Tightly Coupled LiDAR-Inertial Odometry

Taylor Pool

CMU-RI-TR-24-17

May 2024

School of Computer Science
The Robotics Institue
Carnegie Mellon University
Pittsburgh, Pennsylvania

Thesis Committee
Michael Kaess, Chair
Zach Manchester
Shibo Zhao

Submitted in partial fulfillment of the requirements for the Degree of Master of Science

Abstract

In the age of self-driving, LiDAR and IMU represent two of the most ubiquitous sensors in use. Kalman filtering and loosely coupled approaches dominate industry techniques, while current research trends towards a more tighly coupled formulation involving a joint optimization of IMU and LiDAR measurements. After two years of experience working with and creating tightly coupled LiDAR-inertial odometry (LIO) systems for offroad and indoor environments, we detail our findings regarding such implementations. Moreover, we present a general framework involving point-to-point based registration, an adaptive robust kernel, and state-of-the-art preintegration for odometry. Our method operates in real-time on a moderately powerful CPU, and we showcase its capabilities in high speed offroad environments, as well as indoor environments.

Furthermore, we provide an extensive section devoted to background information required to implement our version of LiDAR-inertial odometry. All algorithms and equations (except for IMU preintegration) are given with the goal of this document being completely self-contained. We provide robust analysis of tradeoffs made along the way, and provide direction for future improvements based on real-world observations on real hardware.

The key contributions were developed over two programs: DARPA RACER, a program exploring high-speed off-road autonomy across desert and wooded environments, and MMPUG, a program devoted to indoor navigation with small mobile robots.

Acknowledgements

My advisor, Dr. Michael Kaess, has been a constant source of knowledge and encouragement. This thesis would not have been possible if not for his generous expertise. I also owe much to Dr. Matt Travers for gifting me the opportunity to work on real robots. That has been so very rewarding. Great thanks also goes to my committee members: Shibo Zhao and Dr. Zach Manchester for their key insights and valuable feedback over the course of writing this thesis. I would also be remiss if I didn't acknowledge Darwin Mick, Jay Maier, and Adam Johnson for their grit and intellectually stimulating discussions. I am grateful for their patience with my mistakes, and their selfless help to fix them. My labmates all provided valuable interactions that made me a better student. Thank you (in no partiular order) Montiel, Akshay, Joe, Dan, Easton, Mohamad, Akash, Suddhu, Ray, Chris, Lihong, Tianxiang, and Andrew. My parents, Trevor and Heather also have been a source of encouragement during times of frustration, and I owe much to them. Finally, I would like to thank my wife Anna who has been invariably supportive and unfailingly kind.

Contents

List of Algorithms

List of Figures

List of Tables

Chapter 1

Background

In 2023, the total human population surpassed 8 billion people. This is a remarkable achievement that also brings unprecedented challenges. As demand for food, water, transportation, education, and communication continue to increase, robots can and will provide essential capabilities to cope with these rising needs, stemming largely from autonomous operation. In each situation, robots need to understand their position and velocity. This is the principle task of odometry, and is critical to all other areas of robot operation, including perception, planning, and control. Key sensor technologies have enabled advances in odometry estimation. Among these are LiDAR (Light Ranging and Detection) and IMU (Inertial Measurement Unit).

LiDAR (see Figure 1.1) is a sensor that samples points made by the intersection of nanometer-wavelength light with surfaces. A single LiDAR scan contains thousands of such points, each collected at a discrete time. Because of the small wavelength used, LiDAR ranges measurements are very precise when no motion is present. However, high robot velocity during scans causes motion distortion, which severely limits accuracy. Worse, LiDAR scans have a frequency of 10-20 Hz, which is not sufficient for many real-time controllers.

On the other hand, IMU has a higher output rate (hundreds of Hz), correlated with acceleration and angular velocity in the body frame. Unfortunately, the IMU does not observe these quantities directly, instead measuring specific force: a combination of acceleration and the normal force due to gravity that depends on the orientation of the IMU itself. Additionally, IMU tends to suffer from biases that cannot be factory calibrated because they slowly change over time. Naive reliance on IMU for odometry estimation will

Figure 1.1: An example of a mechanically rotating LiDAR (VLP-16); Credit: Velodyne Lidar

almost inevitably diverge due the double integration of discrete acceleration measurements and slowly varying biases.

Fortunately, combining LiDAR and IMU into a single odometry pipeline mitigates the issues specific to each individual sensor, and creates a robust solution. This is our approach.

Companies such as Near Earth Autonomy (Figure 1.2) and Glydways (Figure 1.3) rely on LIDAR sensors for navigation and perception.

In this document, we explore LiDAR-inertial odometry and detail our findings on a novel algorithm that makes a significant departure from the existing literature.

Over the course of two years, projects such as DARPA RACER (Figure 1.4) and MMPUG (Figure 1.5) have utilized LiDAR-inertial odometry to great effect. The development of this odometry algorithm is the main topic of our thesis. It should be noted that the fusion of IMU and LiDAR has been the study of many research papers in robotics. We seek to push the design of LiDAR-inertial odometry even further through the introduction of a tightly-coupled framework. We present results, and show that we are able to achieve robust results on a number of different environments. The combination of IMU and LiDAR is especially advantageous in long feature-less hallways, which traditionally stymie LiDAR. We present the approach of simpler but more robust methods in order to demonstrate that less is more, even in robotics. While we acknowledge that sophistication may yield state

Figure 1.2: A drone flying with LiDAR sensor attached on top; Credit: Near Earth Autonomy

Figure 1.3: A self-driving car with LiDAR sensor attached to front; Credit: Glydways

of the art results, this often comes at the cost of environment-based tuning, and in some cases, loss of robustness in the algorithm itself. We feel that a single odometry pipeline that works across a variety of sensor configurations and environments is of more benefit to the robotics community at large a highly specialized architecture that performes well within a single environment. This key insight motivates the formulation of our framework. Our core quest is to integrate LiDAR and IMU into a single odometry framework in a tightly coupled fashion that achieves a synergistic effect.

Figure 1.4: A vehicle driving with LiDAR sensors attached; Credit: DARPA RACER Website

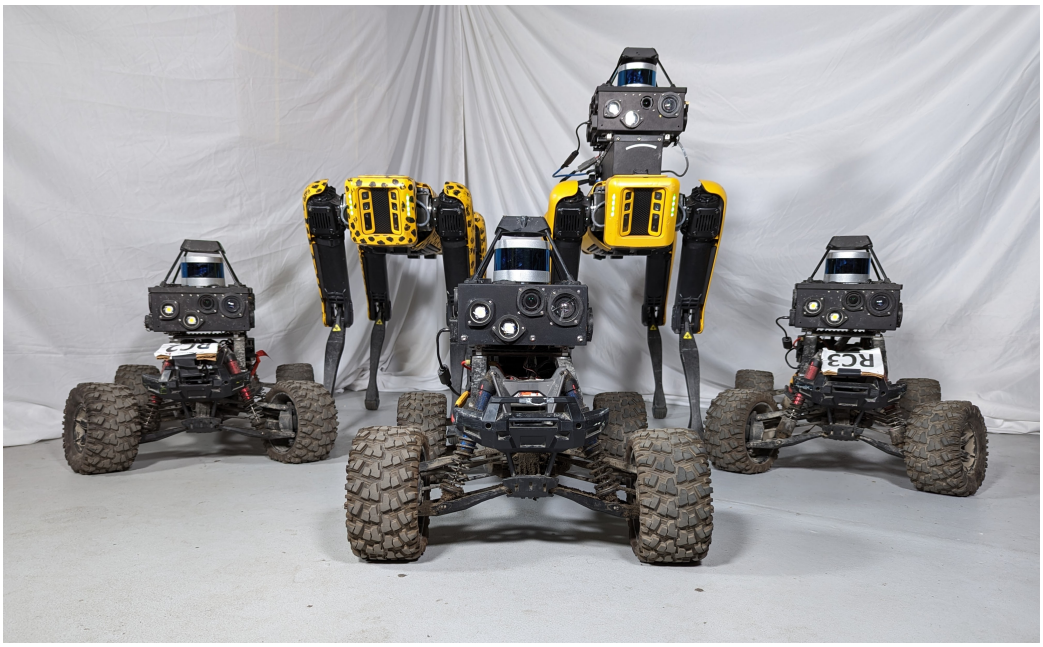


Figure 1.5: Various robots with LiDAR sensors attached; Credit: Matt Lab

Chapter 2

Related Work

For ease of reference, we provide Table 2.1 for a survey of relevant LiDAR and LiDAR-inertial odometry frameworks. We also provide Table 2.2 for a survey of robust methods applied to state estimation.

Table 2.1: Comparison of various LiDAR frameworks

Work	Year	Contribution	IMU
LOAM [33]	2014	Feature extraction and dewarping	Optional
LeGO-LOAM [19]	2018	Segment for group plane features	No
Ye [31]	2019	Tightly coupled feature-based LIO	Yes
LIO-SAM [20]	2020	Smoothing for multiple scans	Yes
FAST-LIO [29]	2021	Extended Kalman filter	Yes
F-LOAM [27]	2021	Faster performance	No
LVI-SAM [21]	2021	Combines IMU, LiDAR, vision	Yes
CT-ICP [9]	2022	Continuous time trajectory estimation	No
FAST-LIO2 [28]	2022	Incremental KD-tree	Yes
KISS-ICP [25]	2023	Adaptive threshold over point-to-point ICP	No
Point-LIO [13]	2023	Sequential processing point-by-point	Yes

The foundations for LiDAR odometry began with [2], which specified the basic iterative closest point algorithm. This method allowed for matching of two separate point clouds that did not have the same number of points. Chen and Medioni added an additional residal corresponding to point-to-plane elements [6]. The seminal textbook, Probabilistic Robotics, [24], remained a go-to source for filtering approaches to LiDAR state estimation.

Table 2.2: Robust methods for LiDAR scan registration

Work	Year	Contribution
Generalized ICP [18]	2009	Probabilistic Residuals for ICP
M-estimators [16]	2019	M-Estimators for residual
Teaser [30]	2020	Graduated Non Convexity
General Adaptive Robust Kernel [1]	2019	Generalized Robust M-Estimator
Adaptive Robust Kernels for NLS [5]	2021	Nonlinear Least Squares

However, vanilla point-to-point registration suffered greatly from outliers. To combat this issue, authors in [18] proposed mapping uncertainty of each LiDAR point position into uncertainty associated with the registration algorithm. However, this approach still did not address how to robustly deal with outliers. In order to decrease the number of outliers, authors of [33] proposed an intelligent extraction of keypoints from a scan. However, instead of simply extracting keypoints using a nearest-neighbor scheme, the authors searched along individual scan lines, which provided more robust feature selection. Variant of LOAM include F-LOAM [27], and LeGO-LOAM [19]. These methods could be combined with a Kalman filtering-based approach like the Unscented Kalman Filter [26].

Orthogonally, methods for large-scale incremental inference like [15] allowed for fast computation over many variables of interest.

However, other approaches sought to find more optimal estimates for larger sets of nonlinear data. These approaches relied on smoothing over multiple timesteps using factor graphs [8]. These factor graphs were borne from Bayes nets [17], and represented a new method for describing cost functions in state estimation.

Other approaches to combine IMU and LiDAR were integrated with vision, resulting in [34].

In an effort to address robustness, Zhang attempted to analyze the degeneracy associated with a given cost function by examining the spectral decomposition of the function [32].

In 2016, a major breakthrough for efficient IMU integration in the smoothing framework was released in [11]. However, people still tried to understand how to resolve outliers by using means such as m-estimators from robust statistics [16]. Additionally, other algorithms such as [30] approached it from

a perspective of graduated non convexity, but this was unsuitable for high-rate online estimation.

Many of these more recent approaches sought to leverage Lie theory [23] for more efficient constraints, especially over rotations. In order to properly talk about rotations, Sola released a brief guide for the roboticist [22].

More tightly coupled approaches like [31] sought to integrate IMU and LiDAR on a larger scale in hope of obtaining more accurate results. Super Odometry [35] and LIO-SAM [20] pioneered efforts for a more tightly coupled framework. Fast-LIO [29] and Fast-LIO 2 [28] attempted to use an incremental KD-tree for efficient hashing and matching of LiDAR points. CT-ICP [9] proposed using continuous time trajectories for estimation. KISS-ICP [25] demonstrated the power of point-to-point ICP, using an intelligent robust cost function that automatically adapted to various different environments. This was possible due to the nature of an adaptive kernel threshold based on the size of the correction assumed from the ICP. Point-LIO [13] showed the power of incorporating IMU measurements as real measurements in the Kalman Filter.

Chapter 3

Mathematical Foundations

3.1 Probability

In order to reason about the state of a robot, the concept of probability is relevant. Rarely, if ever, do we know with exact certainty the position, orientation, or velocity of any agent. Indeed, at the most fundamental level, Heisenberg's Uncertainty Principle [3] asserts that is impossible for one to simultaneously know the position and velocity of any particle.

Thus, we leverage probability theory to quantifiably declare the state of the robot and our confidence respecting said state. A useful reference is [10].

Definition 1. Define a probability space as a triple of (Ω, E, p) where

1. Ω is a set of outcomes.
2. E is a σ-algebra (collection of well-behaved subsets) of events taken from Ω
3. $p : E \to [0, 1]$ is a probability measure

Note that p must satisfy the following three properties:

$$p(\Omega) = 1 \tag{3.1}$$

$$p(\varnothing) = 0 \tag{3.2}$$

$$p\left(\cup_{i=1}^{n} A_i\right) \leq \sum_{i=1}^{n} p\left(A_i\right) \tag{3.3}$$

Example 1 (Fair Coin Toss)**.** Consider the onetime act of tossing a fair coin. Take $\Omega = \{H, T\}$, $E = \{\varnothing, \{H\}, \{T\}, \Omega\}$. Since the coin is fair,

$$p(\varnothing) = 0 \tag{3.4}$$

$$p(\{H\}) = \frac{1}{2} \tag{3.5}$$

$$p(\{T\}) = \frac{1}{2} \tag{3.6}$$

$$p(\{H, T\}) = 1 \tag{3.7}$$

3.1.1 Random Variables

Because it is difficult to directly observe and manipulate expressions of raw events from E, it is useful to define a mapping between E and another space where usual mathematical concepts operate. A random variable is such a mapping.

Definition 2 (Random Variable)**.** $X : E \to \mathbb{R}^n$ is a random variable if it is bijective. Then, with $z \in \mathbb{R}^n$, we say $p(X^{-1}(z)) = p(z \in X) = p(z)$. Furthermore, define $\text{support}(X) = \{z \in \mathbb{R}^n : p(z) > 0\}$.

Example 2 (Fair Coin Toss)**.** Consider again the onetime act of tossing a fair coin. One possible random variable $X : E \to \mathbb{R}$ is the following:

$$X(\{T\}) = 0 \tag{3.8}$$

$$X(\{H\}) = 1 \tag{3.9}$$

$$\tag{3.10}$$

Random variables allow us to take summations and integrals along paths of events. In robotics, this is critical to estimate the state of the robot.

3.1.2 Cumulative Distribution Functions

Recall that the probability of the set of outcomes, Ω is 1. Many times, we would like to understand in terms of random variables how much of the set of outcomes lies below a given outcome, mapped to a random variable. The cumulative distribution function is exactly this:

$$F_X(x) : \mathbb{R}^n \to [0, 1], F_X(x) = p(\{z \in \mathbb{R}^n : z \preccurlyeq x\}) \tag{3.11}$$

Note that $F_X(\inf \text{support}\, X) = 0$, and $F_X(\sup \text{support}\, X) = 1$.

3.1.3 Finite Discrete Case: Probability Mass Functions

Let X be a discrete random variable with a finite number of corresponding outcomes. We define the probability mass function $f_X(x) = p(x)$, and we have

$$F_X(x) = \sum_{k=1}^{n} f_X(s_k)(s_k \preccurlyeq x) \tag{3.12}$$

3.1.4 Finite Discrete Case: Expectation

Now, we will define the expectation, or expected value of the finite discrete random variable, X.

$$\mathbb{E}[X] = \sum_{k=1}^{n} f_X(s_k)s_k \tag{3.13}$$

3.1.5 Continuous Case: Probability Density Functions

For some continuous random variables, we can represent the cumulative distribution function as the integral over another function, which is **not** a probability, but which is called a probability density function.

$$F_X(x) = \int_{\inf \text{support}\,(X)}^{x} f_X(s)\mathrm{d}s \tag{3.14}$$

3.1.6 Continuous Case: Expectation

In the continuous case, we can also define the expectation as the following:

$$\mathbb{E}[X] = \int_{\text{support}\,(X)} s f_X(s)\mathrm{d}s \tag{3.15}$$

Note that not all continuous random variables have an expected value.

Example 3 (Cauchy Distribution)**.** The probability density function for the Cauchy distribution, with support $\mathbb{R}$, is:

$$f_X(x) = \frac{1}{\pi\left(1 + x^2\right)} \tag{3.16}$$

Then computing the expectation of this distribution yields:

$$\mathbb{E}[X] = \int_{\mathbb{R}} f_X(x) x \mathrm{d}x \tag{3.17}$$

$$= \int_{-\infty}^{\infty} \frac{x}{\pi\left(1+x^2\right)} \mathrm{d}x \tag{3.18}$$

$$= \frac{1}{2\pi} \log\left(1+x^2\right)|_{-\infty}^{\infty} \tag{3.19}$$

The last expression is undefined, and so the expectation of the Cauchy distribution is also undefined.

3.1.7 Covariance

Across both continuous and finite discrete random variables, the concept of covariance is useful. It is defined as:

$$\Sigma = \mathbb{E}\left[\left(X - \mathbb{E}[X]\right)\left(X - \mathbb{E}[X]\right)^{\mathrm{T}}\right] \tag{3.20}$$

Note that the diagonal entries of Σ are called variances and denote how much spread exists in that particular dimension. Meanwhile, off-diagonal entries of Σ measure the amount that each dimension varies with other dimensions.

It should be noted that Σ is a symmetric matrix.

3.1.8 Normal Distribution

The normal distribution is of seminal importance in robotics. This is due to its possessing special properties, a few of which we list below:

1. It maximizes differentiable entropy (defined as $\mathbb{E}[-\log(f_X(X))]$) among all continuous random variables with a mean and covariance [7]

2. Sampling from the normal distribution is easy via Gibb's sampling

3. It corresponds to solving a linear system of equations, as we will see later

4. It has a closed form propagation through a linear operator corresponding to another normal distribution

We denote random variable X as being normally distributed with expectation μ and covariance Σ by $X \sim \mathcal{N}(\mu, \Sigma)$. Then $\text{support}(X) = \mathbb{R}^n$, and the probability density function is:

$$f_X(x) = (2\pi \det \Sigma)^{-\frac{1}{2}} \exp\left[-\frac{1}{2}(x-\mu)^{\mathrm{T}} \Sigma^{-1}(x-\mu)\right] \tag{3.21}$$

3.1.9 Bayes Rule

Oftentimes we wish to incorporate prior information into probabilities of other events. Other methods in robotics include maximum likelihood estimation and variants of the Bayes filter. In this case, Bayes rule provides a way to do so:

Let $x \sim \mathcal{X}, y \sim \mathcal{Y}, z \sim \mathcal{Z}$

$$p(x|y,z) = \frac{p(y|x,z)p(x|z)}{p(y|z)} \tag{3.22}$$

3.2 Rotations

Most robots need to estimated orientation as well as position. In three-dimensional space, the set of rotations is not a vector space. To see this fact, note that the commutative property is not satisfied for two arbitraty rotations: a yaw of 90 degrees followed by a pitch of 90 degrees is not equivalent to a pitch of 90 degrees followed by a yaw of 90 degrees. Instead, rotations form a group, which means that they satisfy the following properties:

Definition 3 (Group)**.** Let $\mathcal{G} = (\mathcal{X}, \circ)$ be a pair consisting of a set and an operator $\circ : \mathcal{X} \times \mathcal{X} \to \mathcal{X}$. We say $x \in \mathcal{G}$ if $x \in \mathcal{X}$. Now take $x, y, z \in \mathcal{G}$. Then $\mathcal{G}$ is a group if

$$x \circ y \in \mathcal{X} \qquad \text{(closure)} \tag{3.23}$$

$$\exists I \in \mathcal{G}, x \circ I = I \circ x = x \qquad \text{(identity)} \tag{3.24}$$

$$\forall x \in \mathcal{G}; \exists x^{-1} \in \mathcal{G} : x \circ x^{-1} = x^{-1} \circ x = I \qquad \text{(inverse)} \tag{3.25}$$

$$(x \circ y) \circ z = x \circ (y \circ z) \qquad \text{(associativity)} \tag{3.26}$$

Example 4 ($\mathbb{H}$: Group of Quaternions)**.**

The group of quaternions, $\mathbb{H} = (\mathbb{R}^4 \setminus \{0\}, \circ)$ is defined as follows: with $x = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \end{bmatrix}^{\mathrm{T}}$, $y = \begin{bmatrix} y_1 & y_2 & y_3 & y_4 \end{bmatrix}^{\mathrm{T}}$

$$x \circ y = \begin{bmatrix} x_1y_1 - x_2y_2 - x_3y_3 - x_4y_4 \\ x_1y_2 + x_2y_1 + x_3y_4 - x_4y_3 \\ x_1y_3 - x_2y_4 + x_3y_1 + x_4y_2 \\ x_1y_4 + x_2y_3 - x_3y_2 + x_4y_1 \end{bmatrix} \tag{3.27}$$

Trivially, $x \circ y \in \mathbb{R}^4$. Furthermore, let $e_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \end{bmatrix}^{\mathrm{T}}$. Then $x \circ e_1 = e_1 \circ x = x$. Finally, $x^{-1} = \frac{1}{||x||_2^2} \begin{bmatrix} x_1 & -x_2 & -x_3 & -x_4 \end{bmatrix}^{\mathrm{T}}$ We leave confirmation of the identity, inverse, and associativity properties as an exercise to the reader.

To talk about rotations, we must build upon the group of quaternions by examining a special subgroup: that of quaternions of unit norm.

Example 5 ($\bar{\mathbb{H}}$: Group of Unit Quaternions)**.**

$\bar{\mathbb{H}}$ represents all quaternions of unit length.

$$\bar{\mathbb{H}} = \{x \in \mathbb{H} : ||x||_2 = 1\} \tag{3.28}$$

Proving that $\bar{\mathbb{H}}$ is a group is an exercise for the reader.

Notably, we can also define an operation between a unit quaterion q and a point $p \in \mathbb{R}^3$ that corresponds to rotating p by q.

$$p = q * p = q \circ q(p) \circ q^{-1} \tag{3.29}$$

Note that $q(p) = \begin{bmatrix} 0 & p^{\mathrm{T}} \end{bmatrix}^{\mathrm{T}}$

Example 6 ($\mathcal{S}^3$: Group of 3D Rotations)**.** We note that the set of unit quaternions forms a double cover of the set of 3D rotations. Namely, quaternions q and $-q$ correspond to the same 3D rotation. Because of this, we constrain the quaternions we use to represent rotations to those where the first non-zero element is positive.

3.2.1 Lie Groups

A Lie group is a mathematical construct that has a rich history, spanning back to Sophus Lie.

Definition 4 (Lie Group)**.** A Li group is a group that is also a smooth manifold (locally appears similar to a vector space). A very brief introduction is given here, but further information may be found in [22], [23].

The tangent space to the manifold of a Lie group is isomorphic to $\mathbb{R}^n$ for some n. Thus, we say that a Lie group has dimension n. The key mappings between a Lie group $\mathcal{X}$ and $\mathbb{R}^n$ are as follows:

1. Exponential map from $\mathbb{R}^n$ to the Lie group (denoted $\text{Exp} : \mathbb{R}^n \to \mathcal{X}$)
2. Logarithmic map from the Lie group to $\mathbb{R}^n$ (denoted $\text{Log} : \mathcal{X} \to \mathbb{R}^n$)

The key value of Lie theory is in taking nontrivial group structures and providing a framework for calculus, incremental updates, and cost functions.

Example 7 (Unit Quaternions)**.** Unit Quaternions form a Lie group. They have a 3-dimensional tangent space. Given $\omega \in \mathbb{R}^3$, the exponential is as follows, with $\theta = \frac{||\omega||}{2}$:

$$\text{Exp}(\omega) = \begin{cases} \begin{bmatrix} \cos\theta \\ \frac{\omega}{\theta}\sin\theta \end{bmatrix} & \theta \neq 0 \\ I & \text{otherwise} \end{cases} \tag{3.30}$$

Now, suppose we are given $q \in \bar{\mathbb{H}}$. Let $\theta = 2 * \arccos q_1$. Then the logarithmic map is

$$\text{Log}(q) = \begin{cases} \frac{\theta}{\sin\theta} q_{2,3,4} & \theta \neq 0 \\ \mathbf{0} & \text{otherwise} \end{cases} \tag{3.31}$$

Example 8 (3D Poses)**.** 3D poses form another Lie group, and have a 6 dimensional tangent space corresponding to angular and linear velocities (called a twist). Given a twist $\xi = \begin{bmatrix} \omega^{\mathrm{T}} & v^{\mathrm{T}} \end{bmatrix}^{\mathrm{T}} \in \mathbb{R}^6$, the exponential map is as follows:

$$\text{Exp}(\xi) = (\text{Exp}(\omega), V(\omega)v) \tag{3.32}$$

$$V(\omega) = I + \frac{1-\cos\theta}{\theta^2} \lfloor\omega\rfloor_\times + \frac{\theta - \sin\theta}{\theta^3} \lfloor\omega\rfloor_\times^2 \tag{3.33}$$

$$\theta = ||\omega|| \tag{3.34}$$

The logarithmic map is

$$\text{Log}(R, t) = \begin{bmatrix} \omega \\ V^{-1}(\omega)\, t \end{bmatrix} \tag{3.35}$$

$$\omega = \text{Log}(R) \tag{3.36}$$

3.2.2 Differentiation over Groups

In order to properly optimize over rotations, we need to respect the special structure they posses. One method of doing so is to specially define derivatives with respect to rotations along a tangent space. By doing so, we ensure that all rotations remain valid throughout the optimization. We take inspiration for our approach from [22] and [14].

Let $\mathcal{X}$ be group of dimension n. We define *box-plus* to be a function

$$\boxplus : \mathcal{X} \times \mathbb{R}^n \to \mathcal{X} \tag{3.37}$$

Let $\mathcal{Y}$ be a group of dimension m. We also define *box-minus* to be a function

$$\boxminus : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}^m \tag{3.38}$$

Let $f : \mathcal{X} \to \mathcal{Y}$ be a function.

Then the Frechet derivative of f is defined as $Df : \mathcal{X} \to \mathbb{R}^{m \times n}$ if the following limit exists:

$$\lim_{h \to 0} \frac{|| \left(f(x \boxplus h) \boxminus f(x)\right) - Df(x)h||}{||h||} = 0, \quad h \in \mathbb{R}^n \tag{3.39}$$

Example 9 (Vectors in $\mathbb{R}^n$)**.** Let $x, y \in \mathbb{R}^n$. Then $\boxplus, \boxminus$ are trivial operators:

$$x \boxplus y = x + y \tag{3.40}$$

$$x \boxminus y = x - y \tag{3.41}$$

Example 10 (3D Rotations)**.** Let $R, R_1, R_2 \in \bar{\mathbb{H}}$. Furthermore, let $\omega \in \mathbb{R}^3$. We define $\boxplus, \boxminus$ as follows:

$$R \boxplus \omega = R \circ \operatorname{Exp} \omega \tag{3.42}$$

$$R_1 \boxminus R_2 = \operatorname{Log}({R_2}^{-1} \circ R_1) \tag{3.43}$$

Example 11 (3D Poses)**.** Let $T \in \mathrm{SE}(3), T = (R, p)$. Moreover, let $\xi \in \mathbb{R}^6, \xi = \left[\omega^{\mathrm{T}}, v^{\mathrm{T}}\right]^{\mathrm{T}}$.

$$T \boxplus \xi = (R \boxplus \omega, p + v) \tag{3.44}$$

$$T_1 \boxminus T_2 = \begin{bmatrix} R_1 \boxminus R_2 \\ p_1 - p_2 \end{bmatrix} \tag{3.45}$$

Note that this choice of $\boxplus, \boxminus$ does not conform to the motion of a rigid body, and is not the exponential map for $\mathrm{SE}(3)$. We can use this more efficient operation within an optimization framework, but not for actual motion propagation.

3.3 Optimization

In state estimation we seek to find the most likely state of the robot (Maximum Likelihood Estimation) or the most likely state of the robot given some prior information (Maximum A Posteriori). Both of these problems may be formulated in terms of optimization, and so it is critical to have an understanding of this topic.

We will additionally discuss equality constraints, as these are necessary for rotations. However, we will not discuss inequality constraints.

3.3.1 Unconstrained Optimization

Let $\mathcal{X}$ be a group of dimension n with operations $\boxplus, \boxminus$ defined.

Let $f : \mathcal{X} \to \mathbb{R}$. We seek to find the global optimum $x^\star$

$$x^\star = \min_x f(x) \tag{3.46}$$

Then we define the (Frechet) derivative $Df : \mathcal{X} \to \mathbb{R}^{1\times n}$ where the following limit is defined

$$\lim_{h\to 0} \frac{||\left(f(x \boxplus h) \boxminus f(x)\right) - Df(x)h||}{||h||} = 0, h \in \mathbb{R}^n \tag{3.47}$$

We define the gradient of f as $\nabla f = D^{\mathrm{T}} f$.

Finally, we define the Hessian of f as $D^2 f = D\left[D^{\mathrm{T}} f\right] = D\left[\nabla f\right]$. Note that the Hessian is symmetric.

3.3.2 Equality Constraints

We note that including equality constraints is relatively simple.

We assume that the equality constraints are formulated as $g(x) = 0, g : \mathcal{X} \to \mathbb{R}^m$.

Then we introduce the dual variable $\lambda \in \mathbb{R}^m$, and create the Augmented Lagrangian (so named because we augmented the variables with dual ones):

$$\mathcal{L}\left(x, \lambda\right) = f(x) + \lambda^{\mathrm{T}} g(x) \tag{3.48}$$

The original optimization problem then becomes $\arg\min_{x,\lambda} \mathcal{L}\left(x, \lambda\right)$. All previous machinery still applies. We note that optimization over rotations can be carried out in this form without relying on the $\boxplus, \boxminus$ operators. However, due to the popularity of this approach in state estimation, and its use in our nonlinear optimization library GTSAM, we emphasize that approach.

3.3.3 Taylor's Theorem Applied to the Cost Function

We next examine Taylors Theorem applied twice over for a cost function:

Theorem 1 (Taylor's Theorem)**.** *Let $f : \mathcal{X} \to \mathbb{R}$ be twice-differentiable. Let $x \in \mathcal{X}, \Delta x \in \dim(\mathcal{X})$.*

$$f(x \boxplus \Delta x) = f(x) + Df(x)\Delta x + \frac{1}{2}\Delta x^{\mathrm{T}} D^2 f(x)\Delta x + O\left(\Delta x^3\right) \tag{3.49}$$

$$\approx f(x) + Df(x)\Delta x + \frac{1}{2}\Delta x^{\mathrm{T}} D^2 f(x)\Delta x \tag{3.50}$$

3.3.4 First Order Necessary Condition for Global Optimum

In order for a twice-differentiable function to have a global optimum, the following condition must hold.

$$\forall x \in \mathbb{R}^n, \quad f(x^*) \leq f(x) \implies Df(x^*) = 0 \tag{3.51}$$

It should be noted that this is not a sufficient condition to certify that $x \in \mathcal{X}$ is a global optimum.

3.3.5 Second Order Necessary Conditions for Global Optimum

We begin with some notation for a matrix $A \in \mathbb{R}^{n \times n}$:

$$A > 0 \iff x^{\mathrm{T}} A x > 0, \quad \forall x \in \mathbb{R}^n \tag{3.52}$$

$$A \geq 0 \iff x^{\mathrm{T}} A x \geq 0, \quad \forall x \in \mathbb{R}^n \tag{3.53}$$

We say A is positive definite if $A > 0$, and A is positive semi-definite if $A \geq 0$.

Then, a necessary second order condition for global optimum pertains to the Hessian of the cost function:

$$\forall x \in \mathbb{R}^n, \quad f(x^*) \leq f(x) \implies D^2 f(x^*) \geq 0 \tag{3.54}$$

Again, this is not a sufficient condition to certify that x is a global optimum.

First and Second Order Sufficient Conditions for Local Optimum

Now, we provide first and second order sufficient conditions for local optimum. In other words, if $x^\star$ satisfies the conditions below, it is an optimum within a local neighborhood. However, $x^\star$ is not guarenteed to be the globally optimum solution.

$$Df(x^*) = 0 \wedge D^2 f(x^*) > 0 \implies \tag{3.55}$$

$$\exists \epsilon > 0 \quad \text{s.t} \quad 0 < ||x - x^*|| < \epsilon \implies f(x^*) \leq f(x) \tag{3.56}$$

Analytic Solutions

In order to find the globally optimal solution, we can find all the local optimum according to the first (and second order) sufficient conditions. Then the global optimum is the local optimum that has the smallest cost. To restate, finding the global optimum involves the following steps:

1. Find the set $S = \{x \in \mathbb{R}^n : Df(x) = 0 \wedge D^2 f(x) > 0\}$
2. Find the set $T = \{x \in S : D^2 f(x) > 0\}$
3. $x^* = \min T$

Newton's Method

Unfortunately, finding the analytical solution to optimization problems in robotics is often impossible. Instead we turn to numerical methods for finding the optimal value. The branch of numerical methods we consider are iterative ones that operate given local information. Other global methods for finding optimal values are out of scope for this thesis.

Recall that

$$f(x \boxplus \Delta x) \approx f(x) + Df(x)\Delta x \tag{3.57}$$

$$Df(x \boxplus \Delta x) \approx Df(x) + \Delta x^{\mathrm{T}} D^2 f(x) \tag{3.58}$$

We set $Df(x \boxplus \Delta x) = 0$ and solve for Δx.

$$0 = Df(x) + \Delta x^{\mathrm{T}} D^2 f(x) \tag{3.59}$$

$$D^2 f(x)\Delta x = -D^{\mathrm{T}} f(x) \tag{3.60}$$

This forms the basis for Newton's method as described in Algorithm 1.

Algorithm 1 Newton's method

function NEWTON'S METHOD(f, Df, D^2f, x_0)
    $k \leftarrow 0$
    $\delta \leftarrow \delta_{\max} + 1$
    while $k <$ max_iterations **and** $\delta > \delta_{\max}$ **do**
        solve $D^2f(x_k)\Delta x_k = -D^{\mathrm{T}}f(x_k)$
        $x_{k+1} \leftarrow x_k \boxplus \Delta x_k$
        $\delta \leftarrow ||\Delta x_k||$
        $k \leftarrow k + 1$
    end while
end function

Nonlinear Least Squares

The nonlinear least squares problem is the following:

Minimize $f : \mathcal{X} \to \mathbb{R}$,

$$f(x) = \frac{1}{2}\sum_{i=1}^{l} r_i^{\mathrm{T}}(x) r_i(x), \quad r_i : \mathcal{X} \to \mathbb{R}^n \tag{3.61}$$

We will leverage Newton's method to optimize over f.

Note that differentiation is a linear operator, and so $Df : \mathcal{X} \to \mathbb{R}^{1\times n}$ is:

$$Df(x) = \sum_{i=1}^{l} r_i^{\mathrm{T}}(x) Dr_i(x) \tag{3.62}$$

Now, we must compute the Hessian, D^2f. Note that

$$D^2f(x) = \sum_{i=1}^{l} D\left[D^{\mathrm{T}}[r_i]r_i\right](x) \tag{3.63}$$

From here on, we drop the i-th index notation from r_i, Dr_i to make the reading simpler.

To aid computing $D\left[D^{\mathrm{T}}[r]r\right](x)$ we introduce two operators: vec and $\otimes$.

First, let vec : $\mathbb{R}^{m\times n} \to \mathbb{R}^{mn}$ be the operator which vectorizes a matrix

by sequentially stacking its columns.

$$A = \begin{bmatrix} a_1 & a_2 & \cdots & a_n \end{bmatrix} \tag{3.64}$$

$$\text{vec}(A) = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} \tag{3.65}$$

Second, let $\otimes : \mathbb{R}^{m \times n} \times \mathbb{R}^{l \times p} \to \mathbb{R}^{ml \times np}$ be the Kroneker Product:

$$A \otimes B = \begin{bmatrix} a_{11}B & a_{12}B & \cdots & a_{1n}B \\ a_{21}B & a_{22}B & \cdots & a_{2n}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}B & a_{m2}B & \cdots & a_{mn}B \end{bmatrix} \tag{3.66}$$

With $I_n \in \mathbb{R}^{n \times n}$ as the identity matrix, we have

$$D\left[D^{\mathrm{T}}[r]r\right](x) = D^{\mathrm{T}}r(x)Dr(x) + \left(r^{\mathrm{T}}(x) \otimes I_n\right) D\left[\text{vec}\left(Dr\right)\right](x) \tag{3.67}$$

Gauss-Newton Method

Due to the second term in Equation 3.67, computing the Hessian $D^2 f$ is prohibitively expensive for many applications in state estimation. This is because the size of the Hessian scales quadratically with the number of states to be estimated.

To recify this issue, the Gauss-Newton method simply omits the second term, and approximates

$$D\left[D^{\mathrm{T}}[r]r\right](x) \approx D^{\mathrm{T}}r(x)Dr(x) \tag{3.68}$$

It then proceeds through Newton's Method (Algorithm 1) as normal.

Levenberg-Marquardt Method

While Gauss-Newton performs reasonably well, it can be overconfident due to its approximation of the Hessian. This can result in steps that lead to *increases* in the value of the objective function, which is undesirable.

To resolve this issue, Levenberg proposed regularizing the approximation of the Hessian with a single parameter $\lambda \in \mathbb{R}$.

$$D\left[D^{\mathrm{T}}[r]r\right](x) \approx D^{\mathrm{T}}r(x)Dr(x) + \lambda I \tag{3.69}$$

In practice, one changes λ each iteration over the course of the optimization, increasing it if more regularization is needed, and decreasing it otherwise.

Marquardt rediscovered Levenberg's algorithm and proposed scaling λ by the diagonal of the approximated Hessian, which aims to address scaling discrepancies between different coordinates.

$$D\left[D^{\mathrm{T}}[r]r\right](x) \approx D^{\mathrm{T}}r(x)Dr(x) + \lambda \cdot \operatorname{diag}\left(D^{\mathrm{T}}r(x)Dr(x)\right) \tag{3.70}$$

For full details, see [8].

3.3.6 Robust Estimators (Geman-McClure)

In its naive formulation, nonlinear least squares is susceptible to outliers. Outliers occur in many places in state estimation, from wrong correspondences between points, to bad measurements. For example, see Figure 3.1, where the objective is to fit a straight line to noisy points. A single outlier in the top right corner is enough to severely degrade the curve fitting. A standard method to reduce the impact of outliers is robust estimation using M-Estimators.

Mathematically, M-Estimators modify the cost function associated with nonlinear least squares to

$$f(x) = \sum_{i=1}^{n} \rho\left(\frac{1}{2} r_i^{\mathrm{T}}(x) r_i(x)\right) \tag{3.71}$$

$$\rho \,:\, \mathbb{R}^+ \to \mathbb{R}^+ \tag{3.72}$$

One such M-Estimator is the Geman-McClure function with kernel $\kappa > 0$:

$$\rho(x) = \frac{1}{\kappa + x} x \tag{3.73}$$

Figure 3.2 shows that the Geman-McClure function asymptotically approaches 1 as x approaches ∞.

Thus, the Geman-McClure function induces a weight $w(x) = \frac{1}{k+x}$. For small κ, outliers are extremely deweighted. For large κ, outliers are allowed to affect the result much more.

Figure 3.3 shows the weighting effect for various values of κ.

While a closed form solution to the naive least-squares problem exists as $A^{\mathrm{T}}Ax = A^{\mathrm{T}}b$, adding an M-Estimator generally removes the possibility of

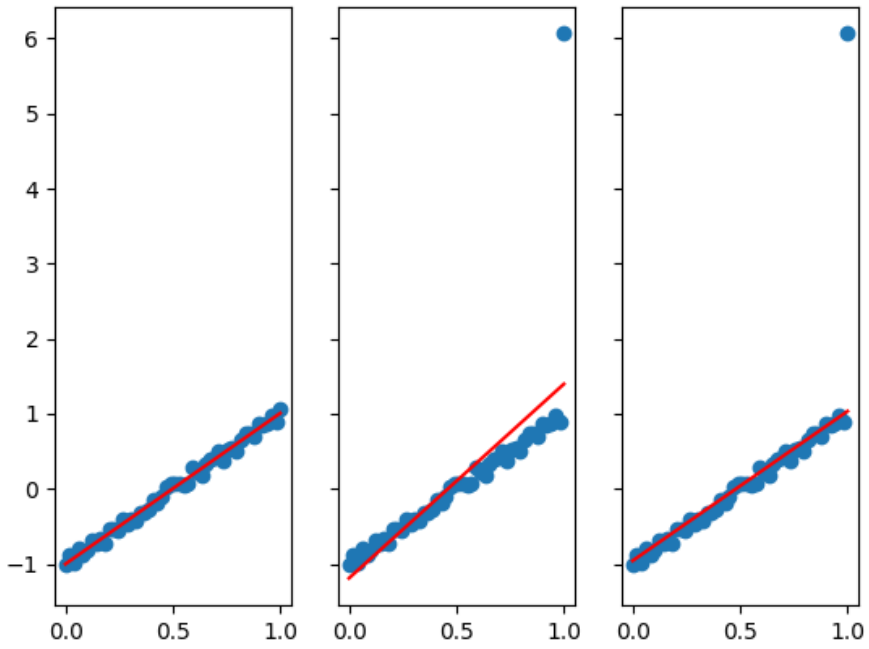


Figure 3.1: Demonstration of outlier rejection scheme; From left to right, naive least squares fitting without outliers, naive least squares fitting with outlier present, robust estimator fitting with outlier present

such a direct path. Hence, we fall back to Newton's method for optimization, and note the following formulations for the first and second derivatives of f:

$$y_i(x) = \frac{1}{2} r_i^{\mathrm{T}}(x) r_i(x) \tag{3.74}$$

$$Df(x) = \sum_{i=1}^{n} D\rho_i(y_i(x))\ Dy_i(x) \tag{3.75}$$

$$Dy_i(x) = r_i^{\mathrm{T}}(x) Dr_i(x) \tag{3.76}$$

$$D^2 y_i(x) \approx D^{\mathrm{T}} r_i(x) Dr_i(x) \tag{3.77}$$

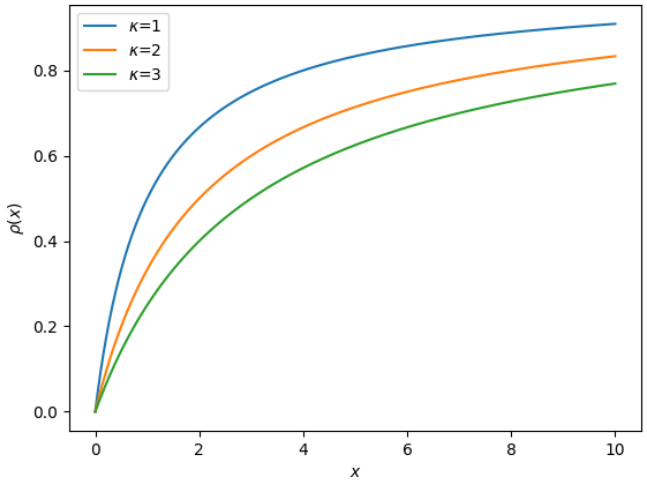


Figure 3.2: Geman-McClure with various kernel values

$$D^2 f(x) = D\left[D^{\mathrm{T}} f\right](x) \tag{3.78}$$

$$= D\left[\sum_{i=1}^{n} D^{\mathrm{T}} y_i(x) D\rho_i(y_i(x))\right] \tag{3.79}$$

$$= \sum_{i=1}^{n} D\left[D^{\mathrm{T}} y_i(x) D\rho_i(y_i(x))\right] \tag{3.80}$$

$$= \sum_{i=1}^{n} \left[D^2 y_i(x) D\rho_i(y_i(x)) + D^{\mathrm{T}} y_i(x) D^2 \rho_i(y_i(x)) D y_i(x)\right] \tag{3.81}$$

$$\approx \sum_{i=1}^{n} \left[D^{\mathrm{T}} r_i(x) D r_i(x) D\rho_i(y_i(x)) + D^{\mathrm{T}} y_i(x) D^2 \rho_i(y_i(x)) D y_i(x)\right] \tag{3.82}$$

$$= \sum_{i=1}^{n} \left[D\rho_i D^{\mathrm{T}} r_i D r_i + D^2 \rho_i D^{\mathrm{T}} y_i D y_i\right] \tag{3.83}$$

Table 3.1 displays various M-Estimators and their respective first and second derivatives.

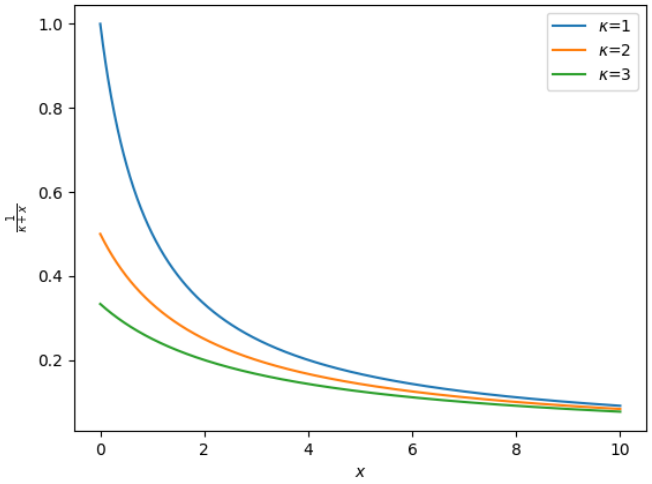


Figure 3.3: Geman-McClure induced weighting

Table 3.1: Comparison of various m-estimators

Name	ρ	$D\rho$	$D^2\rho$	Bounded?
Geman-McClure	$\frac{y}{y+\kappa}$	$\frac{\kappa}{(y+\kappa)^2}$	$-\frac{2\kappa}{(y+\kappa)^3}$	Yes
Welsch	$1-\exp\left(-\frac{y}{\kappa}\right)$	$\frac{1}{\kappa}\exp\left(-\frac{y}{\kappa}\right)$	$-\frac{1}{\kappa^2}\exp\left(-\frac{y}{\kappa}\right)$	Yes
Cauchy	$\log\left(\frac{y+\kappa}{\kappa}\right)$	$\frac{1}{y+\kappa}$	$-\frac{1}{(y+\kappa)^2}$	No

Chapter 4

LiDAR-Inertial Odometry Foundations

4.1 LiDAR

LiDAR, short for LIght Detection and Ranging, is a key sensor present in robotics today. The core technology that enables LiDAR is precise measurement of time, specifically the time taken for an emitted light pulse to return to the sensor. Because the speed of light is constant, we are able to deduce the range of the object off of which the light reflected. Because of the high power requirements associated with LiDAR to overcome the background noise from other light sources, most LiDAR sensors do not fire beams in all directions at the same moment in time. Rather, LiDAR sensors fire a small number of concentrated beams, and then quickly shift the directions of those beams over time in order to obtain greater coverage of a given area.

There are two main methods for moving the beam direction over time. The first is through mechanically rotating the lasers themselves. This is the more common and established approach. One popular vendor in this area is Velodyne (presently acquired by Ouster). The Velodyne Puck has 16 lasers stacked vertically, with varying degrees of pitch. The lasers are rotated around 10-20 Hz for a 360°coverage of the space. Unfortunately, these systems are also reliant on reliable spinning, and need continuous calibration for optimal performance.

The other method for movement of the beams requires no moving parts. Solid state LiDAR sensors, such as those manufactured by Livox (see Figure

4.1), rely on digital technology to manipulate the direction of output of the beams. Table 4.1 displays the various characteristics of two of the most popular sensors.

Figure 4.1: An example of a solid state LiDAR; Credit: Livox

Table 4.1: Comparison between LiDAR sensors

	Velodyne Puck	Livox AVIA
Points/sec	**300,000**	240,000
Cost	$4,600	**$1,600**
Horizontal FOV	**360°**	70.4°
Vertical FOV	30°	**77.2°**

4.2 The Inertial Measurement Unit

An inertial measurement unit is a high rate sensor that is responsible for the short-term state estimation in LiDAR-inertial odometry. It outputs measured angular velocity, $\tilde{\omega}$ and specific force $\tilde{a}$ (the sum of linear acceleration

and the normal force in the IMU frame r). An example of an IMU is shown in Figure 4.2.

Figure 4.2: An Inertial measurement unit (Epson G330)

We choose to model the angular velocity, ${}^{r}\omega$ and linear acceleration ${}^{r}a$ in the body frame with two forms of corruption. The first is a slowly varying bias term, b_ω, b_a, i.e. a random walk associated with a Wiener process [10]. The second source of corruption is normally distributed white noise, η_ω, η_a. This choice of modeling stems from [11], which was leveraged with great success.

In summary

$$\tilde{\omega} = {}^{r}\omega + b_\omega + \eta_\omega \tag{4.1}$$

$$\tilde{a} = {}^{r}a - {}^{r}R_{w}{}^{w}g + b_a + \eta_a \tag{4.2}$$

where

$$\dot{b_\omega} \sim \mathcal{N}(0, \Sigma_{b_\omega}) \tag{4.3}$$
$$\eta_\omega \sim \mathcal{N}(0, \Sigma_{\eta_\omega}) \tag{4.4}$$
$$\dot{b_a} \sim \mathcal{N}(0, \Sigma_{b_a}) \tag{4.5}$$
$$\eta_a \sim \mathcal{N}(0, \Sigma_{\eta_a}) \tag{4.6}$$

Because of the bias associated with the IMU, it is impossible to precisely use the IMU for dead reckoning over long periods of time, hence the value of a LIDAR sensor.

Chapter 5

Tightly Coupled LiDAR-Inertial Odometry

Our contribution can be described in a high level architecture shown in 5.1. It begins the following key assumptions:

1. IMU messages come in at a continuous stream with no dropout
2. LiDAR and IMU messages are time-synchronized
3. IMU message rate is > 200 Hz
4. LiDAR message rate is > 10 Hz

At the outset, we leverage the IMU sensor, fused together with the current pose estimate to propagate our state forward at a high rate.

The measurement model of the IMU from Equation 4.1 helps us understand how to do so. Suppose that our robot's pose and body velocity is known at time t^k. Then suppose that we recieve an IMU measurement, $(\tilde{\omega}, \tilde{a})$ at time t^{k+1}. Then we have in the continuous sense where $*$ represents the quaternion group operator:

$$
\begin{aligned}
{}^{w}\dot{R}_r &= \frac{1}{2}{}^{w}R_r * \begin{bmatrix} 0 \\ {}^{r}\omega \end{bmatrix} && (5.1)\\
{}^{w}\dot{p}_r &= {}^{w}v = {}^{w}R_r{}^{r}v && (5.2)\\
{}^{r}\dot{v} &= {}^{r}a && (5.3)\\
& && (5.4)
\end{aligned}
$$

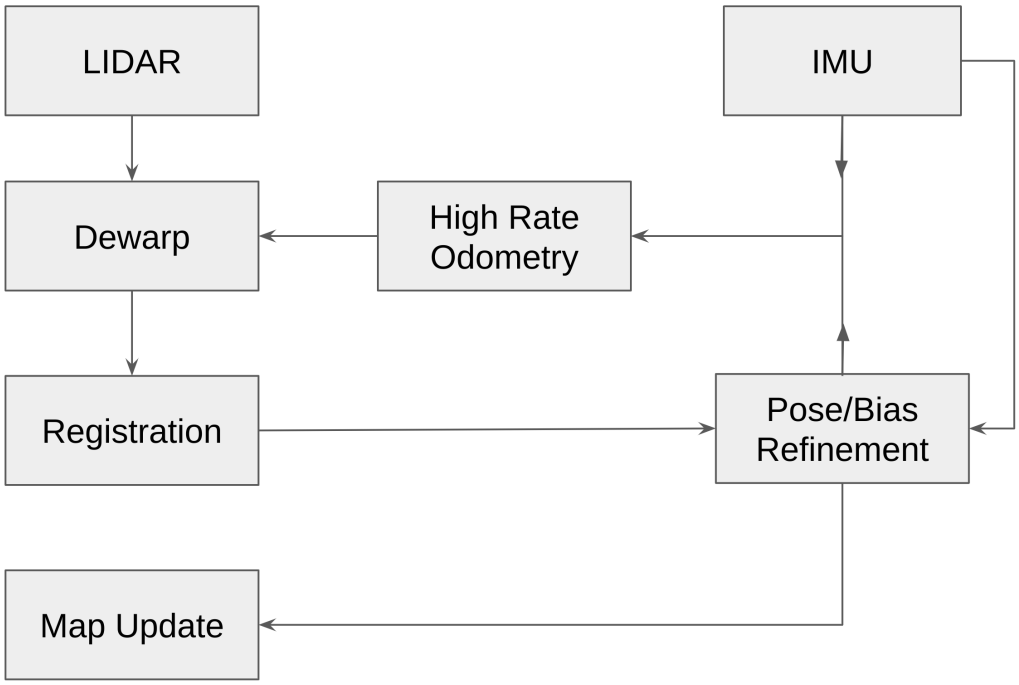


Figure 5.1: High-level architecture for LIO system

We can discretize the continuous equations above into a first order forward Euler update using the 3D pose exponential map from Equation 3.32:

$$\begin{aligned}
\Delta t &= t^{k+1} - t^k && (5.5)\\
{}^r v^{k+1} &= {}^r v^k + {}^r a^{k+1} \Delta t && (5.6)\\
&= {}^r v^k + \tilde{a}^{k+1} + {}^r R_w^k \, {}^w g - b_a && (5.7)\\
({}^w R_r, {}^w p_r)^{k+1} &= ({}^w R_r, {}^w p_r)^k * \operatorname{Exp}\left(\begin{bmatrix} {}^r\omega^{k+1} & {}^r v^{k+1} \end{bmatrix}^{\mathrm{T}} \Delta t \right) && (5.8)\\
&= ({}^w R_r, {}^w p_r)^k * \operatorname{Exp}\left(\begin{bmatrix} \tilde{\omega}^{k+1} - b_\omega & {}^r v^{k+1} \end{bmatrix}^{\mathrm{T}} \Delta t \right) && (5.9)
\end{aligned}$$

The approach above is taken directly from GTSAM [8]. If extra computation is available, then higher order methods may be used. These include Runge-Kutta-Munthe-Kass methods, which we will not cover in this document but [4] is an excellent introduction. Other more efficient methods include Adams-Bashforth; however, these require constant time intervals between measurements which may not hold in practice.

5.1 Dewarping

Because of the nature of mechanical spinning LiDARs, individual points register at different times. This disparity causes motion distortion if the robot is in motion, and this distortion must be corrected for the best possible odometry. Fortunately, each point return from the LiDAR contains a timestamp, which we can use to correct the distortion. We will discuss two methods of conducting the distortion compensation, or dewarping.

5.1.1 Constant Velocity Based Model

The first method makes the assumption that the twist of the vehicle in the body frame ${}^{r_k}\xi = \begin{bmatrix} {}^{r_k}\omega & {}^{r_k}v \end{bmatrix}$ is known at some time t_k. Then, suppose we receive a lidar point ${}^{r_j}p$ in the body frame of the vehicle at time t_j. We further assume that $t_j - t_k$ is small enough to be modeled by a constant twist.

Then the location of the point given in the frame r_k is:

$${}^{r_k}p = \operatorname{Exp}\left({}^{r_k}\xi \cdot (t_j - t_k)\right) {}^{r_j}p \tag{5.10}$$

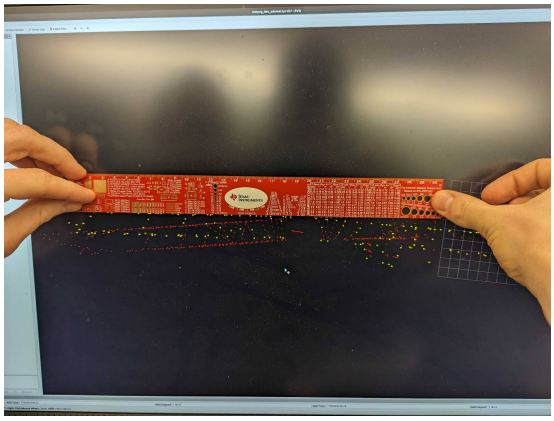


Figure 5.2: Skewed point cloud with constant velocity model

This method is simple, and does not require extremely accurate time synchronization between the IMU and the LiDAR sensor. It works rather well for small accelerations, but struggles significantly when large ones occur, which invalidate the constant velocity assumptions. Such invalidation appeared in our testing during an aggressive turn of a ground vehicle robot in a narrow hallway, as shown in 5.2. For this reason, we switched to a more accurate model based on interpolation.

5.1.2 Interpolation Model

The second method for dewarping relies on a high rate odometry stream (usually provided by IMU fusion). We store this stream of odometry (pose and twist) measurements in a buffer for later access.

Suppose again that we receive a point at time t_p, ${}^{r_p}p$ from a scan at time t_s.

We find the odometry measurements ${}^{w}T_{r_i}, {}^{w}T_{r_j}$ that are immediately before and after t_p respectively. Then, we have

$$ {}^{r_s}p = {}^{r_s}T_w * \text{interpolate}\left({}^{w}T_{r_i}, {}^{w}T_{r_j}, \frac{t_s - t_i}{t_j - t_i}\right) \tag{5.11} $$

Algorithm 2 describes the full interpolation scheme.

Algorithm 2 Interpolation-based dewarping

1: **function** DEWARP($\{{}^{r_i}p_i\}, \{{}^{w}T_j\}$)
2: **for all** $\{{}^{r_i}p_i\}$ **do**
3: Find $k = \sup_j t_j < t_i$, $l = \inf_j t_j > t_i$.
4: $\Delta \leftarrow \frac{t_i - t_k}{t_l - t_k}$
5: Use Equation 5.11 to dewarp p_i
6: **end for**
7: **end function**

5.1.3 Combining Dewarping Methods

While the interpolation model discussed above works well, it relies on having the timestamp for each lidar point contained between two already existing odometry measurements. In reality, this may not be the case due to lag between the IMU and LiDAR sensor. To resolve this issue, it becomes necessary to choose between the following options:

1. Wait for additional high rate odometry measurements to arrive (increases lag time, more accurate)

2. Use the constant velocity model to account for any lidar points that fall outside of high rate odometry measurements (less lag time, less accurate)

5.1.4 Parallelization

It should be noted that the dewarping schemes presented here are highly parallelizable. It is highly advantageous to parallelize this section so that as much time as possible is left for scan-to-map registration. However, the extent to which it should be parallelized depends on the processing power available. For our purposes, C++ 17 provides a native interface to parallel algorithms within its standard library, which is built off of Intel's oneAPI Threading Building Blocks (oneTBB) library.

5.2 Scan to Map Registration

The registration problem seeks to compute the optimal transform between two point clouds, one of them being the most recent LiDAR scan. More

formally, let ${}^aX, {}^bY$ be two sets of points, not necessarily of equal cardinality. Then we seek to find ${}^aT_b$ such that $\sum_i ||{}^aT_b {}^bY - {}^aX||$ is minimized. In the most general case, there are two different categories for the other point cloud: the first being the previous scan. Unfortunately, pure scan-to-scan matching can be difficult due to the sparse nature of the LiDAR point clouds generated by each scan. At high speeds, perceptual aliasing (the idea that two different places can look the same under a given sensor) can significantly degrade such estimation. Because of these issues, we opt for the second approach: scan-to-map registration. A map contains an accumulation of points from previous scans, not limited to just one before. Because of this accumulation, the map contains much more information, decreasing the likelihood of perceptual aliasing and generally improving results. Unfortunately, the map also encodes implicit transform assumptions between previous LiDAR scans. If any of those previous transforms are imprecise, the map will be degraded, and the registration of the current scan will be affected. However, given an IMU and a sufficiently good registration algorithm, we can significantly decrease the risk associated with scan-to-map registration, as we will demonstrate later.

All registration algorithms compute matches, or correspondences between elements of the two point clouds. Some algorithms, like Fast Global Registration [36], never change these correspondences throughout the optimization. Unfortunately, if too many bad correspondences are found, this can lead to bad registration. In contrast, our method recomputes correspondences throughout each step of the optimization, safely ensuring better optimality. Methods like our own are descendents of Iterative Closest Point (ICP) [2], which we willl now discuss in more detail. Iterative closest point [2] represents the conventional approach.

In order to perform good scan to map registration, one must account for the presence of outliers. Outlier techniques such as Graduated Non Convexity are too slow to apply to the registration problem, and so we must use m-estimation techniques. Because of the changing environment, any m-estimator kernel chosen must be able to adapt online. In order to do so, it is useful to measure the norm of the ICP-derived scan-to-map registration [1], [5].

Initialization of the scan registration is very important. This is because scan registration is an iterative method. Like any other iterative method, scan registration is very important. In order to get an initial guess of the transform, we use the IMU.

5.3 Iterative Closest Point

As with any iterative algorithm, the first step in the algorithm is a good initial guess. In the case of pure LIDAR Odometry, typical methods employ the constant velocity model. Unfortunately, such approaches quickly fail when fast disturbances influence the motion of the robot. This was especially true in our case, where the slightest bump in the road would invalidate our constant velocity assumption. Fortunately, the IMU provides a high rate odometry output, which can be used to establish an initial guess with much greater precision. This was our approach.

For each iteration of ICP, matches are computed, followed by computation of residuals and then minimizing said squared residuals with respect to some ΔT. It is important to note the minimization described above is also generally a nonlinear and iterative procedure, except for the special case of pure point-to-point matching. In our case, it is more valuable to recompute matches and step in the right direction, rather than proceed to convergence with wrong correspondences. Thus, only one step was taken in the inner minimization routine to compute ΔT.

These ΔT are then chained together to form the final registration transform estimate. ICP terminates when appropriate convergence criteria are met. Algorithm 3 gives the general procedure.

Algorithm 3 Iterative closest point

function ITERATIVE CLOSEST POINT($^aX, ^bY; \delta, N$)
    $i \leftarrow 0$
    while $||\operatorname{Log}(\Delta T)|| > \delta$ **and** $i < N$ **do**
        compute matches $\left({}^aX, {}^bY\right)$
        compute residuals from matches
        compute ΔT that minimizes the sum of squares of residuals
        ${}^bY \leftarrow \Delta T * {}^bY$
        ${}^aT_b \leftarrow \Delta T * {}^aT_b$
        $i \leftarrow i + 1$
    end while
end function

Figure 5.3 displays various types of features that can be used for the matching process in iterative closest point.

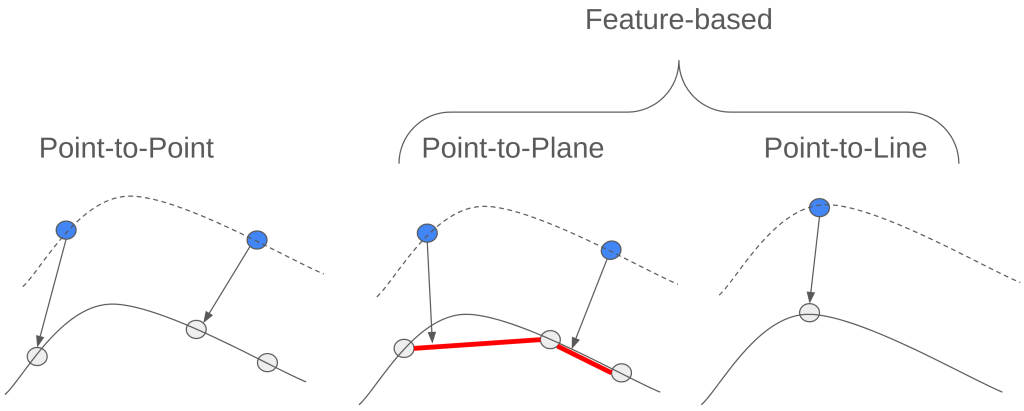


Figure 5.3: From left to right: point, planar, and edge features

The easiest method is to simply match points from one scan to points from other scans. The typical metric to accomplish this is defined by the residual

$$(x, y) = x - y \tag{5.12}$$

However, this method ignores the fact that LiDAR points usually represent the detection of a surface at discrete points where the rays of light intersect said surface. Because of this, point-to-point metrics may overconstrain the estimated motion of the vehicle.

In order to rectify this issue, researchers introduced the point-to-plane metric. Given three points, p_1, p_2, p_3 that lie on a plane, we seek to minimize the orthogonal distance from point x to the plane. It is important to note that p_1, p_2, p_3 should not be colinear, and should not be too close togther, as this will induce unstable computation of the plane normal.

The point-to-plane residual is:

$$r_p(x, p_1, p_2, p_3) = \frac{(x - p_1)^{\mathrm{T}} \left[(p_1 - p_2) \times (p_1 - p_3)\right]}{||(p_1 - p_2) \times (p_1 - p_3)||} \tag{5.13}$$

Finally, as seen in the middle and right sections of Figure 5.3, large surface curvature can make planar approximations rather poor. To this end, the point-to-line residual seeks to minimize the orthogonal distance from a point to the line formed by the area of highest curvature of a given surface. Given

two points, e_1, e_2 on the line, its residual is

$$r_e(x, e_1, e_2) = \frac{||(x - e_1) \times (x - e_2)||}{||e_1 - e_2||} \tag{5.14}$$

Again, e_1 and e_2 should not be too close together, otherwise numerical instability may occur.

The natural next question is how one may discover planar and edge points. This is the topic of the next section.

5.3.1 Extracting Edge and Planar Points

In the ideal case with points uniformly distributed across the LiDAR scan, feature extraction consists of selecting nearest neighbors of any given point, then classifying as an edge or planar point by a metric such as Principle Component Analysis [12], [35].

Unfortunately, points from most LiDAR sensors are not uniformly distributed across the space of the scan. Instead, they tend to lie on lines that are radially distributed across the space as shown in Figure 5.4.

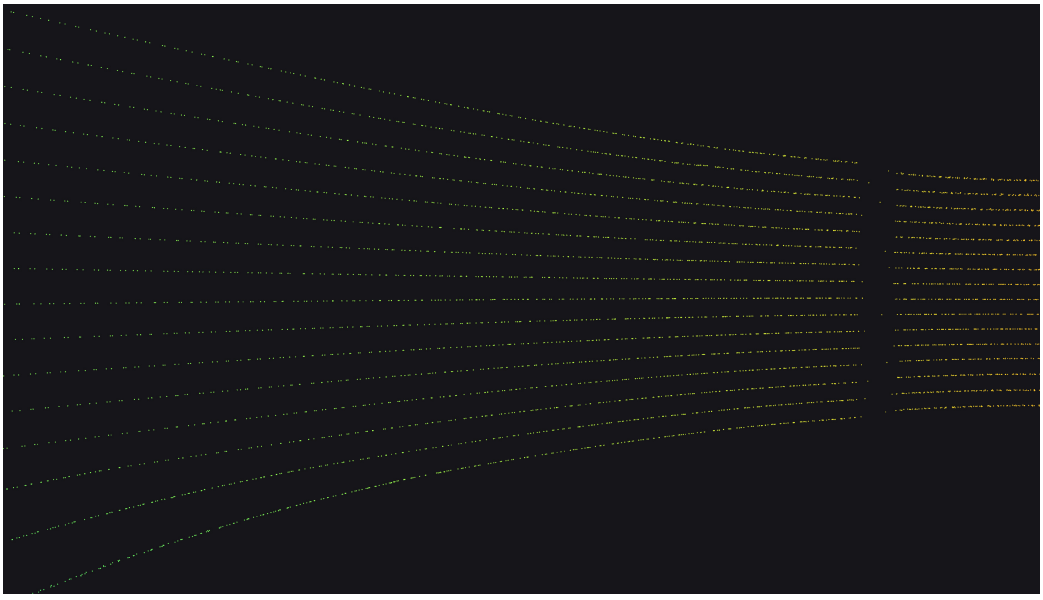


Figure 5.4: Scan generated by a VLP-16 LiDAR sensor

The result is that naive nearest neighbor collection would only return points on the same scan line, thus colinear and destroying any hope of meaningful information.

LiDAR Odometry and Mapping [33] rectified this issue by establishing much more robust criteria for edge and planar point detection. The approach considers each scan line individually, with a sliding window of points within the line evaluated for notions of curvature.

A few special cases exist:

1. If the plane fitted by a point is close parallel to the direction of the scan itself, this plane is unreliable

2. If two feature points lie in the same ray from the LiDAR, then we only select the closest point. This prevents potential occlusions.

Unfortunately, feature extraction is a hard problem that is very dependent on the environment of choice. For this reason, we opted to utilize a point-to-point metric that significantly simplified our approach.

5.3.2 Voxel Map

The purpose of a voxel map is to establish correspondences across the point clouds. Simply speaking, each voxel contains a number of points, and is associated with a hash. In data association, a given point, along with the dimensions of each voxel cell, is computed to belong to a particular voxel at some coordinates in $\mathbb{N}^3$. Then a hash is computed from those coordinates to access the voxel's points. Looping through each of the points in the voxel establishes the correspondence.

In order to do fast data association, we need to optimize for the hashing across voxels. Hashing typically consists of a tradeoff between collisions (different voxels have the same hash) and speed.

A fast but high collision voxel hash function in 2D would be pure addition:

$$c(x, y) = x + y \tag{5.15}$$

This operation is very fast, but unfortunately will result in the voxels at (0,1) and (1,0) having the same hash. This is very undesirable due to the requirement that the correspondence algorithm loop through more points unnecessarily.

Our hash function of choice is the Cantor function:

$$c(x, y) = \frac{(x + y) * (x + y + 1)}{2} + y \tag{5.16}$$

While more complex than the simple addition hash, it guarentees the absence of collisions, which is very desirable.

Table 5.1 shows various values for the Cantor function.

Table 5.1: Various values of the Cantor hash function, c

$c(0,0) = 0$	$c(0,1) = 2$	$c(0,2) = 5$	$c(0,3) = 9$
$c(1,0) = 1$	$c(1,1) = 4$	$c(1,2) = 8$	$c(1,3) = 13$
$c(2,0) = 3$	$c(2,1) = 7$	$c(2,2) = 12$	$c(2,3) = 18$
$c(3,0) = 6$	$c(3,1) = 11$	$c(3,2) = 17$	$c(3,3) = 24$

While the Cantor function, $c : \mathbb{N}^2 \to \mathbb{N}$, is only two-dimensional, we can easily extend this to three dimensions with the double cantor function $d : \mathbb{N}^3 \to \mathbb{N}$

$$d(x, y, z) = c(x, c(y, z)) \tag{5.17}$$

We choose to use this function because it eliminates the entire possibility of collisions.

In summary, our approach is the following.

Algorithm 4 Finding the voxel coordinates

function GET VOXEL COORDINATES($p \in \mathbb{R}^3, \phi \in \mathbb{R}^3, \nu \in \mathbb{R}^3, \epsilon \in \mathbb{N}$)
  $\bar{p} \leftarrow \text{int}\left(\frac{p-\phi}{\nu}\right)$
  coordinates $\leftarrow \text{uint}\left(\bar{p} + \epsilon\right)$
end function

Given a point $p \in \mathbb{R}^3$, the center of the map ϕ, the voxel dimensions ν, and the large number ϵ we use Algorithm 4 to calculate the coordinates of the associated voxel.

In our implementation, we kept the voxel grid in the world frame, so $\phi = 0$, and we set $\nu = \begin{bmatrix}1.0 & 1.0 & 1.0\end{bmatrix}^{\mathrm{T}}$, and $\epsilon = 50,000$.

Once the voxel hash was calculated from the double Cantor hash function, we performed the following insertion as demonstrated in Algorithm 5.

Algorithm 5 Insert into voxel grid

```
function INSERT POINT(p ∈ ℝ³)
    Compute hash associated with voxel
    if voxel exists then
        if size of voxel is less than maximum size then
            append point to voxel
        end if
    else
        Create voxel with point
    end if
end function
```

5.3.3 Outlier Rejection

Because of our point-to-point metric, we needed a strong outlier rejection approach to ensure good registration.

Our method of choice was transforming the registration problem into one that utilized M-Estimator techniques. Specifically, we selected the Geman-McClure kernel for its strong outlier rejection properties. The kernel, κ, was set to a third of the standard deviation of a sliding window of maximum point distances based on previous successful registrations. Our approach was based off of [25].

5.3.4 Summary

In short, we utilized point-to-point matching along with an adaptive kernel for robust registration.

Chapter 6

Pose Refinement

6.1 Building the Bayes Net

Throughout this section, we will construct the Bayes net that lies at the heart of our pose refinement step. The Bayes net encodes causality into a graphical structure, which is directed and acyclic in our case. We refer to Figure 6.1 for the entire construction process.

6.1.1 State Variables

The states we will consider are poses, world velocities, and IMU biases. Specifically, take two of each variable, which corresponds to the state of the robot at two discrete points in time. The first is given by x_0, v_0, b_0, while the second by x_1, v_1, b_1. Figure 6.1a displays the structure.

6.1.2 Bias Propagation

As previously stated, we assume that the bias of the IMU evolves according to a random walk. Then b_0 causes b_1. This is shown in Figure 6.1b by a directed arrow from b_0 to b_1.

6.1.3 Motion

We express the acceleration and angular velocity of the robot from time 0 to time 1 using an additional variable, u. Due to the motion update equations,

it is clear that x_0, v_0, and u cause x_1 and v_1. We refer to Figure 6.1c for visualization.

6.1.4 IMU

We model the IMU using terms previously stated. It is easy to see that x_0, b_0, and u cause z_I. Additionally, due to the Coriolis Effect, v_0 also causes z_I, as shown in 6.1d.

6.1.5 LiDAR

A LiDAR registration fundamentally meausures the relative transform between two poses. To this end, x_0 and x_1 cause z_L. See Figure 6.1e for the complete Bayes net for LiDAR-inertial odometry.

6.2 Forming the Cost Function

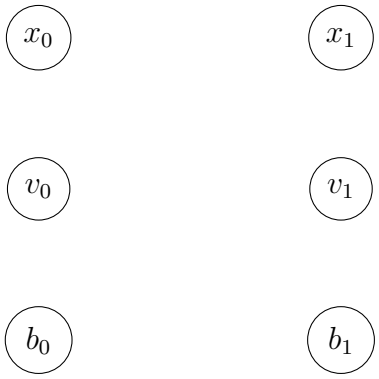
We will now take the Bayes net and factor it into its individual components. Let $S = \{x_0, v_0, b_0, x_1, v_1, b_1\}$ be the set of states we wish to estimate. We can factorize the joint probability of all variables using the causal information encoded in the Bayes net from Figure 6.1. This factorization is as follows

$$
\begin{aligned}
p(S, u, z_I, z_L) = p(x_0) & \quad (6.1)\\
p(v_0) & \quad (6.2)\\
p(b_0) & \quad (6.3)\\
p(x_1|x_0, v_0, u)p(v_1|x_0, v_0, u)p(z_I|x_0, v_0, u)p(u) & \quad (6.4)\\
p(z_L|x_0, x_1) & \quad (6.5)
\end{aligned}
$$

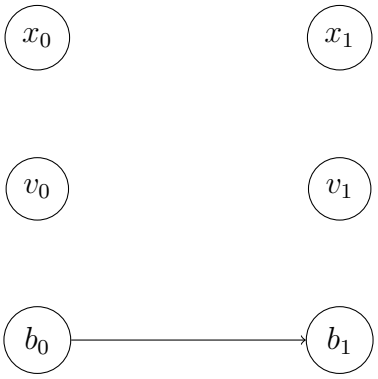
We seek to find the most likely S that explains the IMU and LiDAR measurements. In other words, we can formulate an optimization problem

$$S^\star = \arg\max_S p(S, u, z_I, z_L) \quad (6.6)$$

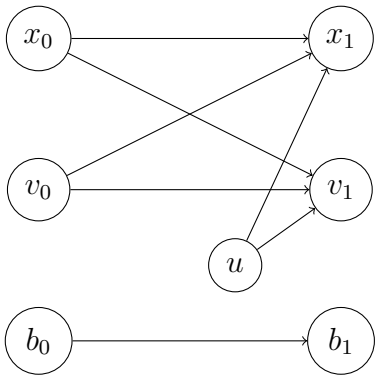
Note that $\log : \mathbb{R}^+ \to \mathbb{R}$ is monotonically increasing; i.e. $x < y \Rightarrow$



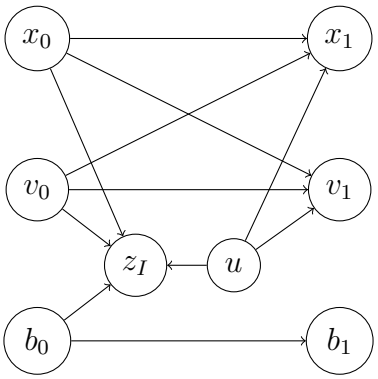
(a) State variables



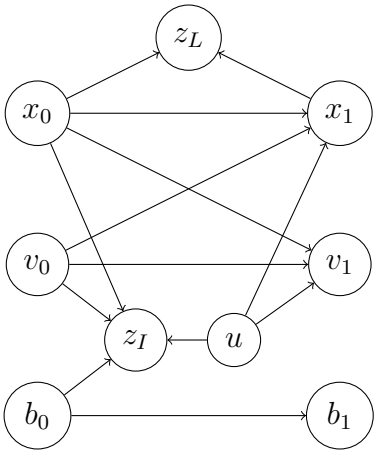
(b) Bias propagation



(c) Motion



(d) IMU



(e) LiDAR

Figure 6.1: Building the Bayes net for LiDAR-inertial odometry

$\log(x) < \log(y)$. Thus,

$$S^\star = \arg\max_S p(S, u, z_I, z_L) \tag{6.7}$$
$$= \arg\max_S \log(p(S, u, z_I, z_L) \tag{6.8}$$
$$\tag{6.9}$$

We now denote $\text{logp}(x) = \log(p(x))$. Because $\log(xy) = \log(x) + \log(y)$, we can transform the factorization from products into sums:

$$S^\star = \arg\max_S f(S) \tag{6.10}$$
$$f(S) = \text{logp}(x_0) \tag{6.11}$$
$$+ \text{logp}(v_0) \tag{6.12}$$
$$+ \text{logp}(b_0) \tag{6.13}$$
$$+ \text{logp}(x_1|x_0, v_0, u) + \text{logp}(v_1|x_0, v_0, u) + \text{logp}(z_I|x_0, v_0, u) \tag{6.14}$$
$$+ \text{logp}(z_L|x_0, x_1) \tag{6.15}$$

Note that we have dropped the $p(u)$ term because it is constant for all values of S.

6.2.1 Normal Implies Nonlinear Least Squares

By assuming a random variable is normally distributed, we obtain a powerful result:

$$\text{logp}(x) \ \alpha \ \frac{1}{2} (x - \mu)^{\mathrm{T}} \Sigma^{-1} (x - \mu) \tag{6.16}$$
$$= \frac{1}{2} r^{\mathrm{T}}(x) r(x), \quad r(x) = \Sigma^{-1/2} (x - \mu) \tag{6.17}$$

Then we have formulated nonlinear least squares, which is a familiar problem in optimization. Because of this simplification, we assume normally distributed noise across our probability densities [8].

6.2.2 Probability of 3D Pose

Given a normal distribution of mean, $y \in \text{SE}(3)$, and covariance, $\Sigma \in \mathbb{R}^{6\times6}$, the random variable $X \in \text{SE}(3)$ is given by

$$\text{Log}\left(y^{-1}X\right) \sim \mathcal{N}(0, \Sigma) \tag{6.18}$$

6.2.3 Probability of 3D Velocity

The probability of a 3D velocity is a standard quantity computed via a normal distribution.

6.2.4 Probability of Imu Bias

The gyroscope bias and accelerometer bias are each normally distributed.

$$b_a \sim \mathcal{N}\left(\mu_{b_a}, \Sigma_{b_a}\right) \tag{6.19}$$

$$b_\omega \sim \mathcal{N}\left(\mu_{b_\omega}, \Sigma_{b_\omega}\right) \tag{6.20}$$

6.2.5 Fixed-lag Smoothing

While the Bayes net in Figure 6.1e only displays two frames, additional frames may be sequentially added to the right of frame 1. Unfortunately, naively appending frames for any reasonable amount of time will soon result in solutions that are too slow for real-time performance. To mitigate this issue, we utilize a fixed-lag smoothing approach, which keeps a sliding window of N frames. The oldest states are towards the rear of the smoother, while the newest states are added to the front. The smoother is "fixed" in the sense that the size of the window is constrained to be less than some maximum number of frames. When the window is full and another frame must be added, the oldest frame is marginalized out and added to the rear states as a prior factor. This marginalization represents the unrecoverable loss of information at the gain of real-time performance. Indeed, the Kalman Filter can be shown as equivalent to a fixed lag smoother of unitary window size.

Specifically, our fixed-lag smoother is a window of poses, velocities, and IMU biases at times corresponding to individual LiDAR scans. Improvements can be made to the fixed-lag smoothing approach by means of keyframes, which represent scans with new information, usually from significant motion of the robot. We did not implement keyframes in our approach due to the desire for simplicity and time constraints.

6.2.6 Factor Graphs

We can display the cost function from Equation 6.10 in graphical terms through a factor graph. Formally, a factor graph is a bipartite graph consisting of two types of nodes. The first type is variable nodes, which encode the state of the robot we wish to estimate. The second type is factor nodes, which represent cost functions associated with variable nodes. The only edges are between variable and factor nodes.

The factor graph has useful properties for incremental smoothing and mapping. However, in our case, we do not need the incremental smoothing and mapping procedure [8], [15] because we rely on fixed-lag smoothing. We simply use the factor graph to represent a cost function consisting of a number of residual blocks for evaluation.

Option 1: Relative Pose Factor

Figure 6.2 shows our initial formulation for the optimization. It consists of prior factors attached to the first pose, velocity in the world frame, and IMU bias of the robot. Each successive LIDAR scan is represented by another frame of pose, velocity, and IMU bias. The frames are connected sequentially by relative pose factors, which are strictly taken from the scan-to-map registration technique discussed earlier.

Let T_k represent the pose of the robot at time t_k. Let ${}^kL_{k+1}$ represent the measured relative pose from LiDAR registration. Then the residual of the relative pose factor is

$$r(T_k, T_{k+1}) = || \left({T_k}^{-1} \circ T_{k+1} \right) \boxminus {}^kL_{k+1} ||^2_\Sigma \tag{6.21}$$

The factors follow directly from the rows of Equation 6.10.

Ideally, this would be the method of choice, as LIDAR does not provide any global constraint over the course of the odometry. Unfortunately, this method was very unstable, and did not perform well, leading to oscillatory behavior in the pose estimation. Attempts to tune uncertainty parameters associated with this formulation also did not yield satisfactory results.

Option 2: Global Pose Factor

The second formulation, shown in Figure 6.3 replaced the relative pose factors present in the first case with absolute pose factors, also taken from the

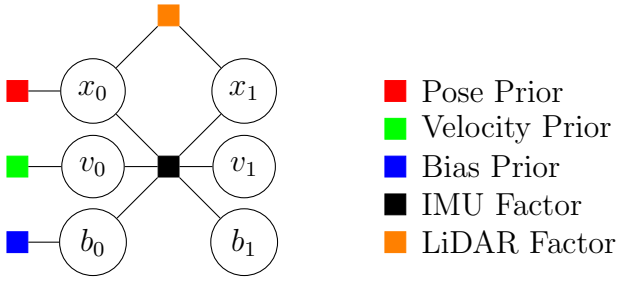


Figure 6.2: Factor graph For LiDAR-inertial odometry with relative pose factors

scan-to-map registration. These absolute pose factors are necessarily overconfident, but lead to better performance in the short term.

Let T_k represent the pose of the robot at time t_k. Let ${}^{w}L_k$ represent the measured global pose from LiDAR registration. Then the residual from the global pose factor is

$$r(T_k, {}^{w}L_k) = ||T_k \boxminus {}^{w}L_k||^2_{\Sigma} \tag{6.22}$$

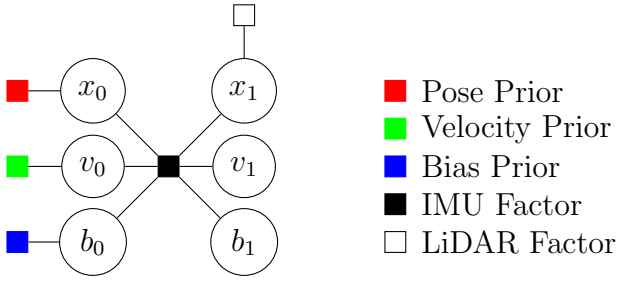


Figure 6.3: Factor graph for LiDAR-inertial odometry with global pose factors

While the reason behind this disparity between the relative and absolute pose factors is still an active question, we present several of our most promising hypothesis:

1. The relative pose constraint provides a constraint along less "dimensions" than an absolute pose constraint

2. The relative pose constraint may not provide total observation of the state space

3. The relative pose constraint may be more sensitive to noise

6.2.7 Preintegrated IMU Factor

Because the previous state and a single IMU measurement determine the next state, bundling many IMU measurements together into one factor means the need to sequentially process (reintegrate) each IMU measurement every time the previous state estimate changes. This can be extremely slow.

Preintegration [11] allows us to accumulate multiple IMU measurements into a single factor as before, but avoid the reintegration. In order to do so, it relies on first-order approximations, which may not be accurate enough if the rate of the IMU is below a certain threshold. However, the discretization of IMU measurements in the first place for integration is already first-order, so performing a higher order numerical integration scheme is most likely unwarrented.

6.3 Map Update

Once the pose refinement is complete, we need to update the map of LiDAR points used for scan registration. Ideally, the map would not drift over time, but we know that this will inevitably happen for odometry systems. As such, we also remove old points from the map; in this way we can reduce the possibility of incorrect loop closures. We refer the reader to the section about Voxel Grids to gain a better understanding of the structure of our map.

In this sense, we are much more tightly coupled than other methods that do not integrate information from the pose refinement into the map update.

Figure 6.4 displays this key difference between loosely coupled and tightly coupled methods.

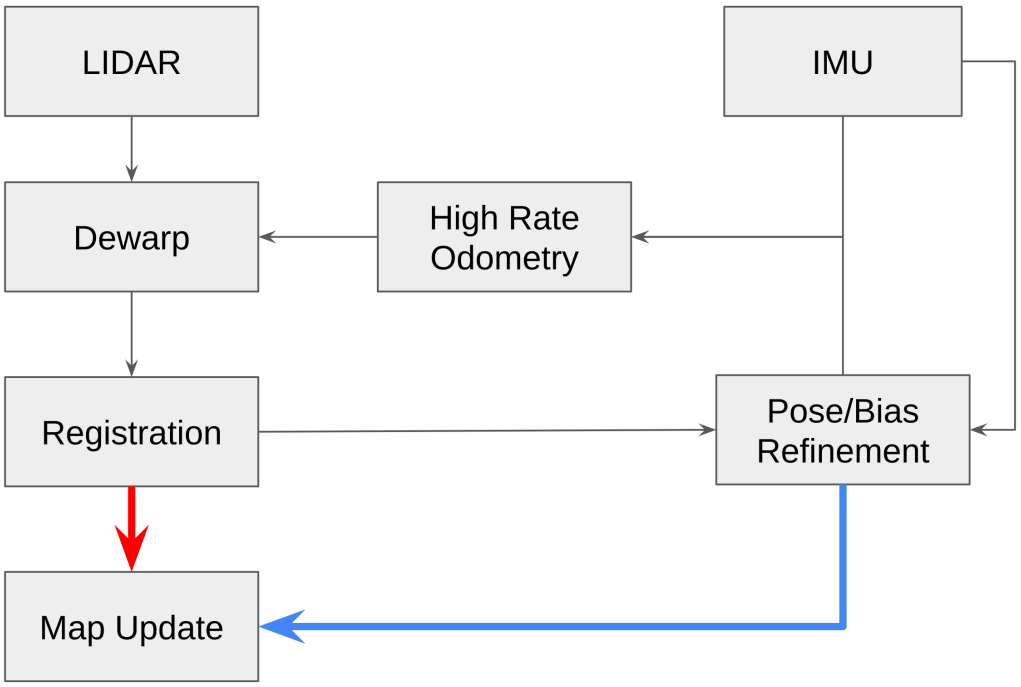


Figure 6.4: Loosely coupled methods (red) take the registration hypothesis for map update. In contrast, tightly coupled methods like ours (blue) integrate information from the pose refinement step

Chapter 7

Results

We display results from our LiDAR-inertial odometry method across two projects. The reason for doing so was to ensure that our algorithm would achieve satisfactory performance across both indoor and outdoor environments. Overall, we met this metric, but also were significantly more robust to failure modes that plagued previous efforts.

7.1 MMPUG

The first project's mission was to drive a small car through narrow corridors indoors at speeds between 2-6 m/s (see Figure 1.5). The car had sudden changes in velocity that made it difficult or impractical for pure LiDAR odometry methods to produce reasonable results. For this reason, our LiDAR-inertial system was needed to ensure optimal state estimation. Figure 7.1 displays the accumulation of individual scans over the length of the trajectory to form a map of the environment. Overall, the figure demonstrates the low drift of our approach, while edges naturally become less sharp as the points are further away from the center of the robot.

Figure 7.2 displays the same environment as Figure 7.1, but from a 3D perspective. The cars to the right of the robot are clearly visible, along with their tires. This view enabled remote driving of the robot with better ergonomics than a simple 2D camera stream provided.

Finally, Figure 7.3 shows the accumulated scans on a larger area of the same hallway, even extending outside to the right of the building. Because of this, we can conclude that our algorithm is able to operate while transitioning

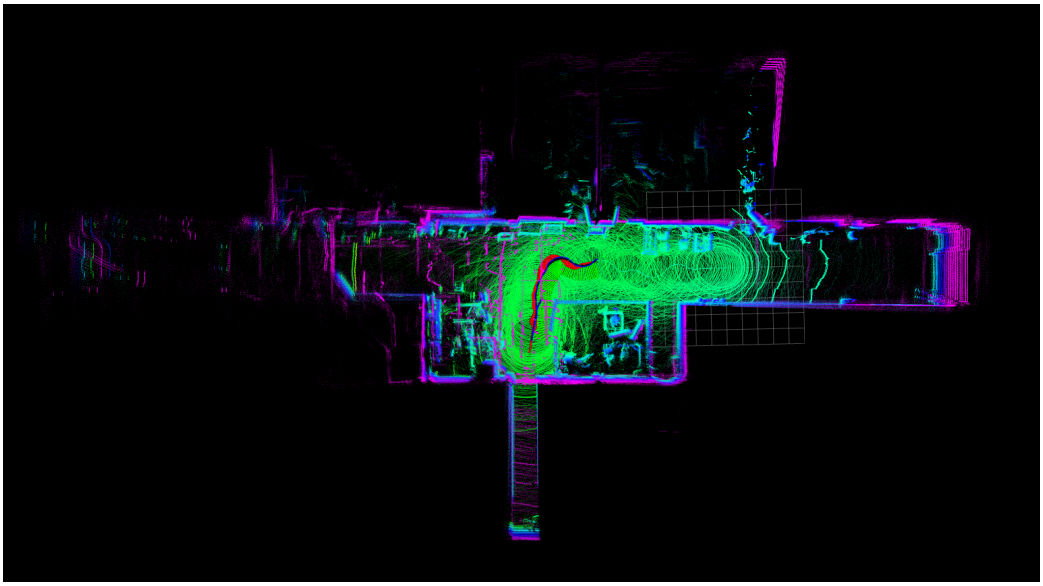


Figure 7.1: MMPUG Results; Credit: Darwin Mick

from traditionally structured to unstructured environments.

7.2 DARPA RACER

The second suite of tests was run in post-processing at real time on data recorded on a Polaris ATV in Penn Hills, PA. The vehicle (displayed in Figure 1.4) was mounted with an Xsens IMU and a Velodyne VLP-32C, then driven at speeds up to 50 miles/hour. Figure 7.4 displays the accumulated LiDAR scans from the estimated robot trajectory, overlayed onto satellite imagery. The overlay was conducted by hand and represents the author's best guess as to the true nature of the robot trajectory. We note that the path traced by the robot closely matches the satellite image, showing that our system has high performance even at high speeds outdoors.

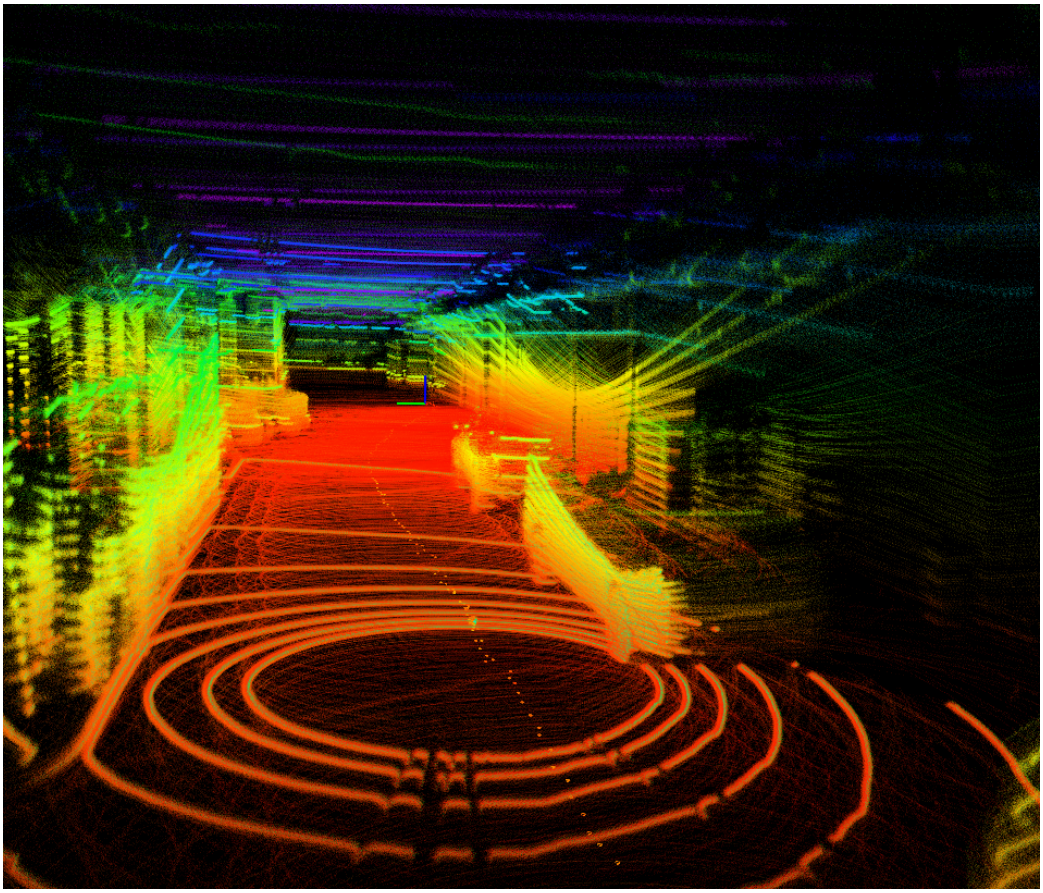


Figure 7.2: 3D MMPUG Results; Credit: Darwin Mick

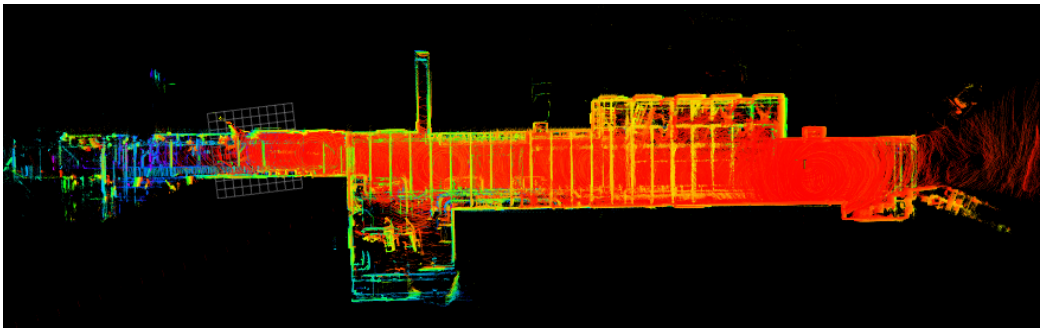


Figure 7.3: Large MMPUG Results; Credit: Darwin Mick

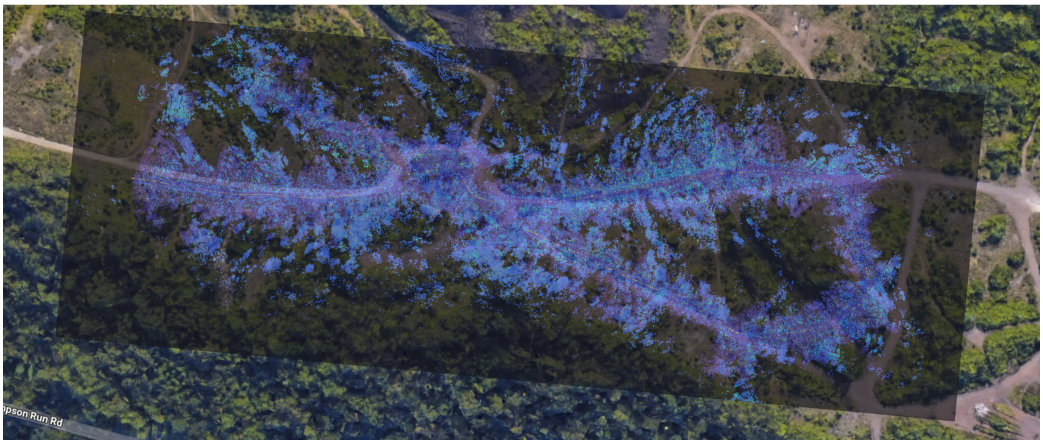


Figure 7.4: Result from DARPA RACER vehicle at Gascola

Chapter 8

Future Work and Conclusion

8.1 Numerical Evaluation

To provide more structured metrics, we plan to compare our estimated trajectory against RTK GPS, which was recorded for several sequences run on the Polaris ATV. Additionally, we plan to provide metrics evaluating the Euclidean distance between the start and end location of the robot, as each sequence ends where it began.

8.2 IMU Dropout Compensation

On DARPA RACER, IMU dropout played a significant factor in state estimation issues. As opposed to variants of the Kalman Filter, preintegration does not natively handle long periods with no measurement update. In order to overcome this issue, we propose the following scheme for IMU dropout shown in Figure 8.1.

$$\Delta t = t_{k+1} - t_k \tag{8.1}$$

$$v_{k+1} = v_k + a_{k+1}\Delta t \tag{8.2}$$

$$x_{k+1} = x_k \operatorname{Exp}\left(\begin{bmatrix}\omega_{k+1} & v_{k+1}\end{bmatrix}^{\mathrm{T}} \Delta t\right) \tag{8.3}$$

$$b_{k+1} = b_k \tag{8.4}$$

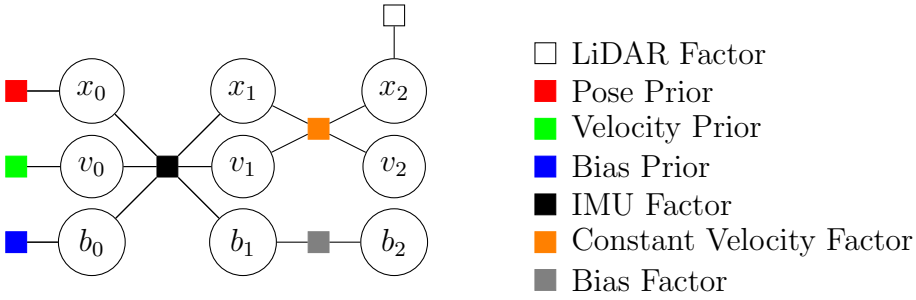


Figure 8.1: Factor graph for LiDAR-inertial odometry with IMU dropout compensation

8.3 Conclusion

In this document, we have presented the necessary fundamentals of LiDAR-inertial odometry. We bagan with a discussion of the nature of LiDAR and IMU sensors. LiDAR is extremely precise but suffers from slow update rates and discrete collection times for individual points. Meanwhile, IMU has a hgih output rate corrupted by a slowly varying bias. Combining LiDAR and IMU together yields a high rate and precise odometry solution. Probability, rotations, and optimization are three indispensible mathematical concepts central to our algorithm. For this reason, we provided a brief background on these areas. Additionally, we highlighted the various methods for point cloud registration, and discussed the use of robust estimators to deweight the impact of potential outliers. Through a factor graph formulation, we can optimize for the most likely set of states over a window of LiDAR scans. Finally, we validated our approach by displaying results across two different robots and environments. Prediction is difficult, but we expect that the concepts outlined in this thesis will provide future readers with the foundational understanding required to contribute in this rapidly developing field.

Bibliography

[1] Jonathan T Barron. "A general and adaptive robust loss function". In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 4331–4339.

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

[3] Paul Busch, Teiko Heinonen, and Pekka Lahti. "Heisenberg's uncertainty principle". In: *Physics reports* 452.6 (2007), pp. 155–176.

[4] Elena Celledoni, Håkon Marthinsen, and Brynjulf Owren. "An introduction to Lie group integrators–basics, new developments and applications". In: *Journal of Computational Physics* 257 (2014), pp. 1040–1061.

[5] Nived Chebrolu, Thomas Läbe, Olga Vysotska, Jens Behley, and Cyrill Stachniss. "Adaptive robust kernels for non-linear least squares problems". In: *IEEE Robotics and Automation Letters* 6.2 (2021), pp. 2240–2247.

[6] Yang Chen and Gérard Medioni. "Object modelling by registration of multiple range images". In: *Image and vision computing* 10.3 (1992), pp. 145–155.

[7] Thomas M Cover. *Elements of information theory*. John Wiley & Sons, 1999.

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

[9] Pierre Dellenbach, Jean-Emmanuel Deschaud, Bastien Jacquet, and François Goulette. "CT-ICP: Real-time elastic lidar odometry with loop closure". In: *2022 International Conference on Robotics and Automation (ICRA)*. IEEE. 2022, pp. 5580–5586.

[10] Lawrence C Evans. *An introduction to stochastic differential equations*. Vol. 82. American Mathematical Soc., 2012.

[11] Christian Forster, Luca Carlone, Frank Dellaert, and Davide Scaramuzza. "On-manifold preintegration for real-time visual–inertial odometry". In: *IEEE Transactions on Robotics* 33.1 (2016), pp. 1–21.

[12] Timo Hackel, Jan D Wegner, and Konrad Schindler. "Fast semantic segmentation of 3D point clouds with strongly varying density". In: *ISPRS annals of the photogrammetry, remote sensing and spatial information sciences* 3 (2016), pp. 177–184.

[13] Dongjiao He, Wei Xu, Nan Chen, Fanze Kong, Chongjian Yuan, and Fu Zhang. "Point-LIO: Robust High-Bandwidth Light Detection and Ranging Inertial Odometry". In: *Advanced Intelligent Systems* 5.7 (2023), p. 2200459.

[14] Jeffrey Humpherys, Tyler J Jarvis, and Emily J Evans. *Foundations of Applied Mathematics, Volume 1: Mathematical Analysis*. SIAM, 2017.

[15] Michael Kaess, Hordur Johannsson, Richard Roberts, Viorela Ila, John J Leonard, and Frank Dellaert. "iSAM2: Incremental smoothing and mapping using the Bayes tree". In: *The International Journal of Robotics Research* 31.2 (2012), pp. 216–235.

[16] Ricardo A Maronna, R Douglas Martin, Victor J Yohai, and Matías Salibián-Barrera. *Robust statistics: theory and methods (with R)*. John Wiley & Sons, 2019.

[17] Judea Pearl. *Causality*. Cambridge university press, 2009.

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

[19] Tixiao Shan and Brendan Englot. "LeGO-LOAM: Lightweight and ground-optimized lidar odometry and mapping on variable terrain". In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2018, pp. 4758–4765.

[20] Tixiao Shan, Brendan Englot, Drew Meyers, Wei Wang, Carlo Ratti, and Daniela Rus. "LIO-SAM: Tightly-coupled lidar inertial odometry via smoothing and mapping". In: *2020 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE. 2020, pp. 5135–5142.

[21] Tixiao Shan, Brendan Englot, Carlo Ratti, and Daniela Rus. "LVI-SAM: Tightly-coupled LiDAR-visual-inertial odometry via smoothing and mapping". In: *2021 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2021, pp. 5692–5698.

[22] Joan Sola, Jeremie Deray, and Dinesh Atchuthan. "A micro Lie theory for state estimation in robotics". In: *arXiv preprint arXiv:1812.01537* (2018).

[23] John Stillwell. *Naive Lie theory*. Springer Science & Business Media, 2008.

[24] Sebastian Thrun. "Probabilistic robotics". In: *Communications of the ACM* 45.3 (2002), pp. 52–57.

[25] Ignacio Vizzo, Tiziano Guadagnino, Benedikt Mersch, Louis Wiesmann, Jens Behley, and Cyrill Stachniss. "KISS-ICP: In defense of point-to-point ICP–simple, accurate, and robust registration if done the right way". In: *IEEE Robotics and Automation Letters* 8.2 (2023), pp. 1029–1036.

[26] Eric A Wan and Rudolph Van Der Merwe. "The unscented Kalman filter for nonlinear estimation". In: *Proceedings of the IEEE 2000 adaptive systems for signal processing, communications, and control symposium (Cat. No. 00EX373)*. Ieee. 2000, pp. 153–158.

[27] Han Wang, Chen Wang, Chun-Lin Chen, and Lihua Xie. "F-LOAM: Fast lidar odometry and mapping". In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2021, pp. 4390–4396.

[28] Wei Xu, Yixi Cai, Dongjiao He, Jiarong Lin, and Fu Zhang. "Fast-LIO2: Fast direct LiDAR-inertial odometry". In: *IEEE Transactions on Robotics* 38.4 (2022), pp. 2053–2073.

[29] Wei Xu and Fu Zhang. "Fast-LIO: A fast, robust LiDAR-inertial odometry package by tightly-coupled iterated kalman filter". In: *IEEE Robotics and Automation Letters* 6.2 (2021), pp. 3317–3324.

[30] Heng Yang, Jingnan Shi, and Luca Carlone. "Teaser: Fast and certifiable point cloud registration". In: *IEEE Transactions on Robotics* 37.2 (2020), pp. 314–333.

[31] Haoyang Ye, Yuying Chen, and Ming Liu. "Tightly coupled 3D LiDAR inertial odometry and mapping". In: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, pp. 3144–3150.

[32] Ji Zhang, Michael Kaess, and Sanjiv Singh. "On degeneracy of optimization-based state estimation problems". In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2016, pp. 809–816.

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

[34] Ji Zhang and Sanjiv Singh. "Visual-LiDAR odometry and mapping: Low-drift, robust, and fast". In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2015, pp. 2174–2181.

[35] Shibo Zhao, Hengrui Zhang, Peng Wang, Lucas Nogueira, and Sebastian Scherer. "Super Odometry: IMU-centric LiDAR-visual-inertial estimator for challenging environments". In: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2021, pp. 8729–8736.

[36] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. "Fast global registration". In: *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam , The Netherlands, October 11-14, 2016, Proceedings, Part II 14*. Springer. 2016, pp. 766–782.