2.45}$$

In equation (2.44), the state is updated by multiplying $\exp\left(\delta\boldsymbol{\xi}^{\wedge}\right)$ from the left. This is because, when computing the gradient in optimization, we consider the change in the residual with respect to perturbations applied from the left on the Lie group. If perturbations from the right are considered instead, $\exp\left(\delta\boldsymbol{\xi}^{\wedge}\right)$ must be multiplied from the right, and care must be taken in this case.

Assuming that $\boldsymbol{\xi} = (\log(T))^{\vee}$, the updated state vector on the Lie algebra becomes $\boldsymbol{\xi}+\delta\boldsymbol{\xi}$. If $\delta\boldsymbol{\xi}$ is regarded as an infinitesimal perturbation, the **Baker-Campbell-Hausdorff** (BCH) expansion leads to the following approximation.

$$\begin{aligned} \exp\left((\boldsymbol{\xi} + \delta\boldsymbol{\xi})^{\wedge}\right) &\simeq \exp(\delta\boldsymbol{\xi}^{\wedge}) \exp(\boldsymbol{\xi}^{\wedge}), \\ &= \exp(\delta\boldsymbol{\xi}^{\wedge}) T. \end{aligned} \tag{2.46}$$

Therefore, it can be confirmed that the state update can be performed as shown in equation (2.44).

2.5.3 Jacobian Computation Using Lie Groups

As described in Section 2.3, deriving the Jacobian is essential when applying the Gauss-Newton method. For example, suppose we have a residual vector $\mathbf{r} \in \mathbb{R}^N$ that depends on $T \in \mathrm{SE}(3)$. To obtain the Jacobian, it suffices to determine J that satisfies the following equation.

$$\mathbf{r}\left(\exp\left(\delta\boldsymbol{\xi}^{\wedge}\right) T\right) - \mathbf{r}(T) = J\delta\boldsymbol{\xi}. \tag{2.47}$$

Note that $J \in \mathbb{R}^{N\times 6}$ (or $\mathbb{R}^{N\times 3}$ in the case of $\mathbf{r}(R) \in \mathbb{R}^N, ; R \in \mathrm{SO}(3)$). Examples of defining residual vectors and deriving their Jacobians are presented in the following chapters.

In differentiating residual vectors, it is sometimes necessary to compute derivatives between elements of Lie groups. To illustrate this, let us first consider the simple case of $\partial T/\partial T$. In this case, it suffices to determine the Jacobian J that satisfies the following equation.

$$T^{-1}\left(\exp\left(\delta\boldsymbol{\xi}^{\wedge}\right) T\right) = I_4 + (J\delta\boldsymbol{\xi})^{\wedge}. \tag{2.48}$$

This corresponds to considering the difference between $\exp\left(\delta\boldsymbol{\xi}^{\wedge}\right) T$ and T, as shown in equation (2.29). In particular, since $T^{-1}T = I_4$, this amounts to analyzing infinitesimal variations around the identity element of the Lie group. Therefore, this is equivalent to considering the expression $f(\mathbf{x} + \delta\mathbf{x}) - f(\mathbf{x}) = J\delta\mathbf{x}$ in a vector space. Note, however, that $J\delta\boldsymbol{\xi}$ is a six-dimensional vector, and thus it cannot be directly added to I_4. Instead, it is mapped to the corresponding element of $\mathfrak{se}(3)$ through the $(\cdot)^{\wedge}$ operator. Then, $T^{-1}\exp\left(\delta\boldsymbol{\xi}^{\wedge}\right) T$ can be written as follows.

$$T^{-1}\exp\left(\delta\boldsymbol{\xi}^{\wedge}\right) T = \exp\left((\mathrm{Ad}_{T^{-1}}\,\delta\boldsymbol{\xi})^{\wedge}\right), \tag{2.49}$$

where $\mathrm{Ad}_T \in \mathbb{R}^{6\times 6}$ is referred to as the **adjoint** operator (its detailed definition will be given in the next subsection). Considering that $\delta\boldsymbol{\xi}$ represents an infinitesimal perturbation, we can approximate it as follows.

$$\exp\left((\mathrm{Ad}_{T^{-1}}\,\delta\boldsymbol{\xi})^{\wedge}\right) \simeq I_4 + (\mathrm{Ad}_{T^{-1}}\,\delta\boldsymbol{\xi})^{\wedge}. \tag{2.50}$$

By comparing equations (2.48) and (2.50), we find that $J = \mathrm{Ad}_{T^{-1}}$, which gives the result of $\partial T/\partial T$.

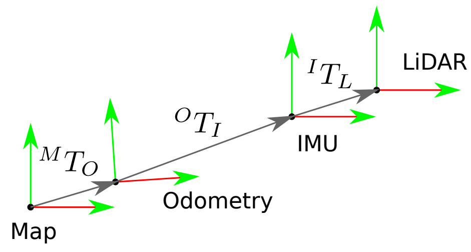

Figure 2.1: Coordinate frames employed in this work.

2.5.4 Adjoint

The adjoint operator is defined as the one that satisfies the following equation.

$$\begin{aligned}(\mathrm{Ad}_R\,\boldsymbol{\phi})^{\wedge} &= R\boldsymbol{\phi}^{\wedge}R^{\top},\\ (\mathrm{Ad}_T\,\boldsymbol{\xi})^{\wedge} &= T\boldsymbol{\xi}^{\wedge}T^{-1},\end{aligned} \tag{2.51}$$

where $\boldsymbol{\phi} \in \mathbb{R}^3$ and $\boldsymbol{\xi} \in \mathbb{R}^6$. Although the detailed derivations are omitted, the adjoint operators are defined as follows.

$$\mathrm{Ad}_R = R, \tag{2.52}$$

$$\mathrm{Ad}_T = \begin{pmatrix} R & \mathbf{t}^{\wedge}R \\ 0 & R \end{pmatrix}. \tag{2.53}$$

2.6 Coordinate Systems and Their Notation

In this book, the coordinate systems considered are primarily the four shown in Fig. 2.1: the map, odometry, IMU, and LiDAR frames. Each frame is denoted by the subscripts M, O, I, and L, respectively. For example, when a point $\mathbf{p}$ is expressed in each of these frames, we place the corresponding superscript in the upper left and denote them as $^{M}\mathbf{p}$, $^{O}\mathbf{p}$, $^{I}\mathbf{p}$, and $^{L}\mathbf{p}$, respectively. When considering a coordinate transformation (from a Source S to a Target T), it is expressed using either of the following notations.

$$\begin{aligned}{}^{T}\mathbf{p} &= {}^{T}R_{S}{}^{S}\mathbf{p} + {}^{T}\mathbf{t}_{S},\\ {}^{T}\mathbf{p} &= {}^{T}T_{S}{}^{S}\mathbf{p}.\end{aligned} \tag{2.54}$$

Note that when using SE(3), we have $\mathbf{p} \in \mathbb{R}^4$, with the fourth element set to 1.

For example, when transforming a point from the IMU frame to the odometry frame, it is expressed as $^{O}\mathbf{p} = {}^{O}R_{I}{}^{I}\mathbf{p} + {}^{O}\mathbf{t}_{I}$, or equivalently as $^{O}\mathbf{p} = {}^{O}T_{I}{}^{I}\mathbf{p}$. Similarly, when considering the direct transformation from the IMU frame to the map frame, either of the following notations can be used.

$$\begin{aligned}{}^{M}\mathbf{p} &= {}^{M}R_{O}\left({}^{O}R_{I}{}^{I}\mathbf{p} + {}^{O}\mathbf{t}_{I}\right) + {}^{M}\mathbf{t}_{O},\\ {}^{M}\mathbf{p} &= {}^{M}T_{O}{}^{O}T_{I}{}^{I}\mathbf{p}.\end{aligned} \tag{2.55}$$

Here, let $^{M}T_{I} = {}^{M}T_{O}{}^{O}T_{I}$, and denote the corresponding translation vector and rotation matrix as $^{M}\mathbf{t}_{I}$ and $^{M}R_{I}$, respectively. With this, equation (2.55) can be rewritten as a single transformation, as shown in equation (2.54).

As for the coordinate transformations illustrated in Fig. 2.1, in general, the transformation between the IMU and LiDAR, $^{I}T_{L}$, is static and thus obtained in advance through calibration.

LIO then estimates ${}^{O}T_{I}$, while SLAM (or localization) estimates ${}^{M}T_{O}$. However, ${}^{I}T_{L}$ can also be optimized online. Moreover, in some cases, one may assume that LIO directly estimates the transformation to the map frame, without relying on SLAM or localization. Therefore, the coordinate relations shown in Fig. 2.1 are not necessarily strict requirements.

Chapter 3

Scan Matching

3.1 Overview of Scan Matching

Scan matching refers to the process of finding a rigid transformation $T = (R \mid \mathbf{t}) \in \mathrm{SE}(3)$ that correctly aligns two point clouds. Suppose we consider aligning two point clouds, $\mathcal{P} = (\mathbf{p}_1, \cdots, \mathbf{p}_N)$ and $\mathcal{Q} = (\mathbf{q}_1, \cdots, \mathbf{q}_M)$, where $\mathbf{p}, \mathbf{q} \in \mathbb{R}^3$. In this case, we consider the following cost function[1].

$$E(\mathbf{t}, R) = \sum_{i=1}^{N} \| \mathbf{q}_i - (R\mathbf{p}_i + \mathbf{t}) \|_2^2 , \tag{3.1}$$

where $\mathbf{q}_i$ denotes the point in $\mathcal{Q}$ that is closest to the transformed point $R\mathbf{p}_i + \mathbf{t}$. To search for such nearest neighbors, data structures such as kd-trees are typically used. In the implementation presented in this book, we employ the library *nanoflann*[2].

Equation (3.1) can also be written as follows.

$$E(T) = \sum_{i=1}^{N} \| \mathbf{q}_i - T\mathbf{p}_i \|_2^2 . \tag{3.2}$$

When expressed in the form of equation (3.2), we have $\mathbf{p}, \mathbf{q} \in \mathbb{R}^4$, with the fourth element of each set to 1. Consequently, $T\mathbf{p}$ is also a four-dimensional vector with its fourth element equal to 1. Since the fourth element of $\mathbf{q}$ is likewise 1, the fourth component of $\mathbf{q} - T\mathbf{p}$ is always zero. As a result, the value of the cost function is identical to that given in equation (3.1). In the following, we define $\mathbf{r} = \mathbf{q} - T\mathbf{p}$ as the residual vector.

In scan matching, the goal is to estimate the rigid transformation shown below.

$$T^* = \underset{T}{\operatorname{argmin}}\, E(T). \tag{3.3}$$

Equation (3.3) indicates that the goal is to find the pose T^* that minimizes the cost function E. Since T^* is generally obtained through an iterative process, this method is referred to as **Iterative Closest Point** (ICP) scan matching. Note that scan matching that minimizes the cost defined in equation (3.2) seeks to minimize the distances between corresponding points, and is therefore called point-to-point ICP.

However, point-to-point ICP is generally considered sensitive to noise. For this reason, in this book we focus on point-to-plane ICP, which minimizes the distance between points and planes

[1] In implementation, the Huber loss shown in equation (2.17) is employed. However, to avoid unnecessary complexity in the explanation, the Huber loss is omitted in this chapter.

[2] `https://github.com/jlblancoc/nanoflann`

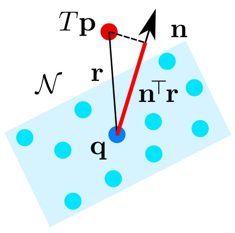

Figure 3.1: Residual error used in point-to-plane ICP.

and is known to provide greater robustness. In point-to-plane ICP, the following cost function is considered.

$$E(T) = \sum_{i=1}^{N} \left(\mathbf{n}_i^\top \mathbf{r}_i\right)^2, \tag{3.4}$$

where $\mathbf{n}_i$ denotes the normal vector in three-dimensional space, computed from the neighboring points of $\mathbf{q}_i$, and we define the residual as $r_i = \mathbf{n}_i^\top \mathbf{r}_i$. These relationships are illustrated in Fig. 3.1. Note that since $\mathbf{r}$ is a four-dimensional vector, $\mathbf{n}$ is also expressed as a four-dimensional vector. However, because the fourth component of $\mathbf{r}$ is always zero, the value of the fourth component of $\mathbf{n}$ has no effect on the computation.

In the following, we first describe the method for computing the normal vectors, then derive the Jacobian associated with the residual vector, and finally explain how this Jacobian is used to perform optimization via the Gauss-Newton method.

3.2 Computation of Normal Vectors

To compute the normal vector at a point $\mathbf{q}$, we first collect the neighboring points around $\mathbf{q}$ and denote this set as $\mathcal{N}$. Using this set, we then compute the mean $\bar{\mathbf{q}}$.

$$\bar{\mathbf{q}} = \frac{1}{|\mathcal{N}|} \sum_{\mathbf{q}_i \in \mathcal{N}} \mathbf{q}_i. \tag{3.5}$$

Next, we compute the covariance matrix C of $\mathcal{N}$.

$$C = \frac{1}{|\mathcal{N}|} \sum_{\mathbf{q}_i \in \mathcal{N}} \left(\mathbf{q}_i - \bar{\mathbf{q}}\right)\left(\mathbf{q}_i - \bar{\mathbf{q}}\right)^\top. \tag{3.6}$$

Then, we perform eigen decomposition of C.

$$C = V \Lambda V^\top, \tag{3.7}$$

where $V = (\mathbf{v}_1\ \mathbf{v}_2\ \mathbf{v}_3)$ and $\Lambda = \mathrm{diag}\left(\lambda_1, \lambda_2, \lambda_3\right)$, where $\mathbf{v}_i$ and λ_i denote the eigenvectors and their corresponding eigenvalues, respectively.

The eigenvalues represent the variance of the points along each direction. The eigenvector corresponding to the smallest eigenvalue indicates the direction with the least variance, i.e., the most compressed direction. Therefore, when the smallest eigenvalue λ_{min} is sufficiently small, the corresponding eigenvector $\mathbf{v}_{\mathrm{min}}$ can be regarded as the normal vector of the local plane on which the points lie. The vector $\mathbf{v}_{\mathrm{min}}$ obtained in this way is defined as the normal vector $\mathbf{n}$ at point $\mathbf{q}$. Moreover, by setting a threshold on λ_{min}, points with insufficient planarity can be excluded.

3.3 Computation of the Jacobian

In solving point-to-plane ICP using the Gauss-Newton method, we derive the Jacobian of the residual $r = \mathbf{n}^\top \mathbf{r}$ with respect to the pose T. This Jacobian can be computed using the chain rule as follows.

$$\frac{\partial r}{\partial T} = \frac{\partial r}{\partial \mathbf{r}} \frac{\partial \mathbf{r}}{\partial T}. \tag{3.8}$$

Since $r = \mathbf{n}^\top \mathbf{r}$, it is clear that $\partial r / \partial \mathbf{r} = \mathbf{n}^\top$. Therefore, in the following, we present the detailed computation only for $\partial \mathbf{r} / \partial T$.

The residual vector $\mathbf{r}$ is clearly a function of the pose T. Thus, the Jacobian J can be derived by considering the following expression.

$$\mathbf{r}\left(\exp\left(\delta\boldsymbol{\xi}^\wedge\right) T\right) - \mathbf{r}\left(T\right) = J \delta\boldsymbol{\xi}, \tag{3.9}$$

where $\delta\boldsymbol{\xi}^\wedge = \left(\delta\mathbf{v}^\top \ \delta\boldsymbol{\phi}^\top\right)^\wedge \in \mathfrak{se}(3)$, and $\exp\left(\delta\boldsymbol{\xi}^\wedge\right) T$ represents the state obtained by applying an infinitesimal perturbation to T from the left in the space of SE(3). Expanding the left-hand side of equation (3.9) yields the following.

$$\begin{aligned}
\mathbf{q} - \exp\left(\delta\boldsymbol{\xi}^\wedge\right) T\mathbf{p} - \left(\mathbf{q} - T\mathbf{p}\right) &= -\left(\exp\left(\delta\boldsymbol{\xi}^\wedge\right) - I_4\right) T\mathbf{p}, \\
&= -\left(\begin{pmatrix} I_3 + \delta\boldsymbol{\phi}^\wedge & \delta\mathbf{v} \\ \mathbf{0}^\top & 1 \end{pmatrix} - I_4\right) \begin{pmatrix} R\mathbf{p} + \mathbf{t} \\ 1 \end{pmatrix}, \\
&= -\begin{pmatrix} \delta\boldsymbol{\phi}^\wedge & \delta\mathbf{v} \\ \mathbf{0}^\top & 0 \end{pmatrix} \begin{pmatrix} \mathbf{p}^{'} \\ 1 \end{pmatrix}, \\
&= -\begin{pmatrix} \delta\boldsymbol{\phi}^\wedge \mathbf{p}^{'} + \delta\mathbf{v} \\ 0 \end{pmatrix}, \\
&= \begin{pmatrix} \left(\mathbf{p}^{'}\right)^\wedge \delta\boldsymbol{\phi} - \delta\mathbf{v} \\ 0 \end{pmatrix}, \\
&= \begin{pmatrix} -I_3 & \left(\mathbf{p}^{'}\right)^\wedge \\ \mathbf{0}^\top & \mathbf{0}^\top \end{pmatrix} \delta\boldsymbol{\xi},
\end{aligned} \tag{3.10}$$

where we set $\mathbf{p}^{'} = R\mathbf{p} + \mathbf{t}$ and use the relation $\delta\boldsymbol{\phi}^\wedge \mathbf{p}^{'} = -\left(\mathbf{p}^{'}\right)^\wedge \delta\boldsymbol{\phi}$. Therefore, the Jacobian with respect to the residual is given as follows.

$$\begin{aligned}
\frac{\partial r}{\partial T} =& \mathbf{n}^\top \begin{pmatrix} -I_3 & \left(\mathbf{p}^{'}\right)^\wedge \\ \mathbf{0}^\top & \mathbf{0}^\top \end{pmatrix}, \\
=& \begin{pmatrix} -\mathbf{n}^\top & \mathbf{n}^\top \left(\mathbf{p}^{'}\right)^\wedge \end{pmatrix} \in \mathbb{R}^{1 \times 6}.
\end{aligned} \tag{3.11}$$

Note that the final $\mathbf{n} \in \mathbb{R}^3$ represents the three-dimensional normal vector.

3.4 Optimization Using the Gauss-Newton Method

Using the Jacobian given in equation (3.11), the Hessian matrix and gradient vector for solving point-to-plane ICP scan matching with the Gauss-Newton method are obtained as follows.

$$\begin{aligned}
H &= \sum_{i=1}^{N} J_i^\top J_i, \\
\mathbf{b} &= \sum_{i=1}^{N} J_i^\top r_i.
\end{aligned} \tag{3.12}$$

When using the Huber loss, we define $w_i = \rho'\left(r_i^2\right)$ based on equation (2.22), and compute $H = \sum_i w_i J_i^\top J_i$ and $\mathbf{b} = \sum_i w_i J_i^\top r_i$. Then, using H and $\mathbf{b}$ as defined in equation (3.12), we solve for $\delta\boldsymbol{\xi}$ such that $H\delta\boldsymbol{\xi} = -\mathbf{b}$. Finally, the state is updated as follows.

$$T \leftarrow \exp\left(\delta\boldsymbol{\xi}^\wedge\right) T. \tag{3.13}$$

This update is repeated until either $\|\delta\boldsymbol{\xi}\|_2 \leq \epsilon$ or a predefined number of iterations is reached, at which point T^* in equation (3.3) is considered to have been obtained. Here, ϵ is an arbitrary constant.

3.5 Practical Considerations

To execute scan matching accurately, obtaining a good initial estimate is of critical importance. If the initial estimate is sufficiently accurate, correspondence search operates reliably, and scan matching converges to the optimal solution with high precision. Conversely, if the initial estimate contains large errors, correspondence search is likely to fail, causing scan matching to break down. It should also be noted that scan matching may converge to a solution that does not minimize the cost function; such solutions are referred to as **local minima**.

To mitigate the effects of incorrect correspondences, it is effective to introduce a robust kernel such as the Huber loss. By suppressing the influence of large residuals, the Huber loss can prevent optimization from being significantly degraded by outliers. However, the Huber loss is not a universal remedy. For instance, when a small number of correct correspondences are mixed with a large number of incorrect ones, the residuals of the correct correspondences may still be large. In such cases, the Huber loss reduces their weights as well, leading to an underestimation of their contribution. As a result, scan matching may succeed without the Huber loss but fail when it is applied. Therefore, the parameter δ of the Huber loss must be carefully tuned. Nevertheless, in most cases, introducing the Huber loss improves the robustness of scan matching.

It is also difficult for scan matching alone to provide a reliable initial estimate at every frame, particularly when the motion involves high translation or rotation speeds. To address this issue, it is effective to integrate external sensors such as IMUs, which can provide more accurate initial estimates. In the next chapter, we discuss a method for loosely coupling LiDAR and IMU measurements.

In addition to the initial estimate, attention must also be paid to the environment in which scan matching is performed. Since scan matching determines rigid transformations based on geometric constraints, obtaining the correct solution requires an environment that provides sufficient constraints. If appropriate constraints cannot be obtained, solving the Gauss-Newton equations $H\delta\boldsymbol{\xi} = -\mathbf{b}$ may fail to yield a valid $\delta\boldsymbol{\xi}$. This occurs because H becomes non-invertible, and its inverse cannot be defined. Such cases are referred to as **degeneracy**, in which scan matching fundamentally fails to function. A common method to detect degeneracy is to perform eigenvalue decomposition of the Hessian matrix defined in equation (3.12) and examine its smallest eigenvalue. If the smallest eigenvalue is extremely small, the Hessian matrix becomes rank-deficient, preventing computation of a valid inverse. Consequently, the optimization process breaks down.

Chapter 4

Loose-Coupled LIO

4.1 State Variables and Problem Formulation

In the scan matching discussed in the previous chapter, the problem was formulated as estimating only the pose $T \in \mathrm{SE}(3)$. In contrast, in the LIO described in this chapter, we consider estimating the following state variables.

$$\mathbf{x} = \left({}^{O}\mathbf{t}_I \ {}^{O}R_I \ {}^{O}\mathbf{v} \ \mathbf{b}^{\omega} \ \mathbf{b}^{a}\right), \tag{4.1}$$

where ${}^{O}\mathbf{t}_I \in \mathbb{R}^3$ and ${}^{O}R_I \in \mathrm{SO}(3)$ denote the translation vector and rotation matrix representing the IMU pose in the odometry frame, ${}^{O}\mathbf{v} \in \mathbb{R}^3$ denotes the velocity vector of the IMU in the odometry frame, and $\mathbf{b}^{\omega}, \mathbf{b}^{a} \in \mathbb{R}^3$ represent the biases of the IMU angular velocity and acceleration, respectively. Since $(\log(R))^{\vee} \in \mathbb{R}^3$, the LIO problem described in this chapter becomes one of estimating a 15-dimensional state. It should be noted that this involves estimating the IMU pose.

Because LIO utilizes both LiDAR and IMU, it is important to account for the temporal alignment of each sensor's data. For example, suppose the LiDAR acquires point clouds $\mathcal{P}_{t-1}$ and $\mathcal{P}_t$ at times $t-1$ and t, respectively. During this interval, the IMU measures data $\mathcal{U}_t = \left(\boldsymbol{\omega}_t^1 \ \mathbf{a}_t^1 \ \cdots \ \boldsymbol{\omega}_t^{M_t} \ \mathbf{a}_t^{M_t}\right)$[1]. The relationships among these sensor data and the timing used in this chapter are illustrated in Fig. 4.1.

In the LIO framework discussed in this chapter, the goal is to estimate the state variables defined in equation (4.1) using both LiDAR and IMU measurements. This is achieved by iteratively performing the following procedure.

1. State prediction using IMU preintegration
2. LiDAR point cloud undistortion
3. Scan matching between a local map and LiDAR point cloud
4. State update based on prediction and scan matching results
5. Keyframe detection and local map construction

It should be noted that the re-update step in item 4 corresponds to loose coupling, for which we employ the Hierarchical Geometric Observer (HGO) used in DirectLIO [12,13]. However, this does not imply that the use of HGO is necessarily the best approach for loose coupling; rather, it is introduced here because its implementation is straightforward. If feasible, it would be preferable to implement an approach such as the Extended Kalman Filter.

[1]In general, the IMU operates at a higher sampling rate than the LiDAR, so multiple IMU measurements exist between two LiDAR scans.

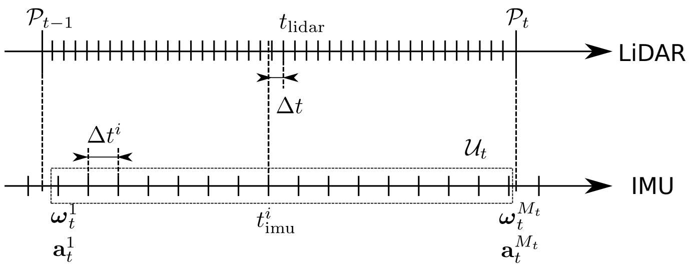

Figure 4.1: Temporal relationship between LiDAR and IMU measurements.

4.2 IMU Preintegration

Suppose the state $\mathbf{x}_{t-1}$ has been estimated up to time $t-1$, and the IMU data $\mathcal{U}_t$ at time t is available. In order to perform scan matching, we employ **IMU preintegration** to update the rotation matrix, translation vector, and velocity vector as follows.

$$\begin{aligned} {}^{O}\mathbf{t}_{I,t} &= {}^{O}\mathbf{t}_{I,t-1} + \sum_{i=1}^{M_t} {}^{O}\mathbf{v}_{t-1}^{j}\Delta t^i + \frac{1}{2}{}^{O}R_{I,t-1}^{j}\left(\mathbf{a}_t^i - \mathbf{b}_{t-1}^a\right)\left(\Delta t^i\right)^2 + \frac{1}{2}\mathbf{g}\left(\Delta t^i\right)^2, \\ {}^{O}R_{I,t} &= {}^{O}R_{I,t-1}\prod_{i=1}^{M_t}\exp\left(\left(\boldsymbol{\omega}_t^i - \mathbf{b}_{t-1}^{\omega}\right)^{\wedge}\Delta t^i\right), \\ {}^{O}\mathbf{v}_t &= {}^{O}\mathbf{v}_{t-1} + \sum_{i=1}^{M_t} {}^{O}R_{I,t-1}^{j}\left(\mathbf{a}_t^i - \mathbf{b}_{t-1}^a\right)\Delta t^i + \mathbf{g}\Delta t^i, \end{aligned} \tag{4.2}$$

where $\mathbf{g}$ denotes the gravity acceleration vector, Δt^i represents the time interval between the i-th and $(i-1)$-th IMU measurements, and ${}^{O}R_{I,t-1}^{j}$ and ${}^{O}\mathbf{v}_{I,t-1}^{j}$ are given as follows.

$$\begin{aligned} {}^{O}R_{I,t-1}^{j} &= {}^{O}R_{I,t-1}\prod_{i=1}^{j}\exp\left(\left(\boldsymbol{\omega}_t^i - \mathbf{b}_{t-1}^{\omega}\right)^{\wedge}\Delta t^i\right), \\ {}^{O}\mathbf{v}_{t-1}^{j} &= {}^{O}\mathbf{v}_{t-1} + \sum_{i=1}^{j} {}^{O}R_{I,t-1}^{j}\left(\mathbf{a}_t^i - \mathbf{b}_{t-1}^a\right)\Delta t^i + \mathbf{g}\Delta t^i. \end{aligned} \tag{4.3}$$

In the scan matching described next, the translation vector and rotation matrix given in equation (4.2) are used as the initial values.

4.3 LiDAR De-skewing

In general, the sampling rate of the IMU is higher than that of the LiDAR. As a result, multiple IMU measurements can be obtained during a single LiDAR scan. By leveraging these measurements, it is possible to compute the poses of the IMU during the LiDAR scan, as shown in equation (4.2). These poses can then be used to correct the positions of the LiDAR points, thereby compensating for motion-induced distortions in the scan. This process is referred to as LiDAR de-skewing.

Before discussing de-skewing in detail, let us denote the set of $M_t + 1$ poses obtained through IMU preintegration as $\left({}^{O}T_I^0, \cdots, {}^{O}T_I^{M_t}\right)$, and the corresponding set of timestamps as $\left(t_{\text{imu}}^0, \cdots, t_{\text{imu}}^{M_t}\right)$. Here, ${}^{O}T_I^j = \left({}^{O}R_{I,t-1}^j \mid {}^{O}\mathbf{t}_{I,t-1}^j\right) \in \mathrm{SE}(3)$, with ${}^{O}T_I^0 = {}^{O}T_{I,t-1}$ and ${}^{O}T_I^{M_t} = {}^{O}T_{I,t}$.

We assume that each LiDAR point is associated with a timestamp. Using this timestamp, we search for the IMU interval such that $t_{\text{imu}}^i \leq t_{\text{lidar}} \leq t_{\text{imu}}^{i+1}$ $(i = 0, \cdots, M_t - 1)$. Suppose that a LiDAR point ${}^{L}\mathbf{p}$ was acquired during the interval between t^{i}_{imu} and t^{i+1}_{imu}. In this case, the corresponding IMU pose at the acquisition time of the point can be computed as follows.

$$\begin{aligned}\Delta t &= t_{\mathrm{lidar}} - t^{i}_{\mathrm{imu}}. \\ {}^{O}R^{d}_{I} &= {}^{O}R^{i}_{I,t-1} \exp\left(\left(\boldsymbol{\omega}^{i}_{t} - \mathbf{b}^{\omega}_{t-1}\right)^{\wedge} \Delta t\right). \\ {}^{O}\mathbf{t}^{d}_{I} &= {}^{O}\mathbf{t}^{i}_{I,t-1} + {}^{O}\mathbf{v}^{i}_{t-1}\Delta t + \frac{1}{2}{}^{O}R^{i}_{I,t-1}\left(\mathbf{a}^{i}_{t} - \mathbf{b}^{a}_{t-1}\right)\Delta t^{2} + \frac{1}{2}\mathbf{g}\Delta t^{2}.\end{aligned} \tag{4.4}$$

Then, we define ${}^{O}T^{d}_{I} = \left({}^{O}R^{d}_{I} \mid {}^{O}\mathbf{t}^{d}_{I}\right)$, and transform ${}^{L}\mathbf{p}$ into a point expressed in the IMU coordinate frame as follows:

$${}^{I}\mathbf{p} = \left({}^{O}T^{M_t}_{I}\right)^{-1} {}^{O}T^{d}_{I} {}^{I}T_{L} {}^{L}\mathbf{p}, \tag{4.5}$$

where ${}^{I}T_{L}$ denotes the rigid transformation matrix between the LiDAR and the IMU, which is assumed to be known in advance through calibration. By applying this operation to all measured points, the motion-induced distortion in the LiDAR point cloud can be corrected. Finally, by applying $\left({}^{O}T^{M_t}_{I}\right)^{-1}$, the corrected point cloud is expressed in a coordinate frame whose origin corresponds to the predicted IMU pose obtained from equation (4.2).

4.4 Scan Matching with a Local Map

After completing the prediction by IMU preintegration and the subsequent motion distortion correction, scan matching is performed with the local map. For the sake of explanation, we first describe scan matching with the local map; the method for constructing the local map will be presented later in Section 4.6. As for scan matching itself, we generally apply the method described in Section 3, but in this subsection we clarify the data and coordinate frames involved.

Assume that the point cloud representing the local map, ${}^{O}\mathcal{M}$, has been constructed in the odometry frame[2]. Furthermore, after applying the distortion correction described in the previous subsection, we obtain the LiDAR point cloud in the IMU frame, denoted as ${}^{I}\mathcal{P}$. In this case, the residual vector for a LiDAR measurement point ${}^{I}\mathbf{p}$ is defined as follows:

$$\mathbf{r} = {}^{O}\mathbf{q} - {}^{O}T_{I} {}^{I}\mathbf{p}, \tag{4.6}$$

where ${}^{O}\mathbf{q}$ denotes the point in ${}^{O}\mathcal{M}$ that is closest to ${}^{O}T_{I}{}^{I}\mathbf{p}$. We then minimize the following cost function:

$$E\left({}^{O}T_{I}\right) = \sum_{i=1}^{N} \rho\left(\left\|\mathbf{n}_{i}^{\top}\mathbf{r}_{i}\right\|_{2}^{2}\right), \tag{4.7}$$

where $\mathbf{n}$ denotes the normal vector associated with ${}^{O}\mathbf{q}$.

4.5 State Update via Loose Coupling

Let ${}^{O}\hat{T}_{I}$ denote the pose predicted by IMU preintegration, and let ${}^{O}T^{*}_{I}$ denote the pose obtained from scan matching. The corresponding translation vectors and quaternions of the rotation matrices are written as ${}^{O}\hat{\mathbf{t}}_{I}$, ${}^{O}\mathbf{t}^{*}_{I}$, ${}^{O}\hat{\mathbf{q}}_{I}$, and ${}^{O}\mathbf{q}^{*}_{I}$, respectively. Using these, the latest state is updated

[2]There are various approaches to building a local map, and in some methods the map is not explicitly constructed in the odometry frame.

as follows:

$$
\begin{aligned}
{}^O\mathbf{q}_I &= {}^O\hat{\mathbf{q}}_I + \Delta t \gamma_1 {}^O\hat{\mathbf{q}}_I \begin{pmatrix} 1 - \left|q_w^d\right| \\ \mathrm{sgn}\left(q_w^d\right) \mathbf{q}_v^d \end{pmatrix}, \\
\mathbf{b}^\omega &= \hat{\mathbf{b}}^\omega - \Delta t \gamma_2 q_w^d \mathbf{q}_v^d, \\
{}^O\mathbf{t}_I &= {}^O\hat{\mathbf{t}}_I + \Delta t \gamma_3 \mathbf{t}^d, \\
{}^O\mathbf{v} &= {}^O\hat{\mathbf{v}}_t + \Delta t \gamma_4 \mathbf{t}^d, \\
\mathbf{b}^a &= \hat{\mathbf{b}}^a - \Delta t \gamma_5 {}^O\hat{R}_I^\top \mathbf{t}^d,
\end{aligned}
\tag{4.8}
$$

where Δt denotes the total integration time used in IMU preintegration, and γ_{1-5} are arbitrary positive constants. The relative pose is expressed as $\mathbf{q}^d = {}^O\hat{\mathbf{q}}_I^{-1} \otimes {}^O\mathbf{q}_I^*$ and $\mathbf{t}^d = {}^O\mathbf{t}_I^* - {}^O\hat{\mathbf{t}}_I$. Furthermore, $\mathbf{q}^d = \left(q_w^d \ \left(\mathbf{q}_v^d\right)^\top\right)^\top$ with $\mathbf{q}_v^d = \left(q_x^d \ q_y^d \ q_z^d\right)^\top$, and ${}^O\hat{R}_I$ is the rotation matrix corresponding to ${}^O\hat{\mathbf{q}}_I$.

4.6 Local Map Construction

There are various ways to construct a local map; in this book, we adopt a keyframe-based approach. Specifically, several poses estimated by the LIO are selected as keyframes, and the corresponding LiDAR point clouds are used together with those poses to construct the local map ${}^O\mathcal{M}$. For keyframe detection, we employ the simplest method: a threshold is set on the amount of motion, and whenever the translational displacement or rotational displacement from the previously selected keyframe exceeds this threshold, a new keyframe is selected.

Let the set of keyframes be denoted as $\left({}^OT_{I,1}, \cdots, {}^OT_{I,K}\right)$, and the corresponding LiDAR point clouds as $\left({}^I\mathcal{P}_1, \cdots, {}^I\mathcal{P}_K\right)$. Here, K represents the number of keyframes used to construct the local map. The local map is then defined as the union of all these point clouds, each transformed into the odometry frame using their corresponding keyframe poses.

$$
{}^O\mathcal{M} = \bigcup_{i=1}^{K} \bigcup_{{}^I\mathbf{p} \in {}^I\mathcal{P}_i} {}^OT_{I,i} {}^I\mathbf{p}. \tag{4.9}
$$

The number of keyframes used may vary, but in general, the local map becomes a large point cloud. As a result, computing normal vectors for all points would incur a very high computational cost. Moreover, as the local map grows, it inevitably includes many points that are not actually used in scan matching. Therefore, in our implementation, normal vectors are computed only for those points involved in the residual vector defined in equation (4.6). Additionally, a flag is implemented to indicate whether a normal has already been computed, further reducing the computational cost associated with local map construction.

4.7 Practical Considerations

Loose-coupled LIO, compared to the tight-coupled LIO described in the next chapter, is generally easier to implement and tune. Moreover, it delivers sufficient performance in most environments. Therefore, it can be considered a suitable approach for those who wish to first implement LIO and understand how to fuse LiDAR and IMU measurements. That said, tight-coupled LIO often achieves higher accuracy in practice, though at the cost of increased implementation complexity and parameter tuning difficulty. To reiterate, loose coupling itself can provide satisfactory performance, and the choice between loose and tight coupling ultimately depends on the user's specific requirements and context.

The most computationally demanding part of LIO lies in the construction of the local map. Reducing the map size lowers the computational cost, but also limits the spatial extent available for scan matching, which in turn increases the likelihood of drift in motion estimation. Conversely, if the local map is too large, the computational burden grows significantly and, in the worst case, may exceed the LiDAR's measurement cycle, causing LIO to fail.

The update frequency of the local map also has a major impact on LIO accuracy. In environments where LiDAR observations are not well suited for scan matching, increasing the update frequency of the local map helps maintain accuracy. However, more frequent updates naturally increase computational cost. In our implementation, local map updates are triggered when the estimated motion exceeds a fixed threshold. Thus, increasing the update frequency results in smaller local maps, which can again make the system more prone to drift errors. When accuracy degradation is observed, adjusting parameters related to the local map or revisiting the update policy often proves effective. It should be noted that these issues concerning local maps also arise in tight-coupled LIO, discussed in the next chapter.

Chapter 5

Tight-Coupled LIO

5.1 State Variables and Problem Formulation

In the previous chapter, the estimated state consisted of the IMU pose (translation vector ${}^{O}\mathbf{t}_I$ and rotation matrix ${}^{O}R_I$), velocity ${}^{O}\mathbf{v}$, and the measurement biases of the IMU angular velocity and acceleration, $\mathbf{b}^{\omega}$ and $\mathbf{b}^{a}$. Although LIO based on tight coupling can be implemented with the same state representation, in this chapter we extend the formulation and define the state as follows:

$$\mathbf{x} = \left({}^{O}\mathbf{t}_I \ {}^{O}R_I \ {}^{O}\mathbf{v} \ \mathbf{b}^{\omega} \ \mathbf{b}^{a} \ \mathbf{g} \ {}^{I}\mathbf{t}_L \ {}^{I}R_L \right), \tag{5.1}$$

where $\mathbf{g} \in \mathbb{R}^3$ denotes the gravity vector, while ${}^{I}\mathbf{t}_L \in \mathbb{R}^3$ and ${}^{I}R_L \in \mathrm{SO}(3)$ represent the relative translation and rotation between the LiDAR and the IMU. Since $\left(\log\left({}^{I}R_L\right)\right)^{\vee} \in \mathbb{R}^3$, the LIO formulation discussed in this chapter estimates a 24-dimensional state. In addition, unlike the LIO described in the previous chapter, we also maintain the covariance matrix of the estimated state, $\Sigma \in \mathbb{R}^{24\times 24}$.

5.2 State Prediction with IMU Preintegration

In the LIO formulation described in this chapter, we also use IMU preintegration, as explained in Section 4.2, to update the IMU pose and velocity. However, unlike the previous chapter, we additionally update the covariance matrix.

To address the covariance update, we first introduce a white noise vector $\boldsymbol{\eta} = \left(\boldsymbol{\eta}^{\omega} \ \boldsymbol{\eta}^{a} \ \boldsymbol{\eta}^{b^{\omega}} \ \boldsymbol{\eta}^{b^{a}}\right)^{\top} \in \mathbb{R}^{12}$. Here, $\boldsymbol{\eta}^{\omega}, \boldsymbol{\eta}^{a}, \boldsymbol{\eta}^{b^{\omega}}, \boldsymbol{\eta}^{b^{a}} \in \mathbb{R}^3$ represent the white noise associated with the IMU angular velocity and acceleration measurements, as well as their respective biases. Now, let $\mathbf{u}_t = \left(\boldsymbol{\omega}_t^{\top} \ \mathbf{a}_t^{\top}\right)^{\top} \in \mathbb{R}^6$ denote the IMU measurement at time t. Using these definitions, the state update based on IMU preintegration can be expressed as follows.

$$\begin{aligned}
{}^{O}\mathbf{t}_{I,t} &= {}^{O}\mathbf{t}_{I,t-1} + {}^{O}\mathbf{v}_{t-1}\Delta t + \frac{1}{2}{}^{O}R_{I,t-1}\left(\mathbf{a}_t - \mathbf{b}^{a}_{t-1} - \boldsymbol{\eta}^{a}_t\right)\Delta t^2 + \frac{1}{2}\mathbf{g}\Delta t^2, \\
{}^{O}R_{I,t} &= {}^{O}R_{I,t-1}\exp\left(\left(\boldsymbol{\omega}_t - \mathbf{b}^{\omega}_{t-1} - \boldsymbol{\eta}^{\omega}_t\right)^{\wedge}\Delta t\right), \\
{}^{O}\mathbf{v}_t &= {}^{O}\mathbf{v}_{t-1} + {}^{O}R_{I,t-1}\left(\mathbf{a}_t - \mathbf{b}^{a}_{t-1} - \boldsymbol{\eta}^{a}_t\right)\Delta t + \mathbf{g}\Delta t, \\
\mathbf{b}^{\omega}_t &= \mathbf{b}^{\omega}_{t-1} + \boldsymbol{\eta}^{b^{\omega}}_t\Delta t, \\
\mathbf{b}^{\omega}_t &= \mathbf{b}^{a}_{t-1} + \boldsymbol{\eta}^{b^{a}}_t\Delta t, \\
\mathbf{g}_t &= \mathbf{g}_{t-1}, \\
{}^{I}\mathbf{t}_{L,t} &= {}^{I}\mathbf{t}_{L,t-1}, \\
{}^{I}R_{L,t} &= {}^{I}R_{L,t-1},
\end{aligned} \tag{5.2}$$

where Δt denotes the time interval of the IMU measurement.

Let the state update given by equation (5.2) be written as $\mathbf{x}_t = \mathbf{f}\left(\mathbf{x}_{t-1}, \mathbf{u}_t, \boldsymbol{\eta}_t\right)$. In this case, the covariance matrix can be updated as follows.

$$\Sigma_t = F_x \Sigma_{t-1} F_x^\top + F_\eta Q F_\eta^\top, \tag{5.3}$$

where $F_x = \partial \mathbf{f} / \partial \mathbf{x}_{t-1} \in \mathbb{R}^{24\times 24}$ and $F_\eta = \partial \mathbf{f} / \partial \boldsymbol{\eta}_t \in \mathbb{R}^{24\times 12}$, while $Q \in \mathbb{R}^{12\times 12}$ denotes the process noise covariance matrix. Since these are large matrices, the computations become cumbersome, but they can be derived as follows[1].

$$F_x = \begin{pmatrix} I_3 & -A\Delta t^2 & I_3\Delta t & 0 & -\frac{1}{2}{}^O R_{I,t-1}\Delta t^2 & \frac{1}{2} I_3 \Delta t & 0 & 0 \\ 0 & J_l^{-1}\left({}^O\mathbf{r}_I\right){}^O R_{I,t}^\top & 0 & -J_l^{-1}\left({}^O\mathbf{r}_I\right) J_r\left(\Delta\boldsymbol{\phi}_t\right)\Delta t & 0 & 0 & 0 & 0 \\ 0 & -A\Delta t & I_3 & 0 & -{}^O R_{I,t-1}\Delta t & I_3\Delta t & 0 & 0 \\ 0 & 0 & 0 & I_3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & I_3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & I_3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & I_3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & J_l^{-1}\left({}^I\mathbf{r}_L\right) \end{pmatrix}, \tag{5.4}$$

$$F_\eta = \begin{pmatrix} 0 & -\frac{1}{2}{}^O R_{I,t-1}\Delta t^2 & 0 & 0 \\ -J_l^{-1}\left({}^O\mathbf{r}_I\right) J_r\left(\Delta\boldsymbol{\phi}_t\right)\Delta t & 0 & 0 & 0 \\ 0 & {}^O R_{I,t-1}\Delta t & 0 & 0 \\ 0 & 0 & I_3\Delta t & 0 \\ 0 & 0 & 0 & I_3\Delta t \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}, \tag{5.5}$$

where the following shorthand notations are used: $\hat{\boldsymbol{\omega}}_t = \boldsymbol{\omega}_t - \mathbf{b}_{t-1}^\omega - \boldsymbol{\eta}_t^\omega$, $\hat{\mathbf{a}}_t = \mathbf{a}_t - \mathbf{b}_{t-1}^a - \boldsymbol{\eta}_t^a$, $A = \left({}^O R_{I,t-1}\hat{\mathbf{a}}_t\right)^\wedge$, ${}^O\mathbf{r}_I = \left(\log\left({}^O R_{I,t}\right)\right)^\vee$, ${}^I\mathbf{r}_L = \left(\log\left({}^I R_{L,t}\right)\right)^\vee$, $\Delta\boldsymbol{\phi}_t = \hat{\boldsymbol{\omega}}_t\Delta t$. In addition, the zeros shown in equations (5.4) and (5.5) all represent 3×3 zero matrices in $\mathbb{R}^{3\times 3}$.

5.3 Update with IEKF

After completing the update with IMU preintegration, the LiDAR point cloud distortion correction described in Section 4.3 is applied. The LiDAR point cloud is then transformed into the IMU frame, and the residual vector and cost function are defined as in equations (4.6) and (4.7). Next, the Jacobian of the residual is derived, and the cost function is minimized. However, since the LIO described in this chapter performs optimization using the state defined in equation (5.1), the Jacobian required here differs from the one derived in Section 3.3.

The Jacobian to be computed for the LIO in this chapter is given as follows.

$$\frac{\partial r_i}{\partial \mathbf{x}} = \left(\frac{\partial r_i}{\partial {}^O\mathbf{t}_I}\ \frac{\partial r_i}{\partial {}^O R_I}\ \frac{\partial r_i}{\partial {}^O\mathbf{v}}\ \frac{\partial r_i}{\partial \mathbf{b}^\omega}\ \frac{\partial r_i}{\partial \mathbf{b}^a}\ \frac{\partial r_i}{\partial \mathbf{g}}\ \frac{\partial r_i}{\partial {}^I\mathbf{t}_L}\ \frac{\partial r_i}{\partial {}^I R_L}\right)^\top \in \mathbb{R}^{1\times 24}. \tag{5.6}$$

The detailed derivations of each Jacobian are omitted here, but their results are given below (the

[1]Although I have repeatedly computed and verified them, I cannot say with absolute confidence that they are entirely correct.

derivation of the Jacobian with respect to the angular velocity bias is presented in Section 5.5).

$$
\begin{aligned}
\frac{\partial r_i}{\partial {}^O\mathbf{t}_I} &= -\mathbf{n}_i^\top, \\
\frac{\partial r_i}{\partial {}^O R_I} &= \mathbf{n}_i^\top \left({}^O R_I {}^I\mathbf{p}_i\right)^\wedge, \\
\frac{\partial r_i}{\partial {}^O\mathbf{v}} &= -\Delta t \mathbf{n}_i^\top, \\
\frac{\partial r_i}{\partial \mathbf{b}^\omega} &= \mathbf{n}_i^\top \left({}^O R_I {}^I\mathbf{p}_i\right)^\wedge J_r\left(\Delta\boldsymbol{\phi}\right)\Delta t, \\
\frac{\partial r_i}{\partial \mathbf{b}^a} &= \frac{1}{2}\Delta t^2 \mathbf{n}_i^\top {}^O R_{I,t-1}, \\
\frac{\partial r_i}{\partial \mathbf{g}} &= -\frac{1}{2}\Delta t^2 \mathbf{n}_i^\top, \\
\frac{\partial r_i}{\partial {}^I\mathbf{t}_L} &= \mathbf{n}_i^\top {}^O R_I, \\
\frac{\partial r_i}{\partial {}^I R_L} &= \mathbf{n}_i^\top {}^O R_I \left({}^I R_L {}^L\mathbf{p}_i\right)^\wedge,
\end{aligned}
\tag{5.7}
$$

where ${}^L\mathbf{p}$ and ${}^I\mathbf{p}$ represent the same point expressed in the LiDAR and IMU frames, respectively, with ${}^I\mathbf{p} = {}^I T_L {}^L\mathbf{p}$. Also, $\Delta\boldsymbol{\phi} = \left(\boldsymbol{\omega}_t - \mathbf{b}_{t-1}^\omega\right)\Delta t$, which is recomputed whenever the state variable $\mathbf{b}_{t-1}^\omega$ is updated.

Once the residuals and Jacobians have been obtained, the update is performed using the **Iterated Extended Kalman Filter** (IEKF), which is achieved by iteratively updating the state according to the following equation.

$$
\begin{aligned}
\mathbf{r}^k &= \left(r_1^k \ \cdots \ r_N^k\right)^\top, \\
J^k &= \left(J_1^k \ \cdots \ J_N^k\right)^\top, \\
H^k &= \begin{pmatrix}
I_3 & 0_{3\times3} & 0_{3\times15} & 0_{3\times3} \\
0_{3\times3} & J_l^{-1}\left(\left(\log\left(\left({}^O R_I^1\right)^\top {}^O R_I^k\right)\right)^\vee\right) & 0_{3\times15} & 0_{3\times3} \\
0_{15\times3} & 0_{15\times3} & I_{15\times15} & 0_{3\times3} \\
0_{3\times3} & 0_{3\times3} & 0_{3\times3} & J_l^{-1}\left(\left(\log\left(\left({}^I R_L^1\right)^\top {}^I R_L^k\right)\right)^\vee\right)
\end{pmatrix}, \\
\bar{\Sigma}^k &= \left(H^k\right)^{-1}\Sigma\left(\left(H^k\right)^{-1}\right)^\top, \\
K^k &= \left(\left(J^k\right)^\top R^{-1} J^k + \left(\bar{\Sigma}^k\right)^{-1}\right)^{-1}\left(J^k\right)^\top R^{-1}, \\
\mathbf{x}^{k+1} &= \mathbf{x}^k \boxplus \left(-K^k\mathbf{r}^k - \left(I_{24} - K^k J^k\right)\left(H^k\right)^{-1}\left(\mathbf{x}^k \boxminus \mathbf{x}^1\right)\right),
\end{aligned}
\tag{5.8}
$$

where $R = \operatorname{diag}\left(\sigma^2, \cdots, \sigma^2\right) \in \mathbb{R}^{N\times N}$, with $\sigma^2 \in \mathbb{R}$ denoting the variance of the residuals. The states marked with the superscript 1 denote the initial values used for the iterative computation. When the update step in the k-th iteration falls below a predefined threshold and convergence is determined, the state and covariance matrix are updated as follows.

$$
\begin{aligned}
\mathbf{x}_t &= \mathbf{x}^k, \\
\Sigma_t &= \left(I_{24} - K^k J^k\right)\bar{\Sigma}^k.
\end{aligned}
\tag{5.9}
$$

5.4 Practical Considerations

In the loose-coupling LIO described in the previous chapter, parameters such as gravitational acceleration and the relative pose between the LiDAR and IMU were not included in the state to be estimated. In contrast, tight-coupling approaches allow these parameters to be jointly optimized. Unlike loose coupling, where estimation is performed in two separate stages, tight coupling eliminates such redundancy and enables unified estimation of all parameters.

That said, including these additional parameters in the estimation does not always lead to dramatic performance improvements. However, in many cases, tight coupling tends to yield more accurate results. It should also be noted that even without optimizing for gravity or the LiDAR-IMU extrinsics, a tight-coupled LIO can still function sufficiently well, meaning these parameters are not necessarily required to be optimized.

Although not covered in this book, recent work has proposed methods that combine LiDAR, cameras, and IMU in a joint optimization framework. In such approaches, accurately estimating the relative pose between the LiDAR and camera is of critical importance. Since these extrinsic parameters are often difficult to determine with sufficient accuracy through offline calibration alone, jointly optimizing them within the tight-coupling framework is a practical and effective approach.

Another interesting property arises in the IEKF. Equation (5.6) shows the Jacobians of the residuals, which include somewhat complex terms. For instance, the Jacobians with respect to the velocity vector and IMU measurement biases must be derived using the chain rule, taking IMU preintegration into account. While these Jacobians can be explicitly computed and used in the implementation, it is noteworthy that the system can still function as a tight-coupled LIO even if all of these Jacobians are simply set to $\mathbf{0}_3$. This is because the correction terms propagate through the Kalman gain K in equation (5.8), thereby updating even those states without explicitly derived Jacobians. Consequently, when the Jacobian computations are cumbersome, or when one wishes to avoid re-computation based on IMU preintegration, the system can still operate effectively without them.

Finally, tight-coupled LIO can also be implemented within a factor graph framework, where IMU preintegration factors are incorporated. This approach has been adopted in LIO-SAM [9] and GLIM [11], enabling not only state estimation but also map smoothing by leveraging past trajectories. Nonetheless, in practice, IEKF-based sequential implementations are often more computationally efficient (though systems such as LIO-SAM and GLIM are capable of running at sufficient real-time rates).

5.5 Derivation of Jacobians w.r.t. Rotation and Gyroscope Bias

Equations (5.4) and (5.5) show the Jacobians used for updating the covariance matrix. However, since the Jacobians with respect to rotation are complicated to derive, we provide a supplementary explanation here.

First, we derive the Jacobian of ${}^{O}\mathbf{t}_{I,t}$ with respect to ${}^{O}R_{I,t-1}$. This can be obtained by considering the perturbed ${}^{O}R_{I,t-1}$, namely $\exp\left(\delta\boldsymbol{\phi}^{\wedge}\right){}^{O}R_{I,t-1}$, and examining the difference in ${}^{O}\mathbf{t}_{I,t}$. For simplicity of notation, let ${}^{O}\mathbf{t}_{I,t}\left(\delta\boldsymbol{\phi}\right)$ denote ${}^{O}\mathbf{t}_{I,t}$ that includes $\exp\left(\delta\boldsymbol{\phi}^{\wedge}\right){}^{O}R_{I,t-1}$. By considering the difference between ${}^{O}\mathbf{t}_{I,t}\left(\delta\boldsymbol{\phi}\right)$ and ${}^{O}\mathbf{t}_{I,t}$ in equation (5.2), we obtain J such that ${}^{O}\mathbf{t}_{I,t}\left(\delta\boldsymbol{\phi}\right) - {}^{O}\mathbf{t}_{I,t} = J\delta\boldsymbol{\phi}$.

$$\begin{aligned}
{}^{O}\mathbf{t}_{I,t}\left(\delta\boldsymbol{\phi}\right) - {}^{O}\mathbf{t}_{I,t} =& \frac{1}{2}\left(\exp\left(\delta\boldsymbol{\phi}^{\wedge}\right) - I_3\right){}^{O}R_{I,t-1}\hat{\mathbf{a}}_t\Delta t^2, \\
=& \frac{1}{2}\delta\boldsymbol{\phi}^{\wedge O}R_{I,t-1}\hat{\mathbf{a}}_t\Delta t^2, \\
=& -\frac{1}{2}\left({}^{O}R_{I,t-1}\hat{\mathbf{a}}_t\right)^{\wedge}\Delta t^2\delta\boldsymbol{\phi}.
\end{aligned} \tag{5.10}$$

Thus, $J = -\frac{1}{2}\left({}^{O}R_{I,t-1}\hat{\mathbf{a}}\right)^{\wedge}\Delta t^2$. The Jacobian of ${}^{O}\mathbf{v}_t$ with respect to ${}^{O}R_{I,t-1}$ can be derived in the same manner.

Next, we consider the Jacobian of ${}^{O}R_{I,t}$. To begin, we restate the rotation update as follows.

$$R_t = R_{t-1} \exp\left(\left(\boldsymbol{\omega}_t - \mathbf{b}^{\omega}_{t-1} - \boldsymbol{\eta}^{\omega}_t\right)^{\wedge} \Delta t\right). \tag{5.11}$$

The covariance matrix with respect to R_t is defined for the corresponding rotation vector $\left(\log\left(R_t\right)\right)^{\vee}$. Therefore, it is necessary to consider the derivatives of $\left(\log\left(R_t\right)\right)^{\vee}$ with respect to R_{t-1} and $\mathbf{b}^{\omega}_{t-1}$.

First, let us consider the Jacobian with respect to R_{t-1}, which can be computed using the following chain rule.

$$\frac{\partial\left(\log\left(R_t\right)\right)^{\vee}}{\partial R_{t-1}} = \frac{\partial\left(\log\left(R_t\right)\right)^{\vee}}{\partial R_t}\frac{\partial R_t}{\partial R_{t-1}}. \tag{5.12}$$

First, to compute $\partial\left(\log\left(R_t\right)\right)^{\vee}/\partial R_t$, we consider the Jacobian that satisfies the following equation.

$$\left(\log\left(\exp\left(\delta\boldsymbol{\phi}^{\wedge}\right)R_t\right)\right)^{\vee} - \left(\log\left(R_t\right)\right)^{\vee} = J\delta\boldsymbol{\phi}. \tag{5.13}$$

By considering the BCH expansion of the first term on the left-hand side, we obtain the first-order approximation $\left(\log\left(\exp\left(\delta\boldsymbol{\phi}^{\wedge}\right)R_t\right)\right)^{\vee} \simeq \left(\log\left(R_t\right)\right)^{\vee} + J_l^{-1}\left(\left(\log\left(R_t\right)\right)^{\vee}\right)\delta\boldsymbol{\phi}$, and thus the Jacobian that satisfies the above equation is given as follows.

$$J_l^{-1}\left(\left(\log\left(R_t\right)\right)^{\vee}\right). \tag{5.14}$$

Next, to compute $\partial R_t/\partial R_{t-1}$, we consider J that satisfies the following equation.

$$\left(R_{t-1}\exp\left(\left(\hat{\boldsymbol{\omega}}_t\right)^{\wedge}\Delta t\right)\right)^{-1}\exp\left(\delta\boldsymbol{\phi}^{\wedge}\right)R_{t-1}\exp\left(\left(\hat{\boldsymbol{\omega}}_t\right)^{\wedge}\Delta t\right) = I_3 + \left(J\delta\boldsymbol{\phi}\right)^{\wedge}, \tag{5.15}$$

where $\hat{\boldsymbol{\omega}}_t = \boldsymbol{\omega}_t - \mathbf{b}^{\omega}_{t-1} - \boldsymbol{\eta}^{\omega}_t$. Expanding the left-hand side of equation (5.15) yields the following.

$$\begin{aligned} &\exp\left(-\left(\hat{\boldsymbol{\omega}}_t\right)^{\wedge}\Delta t\right)R_{t-1}^{-1}\exp\left(\delta\boldsymbol{\phi}^{\wedge}\right)R_{t-1}\exp\left(\left(\hat{\boldsymbol{\omega}}_t\right)^{\wedge}\Delta t\right) \\ =&\exp\left(\left(\mathrm{Ad}_{\exp\left(-\left(\hat{\boldsymbol{\omega}}_t\right)^{\wedge}\Delta t\right)R_{t-1}^{-1}}\delta\boldsymbol{\phi}\right)^{\wedge}\right), \\ \simeq&I_3 + \left(\mathrm{Ad}_{\exp\left(-\left(\hat{\boldsymbol{\omega}}_t\right)^{\wedge}\Delta t\right)R_{t-1}^{-1}}\delta\boldsymbol{\phi}\right)^{\wedge}. \end{aligned} \tag{5.16}$$

Thus, $J = \mathrm{Ad}_{\exp\left(-\left(\hat{\boldsymbol{\omega}}_t\right)^{\wedge}\Delta t\right)R_{t-1}^{-1}}$, which, by using equation (2.51), becomes $\exp\left(-\left(\hat{\boldsymbol{\omega}}_t\right)^{\wedge}\Delta t\right)R_{t-1}^{-1}$. Since $\exp\left(-\left(\hat{\boldsymbol{\omega}}_t\right)^{\wedge}\Delta t\right)R_{t-1}^{-1}$ is equal to R_t^{-1}, this reduces to $R_t^{\top}$.

From the above, equation (5.12) becomes the following.

$$\frac{\partial\left(\log\left(R_t\right)\right)^{\vee}}{\partial R_{t-1}} = J_l^{-1}\left(\left(\log\left(R_t\right)\right)^{\vee}\right)R_t^{\top}. \tag{5.17}$$

Next, we consider the Jacobian with respect to $\mathbf{b}^{\omega}_{t-1}$, which can also be computed using the chain rule as follows.

$$\frac{\partial\left(\log\left(R_t\right)\right)^{\vee}}{\partial \mathbf{b}^{\omega}_{t-1}} = \frac{\partial\left(\log\left(R_t\right)\right)^{\vee}}{\partial R_t}\frac{\partial R_t}{\partial \mathbf{b}^{\omega}_{t-1}}. \tag{5.18}$$

Since $\partial\left(\log\left(R_t\right)\right)^{\vee}/\partial R_t$ is given in equation (5.14), we now consider $\partial R_t/\partial \mathbf{b}^{\omega}_{t-1}$.

To compute $\partial R_t/\partial \mathbf{b}^{\omega}_{t-1}$, we consider J that satisfies the following equation.

$$\left(R_{t-1}\exp\left(\left(\hat{\boldsymbol{\omega}}_t\right)^{\wedge}\Delta t\right)\right)^{-1}R_{t-1}\exp\left(\left(\hat{\boldsymbol{\omega}}_t - \delta\mathbf{b}^{\omega}\right)^{\wedge}\Delta t\right) = I_3 + \left(J\delta\mathbf{b}^{\omega}\right)^{\wedge}. \tag{5.19}$$

The left-hand side of equation (5.19) becomes $\exp\left(\left(-\hat{\boldsymbol{\omega}}_t\right)^{\wedge}\Delta t\right)\exp\left(\left(\hat{\boldsymbol{\omega}}_t - \delta\mathbf{b}^{\omega}\right)^{\wedge}\Delta t\right)$. Using the BCH expansion, the left-hand side of equation (5.19) can be approximated to first order as follows.

$$\begin{aligned} \exp\left(\left(-\hat{\boldsymbol{\omega}}_t\right)^{\wedge}\Delta t\right)\exp\left(\left(\hat{\boldsymbol{\omega}}_t - \delta\mathbf{b}^{\omega}\right)^{\wedge}\Delta t\right) \simeq& \exp\left(-\left(J_r\left(\hat{\boldsymbol{\omega}}\Delta t\right)\delta\mathbf{b}^{\omega}\right)^{\wedge}\Delta t\right), \\ \simeq& I_3 - \left(J_r\left(\hat{\boldsymbol{\omega}}\Delta t\right)\delta\mathbf{b}^{\omega}\right)^{\wedge}\Delta t, \end{aligned} \tag{5.20}$$

where $J_r(\cdot)$ denotes the right Jacobian on SO(3), which is given as follows.

$$J_r(\boldsymbol{\phi}) = I_3 - \frac{1-\cos\theta}{\theta^2}\boldsymbol{\phi}^\wedge + \frac{\theta - \sin\theta}{\theta^3}\left(\boldsymbol{\phi}^\wedge\right)^2, \tag{5.21}$$

Therefore, the J that satisfies equation (5.19) is $-J_r(\hat{\boldsymbol{\omega}}\Delta t)\Delta t$.

From this, equation (5.18) becomes the following.

$$\frac{\partial\left(\log\left(R_t\right)\right)^\vee}{\partial \mathbf{b}^\omega_{t-1}} = -J_l^{-1}\left(\left(\log\left(R_t\right)\right)^\vee\right) J_r\left(\hat{\boldsymbol{\omega}}_t\Delta t\right)\Delta t. \tag{5.22}$$

In addition, the Jacobian of the residual r_i with respect to the gyroscope bias $\mathbf{b}^\omega$, as shown in equation (5.6), can be computed using the chain rule as follows.

$$\frac{\partial r_i}{\partial \mathbf{b}^\omega} = \frac{\partial r_i}{\partial \mathbf{r}_i}\frac{\partial \mathbf{r}_i}{\partial {}^O\mathbf{t}_I}\frac{\partial {}^O\mathbf{t}_I}{\partial {}^O R_I}\frac{\partial {}^O R_I}{\partial \mathbf{b}^\omega} + \frac{\partial r_i}{\partial \mathbf{r}_i}\frac{\partial \mathbf{r}_i}{\partial {}^O R_I}\frac{\partial {}^O R_I}{\partial \mathbf{b}^\omega}. \tag{5.23}$$

The Jacobians with respect to ${}^O R_I$ and $\mathbf{b}^\omega$ are given as follows.

$$\begin{aligned}\frac{\partial {}^O\mathbf{t}_I}{\partial {}^O R_I} &= -\frac{1}{2}\left({}^O R_I \hat{\mathbf{a}}_t\right)^\wedge \Delta t^2,\\ \frac{\partial {}^O R_I}{\partial \mathbf{b}^\omega} &= J_r\left(\Delta\boldsymbol{\phi}\right)\Delta t.\end{aligned} \tag{5.24}$$

Based on these results, equation (5.23) can be written as follows.

$$\begin{aligned}\frac{\partial r_i}{\partial \mathbf{b}^\omega} =& \mathbf{n}_i^\top\left(-I_3\right)\left(-\frac{1}{2}\left({}^O R_I\hat{\mathbf{a}}_t\right)^\wedge\Delta t^2\right)\left(J_r\left(\Delta\boldsymbol{\phi}\right)\Delta t\right) + \mathbf{n}_i^\top\left({}^O R_I {}^I\mathbf{p}_i\right)^\wedge\left(J_r\left(\Delta\boldsymbol{\phi}\right)\Delta t\right),\\ \simeq& \mathbf{n}_i^\top\left({}^O R_I {}^I\mathbf{p}_i\right)^\wedge J_r\left(\Delta\boldsymbol{\phi}\right)\Delta t,\end{aligned} \tag{5.25}$$

where the first term on the right-hand side contains Δt^3 and is therefore neglected as a higher-order infinitesimal.

Chapter 6

Graph-Based SLAM

Up to the previous chapter, we discussed LIO. While LIO can be used to estimate motion, such estimates generally include drift errors. Therefore, simply aligning LiDAR point clouds based on the motion estimated by LIO is not sufficient to build an accurate map. To obtain a precise map, this drift error must be corrected when registering the point clouds. In this book, we address this problem using **graph-based SLAM**.

6.1 Workflow of Graph-Based SLAM

There are various implementations of graph-based SLAM, but this book first outlines the assumed workflow. We begin by running two processes in parallel: odometry and mapping. The mapping process receives the IMU pose in the odometry frame, ${}^{O}T_{I} \in \mathrm{SE}(3)$, estimated by the odometry, along with the corresponding LiDAR point cloud ${}^{I}\mathcal{P}$. As for odometry, the methods described in the previous chapters are used, so in this chapter we focus on explaining the mapping process.

In the mapping process of graph-based SLAM, a graph is constructed consisting of a set of nodes $\mathcal{V}$ and a set of edges $\mathcal{E}$. Here, nodes represent sensor poses or landmark positions, while edges represent the relative poses between these nodes. However, in the implementation described in this book, nodes are used to represent only the sensor poses. Such a graph is referred to as a **pose graph**. The mapping process we present here focuses on constructing this pose graph and performing optimization of the cost function defined on the pose graph.

To build the pose graph, the mapping process receives the poses ${}^{O}T_{I}$ from the odometry process and computes the motion. When the motion exceeds a certain threshold, the corresponding pose is selected as a keyframe ${}^{M}T_{I}$[1]. When a new keyframe ${}^{M}T_{I,i}$ is detected, both the keyframe pose and its corresponding LiDAR point cloud ${}^{I}\mathcal{P}_{i}$ are stored. Additionally, based on the odometry-estimated poses, an edge (odometry edge) between two consecutive keyframes is defined as follows.

$$E_{i-1,i} = {}^{O}T_{I,i-1}^{-1}\,{}^{O}T_{I,i}. \tag{6.1}$$

After computing the odometry edge, loop detection is performed. Loop detection is the process of identifying whether the current location has already been visited in the past, and if so, recognizing the relative pose between the current pose and the previously visited pose. There are various approaches to loop detection, but the method used in this book proceeds as follows: first, from the newly detected keyframe pose, the N nearest keyframes are selected. The proximity between keyframes is measured using only the translation vector. Then, scan matching is performed between the LiDAR point cloud corresponding to the newly added keyframe and those

[1]While various strategies for keyframe detection exist, in this book we adopt a simple method where a threshold is applied to the motion, and a new keyframe is detected once the motion exceeds this threshold.

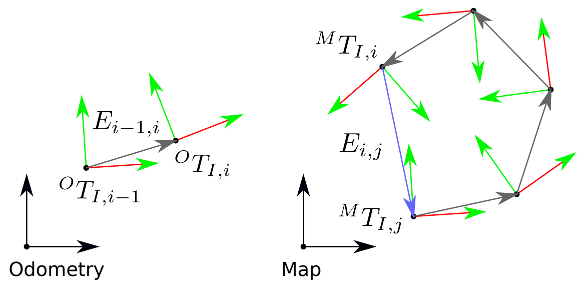

Figure 6.1: Residuals used in pose graph optimization.

corresponding to the previously detected keyframes. In loop detection, accuracy is prioritized over speed. Therefore, although it incurs higher computational cost, Generalized ICP (GICP) [14] is used for scan matching. The details of GICP are described in Section 6.3. If scan matching using GICP is determined to be successful, a loop closure is considered detected, and a loop edge is added.

Now, suppose that for the latest keyframe ${}^{M}T'_{I,i}$, the GICP result with the j-th keyframe is obtained as follows.

$$ {}^{M}T_{I,i} = \underset{{}^{M}T'_{I,i}}{\operatorname{argmin}} E_{\mathrm{GICP}}({}^{M}T'_{I,i}; {}^{M}T_{I,j}, {}^{I}\mathcal{P}_i, {}^{I}\mathcal{P}_j), \tag{6.2} $$

where $E_{\mathrm{GICP}}(\cdot)$ denotes the GICP cost function, and ${}^{M}T_{I,j}$, ${}^{I}\mathcal{P}_i$, and ${}^{I}\mathcal{P}_j$ are assumed to remain fixed. Using the pose ${}^{M}T_{I,i}$ obtained from equation (6.2), the loop edge is defined as follows.

$$ E_{i,j} = {}^{M}T_{I,i}^{-1}\,{}^{M}T_{I,j}. \tag{6.3} $$

When loop detection is performed and a loop edge is newly added to the edge set, pose graph optimization is carried out. The details of pose graph optimization are discussed in the next section. A simple illustration of odometry edges and loop edges is shown in Fig. 6.1.

6.2 Pose Graph Optimization

6.2.1 Cost Function

To analyze pose graph optimization, we begin by considering the residual vectors defined within the pose graph. As shown in equations (6.1) and (6.3), edges in the pose graph are defined as relative poses between two nodes. Assuming these edges represent observations (i.e., fixed values), the following residual vector can be defined.

$$ \mathbf{r}_{i,j}({}^{M}T_{I,i}, {}^{M}T_{I,j}) = \left(\log\left(E_{i,j}^{-1}\,{}^{M}T_{I,i}^{-1}\,{}^{M}T_{I,j}\right)\right)^{\vee}. \tag{6.4} $$

Since equation (6.4) defines the residual vector for the edge between nodes i and j, pose graph optimization is performed by finding the set of poses that minimizes the sum of these residuals.

$$ E(\mathcal{T}) = \sum_{i,j\in\mathcal{E}} \rho\left(\|\mathbf{r}_{i,j}\|^2_{\Omega_{i,j}}\right), \tag{6.5} $$

where $\|\mathbf{r}_{i,j}\|^2_{\Omega_{i,j}} = \mathbf{r}_{i,j}^{\top}\Omega_{i,j}\mathbf{r}_{i,j}$ represents the squared Mahalanobis distance using the information matrix $\Omega_{i,j}$.

6.2.2 Computation of the Jacobian

To minimize the cost function in equation (6.5), we compute the Jacobians of the residual vector in equation (6.4) with respect to ${}^{M}T_{I,i}$ and ${}^{M}T_{I,j}$. For this purpose, we introduce $\Delta_{i,j} = E_{i,j}^{-1}{}^{M}T_{I,i}^{-1}{}^{M}T_{I,j}$. Using $\Delta_{i,j}$, the Jacobians can be derived by applying the chain rule as follows.

$$\begin{aligned}\frac{\partial \mathbf{r}_{i,j}}{\partial {}^{M}T_{I,i}} &= \frac{\partial \mathbf{r}_{i,j}}{\partial \Delta_{i,j}}\frac{\partial \Delta_{i,j}}{\partial {}^{M}T_{I,i}},\\ \frac{\partial \mathbf{r}_{i,j}}{\partial {}^{M}T_{I,j}} &= \frac{\partial \mathbf{r}_{i,j}}{\partial \Delta_{i,j}}\frac{\partial \Delta_{i,j}}{\partial {}^{M}T_{I,j}}.\end{aligned} \tag{6.6}$$

In the following, we consider each derivative separately.

First, let us compute $\partial \mathbf{r}_{i,j}/\partial \Delta_{i,j}$. From equation (6.4), we need to find J that satisfies the following relation.

$$\left(\log\left(\exp\left(\delta\boldsymbol{\xi}^{\wedge}\right)\Delta_{i,j}\right)\right)^{\vee} - \left(\log\left(\Delta_{i,j}\right)\right)^{\vee} = J\delta\boldsymbol{\xi}. \tag{6.7}$$

Here, $\log\left(\exp\left(\delta\boldsymbol{\xi}^{\wedge}\right)\Delta_{i,j}\right)^{\vee}$ can be approximated to first order using the BCH expansion as follows.

$$\left(\log\left(\exp\left(\delta\boldsymbol{\xi}^{\wedge}\right)\Delta_{i,j}\right)\right)^{\vee} \simeq \mathbf{r}_{i,j} + J_l^{-1}\left(\mathbf{r}_{i,j}\right)\delta\boldsymbol{\xi}, \tag{6.8}$$

where $\mathbf{r}_{i,j} = \left(\log\left(\Delta_{i,j}\right)\right)^{\vee}$, and $J_l(\cdot)$ is referred to as the left Jacobian of SE(3), defined as follows.

$$\begin{aligned} J_l(\boldsymbol{\xi}) &= \begin{pmatrix} J_l(\boldsymbol{\phi}) & 0_{3\times 3}\\ Q(\boldsymbol{\xi}) & J_l(\boldsymbol{\phi})\end{pmatrix}\\ J_l(\boldsymbol{\phi}) &= I_3 + \frac{1-\cos\theta}{\|\theta\|^2}\boldsymbol{\phi}^{\wedge} + \frac{\theta - \sin\theta}{\|\theta\|^3}\left(\boldsymbol{\phi}^{\wedge}\right)^2\\ Q(\boldsymbol{\xi}) &= \frac{1}{2}\mathbf{v}^{\wedge} + \frac{1-\alpha}{\|\boldsymbol{\phi}\|^2}\left(\boldsymbol{\phi}^{\wedge}\mathbf{v}^{\wedge} + \mathbf{v}^{\wedge}\boldsymbol{\phi}^{\wedge}\right) + \frac{\beta - 1}{\|\boldsymbol{\phi}\|^4}\boldsymbol{\phi}^{\wedge}\mathbf{v}^{\wedge}\boldsymbol{\phi}^{\wedge}\\ \alpha &= \frac{\sin\theta}{\theta}\cdot\frac{\theta + \sin\theta}{2\left(1-\cos\theta\right)}\\ \beta &= \frac{1}{\theta^2}\left(1 - \frac{\sin\theta}{\theta}\right),\end{aligned} \tag{6.9}$$

where $\boldsymbol{\xi}^{\wedge} = \left(\begin{pmatrix}\mathbf{v}^{\top} & \boldsymbol{\phi}^{\top}\end{pmatrix}^{\top}\right)^{\wedge} \in \mathfrak{se}(3)$, $\boldsymbol{\phi}^{\wedge} \in \mathfrak{so}(3)$, and $\theta = \|\boldsymbol{\phi}\|_2{}^2$.

From equations (6.7) and (6.8), $\partial \mathbf{r}_{i,j}/\partial \Delta_{i,j}$ is given as follows.

$$\frac{\partial \mathbf{r}_{i,j}}{\partial \Delta_{i,j}} = J_l^{-1}(\mathbf{r}_{i,j}) \tag{6.10}$$

Next, to compute the derivative $\partial \Delta_{i,j}/\partial {}^{M}T_{I,i}$, we first introduce $\Delta_{i,j}\left(\delta\boldsymbol{\xi}\right) = E_{i,j}^{-1}\left(\exp\left(\delta\boldsymbol{\xi}^{\wedge}\right){}^{M}T_{I,i}\right)^{-1}{}^{M}T_{I,j}$, where $\delta\boldsymbol{\xi}^{\wedge} \in \mathfrak{se}(3)$. By multiplying $\Delta_{i,j}\left(\delta\boldsymbol{\xi}\right)$ from the left with $\Delta_{i,j}^{-1}$, we consider the variation around the identity element I_4.

$$\begin{aligned}\Delta_{i,j}^{-1}\Delta_{i,j}\left(\delta\boldsymbol{\xi}\right) &= {}^{M}T_{I,j}^{-1}{}^{M}T_{I,i}E_{i,j}E_{i,j}^{-1}{}^{M}T_{I,i}^{-1}\exp\left(-\delta\boldsymbol{\xi}^{\wedge}\right){}^{M}T_{I,j},\\ &= {}^{M}T_{I,j}^{-1}\exp\left(-\delta\boldsymbol{\xi}\right){}^{M}T_{I,j},\\ &= \exp\left(\left(-\operatorname{Ad}_{{}^{M}T_{I,j}^{-1}}\delta\boldsymbol{\xi}\right)^{\wedge}\right),\\ &\simeq I_4 + \left(-\operatorname{Ad}_{{}^{M}T_{I,j}^{-1}}\delta\boldsymbol{\xi}\right)^{\wedge}.\end{aligned} \tag{6.11}$$

[2]Equation (6.9) includes both the left Jacobians of SE(3) and SO(3). They are distinguished by whether the argument lies in $\mathfrak{se}(3)$ or $\mathfrak{so}(3)$.

By comparing with equation (2.48), it follows that $\partial \Delta_{i,j} / \partial^M T_{I,i}$ is given by:

$$\frac{\partial \Delta_{i,j}}{\partial^M T_{I,i}} = -\operatorname{Ad}_{{}^M T_{I,j}^{-1}} . \tag{6.12}$$

From equations (6.10) and (6.12), it follows that $\partial \mathbf{r}_{i,j} / \partial^M T_{I,i}$ is given by:

$$\frac{\partial \mathbf{r}_{i,j}}{\partial^M T_{I,i}} = -J_l^{-1}(\mathbf{r}_{i,j}) \operatorname{Ad}_{{}^M T_{I,j}^{-1}} . \tag{6.13}$$

Similarly, by performing the same derivation, $\partial \mathbf{r}_{i,j} / \partial^M T_{I,j}$ can be obtained as follows:

$$\frac{\partial \mathbf{r}_{i,j}}{\partial^M T_{I,j}} = J_l^{-1}(\mathbf{r}_{i,j}) \operatorname{Ad}_{{}^M T_{I,i}} . \tag{6.14}$$

6.2.3 Optimization with the Gauss-Newton Method

To minimize the cost function shown in equation (6.5), we employ the Gauss-Newton method. First, the Jacobians of the residual vector $\mathbf{r}_{i,j}$ in equation (6.4) with respect to ${}^M T_{I,i}$ and ${}^M T_{I,j}$ are given by equations (6.13) and (6.14), respectively. Let us denote these Jacobians as J_i and J_j. Now, the Jacobian of the residual vector $\mathbf{r}_{i,j}$ corresponding to the i-th and j-th edge is expressed as $J_{i,j} = (0 \cdots 0 \; J_i \; 0 \cdots 0 \; J_j \; 0 \cdots 0) \in \mathbb{R}^{6 \times 6N}$, since only the i-th and j-th poses affect the residual, and all other entries are zero. Using this, we can compute the Hessian matrix $H \in \mathbb{R}^{6N \times 6N}$ and the gradient vector $\mathbf{b} \in \mathbb{R}^{6N}$.

$$\begin{aligned} H &= \sum_{ij \in \mathcal{E}} w_{i,j} J_{i,j}^\top J_{i,j}, \\ \mathbf{b} &= \sum_{ij \in \mathcal{E}} w_{i,j} J_{i,j}^\top \mathbf{r}_{i,j}, \end{aligned} \tag{6.15}$$

where $w_{i,j} = \rho'\left(\|\mathbf{r}_{i,j}\|^2_{\Omega_{i,j}}\right)$. Using this Hessian matrix and gradient, we solve for $\delta\boldsymbol{\xi}$ such that $H\delta\boldsymbol{\xi} = -\mathbf{b}$. The vector $\delta\boldsymbol{\xi} \in \mathbb{R}^{6N}$ is composed of N blocks, each corresponding to the update for one pose. That is, if the i-th block is denoted as $\delta\boldsymbol{\xi}_i \in \mathbb{R}^6$, then the corresponding i-th pose ${}^M T_{I,i}$ is updated as follows:

$${}^M T_{I,i} \leftarrow \exp\left(\delta\boldsymbol{\xi}_i^\wedge\right) {}^M T_{I,i}. \tag{6.16}$$

When implementing graph-based SLAM, the Hessian matrix has a size of $6N \times 6N$, and directly computing its inverse would be computationally very expensive. However, in most cases, a typical graph-based SLAM problem results in H being sparse, where most of the off-diagonal elements are zero. Such a matrix is called a **sparse matrix**, and by using dedicated solvers, it is possible to efficiently solve for $\delta\boldsymbol{\xi}$ in $H\delta\boldsymbol{\xi} = -\mathbf{b}$. Therefore, in practice, instead of explicitly constructing a large Jacobian as in equation (6.15), the computation is carried out by evaluating J_i and J_j and incrementally adding their contributions to the corresponding entries of the matrix.

6.3 Loop Detection

6.3.1 Optimization of GICP

As mentioned earlier, GICP is used for loop detection. GICP is a method that can be applied to align two point clouds: the source point cloud $\mathcal{P}$ and the target point cloud $\mathcal{Q}$. In GICP, each point cloud is represented using Gaussian distributions. To achieve this, nearest-neighbor searches are first performed for all points in each point cloud to retrieve their neighboring points. Using

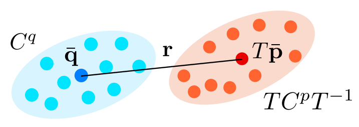

Figure 6.2: Residual error used in GICP.

these neighboring points, the mean and covariance matrices are then computed. The mean and covariance are calculated according to equations (3.5) and (3.6). Specifically, the i-th point of $\mathcal{P}$ is represented by $\bar{\mathbf{p}}_i$ and C_i^p, while the i-th point of $\mathcal{Q}$ is represented by $\bar{\mathbf{q}}_i$ and C_i^q. Based on these, the cost function in GICP is defined as follows.

$$E(T) = \sum_{i=1}^{N} \rho \left((\bar{\mathbf{q}}_i - T\bar{\mathbf{p}}_i)^\top \left(\hat{C}_i^q + T\hat{C}_i^p T^{-1} \right)^{-1} (\bar{\mathbf{q}}_i - T\bar{\mathbf{p}}_i) \right), \tag{6.17}$$

where $\hat{C}^p$ and $\hat{C}^q$ are defined as follows.

$$\begin{aligned} \hat{C}^p &= \begin{pmatrix} C^p & \mathbf{0} \\ \mathbf{0}^\top & 0 \end{pmatrix} \in \mathbb{R}^{4\times 4}, \\ \hat{C}^q &= \begin{pmatrix} C^q & \mathbf{0} \\ \mathbf{0}^\top & 0 \end{pmatrix} \in \mathbb{R}^{4\times 4}. \end{aligned} \tag{6.18}$$

Figure 6.2 illustrates the relationship of the residual used in GICP.

In this book, GICP is also solved using the Gauss-Newton method. For notational simplicity, let $\mathbf{r}_i = \bar{\mathbf{q}}_i - T\bar{\mathbf{p}}_i$ and $\Omega_i = \left(\hat{C}_i^q + T\hat{C}_i^p T^\top \right)^{-1}$. First, to compute the Jacobian of $\mathbf{r}_i$ with respect to T, we seek J_i such that the following holds.

$$\bar{\mathbf{q}}_i - \exp\left(\delta\boldsymbol{\xi}^\wedge\right) T\bar{\mathbf{p}}_i - (\bar{\mathbf{q}}_i - T\bar{\mathbf{p}}_i) = J_i \delta\boldsymbol{\xi}. \tag{6.19}$$

Expanding the left-hand side of equation (6.19) yields the following.

$$\begin{aligned} -\exp\left(\delta\boldsymbol{\xi}^\wedge\right) T\bar{\mathbf{p}}_i + T\bar{\mathbf{p}}_i &= -\left(\exp\left(\delta\boldsymbol{\xi}^\wedge\right) - I_4\right) T\bar{\mathbf{p}}_i, \\ &= -\begin{pmatrix} \delta\boldsymbol{\phi}^\wedge & \delta\mathbf{v} \\ \mathbf{0}^\top & 0 \end{pmatrix} \begin{pmatrix} \bar{\mathbf{p}}_i^{'} \\ 1 \end{pmatrix}, \\ &= -\begin{pmatrix} -\left(\bar{\mathbf{p}}_i^{'}\right)^\wedge \delta\boldsymbol{\phi} + \delta\mathbf{v} \\ 0 \end{pmatrix}, \\ &= \begin{pmatrix} -I_3 & \left(\bar{\mathbf{p}}_i^{'}\right)^\wedge \\ \mathbf{0}^\top & \mathbf{0}^\top \end{pmatrix} \delta\boldsymbol{\xi}, \end{aligned} \tag{6.20}$$

where $\bar{\mathbf{p}}_i^{'} = R\bar{\mathbf{p}}_i + \mathbf{t}$. Therefore, the J_i that satisfies equation (6.19) is given as follows.

$$J_i = \begin{pmatrix} -I_3 & \left(\bar{\mathbf{p}}_i^{'}\right)^\wedge \\ \mathbf{0}^\top & \mathbf{0}^\top \end{pmatrix}. \tag{6.21}$$

Using this Jacobian, the Hessian and gradient can be computed as follows.

$$\begin{aligned} H &= \sum_{i=1} w_i J_i^\top \Omega_i J_i, \\ \mathbf{b} &= \sum_{i=1} w_i J_i^\top \Omega_i \mathbf{r}_i, \end{aligned} \tag{6.22}$$

where $w_i = \rho'\left(\mathbf{r}_i^\top \Omega_i \mathbf{r}_i\right)$. Using H and $\mathbf{b}$, we solve for $\delta\boldsymbol{\xi}$ such that $H\delta\boldsymbol{\xi} = -\mathbf{b}$, and update the pose as $T \leftarrow \exp\left(\delta\boldsymbol{\xi}^\wedge\right) T$.

A brief note on the derivation of equation (6.22): as discussed in Section 2.3, it is obtained by considering the cost function that uses the linear approximation of the residual vector when the state undergoes a small perturbation.

$$\sum_{i=1}^{N} \left(\mathbf{r}_i + J_i\delta\boldsymbol{\xi}\right)^\top \Omega_i \left(\mathbf{r}_i + J_i\delta\boldsymbol{\xi}\right) = \sum_{i=1}^{N} \mathbf{r}_i^\top \Omega_i \mathbf{r}_i + 2\delta\boldsymbol{\xi}^\top \Omega_i J_i \mathbf{r}_i + \delta\boldsymbol{\xi}^\top J_i^\top \Omega_i J_i \delta\boldsymbol{\xi}, \tag{6.23}$$

Differentiating the right-hand side with respect to $\delta\boldsymbol{\xi}$ and setting it equal to 0 yields the following result (note that, compared to equation (6.22), w_i is omitted in the equation below).

$$\sum_{i=1}^{N} J_i^\top \Omega_i J_i \delta\boldsymbol{\xi} = -\sum_{i=1}^{N} J_i^\top \Omega_i \mathbf{r}_i. \tag{6.24}$$

6.3.2 Loop Detection Success Criteria

To perform loop detection, GICP is used to minimize the cost function defined in equation (6.17), thereby aligning the point clouds. Based on this alignment result, it is necessary to determine whether loop detection has succeeded or failed. However, it is difficult to establish explicit criteria that reliably indicate the correctness of the alignment. For instance, thresholds on the final value of the cost function or on the mean residual can serve as indicators to some extent, but they do not always provide a definitive judgment. In practice, however, such indicators are often useful. In the implementation presented in this book, thresholds are applied to the mean residual and to the alignment ratio (i.e., the proportion of points whose distance to the nearest neighbor is below a threshold). These measures are then used to decide whether loop detection is deemed successful.

That said, this approach may still incorrectly classify failed alignments as successful. If such erroneous information is incorporated into the graph, it may cause the optimization to diverge or fail. For this reason, it is also effective to employ robust kernels, such as the Huber loss, in the graph optimization described in equation (6.15).

6.4 Computation of Coordinate Transformations and Map Construction

6.4.1 Transformations

Using LIO, we obtain the IMU pose ${}^{O}T_I$ in the odometry frame. When pose graph optimization is performed, the poses of newly detected keyframes are corrected, resulting in the IMU pose ${}^{M}T_I$ in the map frame. To compute the transformation ${}^{M}T_O$ between the map and odometry frames, as illustrated in Figure 6.3, we assume that the IMU pose in the map frame and the IMU pose in the odometry frame coincide. From this relationship, ${}^{M}T_O$ can be obtained as follows.

$$\begin{aligned} {}^{M}T_O &= {}^{M}T_I {}^{O}T_I^{-1}, \\ &= {}^{M}T_I {}^{I}T_O. \end{aligned} \tag{6.25}$$

Although the details are not discussed in this book, the transformation ${}^{M}T_O$ is obtained in the same manner when performing localization.

6.4.2 Map Construction

Once the pose graph optimization is completed, each keyframe in the map frame is corrected to ${}^{M}T_{I,i}$. Using these poses together with the corresponding point clouds ${}^{I}\mathcal{P}_i$ for each keyframe, the

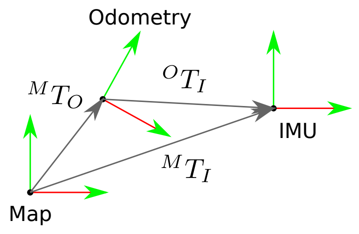

Figure 6.3: Residuals used in pose graph optimization.

point cloud map is constructed as follows.

$$ {}^{M}\mathcal{M} = \bigcup_{i=1}^{K} \bigcup_{{}^{I}\mathbf{p} \in {}^{I}\mathcal{P}_i} {}^{M}T_{I,i} {}^{I}\mathbf{p}, \tag{6.26} $$

where K denotes the number of keyframes. In the graph-based SLAM implementation presented in this book, however, this map point cloud is not directly used in the processing. Therefore, it is sufficient to generate it only once at the end of the SLAM process. That said, executing it after each pose graph optimization allows one to verify online whether the map is being correctly constructed. For this reason, the implementation described in this book performs this operation after every optimization.

6.5 Practical Considerations

In the method presented in this chapter, keyframes are detected by applying a threshold on the amount of motion, and the LiDAR point cloud corresponding to each keyframe is stored. A map is then constructed by transforming these point clouds according to their associated keyframe poses. One drawback of this approach is that not all LiDAR point clouds are used in the mapping process, which results in a lower map density. To increase the density, one could, for example, accumulate LiDAR scans over a fixed time interval and then detect a keyframe. However, this approach causes memory usage to grow over time. On the other hand, the motion-based keyframe detection method scales memory usage with motion rather than time, making it more memory-efficient. Thus, there is a trade-off between density and memory cost.

If the sole goal is to create a denser point cloud map, one practical approach is to reuse the same dataset used for SLAM-based mapping, perform localization again on this map, and then remap the point clouds according to the refined poses. While slightly more cumbersome, this method enables the creation of a high-density point cloud map without significant memory concerns. However, it should be noted that the result will not necessarily guarantee globally consistent mapping, since it is not based on optimizing global alignment of the point cloud data.

The method introduced in this chapter falls under graph-based SLAM using a pose graph. When using a pose graph, LiDAR point clouds are not directly optimized, and therefore, mapping accuracy may be lower compared to full graph-based SLAM. In general graph-based SLAM, however, all LiDAR point clouds are rarely optimized due to memory constraints. Instead, feature points or landmarks are extracted from the LiDAR point clouds and incorporated into the optimization. Consequently, the quality of mapping depends heavily on the performance of feature/landmark detection and association.

With modern 3D LiDARs providing long-range measurements and high-density point clouds, even simple point cloud registration between two scans can achieve highly accurate relative pose estimates. Therefore, under such conditions, pose graph optimization alone is often sufficient to generate an accurate map. Nonetheless, as demonstrated in works such as [11], incorporating full map optimization can further improve accuracy, meaning whether pose graph optimization alone is sufficient depends on the use case.

In the method described here, loop detection is performed simply by applying scan matching between nearby keyframes. As mentioned in Section 3.5, one of the prerequisites for successful scan matching is having a sufficiently accurate initial pose. However, odometry inherently accumulates drift error, and as the trajectory grows longer during SLAM execution, the accuracy of initial poses used for loop detection scan matching decreases. Thus, relying solely on scan matching may fail to close large loops. In such cases, alternative approaches such as extracting features or descriptors from LiDAR scans and detecting similar scans based on these features are required. However, such methods generally tend to be less accurate than loop detection based purely on scan matching. Therefore, additional considerations such as outlier detection during pose graph optimization become important.

Even when loop detection relies solely on scan matching, correctly closing loops effectively compensates for accumulated odometry drift. From a practical standpoint, designing trajectories that facilitate loop closures is often a strategy to ensure successful map building. However, when loops are too large, the system may reach its performance limits, and incorporating alternative approaches becomes necessary.

Bibliography

[1] F. Dellaert, D. Fox, W. Burgard, and S. Thrun. Monte Carlo localization for mobile robots. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 2, pages 1322–1328, 1999.

[2] N. Akai. Reliable Monte Carlo localization for mobile robots. *Journal of Field Robotics*, 40(3):595–613, 2023.

[3] W. Xu and F. Zhang. FAST-LIO: A fast, robust LiDAR–inertial odometry package by tightly-coupled iterated kalman filter. *IEEE Robotics and Automation Letters*, 6(2):3317–3324, 2021.

[4] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. The MIT Press, 2005.

[5] J. Borenstein, H.R. Everett, L. Feng, and D. Wehe. Mobile robot positioning: Sensors and techniques. *Journal of Robotic Systems*, 14(4):229–340, 1997.

[6] D. Nister, O. Naroditsky, and J. Bergen. Visual odometry. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, volume 1, 2004.

[7] J. Zhang and S. Singh. LOAM: lidar odometry and mapping in real-time. In *Robotics: Science and Systems*, 2014.

[8] Christian Forster, Luca Carlone, Frank Dellaert, and Davide Scaramuzza. On-manifold preintegration for real-time visual–inertial odometry. *IEEE Transactions on Robotics*, 33(1):1–21, 2017.

[9] T. Shan, B. Englot, D. Meyers, W. Wang, C. Ratti, and Rus D. LIO–SAM: Tightly-coupled lidar inertial odometry via smoothing and mapping. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5135–5142. IEEE, 2020.

[10] W. Xu, Y. Cai, D. He, J. Lin, and F. Zhang. FAST-LIO2: Fast direct LiDAR–inertial odometry. *IEEE Transactions on Robotics*, 38(4):2053–2073, 2022.

[11] K. Koide, M. Yokozuka, S. Oishi, and A. Banno. GLIM: 3D range-inertial localization and mapping with GPU-accelerated scan matching factors. *Robotics and Autonomous Systems*, 179(2):104750, 2024.

[12] K. Chen, R. Nemiroff, and B. T. Lopez. Direct lidar-inertial odometry: Lightweight LIO with continuous-time motion correction. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3983–3989, 2023.

[13] B. T. Lopez. A contracting hierarchical observer for pose-inertial fusion. *arXiv:2303.02777*, 2023.

[14] A. Segal, D. Hähnel, and S. Thrun. Generalized–ICP. In Jeff Trinkle, Yoky Matsuoka, and José A. Castellanos, editors, *Robotics: Science and Systems*. The MIT Press, 2009.