SuperLoop: a LIDAR-based SLAM Back-end for Underground Exploration

Lucas Nogueira

CMU-RI-TR-23-50

August 8, 2023

The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA

Thesis Committee:
Prof. Sebastian Scherer, *chair*
Prof. Michael Kaess
Jay Patrikar

Submitted in partial fulfillment of the requirements for the degree of Master in Robotics.

To my wife, Gabriela.
To my father, Thomaz.
To my mother, Ozeneide.

Tudo vale a pena, se a alma não é pequena
Fernando Pessoa

Abstract

Robots deployed in underground scenarios require a SLAM system that can handle a variety of challenges, such as the absence of GPS, large scale maps, bad illumination, and geometrically degenerate environments. It is nearly impossible for any SLAM solution to handle all these challenges perfectly, specially if the robot is exploring its environment for the first time. In this case, the SLAM system must adapt to the diverse scenarios it finds and recognize any eventual failure so the other robot modules can handle it appropriately. Therefore, it is imperative to design fail-aware SLAM systems. A common architecture for modern SLAM systems is to separate a sensor-dependent front-end algorithm from a back-end based on pose graph optimization (PGO). In these cases, the PGO needs to estimate the uncertainty of the relative keypose transformation generated by the front-end. However, important state-of-the-art LIDAR-based odometry algorithms do not provide this information at all. This work proposes an algorithm that can estimate this uncertainty by analysing only the sequence of odometry poses produced in the keyframe window and comparing them to the gyroscope and accelerometer readings of an inertial measurement unit (IMU). This uncertainty estimation (UE) algorithm is used to build a complete SLAM solution, with LIDAR-inertial SuperOdometry as the front-end, and a PGO back-end. Additionally, underground place recognition capabilities are added this back-end via a slightly modified ScanContext descriptor. The UE algorithm is tested with the use of simulated and motion capture data, due to the availability of a ground-truth in these cases. The back-end SLAM system, named SuperLoop, is tested with data from wheeled robots exploring an abandoned hospital. The results show that the proposed UE algorithm may be better than using a constant diagonal covariance, as is common practice. It is also shown how it can detect odometry failure cases and may fix them before they contaminate the back-end.

Acknowledgments

I have had the privilege of crossing path with a host of great people that help make possible the long journey toward this degree. In general, they have showed me that being a great professional and great human being at the same time is typically highly-correlated.

From my large Brazilian family, I thank my parents, Ozeneide and Thomaz, for all their hard work put towards ensuring I had a good health and a good education. Not sure which one was the most challenging. I also thank my grandmothers Glória and Zenaide for their never-ending care. You are deeply missed. I thank my sisters and brothers-in-law: Mariana, Rebeca, Tomás and João. Your support and friendship were invaluable. I also thank my little nephew and nieces and goddaughters, Lorenzo, Lua, Sofia, Marina, and Pérola; who are the joy of my life.

I also thank the people that gave me my first work opportunity in Pittsburgh. Samuel Bueno, my research advisor, who sent what turned out to be a very fruitful email asking for a internship on my behalf. Marcel Bergermann, who received that email and gave me a life-changing opportunity. Ji Zhang, who wrote this little piece of code which became my main work specialty. Dave Duggins, who made sure I always had work coming my way. Finally, Kevin Dowling, for demonstrating such a trust in me even before we had actually met in person.

Once at CMU, I thank first my Subt colleagues who were incredibly important in my adaptation to a new city in the middle of a epidemic and under the pressure of a DARPA Challenge. Thanks Chao Cao and Hongbiao Zhu, specially. You guys were like brothers in those days. I thank Shibo Zhao, a great friend, co-inventor, and just a really nice guy that I was incredibly fortunate to have as the closest work collaborator. I also thank my fun and very resilient friends from the MMPUG project, Damanpreet Singh, Namya Bagree, Sourojit Saha and Prasanna Sriganesh. I hope I can keep your friendship for life.

Thanks to my advisor Sebastian Scherer, who always showed that he cared a lot for his students and their own personal development, with great respect for their mental health. I also thank the AirLab family and the RI community, of which I am very proud of having been a part. In particular, I thank Andrew Saba, Josh Spisak, Peigen Sun, Ian Higgins, John Keller, Greg Armstrong.

Funding

The content is approved for public release and the distribution is unlimited. This work is partially sponsored by DARPA under agreement No. HR00111820044. The content is not endorsed by and does not necessarily reflect the position or policy of the government or our sponsors.

Contents

When this dissertation is viewed as a PDF, the page header is a link to this Table of Contents.

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

The DARPA Subterranean Challenge [9] represented an important research thrust into resilient real-world SLAM systems. In this competition, teams were required to deploy robotic systems that explored underground environments in search for a pre-defined list of artifacts. Point were scored by reporting artifact locations within a 5 meter error threshold. The robots were managed by a single human operator via a computer interface. The most efficient solutions deployed multiple robots simultaneously, and the human operator only sent high-level commands to them. Therefore, each robot required enough autonomy to spend most of their time unattended and still make progress towards their mission. This requirement imposed several challenges for the SLAM system, since subterranean environments are large scale GPS-denied environments. Solving these type of scenarios will increase the deployment of robots in search-and-rescue missions, such as the Surfside Condo building collapse in Florida [29].

The Challenge took place mainly over 3 separate events: the Tunnel Circuit (2019); the Urban Circuit (2020) and the Final Event (2021). An additional Cave Circuit event was supposed to take place in 2020, but it was cancelled due to concerns regarding the COVID epidemic. A good review of the SLAM solutions developd by all teams for the Final Event can be found in [16]. Carnegie Mellon University was represented in the Challenge by Team Explorer [7].

Figure 1.1: Top: Complete map generated by Team Explorer after 30 minutes in the Final Event. Multiple colors represent contributions from different robots. At this stage, Team Explorer had reached 26 out of 28 map sectors, as defined by DARPA. Bottom: Final complete map colorized by the deviation metric. Green are inlier points, within 1 meter of ground truth. Orange are outlier points.

The Tunnel Circuit consisted of a network of tunnels with extended length and constrained passages, such as those found in mining operations. Team Explorer achieved first place in this event and an additional award for most accurate artifact location. The Urban Circuit took place in a an abandoned nuclear power plant, representing man-made underground environments with vertical elements. The team finished in second place. For both of these events, the SLAM submodule consisted of a LOAM [37] front-end and a complementary back-end based on Pose Grapho Optimizaiton (PGO) and a radius search loop closure method [31].

The Final Event featured a new SLAM front-end algorithm: SuperOdometry (SO) [40]. SO was developed entirely within the backdrop of the DARPA Subterranean Challenge to tackle data fusion challenges. Its first design principle is modularity. It dictates the goal of being able to add new sensor modalities to the estimation algorithm without modifying the existing framework, and with minimal dependencies between them. The second design principle is resiliency. It expresses the idea that failure in a single modality should not impact the whole system, as long as there is a single working modality available. This is partly achieved by the first principle. The key insight to SO is the focus on the IMU as a environment-independent sensor. This leads to an IMU-centric architecture that combines advantages from loosely- and tightly-coupled methods. It allows for easy integration of different sensor modules, which can vary depending on the robot platform.

The back-end for the Final Event remained the same as the previous events, however. Integration efforts ensured that each module operated mostly independent. SO was heavily adapted to conform to the same LOAM output formats, ensuring compatibility. This solution, named ExplorerSLAM, is described in more details in [16]. Overall, our combined system achieved the deviation of 11.6% of the map while covering 84.7% of the map, as can be seen in Fig. 1.1. The deviation metric is defined by DARPA as the number of points that are further than 1 meter away from any point in the ground truth pointcloud. All of our visual artifact detections were correctly localized within the DARPA scoring tolerance. Most SLAM drift was concentrated in 3 out of 7 robots, while the other 4 had near-perfect runs. Fig. 1.2 shows the evolution of the front-end and back-end components used by Team Explorer during the DARPA Subterranean Challenge

These results put ExplorerSLAM performance in the Final Event as second only to

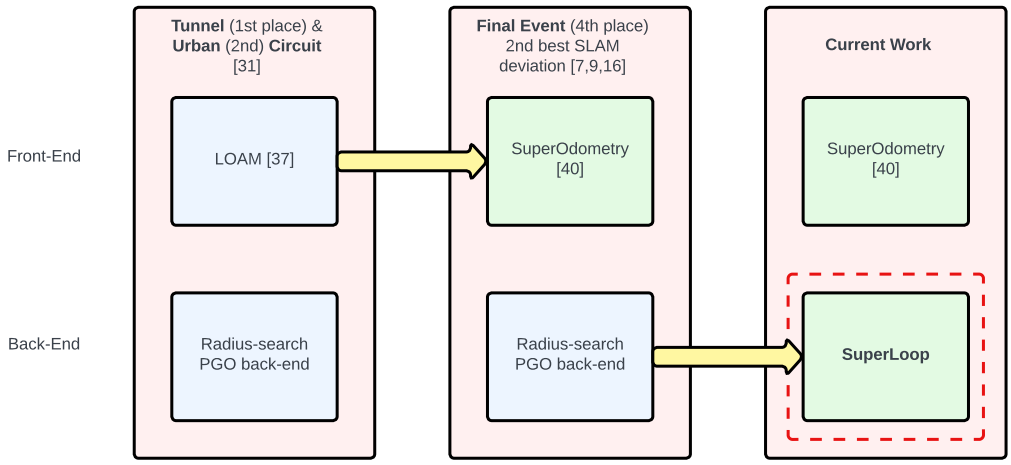


Figure 1.2: Evolution of SLAM Systems used by Team Explorer in the DARPA Subterranean Challenge. The proposed method is highlighted in red.

Team CSIRO's Data61 solution based on Wildcat SLAM [30], which is already in early commercialization phase. Despite this good result, ExplorerSLAM still leaves room for improvement, specially in the back-end component. During testing, important issues arose that were not present in the Final Event, and remained unsolved. For instance, long and geometrically degenerate corridors were a pain point for ExplorerSLAM. They caused enough drift that the back-end found it impossible to recover, because it exceeded the maximum radius allowed for loop closure detection, as seen in Fig. 1.3. These issues motivated the research on a SuperOdometry-compatible back-end that is better at handling odometry failure and possibly recovering from it.

Existing LIDAR-based SLAM systems such as LOAM and SuperOdometry do not have an estimate of how good their own estimation is. When they do create metrics, they are not directly translatable to the factor graph covariance formulation. Therefore, the current practice has been for the SLAM engineer to choose a constant diagonal matrix to be used in the back-end for all the odometric constraints. This is done by trial-and-error method, and with the assumption that the test conditions will be a good representation of the final deployment scenario. However, our preparation for the Subterranean Challenge showed that a) the quality of pointcloud registration varies heavily in the same run depending on environment geometry; and that b) it is

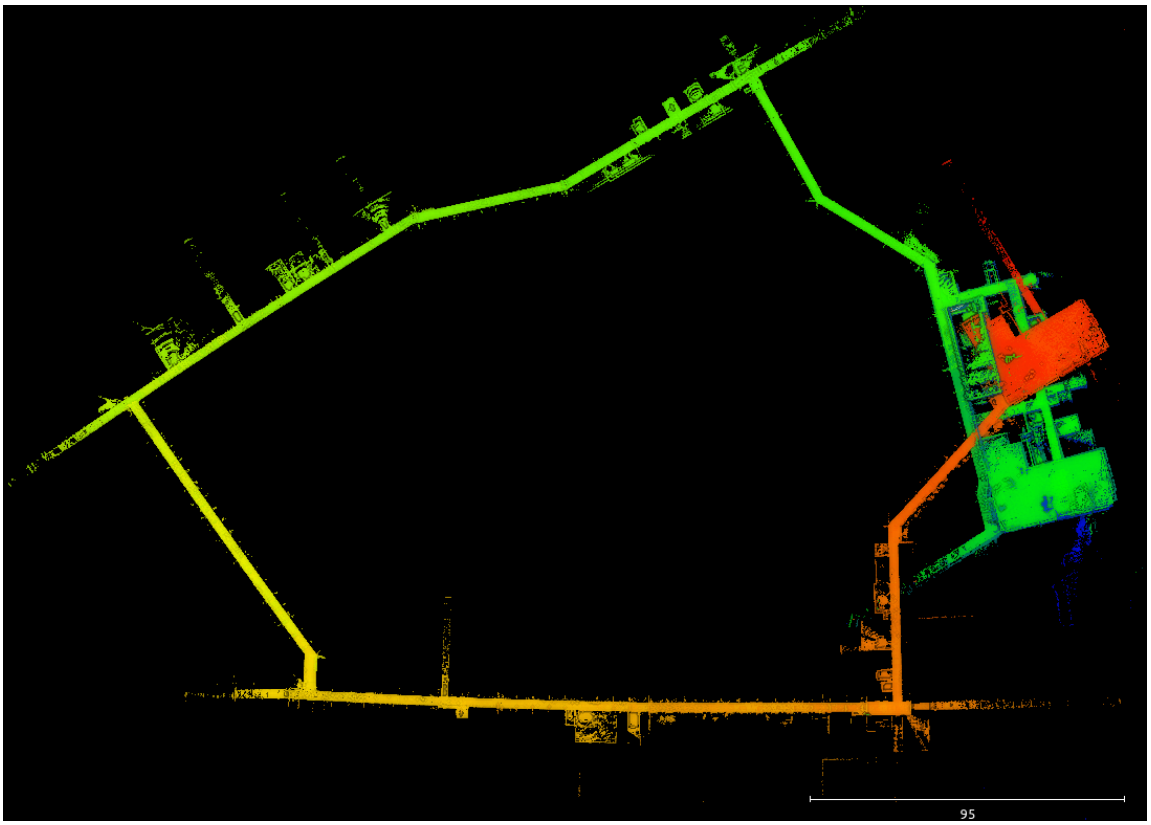


Figure 1.3: Example of a failed SLAM result during preparation for the Subterranean Challenge. The scale of the environment prevents closing the loop when the only mechanism is a radius search.

very hard to predict and recreate the deployment scenario during the testing phase. Thus, it is important to create algorithms that are robust to diverse geometry and, when that is not possible, aware of its failure cases.

This work develops a SLAM back-end that improves Team Explorer's solution and improves its robustness and fail-aware capabilities in two ways. First, introducing a place recognition mechanism that can work within the subterranean environment. Second, developing a mechanism for uncertainty estimation that allows the SLAM system to properly deal with low-quality sections of the map, decreasing the confidence in the bad odometry results. Additionally, the uncertainty estimation method should ideally generalize for different sensor types that might be used in the SLAM module in the future.

1.2 Related Work

1.2.1 Lidar-based SLAM State-of-the-Art

Modern SLAM research aims to design algorithms that can leverage the strengths of different sensors and alleviate their weakness. Common exteroceptive sensors in use are RGB and infrared cameras; radar; and LIDAR ("light detection and ranging"). LIDAR sensors are relatively expensive, but also very reliable. As evidence, we note that among the top 5 teams in the Subterranean Challenge, all of them relied heavily on LIDARs, sometimes even deploying multiple copies in a single robot. A detailed review of the solutions deployed in the Final Round of the Challenge is found in [16]. The few robots that tried to use visual-inertial solutions without LIDAR experienced severe issues during their runs. The LIDAR prevalence is due to the nature of underground exploration, with its lack of illumination and visual degeneracies. Additionally, the Challenge required artifacts with maximum error of 5 meters, and most teams deemed that camera-based SLAM does not offer that guarantee. For these reasons, LIDAR-based SLAM is still the benchmark for underground exploration.

Popular LIDAR-based SLAM algorithms are actually better characterized as LIDAR-odometry, since the map is static and there are no loop closures. LOAM[37], the best performing LIDAR algorithm in the KITTI benchmark, is a prime example. SuperOdometry [40] is similar in this aspect. LOAM uses a loosely coupled approach, where each sensor modality is processed with a separate optimization module, generating an initial guess for the subsequent ones, ending with the LIDAR module. SuperOdometry uses a combination of tightly- and loosely-coupled solutions, and fuses information from different modules using an IMU-centric factor graph. CompSLAM [18] was the solution used by the winners of the Subterranean Challenge. It is based on LOAM, with the addition of visual odometry as prior to the LIDAR processing and a D-optimality criterion for rejecting the visual estimates.

Other algorithms do include a loop closure step and may be considered a full SLAM solution [28]. LIO-SAM [32] fuses LIDAR odometry, IMU preintegration factors [17], GPS measurements and loop closure constraints in a single factor graph. Its initial implementation only uses Euclidian distance-based loop closure detection, but is designed to be compatible with arbitraty place recognition methods. Another

example is WildcatSLAM [30], the SLAM solution from TEAM CSIRO that achieved the smallest deviation result in the Final Round, according to DARPA's metrics. It uses a continuous-time trajectory formulation in its LIDAR registration process, and pose graph optimization (PGO) in the back-end, with the possibility of integrating ScanContext [19] place recognition. This method inspires some features of our proposed method, such as the use of B-splines as a trajectory representation and PGO.

1.2.2 SLAM Uncertainty Metrics

State-of-the-art SLAM systems such as the ones mentioned in the previous section usually have modules dedicated to processing specific sensor modalities, such as LIDAR Odometry (LO) and Visual odometry (LO). Another sensor fusion step is required to merge the information from these different modules. This steps can be implemented with a filter such as the EKF [10], factor graphs [32], or loosely-coupled as an initialization value for another module optimization [37]. The merged information may be in the form of pose estimates constraints or even raw constraints such as landmarks associations. In any case, the merge step requires a model of the uncertainty associated with each sensor modality. This model allows the sensor fusion algorithm to appropriately weigh the redundant informtion from each module, generating a consensus estimate.

Some of the uncertainty metrics in use are not defined properly as a distribution, and act more like a quality measure. For instance, LOAM uses the eigenvalues of the optimization covariance matrix as a way to determine degenerate directions [38]. Then, these directions are not updated during the optimization, maintaining the estimates from the previous modules in the pipeline. SuperOdometry and LIO-SAM assume that the estimates from its exteroceptives modules are governed by a constant and diagonal Gaussian noise, determined by hand-tuning and usually setting the LIDAR with a relatively small covariance. During normal operation, this is a reasonable assumption, but it is exactly during anormal operation that the estimation problem becomes interesting.

One possibility is to estimate the uncertainty in LIDAR scan matching analysing the properties of the underlying Iterative Closest Point (ICP) algorithm. For instance,

some methods [2, 8] propose to use the Hessian matrix of the scan-matching error function to derive a covariance estimate. Other methods [4, 25, 26] use sampling-based approaches where a Monte Carlo-style method generates samples of pose parameters. However, these methods are considered computationally expensive. An unscented transform is to calculate the ICP covariance in [3], postulating that the covariance of the initial estimate dominates the ICP error. The suitability of each of these methods for sensor fusion purposes is still debatable. For instance, the authors of [28] find that the method in [8] is over-optmistic, underestimating the actual uncertainty, and thus it may lead to numerical issues in the factor graph optimization.

Our method abstracts away the sensor modality, and looks only at the sequence of pose estimates it produces. This was done because it could later be applied to other sensor types such as cameras and GPS. Therefore, it mainly manipulates the uncertainty quantities expressed in the $SE(3)$ manifold and its derivatives. The Lie Algebra formulation of uncertainty offers accuracy advantages compared to pure Cartesian coordinates [24]. Mangelson et al. [27] develops expressions for the resulting covariances from pose composition, pose inversion, and relative pose operations using Lie groups. It expands upon [1] by modeling correlated poses. A practical summary of the Lie Algebra and its applications to robotics can be found in [33]. These concepts are used in this work to model the uncertainty in the rigid body motion group $SE(3)$.

1.3 Contributions

The main contributions of this thesis are:

1. A method for estimating the uncertainty in a sequence of pose estimates using the IMU sensor as an arbitrator. It creates continuous-time trajectories from the odometry data using B-splines. Then these splines are efficiently differentiated to provide linear acceleration and angular velocity estimates at the IMU timestamps. A covariance estimate is calculated by comparing the spline-based estimates and the IMU measurements inside a given window of time. This is explained in Chapter 3.
2. SuperLoop, a LIDAR-based SLAM back-end that works synergistically with SuperOdometry. It has built-in place recognition via ScanContext that has been

adapted for underground environments. Also, it uses our uncertainty estimation method to build its internal factor graph. Finally, it can handle some level of SLAM failure and recover from it. This is presented in Chapter 4.

Chapter 2

Background

This chapter presents theoretical background information necessary to understand the methods proposed in this work. Section 2.1 introduces to the basic concepts of Lie Algebra; and Section 2.2 summarizes the theory of B-spline interpolation. These are required to understand the Uncertainty Estimation method presented in Chapter 3. Section 2.3 presents the Factor Graph theory that underpins a large part of modern SLAM systems. Finally, Section 2.4 describes the ScanContext place recognition module. These last two sections are important to understand the SLAM system presented in Chapter 4.

2.1 Lie Algebra

One important challenge that arises in the formulation of estimation algorithms relates to the mathematical nature of rotations. A rotation matrix $\mathbf{R} \in SO(3)$ is 3×3 matrix with 9 elements, but only 3 degrees of freedom (DOF), thus it is *overparameterized*. This is contrast to translation, typically described with a vector $\mathbf{t} \in \mathbb{R}^3$, and has exactly 3 parameters for 3 DOF. Overparameterization causes issues in mathematical optimization, and wrong formulations may result in situations where a variable that should represent a physical rotation becomes a 3×3 matrix that is no longer in $SO(3)$. An additional projection step is needed in this case to bring the variable back into the $SO(3)$ space.

However, there is an important property of the $SO(3)$ that allows for a better

modeling of the problem. These spaces are *manifolds*, i.e. every point in it has a neighborhood that is homeomorphic to Euclidian space. Therefore, around any given point it is possible to build a vector space (the tangent space) representing incremental changes that can be mapped back to the manifold. As will be shown, using this mechanism allows us to avoid the overparameterization issue and properly perform optimizations and model uncertainties.

These ideas give rise to the Lie Theory, formulated by mathematician Sophus Lie in the XIX century and currently an important tool in estimation theory. A better practical introduction on Lie Theory can be found in [33]. This section introduces only the parts of the theory that are relevant to understanding the proposed methods.

2.1.1 Lie Group and Lie Algebra

A Lie Group is a *smooth manifold* whose elements satisfy the group axioms. A smooth manifold is a topological space that is locally similar to an Euclidian (i.e. linear) space. A group $(\mathcal{G}, \circ)$ is composed of a set $\mathcal{G}$ and a composition operation $\circ$ such that the following axioms are valid:

- **Closure under** $\circ$: The composition of any two elements of the group remains in the group

$$a, b \in \mathcal{G} \ \cap \ a \circ b = c \quad \Longrightarrow \quad c \in \mathcal{G}$$

- **Identity**: There exists an identity element that whenever composed with another element of the group, returns that same element.

$$\exists i \in \mathcal{G} : a \circ i = i \circ a = a; \ \ \forall a \in \mathcal{G}$$

- **Inverse**: For every element of the group, there exists exactly one other element such that composition of the two result in the identity element:

$$\forall a \in \mathcal{G} \implies \exists b \in \mathcal{G} : a \circ b = b \circ a = i$$

- **Associativity**: In an expression containing multiple occurrences of the composition, the order in which these operations are performed does not alter the final result.

$$\forall a, b, c \in \mathcal{G} \implies (a \circ b) \circ c = a \circ (b \circ c)$$

Lie groups bridge the properties of smooth manifolds making it possible to perform

calculus with these mathematical entities. Given a Lie Group $\mathcal{M}$, the tangent space *at the identity*, $\mathcal{T}_{\mathcal{E}}\mathcal{M}$, is the *Lie algebra* of $\mathcal{M}$, and noted $\mathfrak{m}$, such that $\mathfrak{m} \triangleq \mathcal{T}_{\mathcal{E}}\mathcal{M}$. The folowing remarks connect the Lie Algebra and its group:

- The Lie Algebra $\mathfrak{m}$ is a vector space and therefore its elements can be identified to $\mathbb{R}^m$, where m is exactly the number of DOF in the manifold $\mathcal{M}$.
- The is an operation called the *exponential map* $\exp : \mathfrak{m} \to \mathcal{M}$ that exactly converts elements of the Lie Algebra into elements of the group and its inverse is the *logarithmic map*.
- Vectors of the tangent space at $\mathcal{X} \in \mathcal{M}$ can be transformed to the tangent space at identity $\mathcal{I}$ through a linear transform called the *adjoint* transformation.

A Lie Algebra can be defined in any point $\mathcal{X}$ of the manifold, creating a vector space for the tangent space $\mathcal{T}_{\mathcal{X}}\mathcal{M}$. One can observe how the structure of the Lie algebra arises using the $SO(3)$ group as an example.

The $SO(3)$ rotation group All elements of this group respect the orthonormality condition $\mathbf{R}^\top\mathbf{R} = \mathbf{I}$. Taking the time derivative of this expression yields: $\mathbf{R}^\top\dot{\mathbf{R}} = -(\mathbf{R}^\top\dot{\mathbf{R}})^\top$. This implies that the expression $\mathbf{R}^\top\dot{\mathbf{R}}$ is the negative of its transpose, a property known as skew-symmetry. These special matrices in 3-dimensions are commonly written as:

$$[\omega]_\times = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ \omega_y & \omega_x & 0 \end{bmatrix}; \quad \omega = [\omega_x, \omega_y, \omega_z]^\top \tag{2.1}$$

Therefore $\mathbf{R}^\top\dot{\mathbf{R}} = [\omega]_\times$ and $\dot{\mathbf{R}} = \mathbf{R}[\omega]_\times$. This last expression gives us the form of the tangent space for a given element $\mathbf{R} \in SO(3)$. For the identity case where $\mathbf{R} = \mathbf{I}$, we obtain the Lie algebra of $SO(3)$, denoted $\mathfrak{so}(3)$:

$$\dot{\mathbf{R}} = [\omega]_\times \implies [\omega]_\times \in \mathfrak{so}(3) \tag{2.2}$$

The DOF of this Lie algebra is 3, and therefore it can be related to $\mathbb{R}^3$. For this purpose, the *hat* and *vee* operators are defined.

- Hat Operator $^\wedge$: $\mathbf{R}^3 \to \mathfrak{so}(3); \quad \omega \mapsto \omega^\wedge = [\omega]_\times$
- Vee Operator $^\vee$: $\mathfrak{so}(3) \to \mathbf{R}^3; \quad [\omega]_\times \mapsto [\omega]_\times^\vee = \omega$

Note that the skew-symmetric nature of the tangent space here is a direct consequence of the group constraint of $SO(3)$ and therefore might be different for different Lie Groups. Also the hat and vee operators are always available in any Lie Groups, although their specific format will vary accordingly. More generally, they can be defined by considering the derivatives around the origin of the Lie group $\mathcal{M}$ in the i-th direction E_i, also known as the *generators* of $\mathfrak{m}$. Thus we can rewrite the hat and vee operators for any Lie algebra as:

- Hat: $\mathbb{R}^m \to \mathfrak{m}; \boldsymbol{\tau} \mapsto \boldsymbol{\tau}^\wedge = \sum_{i=1}^m \tau_i E_i$
- Vee: $\mathfrak{m} \to \mathbb{R}^m; \boldsymbol{\tau}^\wedge \mapsto (\boldsymbol{\tau}^\wedge)^\vee = \boldsymbol{\tau} = \sum_{i=1}^m \tau_i \mathbf{e}_i$

The vectors $\mathbf{e}_i$ form the basis of $\mathbb{R}^m$. This shows how the Lie algebra $\mathfrak{m}$ is isomorphic to the vector space $\mathbb{R}^m$. This allow the two spaces to be used interchangeably, and $\mathbb{R}^m$ is typically preferred in this work.

2.1.2 Exponential and Logarithmic Map

The exponential and the logarithmic maps are operations that allow us to convert elements from the Lie algebra to the group and vice-versa. The exponential map can be derived by considering the inverse equation and taking its derivative. Let $\mathcal{X}$ be an element of the Lie Group $\mathcal{M}$ and $\mathcal{I}$ the group's identity. Then:

$$\mathcal{X}^{-1}\mathcal{X} = \mathcal{I} \tag{2.3}$$

$$\mathcal{X}^{-1}\dot{\mathcal{X}} + \dot{\mathcal{X}^{-1}}\mathcal{X} = 0 \tag{2.4}$$

$$\mathcal{X}^{-1}\dot{\mathcal{X}} = -\dot{\mathcal{X}^{-1}}\mathcal{X} = \tag{2.5}$$

$$\dot{\mathcal{X}} = \mathcal{X}\boldsymbol{\tau}^\wedge \tag{2.6}$$

This derivation is very similar as the one shown for the $SO(3)$ rotation group. This is a generalization of that one. The last equation may be recognized as an ordinary differential equation (ODE) and its solution is:

$$\mathcal{X}(t) = \mathcal{X}(0)\ \exp(\boldsymbol{\tau}^\wedge) \tag{2.7}$$

$$\exp(\boldsymbol{\tau}^\wedge) = \mathcal{X}(0)^{-1}\mathcal{X}(t) \tag{2.8}$$

Therefore, by the group property, $\exp(\boldsymbol{\tau}^\wedge)$ is also a group member and the exponential map takes an element of the Lie algebra and returns an element of the Lie group. The inverse process is the logarithmic map, and they can be summarized

as:

- Exponential Map exp: $\mathfrak{m} \to \mathcal{M}; \boldsymbol{\tau}^\wedge \mapsto \mathcal{X} = \exp(\boldsymbol{\tau}^\wedge)$
- Logarithmic Map log : $\mathcal{M} \to \mathfrak{m}; \mathcal{X} \mapsto \boldsymbol{\tau}^\wedge = \log(\mathcal{X})$

The exponential map may be calculated using the Taylor series:

$$\exp(\boldsymbol{\tau}^\wedge) = \mathcal{I} + \boldsymbol{\tau}^\wedge + \frac{1}{2}\boldsymbol{\tau}^{\wedge 2} + \dots \tag{2.9}$$

Some properties of the exponential map are intuitive:

$$\exp((t+s)\boldsymbol{\tau}^\wedge) = \exp(t\boldsymbol{\tau}^\wedge)\exp(s\boldsymbol{\tau}^\wedge) \tag{2.10}$$

$$\exp(t\boldsymbol{\tau}^\wedge) = \exp(\boldsymbol{\tau}^\wedge)^t \tag{2.11}$$

$$\exp(-\boldsymbol{\tau}^\wedge) = \exp(\boldsymbol{\tau}^\wedge)^{-1} \tag{2.12}$$

with t, s being scalars. On the other hand, a less intuitive property of the exponential map is:

$$\exp(\mathcal{X}\boldsymbol{\tau}^\wedge\mathcal{X}^{-1}) = \mathcal{X}\exp(\boldsymbol{\tau}^\wedge)\mathcal{X}^{-1} \tag{2.13}$$

Following the convention in [33], we define the capitalized exponential and logarithmic maps, that map directly from and to $\mathbb{R}^m$ to the Lie group. This is just a shortcut that simplifies the notation.

- Exp: $\mathbb{R}^m \to \mathcal{M} : \mathrm{Exp}(\boldsymbol{\tau}) \triangleq \exp(\boldsymbol{\tau}^\wedge) = \mathcal{X}$
- Log: $\mathcal{M} \to \mathbb{R}^m : \mathrm{Log}(\mathcal{X}) \triangleq \log(\mathcal{X})^\vee = \boldsymbol{\tau}$

2.1.3 Group and Increment Operations

Using the theory presented so far, it is possible to create operations that compose Lie group elements with small increments in the Lie algebra, which is particularly important for optimization and uncertainty modeling. Specifically, we define plus and minus operators, and these can be separated into right- and left- versions. The right operators are:

- right-$\oplus$:

$$\mathcal{Y} = \mathcal{X} \oplus {}^{\mathcal{X}}\boldsymbol{\tau} \triangleq \mathcal{X} \circ \mathrm{Exp}({}^{\mathcal{X}}\boldsymbol{\tau}) \in \mathcal{M}$$

- right-$\ominus$:

$${}^{\mathcal{X}}\boldsymbol{\tau} = \mathcal{Y} \ominus \mathcal{X} \triangleq \mathrm{Log}(\mathcal{X}^{-1} \circ \mathcal{Y}) \in \mathcal{T}_{\mathcal{X}}\mathcal{M}$$

For the right operators, the increment ${}^{\mathcal{X}}\boldsymbol{\tau}$ is an element of the tangent space at

$\mathcal{X}$, indicated by the left-side superscript. It is also common practice to say that the increment is expressed in the local frame at $\mathcal{X}$.

The left operators are:

- left-$\oplus$:
$$\mathcal{Y} = {}^{\mathcal{I}}\boldsymbol{\tau} \oplus \mathcal{X} \triangleq \operatorname{Exp}({}^{\mathcal{I}}\boldsymbol{\tau}) \circ \mathcal{X} \in \mathcal{M}$$
- left-$\ominus$:
$${}^{\mathcal{I}}\boldsymbol{\tau} = \mathcal{Y} \ominus \mathcal{X} \triangleq \operatorname{Log}(\mathcal{Y} \circ \mathcal{X}^{-1}) \in \mathcal{T}_{\mathcal{I}}\mathcal{M}$$

For the left operators, the increments are defined in the tangent space of the identity element $\mathcal{I}$ of the Lie group. The increment is now expressed in the *global* frame. In the physical spaces considered in this work, the identity is the origin of the reference frame. Most of the time in this work, the $\oplus$ and $\ominus$ operators will be referring to the right-forms, except when noted otherwise.

2.1.4 The Adjoint Operation

The adjoint operation enables a transformation of the increments between the global and local frames. Its derivation is:

$${}^{\mathcal{I}}\boldsymbol{\tau} \oplus \mathcal{X} = \mathcal{X} \oplus {}^{\mathcal{X}}\boldsymbol{\tau} \tag{2.14}$$
$$\operatorname{Exp}({}^{\mathcal{I}}\boldsymbol{\tau})\mathcal{X} = \mathcal{X} \operatorname{Exp}({}^{\mathcal{X}}\boldsymbol{\tau}) \tag{2.15}$$
$$\exp({}^{\mathcal{I}}\boldsymbol{\tau}^{\wedge}) = \mathcal{X}\exp({}^{\mathcal{X}}\boldsymbol{\tau}^{\wedge})\mathcal{X}^{-1} = \exp(\mathcal{X}\ {}^{\mathcal{X}}\boldsymbol{\tau}^{\wedge}\ \mathcal{X}^{-1}) \tag{2.16}$$
$${}^{\mathcal{I}}\boldsymbol{\tau}^{\wedge} = \mathcal{X}\ {}^{\mathcal{X}}\boldsymbol{\tau}^{\wedge}\ \mathcal{X}^{-1} \tag{2.17}$$

Where the property from Eq. 2.13 was used. Now we can define the adjoint of $\mathcal{M}$ at $\mathcal{X}$:

$$\operatorname{Adj}_{\mathcal{X}} : \mathfrak{m} \to \mathfrak{m}; \quad \boldsymbol{\tau}^{\wedge} \mapsto \operatorname{Adj}_{\mathcal{X}}(\boldsymbol{\tau}^{\wedge}) \triangleq \mathcal{X}\ \boldsymbol{\tau}^{\wedge}\ \mathcal{X}^{-1} \tag{2.18}$$

and therefore ${}^{\mathcal{I}}\boldsymbol{\tau}^{\wedge} = \operatorname{Adj}_{\mathcal{X}}({}^{\mathcal{X}}\boldsymbol{\tau}^{\wedge})$. The adjoint is a linear operator, and has an equivalent matrix operator that acts on the $\mathbb{R}^m$ vectors ${}^{\mathcal{I}}\boldsymbol{\tau}$ and ${}^{\mathcal{X}}\boldsymbol{\tau}$.

$$\mathbf{Adj}_{\mathcal{X}} \in \mathbb{R}^{m\times m} : \quad \mathbb{R}^m \to \mathbb{R}^m; \quad {}^{\mathcal{X}}\boldsymbol{\tau} \mapsto {}^{\mathcal{I}}\boldsymbol{\tau} = \mathbf{Adj}_{\mathcal{X}}\ {}^{\mathcal{X}}\boldsymbol{\tau} \tag{2.19}$$

2.1.5 Uncertainty in Manifolds

Lie theory allows us to obtain a unified formulation for dealing with uncertainties. A noise model can be defined by considering considering a local perturbation $\boldsymbol{\tau} \in T_{\mathcal{X}}\mathcal{M}$ around a mean value $\mathcal{X} \in \mathcal{M}$. In this work, we denote noisy estimates with a $\sim$ over the variable. This allows us to write:

$$\tilde{\mathcal{X}} = \mathcal{X} \oplus \boldsymbol{\tau} \iff \boldsymbol{\tau} = \tilde{\mathcal{X}} \ominus \mathcal{X} \tag{2.20}$$

With this formulation, covariance can be obtained with its standard definition using the expectation operation $\mathbb{E}[\cdot]$

$$\begin{aligned} \mathrm{Cov}(\tilde{\mathcal{X}}) \triangleq \Sigma_{\tilde{\mathcal{X}}} &= \mathbb{E}[(\tilde{\mathcal{X}} \ominus \mathbb{E}[\tilde{\mathcal{X}}])(\tilde{\mathcal{X}} \ominus \mathbb{E}[\tilde{\mathcal{X}}])^\top] && (2.21)\\ &= \mathbb{E}[(\tilde{\mathcal{X}} \ominus \mathcal{X})(\tilde{\mathcal{X}} \ominus \mathcal{X})^\top] && (2.22)\\ &= \mathbb{E}[\boldsymbol{\tau}\boldsymbol{\tau}^\top] && (2.23) \end{aligned}$$

Therefore, $\Sigma_{\tilde{\mathcal{X}}} \in \mathbb{R}^{m \times m}$ and it is possible to define a Gaussian distribution on the manifold such that $\tilde{\mathcal{X}} \sim \mathcal{N}(\mathcal{X}, \Sigma_{\tilde{\mathcal{X}}})$. Note that the covariance is defined in the tangent space instead of the manifold space itself.

Global-frame covariances can also be defined using the left- operators. In this case, we have:

$$\begin{aligned} \tilde{\mathcal{X}} = {}^{\mathcal{I}}\boldsymbol{\tau} \oplus \mathcal{X} &\iff {}^{\mathcal{I}}\boldsymbol{\tau} = \tilde{\mathcal{X}} \ominus \mathcal{X} && (2.24)\\ {}^{\mathcal{I}}\Sigma_{\tilde{\mathcal{X}}} &= \mathbb{E}[{}^{\mathcal{I}}\boldsymbol{\tau}\ {}^{\mathcal{I}}\boldsymbol{\tau}^\top] && (2.25) \end{aligned}$$

Using the adjoint operation, it is possible to convert the global covariance to local and vice-versa:

$${}^{\mathcal{I}}\Sigma_{\tilde{\mathcal{X}}} = \mathbf{Adj}_{\tilde{\mathcal{X}}}\ {}^{\mathcal{X}}\Sigma_{\tilde{\mathcal{X}}}\ \mathbf{Adj}_{\tilde{\mathcal{X}}}^\top \tag{2.26}$$

2.1.6 Derivatives and Jacobians in Lie Groups

Given a function f that maps elements from a manifold $\mathcal{M}$ to a manifold $\mathcal{N}$, it is possible to define a Jacobian matrix of $f : \mathcal{M} \to \mathcal{N}$ using the right-$\oplus, \ominus$ operators, using a form similar to the standard derivative.

$$\frac{{}^{\mathcal{X}}Df(\mathcal{X})}{D\mathcal{X}} \triangleq \lim_{\boldsymbol{\tau} \to 0} \frac{f(\mathcal{X} \oplus \boldsymbol{\tau}) \ominus f(\mathcal{X})}{\boldsymbol{\tau}} \tag{2.27}$$

This the *right Jacobian of f*. It maps variations in the local tangent space $T_{\mathcal{X}}M$

to the local tangent space of its image $T_{f(\mathcal{X})}N$. It is a $n \times m$ matrix, where n and m are the DOF of $\mathcal{N}$ and $\mathcal{M}$, respectively.

This allows for the small-tau ${}^{\mathcal{X}}\boldsymbol{\tau} \to 0$ approximation:

$$f(\mathcal{X} \oplus {}^{\mathcal{X}}\boldsymbol{\tau}) \approx f(\mathcal{X}) \oplus \frac{{}^{\mathcal{X}}Df(\mathcal{X})}{D\mathcal{X}} {}^{\mathcal{X}}\boldsymbol{\tau} \tag{2.28}$$

When $f = \text{Exp}(\boldsymbol{\tau})$, then we have the *right Jacobian of M.*

$$\mathbf{J}_r(\boldsymbol{\tau}) \triangleq \frac{{}^{\boldsymbol{\tau}}D \text{ Exp}(\boldsymbol{\tau})}{D\boldsymbol{\tau}} \in \mathbb{R}^{m \times m} \tag{2.29}$$

The right Jacobian of $\mathcal{M}$ allows for the following approximations:

$$\text{Exp}(\boldsymbol{\tau} + \delta\boldsymbol{\tau}) \approx \text{ Exp}(\boldsymbol{\tau}) \text{ Exp}(\mathbf{J}_r(\boldsymbol{\tau})\delta\boldsymbol{\tau}) \tag{2.30}$$

$$\text{Exp}(\boldsymbol{\tau}) \text{ Exp}(\delta\boldsymbol{\tau}) \approx \text{ Exp}(\boldsymbol{\tau} + \mathbf{J}_r^{-1}(\boldsymbol{\tau})\delta\boldsymbol{\tau}) \tag{2.31}$$

2.1.7 The SE(3) Rigid Motion Lie Group Properties

This section summarizes the application of Lie Theory to the 3D rigid motion space, $SE(3)$. This is the space of interest in this work, as it describe the motion of our robots.

An element $\mathcal{X} = \{\mathbf{R}, \mathbf{t}\}$ of $SE(3)$ is fully described by a 4×4 matrix $\mathbf{M}$ such that:

$$\mathbf{M} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix} \tag{2.32}$$

where $\mathbf{R} \in SO(3)$ and $\mathbf{t} \in \mathbb{R}^3$. The group constraint, inherited from $SO(3)$, is $\mathbf{R}^\top\mathbf{R} = \mathbf{I}$. This results in the element having size of 16, but only 6 DOF. The tangent space is $\mathbb{R}^6$, where the perturbation $\boldsymbol{\tau}$ is defined. It has two components:

$$\boldsymbol{\tau} = \begin{bmatrix} \boldsymbol{\rho} \\ \boldsymbol{\theta} \end{bmatrix} ; \quad \boldsymbol{\rho}, \boldsymbol{\theta} \in \mathbb{R}^3 \tag{2.33}$$

The translational component is $\boldsymbol{\rho}$ and the rotational one is $\boldsymbol{\theta}$. The element $\boldsymbol{\tau}^\wedge$ of the Lie algebra $\mathfrak{m}$ is:

$$\boldsymbol{\tau}^\wedge = \begin{bmatrix} [\boldsymbol{\theta}]_\times & \boldsymbol{\rho} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \tag{2.34}$$

The inverse of $\mathcal{X}$ is a consequence of matrix inversion and establishes that:

$$\mathcal{X}^{-1} = \{\mathbf{R}^\top, -\mathbf{R}^\top\mathbf{t}\} \tag{2.35}$$

The composition of $\mathcal{X} = \{\mathbf{R}_x, \mathbf{t}_x\} \in SE(3)$ with $\mathcal{Y} = \{\mathbf{R}_y, \mathbf{t}_y\} \in SE(3)$ is similarly obtained:

$$\mathcal{X} \circ \mathcal{Y} = \{\mathbf{R}_x\mathbf{R}_y, \mathbf{t}_x + \mathbf{R}_x\mathbf{t}_y\} \tag{2.36}$$

The Exp and Log operations maps from the vector tangent space $\mathbb{R}^6$ to the $SE(3)$ elements.

$$\mathbf{M} = \text{Exp}(\boldsymbol{\tau}) \triangleq \begin{bmatrix} \text{Exp}(\boldsymbol{\theta}) & \mathbf{V}(\boldsymbol{\theta})\boldsymbol{\rho} \\ \mathbf{0} & 1 \end{bmatrix} \tag{2.37}$$

$$\boldsymbol{\tau} = \text{Log}(\mathbf{M}) \triangleq \begin{bmatrix} \mathbf{V}^{-1}(\boldsymbol{\theta}\mathbf{t}) \\ \text{Log}(\mathbf{R}) \end{bmatrix} \tag{2.38}$$

Note that $\text{Log}(\mathbf{R}) = \boldsymbol{\theta} = \theta\mathbf{u}$, with θ a scalar and $\mathbf{u}$ a unit vector. And $\mathbf{V}(\boldsymbol{\theta})$ is defined as:

$$\mathbf{V}(\boldsymbol{\theta}) = \mathbf{I} + \frac{1-\cos(\theta)}{\theta^2}[\boldsymbol{\theta}]_\times + \frac{\theta - \sin\theta}{\theta^3}[\boldsymbol{\theta}]_\times^2 \tag{2.39}$$

When there is no rotational component, $\boldsymbol{\theta} = \mathbf{0}$, then $\mathbf{V}(\mathbf{0}) = \mathbf{I}$ and the operations simplify to the 3D translation space.

The adjoint is calculated by taking the right-hand side of Eq. 2.18, applying the vee operator to it, and isolating $\boldsymbol{\tau} = [\boldsymbol{\rho}^\top, \boldsymbol{\theta}^\top]^\top$.

$$\mathbf{Adj}_\mathbf{M}\,\boldsymbol{\tau} = (\mathbf{M}\boldsymbol{\tau}^\wedge\mathbf{M}^{-1})^\vee \tag{2.40}$$

$$= \begin{bmatrix} \mathbf{R}\boldsymbol{\rho} + [\mathbf{t}]_\times\mathbf{R}\boldsymbol{\theta} \\ \mathbf{R}\boldsymbol{\theta} \end{bmatrix} \tag{2.41}$$

$$= \begin{bmatrix} \mathbf{R} & [\mathbf{t}]_\times\mathbf{R} \\ 0 & \mathbf{R} \end{bmatrix} \begin{bmatrix} \boldsymbol{\rho} \\ \boldsymbol{\theta} \end{bmatrix} \tag{2.42}$$

$$\mathbf{Adj}_\mathbf{M} = \begin{bmatrix} \mathbf{R} & [\mathbf{t}]_\times\mathbf{R} \\ 0 & \mathbf{R} \end{bmatrix} \in \mathbb{R}^{6\times6} \tag{2.43}$$

The right Jacobian $\mathbf{J}_r(\boldsymbol{\rho}, \boldsymbol{\theta})$ of $SE(3)$ has the form

$$\mathbf{J}_r(\boldsymbol{\rho}, \boldsymbol{\theta}) = \begin{bmatrix} \mathbf{V}(-\boldsymbol{\theta}) & \mathbf{Q}(-\boldsymbol{\rho}, -\boldsymbol{\theta}) \\ 0 & \mathbf{V}(-\boldsymbol{\theta}) \end{bmatrix} \tag{2.44}$$

$$\begin{aligned}\mathbf{Q}(\boldsymbol{\rho},\boldsymbol{\theta}) = \frac{1}{2}\boldsymbol{\rho}_\times + \frac{\theta-\sin\theta}{\theta^3}(\boldsymbol{\theta}_\times\boldsymbol{\rho}_\times + \boldsymbol{\rho}_\times\boldsymbol{\theta}_\times + \boldsymbol{\theta}_\times\boldsymbol{\rho}_\times\boldsymbol{\theta}_\times) \\ -\frac{1-\frac{\theta^2}{2}-\cos\theta}{\theta^4}(\boldsymbol{\theta}_\times^2\boldsymbol{\rho}_\times + \boldsymbol{\rho}_\times\boldsymbol{\theta}_\times^2 - 3\boldsymbol{\theta}_\times\boldsymbol{\rho}_\times\boldsymbol{\theta}_\times) \\ -\frac{1}{2}\left(\frac{1-\frac{\theta^2}{2}-\cos\theta}{\theta^4} - 3\frac{\theta-\sin\theta-\frac{\theta^3}{6}}{\theta^5}\right)(\boldsymbol{\theta}_\times\boldsymbol{\rho}_\times\boldsymbol{\theta}_\times^2 + \boldsymbol{\theta}_\times^2\boldsymbol{\rho}_\times\boldsymbol{\theta}_\times)\end{aligned} \tag{2.45}$$

2.2 B-Spline Interpolation

B-splines are functions that allow us to represent continuous trajectories $\mathbf{p}(t)$ from discrete inputs (knots) $\mathbf{p}_i$, and have a number of desirable properties. The first one is *locality*. It dictates that the value of the function at a given time is only controlled by a small subset of knots. The number of knots k in this subset is the *order* of the spline. Another desirable property is that B-splines are C^{k-1} smooth. This means that derivatives are guaranteed to exist for the function $\mathbf{p}(t)$ up to the $k-1$ order.

We define a *uniform B-spline* as

$$\mathbf{p}(t) = \sum_{i=0}^{N} B_{i,k}(t)\mathbf{p}_i, \quad 0 \le i \le N \tag{2.46}$$

where $B_{i,k}(t)$ are the spline coefficients. In the uniform spline, each control point $\mathbf{p}_i$ is associated with time $t_i = t_0 + i\Delta t$, such that the knots are uniformly spaced in time. The coefficients are given by the De Boor-Cox [11, 12] recurrence relation:

$$B_{i,0}(t) = \begin{cases} 1 & \text{for } \; t_i \le t < t_{i+1} \\ 0 & \text{otherwise} \end{cases} \tag{2.47}$$

$$B_{i,j}(t) = \frac{t-t_i}{j\Delta t}B_{i,j-1}(t) + \frac{t_{i+j+1}-t}{j\Delta t}B_{i+1,j-1}(t) \tag{2.48}$$

It is possible to represent B-splins using a matrix representation. First, note that at time $t \in [t_i, t_{i+1})$, the value of $\mathbf{p}(t)$ depends only on $\mathbf{p}_i, \ldots, \mathbf{p}_{i+k-1}$. A simpler time unit is defined as $s(t) := (t-t_0)/\Delta t$, such that the time instants t_i transform to i. The temporal variable $u(t) := s(t) - i$ then represents the time since the start of the segment $[t_i, t_{i+1}]$. Now, it is possible to write

$$\mathbf{p}(u) = [\mathbf{p}_i \;\; \mathbf{p}_{i+1} \;\; \ldots \;\; \mathbf{p}_{i+k-1}]\mathbf{M}^{(k)}\mathbf{u} \tag{2.49}$$

where $\mathbf{u}$ is a column vector such that $u_n = u^n$ and $M^{(k)}$ is a blending matrix with

the entry at the s-th column and n-th row equal to

$$m_{s,n}^{(k)} = \sum_{l=s}^{k-1} (-1)^{l-s} C_k^{l-s} (k-1-l)^{k-1-n}, \quad C_k^s = \frac{k!}{s!(k-s)!} \tag{2.50}$$

An alternative representation is the *cumulative B-spline*, where the difference between knots is used, instead of their absolute values. This representation is:

$$\mathbf{p}(t) = \tilde{B}_{0,k}(t)\mathbf{p}_0 \oplus \sum_{i=1}^{N} \tilde{B}_{i,k}(t)(\mathbf{p}_i \ominus \mathbf{p}_{i-1}) \tag{2.51}$$

$$\tilde{B}_{i,k}(t) = \sum_{s=i}^{N} B_{s,k}(t) \tag{2.52}$$

These equations also admit a matrix representation, using the cumulative matrix $\tilde{\mathbf{M}}^{(k)}$ with entries $\tilde{m}_{s,n}^{(k)} = \sum_{s=j}^{k-1}$ and difference vectors $\mathbf{d}_\mathbf{j}^\mathbf{i} = \mathbf{p}_{i+j} \ominus \mathbf{p}_{i+j-1}$ such that

$$\mathbf{p}(u) = [\mathbf{p}_i \ \mathbf{d}_1^i \ \mathbf{d}_2^i \ \dots \ \mathbf{d}_{k-1}^i]\tilde{\mathbf{M}}^{(k)}\mathbf{u} \tag{2.53}$$

$$\mathbf{p}(u) = [\mathbf{p}_i \ \mathbf{d}_1^i \ \mathbf{d}_2^i \ \dots \ \mathbf{d}_{k-1}^i]\boldsymbol{\lambda}(u) \tag{2.54}$$

$$\mathbf{p}(u) = \mathbf{p}_i(u) \oplus \sum_{j=1}^{k-1} \lambda_j(u)\mathbf{d}_j^i \tag{2.55}$$

where this derivation uses the fact that $\lambda_0(u) = 1$. Eq. 2.55 has the added benefit of allowing for functions over Lie groups. Note that it is not possible to use Eq. 2.49 because there are no scaling operations of Lie Group elements. Therefore it is necessary to scale the increments in the Lie algebra (obtained with the $\ominus$) and add them together with the $\oplus$ operation.

To obtain derivatives of B-splines, it suffices to use the time derivative of $\mathbf{u}$, such that

$$\dot{\mathbf{p}}(u) = [\mathbf{p}_i \ \mathbf{d}_1^i \ \mathbf{d}_2^i \ \dots \ \mathbf{d}_{k-1}^i]\tilde{\mathbf{M}}^{(k)}\dot{\mathbf{u}} \tag{2.56}$$

$$\ddot{\mathbf{p}}(u) = [\mathbf{p}_i \ \mathbf{d}_1^i \ \mathbf{d}_2^i \ \dots \ \mathbf{d}_{k-1}^i]\tilde{\mathbf{M}}^{(k)}\ddot{\mathbf{u}} \tag{2.57}$$

An efficient method for calculating these derivatives for Lie groups is proposed in [34] and their computational package is used in this work whenever a B-spline interpolation is calculated.

2.3 Factor Graphs

In recent years, factor graphs have become prevalent in the SLAM domain. SLAM typically involves optimizing a set of 3D poses that represent the trajectory of a robot. The information used to perform this optimization usually comes from observing distances and angles to landmarks in the environment, obtaining pose constraints. All these observations carry their own uncertainty, and therefore it is common to optimize the robot and landmark poses using a MAP (maximum a posteriori) formulation.

An important insight into the structure of these problems lead to the emergence of factor graphs as a modeling tool: the locality property. Although there are many variables in the SLAM problem, most of them only depend on a small subset of other variables and measurements. For example, a odometry estimate relates two consecutive poses in a trajectory that might contain hundreds of them.

Factor graphs contain *variables* and *factors*. Variables are quantities that will be estimated and factors represent constraints on these variables. An edge always connects a variable and a factor, indicating that the factor depends on that variable. The locality property allows this graph to be relatively *sparse*, and in this case specific techniques are used to solve the underlying optimization problem.

2.3.1 Definition

A factor graph is a bipartite graph $\mathcal{F} = \{\mathcal{U}, \mathcal{V}, \mathcal{E}\}$ with two types of nodes: factors $\phi_i \in \mathcal{U}$ and variables $x_j \in \mathcal{V}$. Additionally, edges $e_{ij} \in \mathcal{E}$ connect a factor ϕ_i and variable x_j. Then, a factor graph F defines the factorization

$$\phi(X) = \prod_i \phi_i(X_i) \tag{2.58}$$

where X_i are all the variables that connect to ϕ_i. Typically, the factors $\phi(X_i)$ have the form

$$\phi_i(X_i) \propto \exp\left(-\frac{1}{2}\|h_i(X_i) \ominus z_i\|^2_{\Sigma_i}\right) \tag{2.59}$$

where h_i is a measurement prediction function that takes a subset of variables X_i and outputs the expected measurement. z_i is the actual measurement received and Σ_i is the covariance of the zero-mean Gaussian noise involved in the measurement function. Applying the logarithm to Eq. 2.59 and replacing it in Eq. 2.58 allows

us to understand the equivalency between the factor graph and a sum of nonlinear least-squares

$$\log(\phi(X)) = \sum_i r_i(X_i),\ r_i := -\frac{1}{2}\|h_i(X_i) \ominus z_i\|^2_{\Sigma_i} \tag{2.60}$$

2.3.2 Pose Graph Optimization

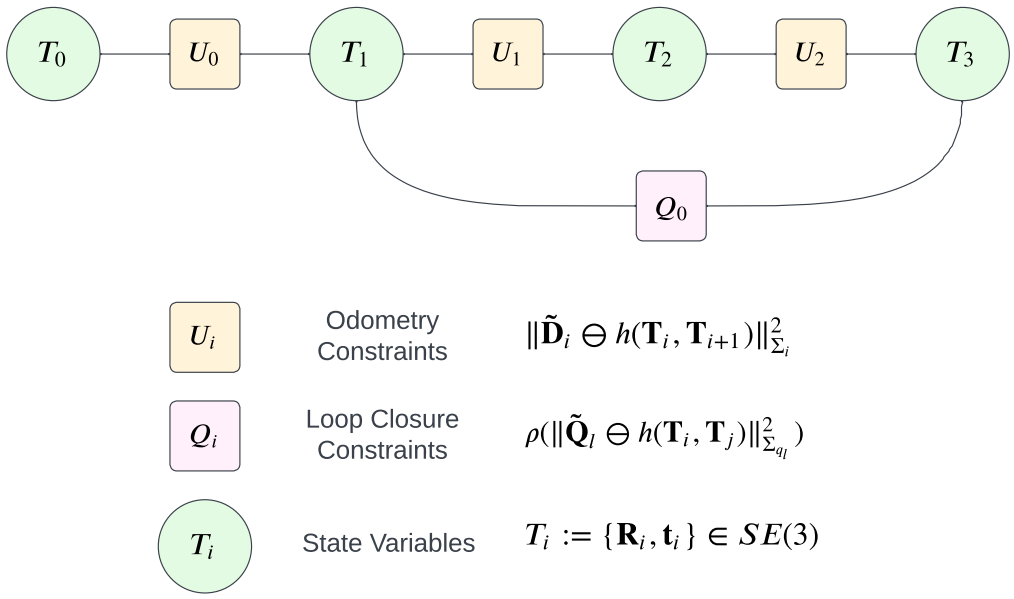


Figure 2.1: PoseSLAM Factor Graph

PoseSLAM is an instance of the SLAM problem where only the robot's trajectory is optimized. When it is solved through the factor graph formulation, it gives rise to the Pose Graph Optimization (PGO) problem. The estimated variables are poses $\mathbf{T}_i = \{\mathbf{R}_i, \mathbf{t}_i\} \in SE(3)$ and the set of variables is $X := \{x_i := \mathbf{T}_i\}$. Typically, there are two factor types. Both involve the relative pose measurement prediction function $h(X)$.

$$h(\mathbf{T}_i, \mathbf{T}_j) = \mathbf{T}_i^{-1}\mathbf{T}_j \tag{2.61}$$

The factor types are:

- Odometry Factors U: Relates two consecutive pose variables $\mathbf{T}_i$ and $\mathbf{T}_{i+1}$ with a relative pose measurement $\tilde{\mathbf{D}}_i$. The measurement model is:

$$\tilde{\mathbf{D}}_i = h(\mathbf{T}_i, \mathbf{T}_{i+1}) \oplus \xi_i, \quad \xi_i \sim \mathcal{N}(\mathbf{0}, \Sigma_i) \tag{2.62}$$

  It adds a residual to the optimization problem in the form

$$u_i(\mathbf{T}_i, \mathbf{T}_{i+1}) = \|\tilde{\mathbf{D}}_i \ominus h(\mathbf{T}_i, \mathbf{T}_{i+1})\|^2_{\Sigma_i} \tag{2.63}$$

  where Σ_i is the covariance of the measurement noise ξ_i that affects *this specific relative keypose measurement*.

- Loop Closure Factors Q: Relates arbitrary pose variables $\mathbf{T}_i$ and $\mathbf{T}_j$ with a relative loop measurement $\mathbf{Q}$. The naive measurement model would be:

$$\tilde{\mathbf{Q}}_l = h(\mathbf{T}_i, \mathbf{T}_j) \oplus \xi_{q_l}, \quad \xi_{q_l} \sim \mathcal{N}(\mathbf{0}, \Sigma_{q_l}) \tag{2.64}$$

  It adds a residual to the optimization in the form

$$q_l(\mathbf{T}_i, \mathbf{T}_j) = \|\tilde{\mathbf{Q}}_l \ominus h(\mathbf{T}_i, \mathbf{T}_j)\|^2_{\Sigma_{q_l}} \tag{2.65}$$

  where Σ_{q_l} is the covariance of the loop closure measurement noise ξ_{q_l} that affects this loop closure pose measurement. However, loop closure constraints are known to contain a large amount of outliers. Therefore, it is common practice to use a M-estimator. In this case, the noise ξ_{q_l} is not Gaussian anymore, and would be modeled with another distribution such as Cauchy [23, 39]. The residual is then modified to:

$$q_l(\mathbf{T}_i, \mathbf{T}_j) = \rho(\|\tilde{\mathbf{Q}}_l \ominus h(\mathbf{T}_i, \mathbf{T}_j)\|^2_{\Sigma_{q_l}}) \tag{2.66}$$

  where $\rho(.)$ is a robust loss function that down-weighs large error values, preventing outliers from having a disproportional effect on the optimization result.

Fig. 2.1 shows a factor graph example containing all the aforementioned elements. Aggregating the residuals leads to the PoseSLAM minimization problem:

$$X^* = \underset{X}{\operatorname{argmin}} \sum_{i=0}^{|U|-1} u_i + \sum_{l=0}^{|Q|-1} q_l \tag{2.67}$$

which resembles Eq. 2.60, without the $-1/2$ factor that is not important for the minimization. This is a sum of nonlinear least-squares minimization problem. It is typically solved by a) choosing an initial solution guess, b) linearizing the equations around the current solution point and c) calculated the solution increment by inverting

the equations. The presence of a robust loss slightly modifies this basic algorithm by deploying Iteratively Reweighted Least Squares techniques (IRLS). Using the factor graph formulation is helpful specifically because it exploits the problem sparsity with dedicated techniques. For more details into factor graph theory and its applicability to robotic estimation problems, see [15] and [13]. There are many computational packages readily available nowadays to tackle this problem. In this work, we have used GTSAM [14].

2.4 ScanContext

Place recognition (PR) is the mechanism by which a SLAM algorithm may understand the topology of the environment. It allows the robot to realize it has visited the same place multiple times. Without it, the robot will assume it is always exploring new territory.

The PR problem is a form of long-term data association. It builds a *database* from sensor measurements as the robot traverses the environment. Each measurement z_t is considered a *place* when combined with time t and location l_t. The goal is, given a *query* place, decide if the robot has already visited it, and return an existing entry in the place database that matches the query place. In the implementation side, the criteria for *same place* is usually defined within a Euclidian distance threshold.

There are two steps involved in PR: database insertion and database retrieval. Insertion is the process where a sensor measurement is typically encoded into a more compact representation. Common sensors used in this step are LIDAR and cameras, and suffer from noise and large raw data size. The compact representation is called a *descriptor* $\mathbf{f}_t = f(z_t)$. Retrieval relies on a distance function $D(\mathbf{f}_i, \mathbf{f}_j)$ that allows the query database entry to be efficiently compared to all other entries in the database, returning a match if one exists in a timely manner. When this distance is lower than a predefined threshold ϵ, the match is obtained.

This work uses the ScanContext (SC) descriptor[20]. It is a LIDAR-only place recognition system that was originally designed for outdoor use, specifically for self-driving cars. Only a small modification was needed to adapt it to our underground use-case, as will be shown in Chapter 4. Here, we briefly describe how the insertion and retrieval steps work for SC.

2.4.1 Database Insertion

The idea of the SC descriptor is to encode the raw pointcloud data into a top-down "image". Consider the *scan* $C := \{\mathbf{p}_i\}$ as a set of unordered points $\mathbf{p}_i = [x_i, y_i, z_i]^\top$ produced by the LIDAR sensor. SC divides it into azimuthal and radial bins P_{ij}, with $i \in \{1, \ldots, N_r\}$ and $j \in \{1, \ldots, N_s\}$ centered around the sensor origin, along the XY plane. N_s and N_r are the number of sectors (radial divisions) and rings (azimuthal divisions). For example, $N_s = 60$ implies that each sector comprises an angle of $360^o/60 = 6^o$. Each ring has a thickness of L_{max}/N_r, where L_{max} is the maximum sensing range of the LIDAR sensor. Fig. 2.2a illustrates the partitioning mechanism.

For each bin P_{ij}, a single real value is chosen to represent all the bin elements. The SC original selection function is:

$$s(P_{ij}) = \max_{\mathbf{p} \in P_{ij}} z(\mathbf{p}) \tag{2.68}$$

where $z(\mathbf{p})$ is a function that extracts the z-value from point $\mathbf{p}$. This selection function simply chooses the highest height value of any point in the bin to build its descriptor. The final descriptor, as shown in Fig. 2.2b is an image I such that

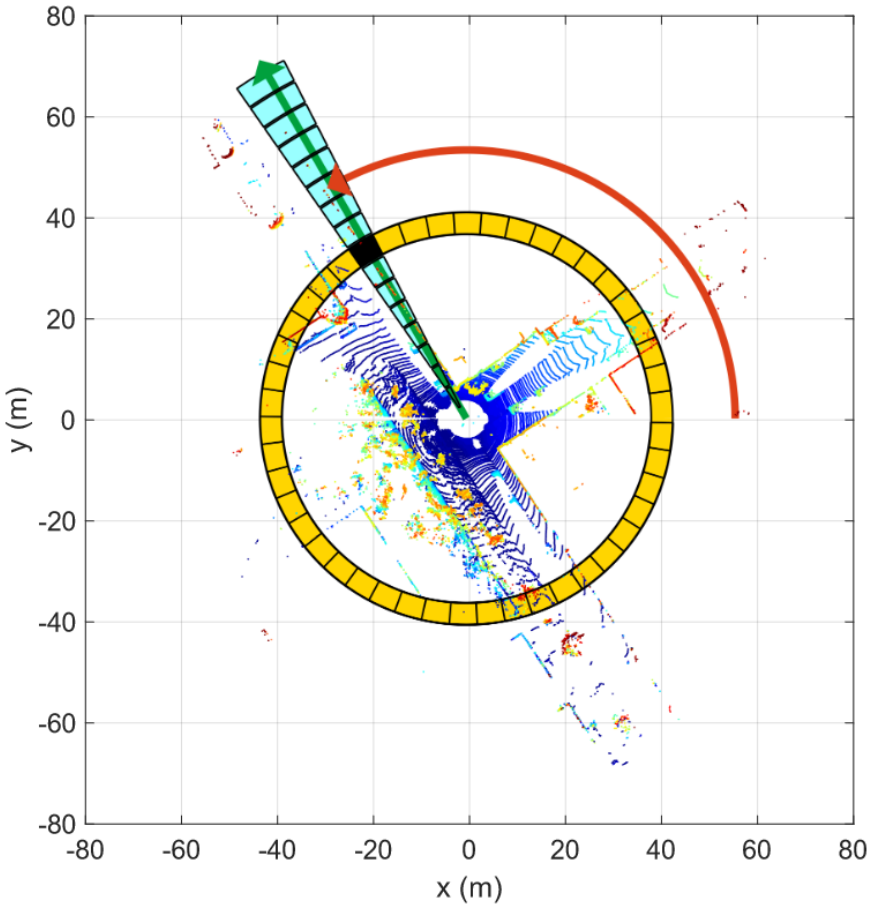
$$I = (a_{ij}) \in \mathbb{R}^{N_r \times N_s}, a_{ij} = s(P_{ij}) \tag{2.69}$$

2.4.2 Database Retrieval

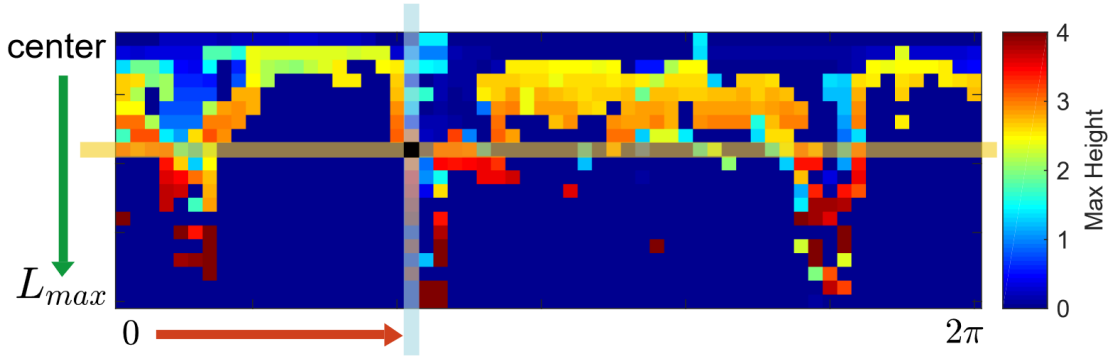
The distance function takes in the query descriptor image I^q and a given element from the database I^c and outputs a disparity measure. For SC, this is done by considering each sector at a time, and obtaining the cosine distance between the vectors c_j^q and c_j^c, which are the $j-th$ column vector of each image. These columns vectors represent the heights along all the rings for a single sector. The cosine distance is similarity value between -1 and 1. The distance function subtracts the cosine distance from 1, making it a proper disparity measure, and averages it over all columns (sectors). Its final form is

$$d(I^q, I^c) = \frac{1}{N_s} \sum_{j=1}^{N_s} \left(1 - \frac{c_j^q \cdot c_j^c}{\|c_j^q\| \|c_j^c\|}\right) \tag{2.70}$$

This distance function is sensitive to the orientation of the robot when data acquisition happened. To remedy this, the distance is taken along all possible column-shifted descriptor images and the minimum among them is returned as the



(a) Partitioning of the pointcloud in sectors and rings, using a top-down view.



(b) Scan Context descriptor generated. An image of size $N_r \times N_s$, where each pixel value is the maximum height of all points in the associated partition.

Figure 2.2: Scan Context Descriptor. Reproduced from Kim *et al.* [19]. Copyright © 2018 IEE.

final distance. This process has the byproduct of roughly estimating the yaw offset between the two robot poses, which can be used to initialize a more precise pointcloud registration later.

ScanContext uses another encoding mechanism to speed up the comparisons. It uses the occupancy ratio of each ring to build another descriptor. A ring is a row of the image descriptor, and has length equal to the number of sectors. The occupancy ratio $\psi(r_i)$ is the number of non-zero elements of the ring divided by the number of sectors. Then, the **ring key** descriptor $\mathbf{k}$ is built as

$$\mathbf{k} = [\psi(r_1), \ldots, \psi(r_{N_r})] \in \mathbb{R}^{N_r} \tag{2.71}$$

The ring key descriptor is used in the first step of the retrieval. It is used to build a KD tree structure that is searched for the closest N matches in ring key space. These matches are the only ones where the full distance function from Eq. 2.70 is applied, and the final match is obtained by taking the smallest distance among them. More details about ScanContext can be found in [19, 20, 21].

Chapter 3

IMU-centric Trajectory Uncertainty Estimation

3.1 Introduction

A common process in SLAM systems is merging multiple sources of information, specifically trajectory information. Typically, there are redundant sensor modalities that can serve as input for trajectory estimation, such as cameras and lidar. In order to fuse these information successfully in a probabilistic framework, it is important to obtain an estimation of the uncertainty associated with each estimate. This is usually modeled as a Gaussian distribution characterized by a covariance and a mean estimate. However, many visual- and lidar-odometry methods such as SuperOdometry and LOAM are single point estimates and do not provide this covariance information.

For the methods that do provide the uncertainty estimation, the calculation is typically custom tailored to that method and depends on the environment[28, 36]. For optimization-based methods, the Fisher information matrix may be used to build an uncertainty estimation[35]. In LIDAR and Visual odometry, the covariance depends on the spatial distribution of features and its ability to constrain all the 6 DOFs.

The key idea used in our solution is to abstract away the particularities of each method and/or sensors and consider only the sequence of estimates generated by it. Thus, this method may work independently of the sensor type, under a reasonable

set of assumptions. A possible downside is that we need to accumulate a number of pose estimates before calculating the uncertainty.

The core insight of the method is to follow the same principle as SuperOdometry and use an IMU-centric approach. Therefore, we use the inertial data as the arbitrator to evaluate different trajectories. To accomplish this, we create a continuous-time trajectory using B-splines from the low-frequency (10Hz or 5Hz) odometric data. We then differentiate this trajectory at the imu timestamps (200Hz) to obtain acceleration and angular velocity estimates. Finally, we obtain a covariance measurement using the sample covariance of the residual between the imu measurements and the values obtained from the splines.

3.2 Problem Definition

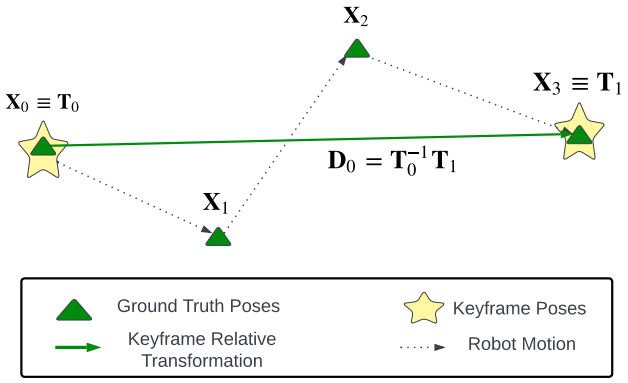
Consider a robot exploring its environment. We define the state of this robot at time t as the pose $\mathbf{X}(t) = \{\mathbf{R}(t), \mathbf{t}(t)\} \in SE(3)$, containing the 3D position $\mathbf{t}(t)$ and orientation $\mathbf{R}(t)$ of its body frame B w.r.t to a world frame W. For convenience, the body frame is defined to coincide with the IMU frame. An odometry (or front-end) algorithm produces a sequence of points estimates $\tilde{\mathbf{X}}_i = \tilde{\mathbf{X}}(i\Delta t)$ of the true pose in regular intervals Δt. An inertial measuerement unit (IMU) uses an accelerometer and a gyroscope to produce estimates of its linear acceleration $\tilde{\mathbf{a}}(t) \in \mathbb{R}^3$ and the angular velocity $\tilde{\boldsymbol{\omega}}(t) \in \mathbb{R}^3$ of frame B relative to W. Both quantities are expressed in the body frame.

A back-end algorithm consumes the estimates $\tilde{\mathbf{X}}_i$ and uses a *keyframe process* to select specific estimates as *keyposes*, which we write as $\tilde{\mathbf{T}}_k := \text{keyframe}(\tilde{\mathbf{X}}_i, k)$. Each keyframe is associated with its time t_k. The interval between two consecutives keyposes is referred to as the *keyframe window* $K := \{t_k < t < t_{k+1}\}$. The estimated relative transformation $\tilde{\mathbf{D}}_k$ connects two consecutive keyposes such that: $\tilde{\mathbf{D}}_k = \tilde{\mathbf{T}}_k^{-1}\tilde{\mathbf{T}}_{k+1}$. Similarly, the true relative transformation $\mathbf{D}_k$ connects the true robot poses at times t_k and t_{k+1}, such that $\mathbf{D}_k = \mathbf{T}_k^{-1}\mathbf{T}_{k+1}$. Typically, the back-end algorithm requires an indication of how $\tilde{\mathbf{D}}_k$ is related to the true value $\mathbf{D}_k$. This relation is referred to as the *noise model*. In this work, the noise model used is:

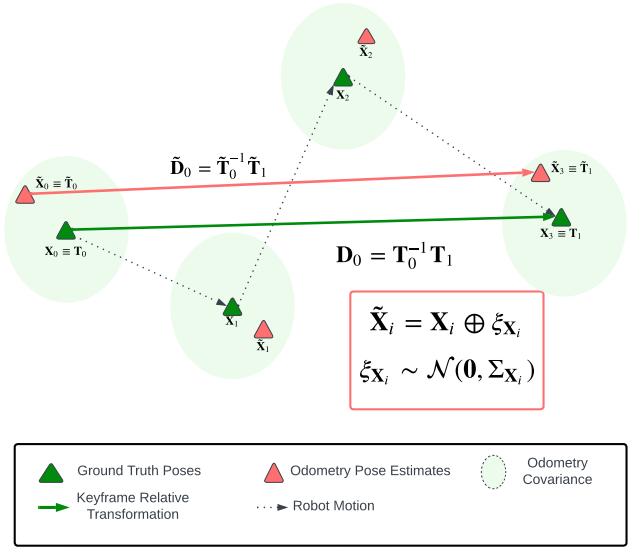
$$\tilde{\mathbf{D}}_k = \mathbf{D}_k \oplus \xi_k = \mathbf{D}_k \ \text{Exp}(\xi_k), \qquad \xi_k \sim \mathcal{N}(\mathbf{0}, \Sigma_k) \tag{3.1}$$

Thus, the relative keypose transformation noise $\xi_k \in \mathbb{R}^6$ follows a zero-mean normal distribution with covariance $\Sigma_k \in \mathbb{R}^{6\times6}$. This noise is defined in the tangent space of $\mathbf{D}_k$ and is mapped onto $SE(3)$ via the exponential map.

We can now define our uncertainty estimation problem as the estimation of the keyframe covariance Σ_k from the sequence of odometry estimates $\{\tilde{\mathbf{X}}_i\}$ and IMU measurements $\{\tilde{\mathbf{a}}(t)\}$ and $\{\tilde{\boldsymbol{\omega}}(t)\}$, with $i\Delta t, t \in K$.

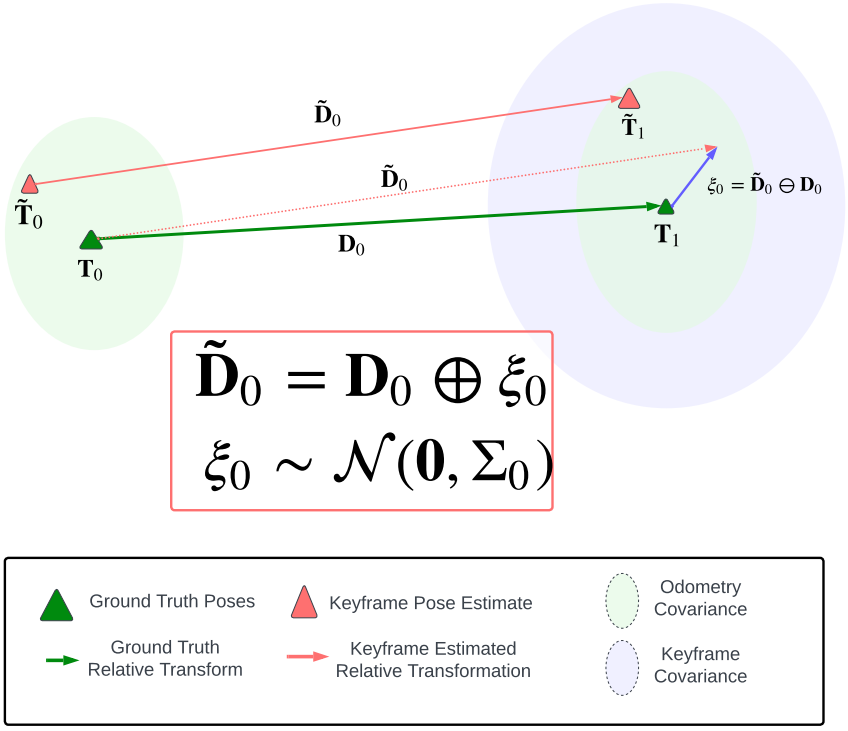


(a) The robot moves in the environment, and its ground truth trajectory is discretized to create sequence $\mathbf{X}_i \in SE(3)$. Some of the poses are selected to be keyposes $\mathbf{T}_i$ and originate *keyframes*. The keyframe relative transform $\mathbf{D}_i$ is relevant to the Pose Graph Optimization.



(b) An odometry algorithm produces pose estimates $\tilde{\mathbf{X}}_i \sim \mathcal{N}(\mathbf{X}_i, \Sigma_{\mathbf{X}_i})$ that are normally distributed around the ground truth. An estimate of the keyframe relative transform $\tilde{\mathbf{D}}_i = \tilde{\mathbf{T}}_i^{-1}\tilde{\mathbf{T}}_{i+1}$ is created from two noisy odometry estimates.

Figure 3.1: Illustration of the Absolute Noise model for odometry estimates.



(c) The keyframe covariance Σ_k characterizes the distribution of the keyframe relative transform error $\xi = \tilde{\mathbf{D}}_k \ominus \mathbf{D}_k$ and is the quantity to be determined in the proposed method.

Figure 3.1: Illustration of the Absolute Noise model for odometry estimates.

3.3 Odometry Absolute Noise Model

For simplicity, we will consider a keyframing processing based on a fixed-time interval. If our odometry estimates are produced with frequency $f = 1/\Delta t$, and our keyframing selects keyposes every T seconds, then we have:

$$\mathbf{T}_k = \mathbf{X}_{Nk}, \quad \text{with} \quad N = \frac{T}{\Delta t} \tag{3.2}$$

The number $N = \frac{T}{\Delta t}$ defines the keyframe window size. Therefore, a keyframe window is comprised of odometry estimates $\{\mathbf{X}_m |\ m \in \{i, i+1, \ldots,\ i+N\}\ \}$, where the first and last poses in the window ($\mathbf{X}_i$ and $\mathbf{X}_{i+N}$) are also keyposes ($\mathbf{T}_k$ and $\mathbf{T}_{k+1}$, respectively). Fig. 3.1a illustrates this keyframing process.

We propose the following noise model for the odometry results:

$$\tilde{\mathbf{X}}_i = \mathbf{X}_i \oplus \xi_{X_i}, \quad \xi_{X_i} \sim \mathcal{N}(\mathbf{0}, \Sigma_{X_i}) \tag{3.3}$$

The noise ξ_{X_i} is defined in the tangent space of $\mathbf{X}_i$. This noise model is typically more appropriate for sensors and/or methods that are able to gauge the absolute position of the robot directly, such as GPS, or Ultra Wide Band sensors. Pure odometry algorithms, on the other hand, are better characterized by an relative noise model, where the error is accrued incrementally. However, we argue that this model is acceptable for a sensor such as Lidar when the *proportion of newly observed environment in each sensor acquisition is small compared to the previously observed section*. This is common when the robot observes the same room with Lidar continously, or explores large areas with slower speeds. This noise model is shown in Fig. 3.1b.

Moreover, we will make the simplifying assumption that the noises ξ_{X_i} are constant throughout a keyframe window. In this case, determining this odometry noise will be enough to determine the keyframe covariance. Consider a given keyframe defined by

the start keypose $\tilde{\mathbf{T}}_0$ and end keypose $\tilde{\mathbf{T}}_1$, the relative transform is

$$\begin{aligned}
\tilde{\mathbf{D}}_0 &= \tilde{\mathbf{T}}_0^{-1}\tilde{\mathbf{T}}_1 && (3.4)\\
&= (\mathbf{T}_0 \oplus \xi_0)^{-1}(\mathbf{T}_1 \oplus \xi_1) && (3.5)\\
&= \ \mathrm{Exp}(-\xi_0)\mathbf{T}_0^{-1}\mathbf{T}_1\ \mathrm{Exp}(\xi_1) && (3.6)\\
&= \ \mathrm{Exp}(-\xi_0)\mathbf{D}_0\ \mathrm{Exp}(\xi_1) && (3.7)\\
&= \mathbf{D}_0\ \mathrm{Exp}(-\mathbf{Adj}_{\mathbf{D}_0}^{-1}\xi_0)\ \mathrm{Exp}(\xi_1) && (3.8)\\
\tilde{\mathbf{D}}_0 &= \mathbf{D}_0\ \mathrm{Exp}(\tau + \mathbf{J}_l(\tau)\xi_1),\ \text{with } \tau = -\mathbf{Adj}_{\mathbf{D}_0}^{-1}\xi_0 && (3.9)
\end{aligned}$$

Now, we can determine the covariance of $\mathbf{D}_0$.

$$\begin{aligned}
Cov(\mathbf{D}_0) &= E\left[(\tau + \mathbf{J}_l(\tau)\xi_1)(\tau + \mathbf{J}_l(\tau)\xi_1)^\top\right] && (3.10)\\
&\approx E\left[\tau\tau^\top\right] + E\left[\mathbf{J}_l\xi_1\xi_1^\top\mathbf{J}_l^\top\right] && (3.11)\\
&\approx \mathbf{Adj}_{\mathbf{D}_0}^{-1}Cov(\mathbf{T}_0)\mathbf{Adj}_{\mathbf{D}_0}^{-\top} + E\left[\mathbf{J}_l\xi_1\xi_1^\top\mathbf{J}_l^\top\right] && (3.12)
\end{aligned}$$

The above expression allows us to obtain the keyframe uncertainty once we have obtained the odometry covariances at the start and end poses that define the keyframe, as shown in Fig. 3.1c. It also agrees with the composition rule proposed in [27], if we consider the noises affecting the keyposes to be uncorrelated. The next section will explain how the proposed method estimates these odometry covariances.

3.4 Odometry Covariances

As explained in the previous section, it is assumed that the odometry estimates $\tilde{\mathbf{X}}_i = \{\tilde{\mathbf{R}}_i, \tilde{\mathbf{t}}_i\} \in SE(3)$ have the following noise model:

$$\tilde{\mathbf{X}}_i = \mathbf{X}_i \oplus \xi_{X_i}, \quad \xi_{X_i} \sim \mathcal{N}(\mathbf{0}, \Sigma_{X_i}) \tag{3.13}$$

The proposed method will estimate the odometry covariance as:

$$\Sigma_{\mathbf{X}_i} = \begin{bmatrix} \Sigma_{\mathbf{t}_i} & 0 \\ 0 & \Sigma_{\mathbf{R}_i} \end{bmatrix} \tag{3.14}$$

This is done considering the covariances of the position $\Sigma_{\mathbf{t}_i} \in \mathbb{R}^{3\times3}$ and orientation $\Sigma_{\mathbf{R}_i} \in \mathbb{R}^{3\times3}$ parts separately. Note that we have applied the simplifying assumption that the position and orientation noises are uncorrelated. First, the IMU gyroscope values are used to determine the rotational uncertainty, and then the accelerometer

is used for the position uncertainty. Finally, they are merged together to produce the desired result.

3.4.1 Rotational Covariance

In this section, we will look at the rotational component separately, such that the relevant odometry model becomes:

$$\tilde{\mathbf{R}}_i = \mathbf{R}_i \oplus \xi_{\mathbf{R}_i}; \quad \xi_{\mathbf{R}_i} \sim \mathcal{N}(\mathbf{0}, \Sigma_{\mathbf{R}_i}); \quad \xi_{\mathbf{R}_i} \in \mathbb{R}^3, \Sigma_{\mathbf{R}_i} \in \mathbb{R}^{3\times 3} \tag{3.15}$$

The first step is to build a B-Spline from the odometry estimates. This creates a spline interpolation function that produces an estimate of the orientation and its derivatives at the same timestamps as the IMU measurements. In this case, the value that we are interested in interpolating with the B-spline is the angular velocity expressed in body frame $\tilde{\boldsymbol{\omega}}_s(t)$. The interpolation function f has the following form, with $\mathbf{R}_i$ being the rotational component of odometry pose $\mathbf{X}_i$:

$$\tilde{\boldsymbol{\omega}}_s(t) = f_{\boldsymbol{\omega}}(\mathbf{R}_i, \ldots, \mathbf{R}_{i+N}, t) \tag{3.16}$$

Remember that B-Spline uses only a subset of the odometry estimates $\mathbf{R}_i$ for the interpolation at any given time. We assume that this estimate of the angular velocity has the following noise model:

$$\tilde{\boldsymbol{\omega}}_s = \boldsymbol{\omega} + \boldsymbol{\epsilon}_{\omega_s}, \quad \boldsymbol{\epsilon}_{\omega_s} \sim N(\mathbf{0}, \Sigma_{\omega_s}) \tag{3.17}$$

where $\boldsymbol{\omega}$ is the true body-frame angular velocity and ϵ_{ω_s} a zero-mean Gaussian noise with covariance $\Sigma_{\omega_s} \in \mathbb{R}^3$.

Now, we turn to the IMU gyroscope noise model. The body-frame angular velocity measurements $\tilde{\boldsymbol{\omega}}_{IMU}(t)$ are affected by bias $\mathbf{b}_\omega \in \mathbb{R}^3$ and white noise $\boldsymbol{\epsilon}_{\omega_{IMU}} \in \mathbb{R}^3$.

$$\tilde{\boldsymbol{\omega}}_{IMU}(t) = \boldsymbol{\omega} + \mathbf{b}_\omega + \boldsymbol{\epsilon}_{\omega_{IMU}}, \quad \boldsymbol{\epsilon}_{\omega_{IMU}} \sim N(\mathbf{0}, \Sigma_{\boldsymbol{\omega}_{IMU}}), \quad \Sigma_{\boldsymbol{\omega}_{IMU}} \in \mathbb{R}^{3\times 3} \tag{3.18}$$

We define a residual $\mathbf{r}_\omega$ and obtain its noise model:

$$\mathbf{r}_{\boldsymbol{\omega}}(t) := \tilde{\boldsymbol{\omega}}_s(t) - \tilde{\boldsymbol{\omega}}_{IMU}(t) + \mathbf{b}_{\boldsymbol{\omega}}(t) \tag{3.19}$$

$$\mathbf{r}_{\boldsymbol{\omega}}(t) = \boldsymbol{\omega}(t) + \boldsymbol{\epsilon}_{\boldsymbol{\omega}_s}(t) - (\boldsymbol{\omega}(t) + \mathbf{b}_{\boldsymbol{\omega}}(t) + \boldsymbol{\epsilon}_{\boldsymbol{\omega}_{IMU}}(t)) + \mathbf{b}_{\boldsymbol{\omega}}(t) \tag{3.20}$$

$$\mathbf{r}_{\boldsymbol{\omega}}(t) = \boldsymbol{\epsilon}_{\boldsymbol{\omega}_s}(t) - \boldsymbol{\epsilon}_{\boldsymbol{\omega}_{IMU}}(t) \tag{3.21}$$

Therefore, the covariances of the residual are:

$$\Sigma_{\mathbf{r}_{\boldsymbol{\omega}}}(t) = \Sigma_{\boldsymbol{\omega}_s}(t) + \Sigma_{\boldsymbol{\omega}_{IMU}} \tag{3.22}$$

$$\Sigma_{\boldsymbol{\omega}_s}(t) = \Sigma_{\mathbf{r}_{\boldsymbol{\omega}}}(t) - \Sigma_{\boldsymbol{\omega}_{IMU}} \tag{3.23}$$

This relation will allow us to later estimate the odometry rotation covariance $\Sigma_{\mathbf{R}_i}$.

3.4.2 Positional Covariance

Now, we handle the positional covariance. First, consider the interpolation function that is able to produce linear acceleration measurements in world frame ${}_{\mathrm{W}}\tilde{\mathbf{a}}_s$ from the odometry position estimates $\tilde{\mathbf{t}}_i$.

$${}_{\mathrm{W}}\tilde{\mathbf{a}}_s(t) = f_{\mathbf{a}}(\tilde{\mathbf{t}}_i, \ldots, \tilde{\mathbf{t}}_{i+N}, t) \tag{3.24}$$

We model the noise on the odometry position estimates as:

$$\tilde{\mathbf{t}}_i = \mathbf{t}_i + {}_{\mathrm{W}}\xi_{\mathbf{t}}, \quad {}_{\mathrm{W}}\xi\mathbf{t} \sim \mathcal{N}(\mathbf{0}, {}_{\mathrm{W}}\Sigma_{\mathbf{t}_i}) \tag{3.25}$$

We make the simplifying assumption that ${}_{\mathrm{W}}\Sigma_{\mathbf{t}_i}$ is constant inside the keyframe window. Note that this is different from the position covariance in Eq. 3.14, which is defined in the local frame. This assumption implies that the position noise distribution *in the world frame* is the same. It is is the result of experimental observation that the environment geometry is determinant in the types of errors that odometry algorithms makes, specially LIDAR-based ones.

We assume that the spline-based estimate of linear acceleration has the following noise model:

$${}_{\mathrm{W}}\tilde{\mathbf{a}}_s = {}_{\mathrm{W}}\mathbf{a} + \boldsymbol{\epsilon}_{\mathbf{a}_s}, \quad \boldsymbol{\epsilon}_{\mathbf{a}_s} \in \mathcal{N}(\mathbf{0}, \Sigma_{\mathbf{a}_s}) \tag{3.26}$$

where ${}_{\mathrm{W}}\mathbf{a}$ is the true world-frame acceleration and $\Sigma_{\mathbf{a}_s} \in \mathbb{R}^{3\times3}$ is the covariance of the spline-based estimate. It will also be necessary to estimate the body-to-world rotation with timestamps that match the IMU data. For this, the B-spline interpolation is again used:

$$\tilde{\mathbf{R}}_s(t) = f_{\mathbf{R}}(\mathbf{R}_i, \ldots, \mathbf{R}_{i+N}, t) \tag{3.27}$$

The IMU accelerometer noise model is:

$${}_{\mathrm{B}}\tilde{\mathbf{a}}_{IMU} = \mathbf{R}^{\top}({}_{\mathrm{W}}\mathbf{a} - {}_{\mathrm{W}}\mathbf{g}) + \mathbf{b}_{\mathbf{a}} + \epsilon_{\mathbf{a}_{IMU}}, \quad \epsilon_{\mathbf{a}_{IMU}} \sim \mathcal{N}(\mathbf{0}, \Sigma_{\mathbf{a}_{IMU}}) \tag{3.28}$$

where ${}_{\mathrm{W}}\mathbf{g} \in \mathbb{R}^3$ is the gravity vector; $\mathbf{b}_{\mathbf{a}} \in \mathbb{R}^3$ is the IMU bias, and $\epsilon_{\mathbf{a}_{IMU}} \in \mathbb{R}^3$ is a zero-mean Gaussian noise with covariance $\Sigma_{\mathbf{a}_{IMU}}$. Similarly as before, we define a

residual $\mathbf{r_a}$ as:

$$ {}_{\mathrm{w}}\mathbf{r_a} = {}_{\mathrm{w}}\tilde{\mathbf{a}}_s - \tilde{\mathbf{R}}_s({}_{\mathrm{B}}\tilde{\mathbf{a}}_{IMU} - \mathbf{b}_a) - {}_{\mathrm{w}}\mathbf{g} \tag{3.29} $$

$$ {}_{\mathrm{w}}\mathbf{r_a} = {}_{\mathrm{w}}\mathbf{a} + \epsilon_{\mathbf{a}_s} - \tilde{\mathbf{R}}_s(\mathbf{R}^\top({}_{\mathrm{w}}\mathbf{a} - {}_{\mathrm{w}}\mathbf{g}) + \mathbf{b_a} + \epsilon_{\mathbf{a}_{IMU}} - \mathbf{b}_a) - {}_{\mathrm{w}}\mathbf{g} \tag{3.30} $$

$$ {}_{\mathrm{w}}\mathbf{r_a} = {}_{\mathrm{w}}\mathbf{a} + \epsilon_{\mathbf{a}_s} - \tilde{\mathbf{R}}_s(\mathbf{R}^\top({}_{\mathrm{w}}\mathbf{a} - {}_{\mathrm{w}}\mathbf{g}) + \epsilon_{\mathbf{a}_{IMU}}) - {}_{\mathrm{w}}\mathbf{g} \tag{3.31} $$

$$ {}_{\mathrm{w}}\mathbf{r_a} \approx \epsilon_{\mathbf{a}_s} - \tilde{\mathbf{R}}_s\epsilon_{\mathbf{a}_{IMU}} \tag{3.32} $$

Therefore, the covariance of the residual is:

$$ {}_{\mathrm{w}}\Sigma_{\mathbf{r_a}} \approx \Sigma_{\mathbf{a}_s} + \tilde{\mathbf{R}}_s\Sigma_{\mathbf{a}_{IMU}}\tilde{\mathbf{R}}_s^\top \tag{3.33} $$

$$ \Sigma_{\mathbf{a}_s} \approx {}_{\mathrm{w}}\Sigma_{\mathbf{r_a}} + \tilde{\mathbf{R}}_s\Sigma_{\mathbf{a}_{IMU}}\tilde{\mathbf{R}}_s^\top \tag{3.34} $$

In this derivation, we considered the approximation $\tilde{\mathbf{R}}_s\mathbf{R}^\top \approx \mathbf{I}$ and ignored the extra noise term that arises when the noise model of $\tilde{\mathbf{R}}_s$ is considered.

3.4.3 Recovering Pose Covariances from Spline Derivative Covariances

In the proposed method, B-Splines are used to provide estimates of orientation, linear acceleration and angular velocity at the same timestamps as IMU measurements, from sparse odometry estimates. We want to derive a relationship between the uncertainty of these splines derivatives and the uncertainty in the odometry estimates. In order to this, consider the matrix representation of a cubic B-spline when it is desired to obtain an interpolation at a given time $t \in [t_i, t_{i+1})$. The control points are $\mathbf{p}_{i+k}$ with $k \in \{0, 1, 2, 3\}$. We use an uniform representation for time such that $s(t) = (t - t_0)/\Delta t$. After doing this, the control points are transformed into $\mathbf{p}_n$ with $n \in \{0, 1, 2, 3\}$. Then, we define $u(t) = s(t) - i$ as the normalized time elapsed since the start of the segment $[t_i, t_{i+1})$ and start using u as the time variable. For instance, if $t = t_i$ then $u = 0$ and as approaches $t = t_{i+1}$ then u approaches 1. It will be useful to create a column vector $\mathbf{u} = [1, u, u^2, u^3]^\top$. Then, the interpolate $\mathbf{p}(u)$ becomes:

$$ \mathbf{p}(u) = \begin{bmatrix} \mathbf{p}_0 & \mathbf{p}_1 & \mathbf{p}_2 & \mathbf{p}_3 \end{bmatrix} \mathbf{M}^{(4)} \begin{bmatrix} 1 \\ u \\ u^2 \\ u^3 \end{bmatrix} \tag{3.35} $$

where $M^{(4)}$ is a 4×4 blending matrix with pre-defined entries. Note from this structure that obtaining a derivative is as easy as differentiating vector $\mathbf{u}$, such that $\dot{\mathbf{u}} = [0, 1, 2u, 3u^2]^\top$ and $\ddot{\mathbf{u}} = [0, 0, 2, 6u]^\top$ and replacing it appropriately in the equation above. Now, we can combine the two matrices on the right to obtain:

$$\mathbf{p}(u) = [\mathbf{p}_0 \quad \mathbf{p}_1 \quad \mathbf{p}_2 \quad \mathbf{p}_3]\mathbf{c}(u) \tag{3.36}$$

$$\mathbf{p}(u) = \sum_n c_n(u)\mathbf{p}_n \tag{3.37}$$

Consider that the control points are actually estimates with the identical noise model such that $\tilde{\mathbf{p}}_n \sim \mathcal{N}(\mathbf{p}_n, \Sigma_\mathbf{p})$. Then, we can obtain the covariance of the interpolate at time u:

$$Cov(\tilde{\mathbf{p}}(u)) = \sum_n c_n^2(u)\Sigma_\mathbf{p} \tag{3.38}$$

$$= \Sigma_\mathbf{p} \sum_n c_n^2(u) \tag{3.39}$$

This shows that the covariance of the interpolate will change in the interval $u \in [0, 1)$. We obtain the average covariance $\overline{\text{Cov}}(\tilde{\mathbf{p}})$ in that interval by:

$$\overline{\text{Cov}}(\tilde{\mathbf{p}}) = \int_0^1 \Sigma_\mathbf{p} \sum_n c_n^2(u)du \tag{3.40}$$

$$= \Sigma_\mathbf{p} \int_0^1 \sum_n c_n^2(u)du \tag{3.41}$$

$$= \Sigma_\mathbf{p} C \tag{3.42}$$

where C is the resulting constant from the integral. Note that if the interpolate is a derivative of the control points (such as velocity or acceleration), then the only difference is that we would replace functions $c_n(u)$ with their derivatives. But the final result of the integral will still be a single constant. Now, we examine the concrete cases where the interpolated values of interest are the angular velocities $\omega_s(t)$ and the linear acceleration $\mathbf{a}_s(t)$.

First, consider the spline-based interpolation of the angular velocity, as presented in Eq. 3.16. In this case $\mathbf{p} := \boldsymbol{\omega}_s$. Consider that there are M angular velocity measurements in the keyframe window. In the following derivation, we omit the index i, applying the assumption that the body-frame rotational covariance is constant

inside the keyframe window ($\Sigma_{\mathbf{R}_i} \equiv \Sigma_{\mathbf{R}}$). Thus, we can write:

$$\Sigma_{\mathbf{p}} C = \overline{\mathrm{Cov}}(\tilde{\mathbf{p}}) \tag{3.43}$$

$$\Sigma_{\mathbf{R}} C_{\mathbf{R}} = \overline{\mathrm{Cov}}(\tilde{\boldsymbol{\omega}}_s) \tag{3.44}$$

$$\approx \frac{1}{M} \sum_t \Sigma_{\boldsymbol{\omega}_s}(t) \tag{3.45}$$

$$\approx \frac{1}{M} \sum_t \left(\Sigma_{\mathbf{r}_{\boldsymbol{\omega}}}(t) - \Sigma_{\boldsymbol{\omega}_{IMU}}\right) \tag{3.46}$$

$$\approx \frac{1}{M} \sum_t \left(\mathbf{r}_{\boldsymbol{\omega}}(t)\mathbf{r}_{\boldsymbol{\omega}}(t)^\top - \Sigma_{\boldsymbol{\omega}_{IMU}}\right) \tag{3.47}$$

$$\Sigma_{\mathbf{R}} \approx \frac{1}{C_{\mathbf{R}} M} \sum_t \left(\mathbf{r}_{\boldsymbol{\omega}}(t)\mathbf{r}_{\boldsymbol{\omega}}(t)^\top - \Sigma_{\boldsymbol{\omega}_{IMU}}\right) \tag{3.48}$$

For the case of linear acceleration, we now consider that the *world-frame* positional covariance is constant inside the keyframe window, such that ${}_{\mathrm{w}}\Sigma_{\mathbf{t}_i} \equiv {}_{\mathrm{w}}\Sigma_{\mathbf{t}}$:

$$\Sigma_{\mathbf{p}} C = \overline{\mathrm{Cov}}(\tilde{\mathbf{p}}) \tag{3.49}$$

$${}_{\mathrm{w}}\Sigma_{\mathbf{t}} C_{\mathbf{t}} = \overline{\mathrm{Cov}}(\tilde{\mathbf{a}}_s) \tag{3.50}$$

$$\approx \frac{1}{M} \sum_t \Sigma_{\mathbf{a}_s}(t) \tag{3.51}$$

$$\approx \frac{1}{M} \sum_t \left({}_{\mathrm{w}}\Sigma_{\mathbf{r}_{\mathbf{a}}} + \tilde{\mathbf{R}}_s(t)\Sigma_{\mathbf{a}_{IMU}}\tilde{\mathbf{R}}_s(t)^\top\right) \tag{3.52}$$

$$\approx \frac{1}{M} \sum_t \left({}_{\mathrm{w}}\mathbf{r}_{\mathbf{a}}(t){}_{\mathrm{w}}\mathbf{r}_{\mathbf{a}}(t)^\top + \tilde{\mathbf{R}}_s(t)\Sigma_{\mathbf{a}_{IMU}}\tilde{\mathbf{R}}_s(t)^\top\right) \tag{3.53}$$

$${}_{\mathrm{w}}\Sigma_{\mathbf{t}} \approx \frac{1}{C_{\mathbf{t}} M} \sum_t \left({}_{\mathrm{w}}\mathbf{r}_{\mathbf{a}}(t)\ {}_{\mathrm{w}}\mathbf{r}_{\mathbf{a}}(t)^\top + \tilde{\mathbf{R}}_s(t)\Sigma_{\mathbf{a}_{IMU}}\tilde{\mathbf{R}}_s(t)^\top\right) \tag{3.54}$$

3.4.4 Transformation to Body-Frame and Covariance Composition

The final part of our method assures we are expressing the uncertainty correctly as in the format shown in Eq. 3.3 and Eq. 3.14:

$$\tilde{\mathbf{X}} = \mathbf{X} \oplus \xi_X \tag{3.55}$$

$$\begin{bmatrix} \tilde{\mathbf{R}} & \tilde{\mathbf{t}} \\ \mathbf{0} & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix} \operatorname{Exp}(\xi_X) \tag{3.56}$$

$$\begin{bmatrix} \tilde{\mathbf{R}} & \tilde{\mathbf{t}} \\ \mathbf{0} & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R} \operatorname{Exp}(\xi_{\mathbf{R}}) & \mathbf{t} + \mathbf{R}\mathbf{V}(\xi_{\mathbf{R}})\xi_{\mathbf{t}} \\ \mathbf{0} & 1 \end{bmatrix} \tag{3.57}$$

It is possible to recognize that the noise model as shown in Eq. 3.57 is exactly the same as described in Eq. 3.15. Therefore, the previously obtained $\Sigma_{\mathbf{R}}$ is already in the format it needs to be.

For the translational part, note that its noise model was previously defined as $\tilde{\mathbf{t}} = \mathbf{t} + {}_{\mathrm{w}}\xi_{\mathbf{t}}$. Comparing this with Eq. 3.57, then ${}_{\mathrm{w}}\xi_{\mathbf{t}} = \mathbf{R}\mathbf{V}(\xi_{\mathbf{R}})\xi_{\mathbf{t}}$. From this, it is possible to calculate the body-frame covariance $\Sigma_{\mathbf{t}}$ which results in:

$$\Sigma_{\mathbf{t}} = \mathbf{R}^{\top}{}_{\mathrm{w}}\Sigma_{\mathbf{t}}\mathbf{R} \tag{3.58}$$

Defining the error in the world frame is equivalent to considering left-$\oplus$ operation. The Adjoint can then be used to move it to the right-side such that: $\tilde{\mathbf{X}} = \xi_X \oplus \mathbf{X} = \mathbf{X} \oplus (\mathbf{Adj}_{\mathbf{X}}^{-1}\xi_X)$. And this produces the same result as Eq. 3.58.

Therefore, the covariance for any odometry estimate in the keyframe window may be written as:

$$\Sigma_{\mathbf{X}_i} = \begin{bmatrix} \Sigma_{\mathbf{R}_i} & \mathbf{0} \\ \mathbf{0} & \mathbf{R}_i^{\top}{}_{\mathrm{w}}\Sigma_{\mathbf{t}}\mathbf{R}_i \end{bmatrix} \tag{3.59}$$

Then, the keyframe covariance is given by Eq. 3.12, with the starting keyframe pose $\tilde{\mathbf{T}}_0 = \tilde{\mathbf{X}}_i$ and the endpoint one $\tilde{\mathbf{T}}_1 = \tilde{\mathbf{X}}_{i+N}$. With an abuse of notation, let's denote $\tilde{\mathbf{D}}_0 = \tilde{\mathbf{T}}_0^{-1}\tilde{\mathbf{T}}_1 \equiv \{\mathbf{R}_{01}, \mathbf{t}_{01}\}$ and $\mathbf{Adj}_{\mathbf{D}_0} = \begin{bmatrix} \mathbf{R}_{01} & [\mathbf{t}_{01}]_{\times}\mathbf{R}_{01} \\ 0 & \mathbf{R}_{01} \end{bmatrix}$.

$$Cov(\mathbf{D}_0) \approx \mathbf{Adj}_{\mathbf{D}_0} Cov(\mathbf{T}_0) \mathbf{Adj}_{\mathbf{D}_0}^\top + E\left[\mathbf{J}_l \xi_1 \xi_1^\top \mathbf{J}_l^\top\right] \tag{3.60}$$

$$\Sigma_{\mathbf{D}_0} \approx \mathbf{Adj}_{\mathbf{D}_0} \begin{bmatrix} \Sigma_{\mathbf{R}} & \mathbf{0} \\ \mathbf{0} & \mathbf{R}_{0\,\mathrm{w}}^\top \Sigma_{\mathbf{t}} \mathbf{R}_0 \end{bmatrix} \mathbf{Adj}_{\mathbf{D}_0}^\top + E\left[\mathbf{J}_l \xi_1 \xi_1^\top \mathbf{J}_l^\top\right] \tag{3.61}$$

$$\approx \mathbf{Adj}_{\mathbf{D}_0} \begin{bmatrix} \Sigma_{\mathbf{R}} & \mathbf{0} \\ \mathbf{0} & \mathbf{R}_{0\,\mathrm{w}}^\top \Sigma_{\mathbf{t}} \mathbf{R}_0 \end{bmatrix} \mathbf{Adj}_{\mathbf{D}_0}^\top + \Sigma_{\mathbf{T}_1} \tag{3.62}$$

$$\approx \mathbf{Adj}_{\mathbf{D}_0} \begin{bmatrix} \Sigma_{\mathbf{R}} & \mathbf{0} \\ \mathbf{0} & \mathbf{R}_{0\,\mathrm{w}}^\top \Sigma_{\mathbf{t}} \mathbf{R}_0 \end{bmatrix} \mathbf{Adj}_{\mathbf{D}_0}^\top + \begin{bmatrix} \Sigma_{\mathbf{R}} & \mathbf{0} \\ \mathbf{0} & \mathbf{R}_{1\,\mathrm{w}}^\top \Sigma_{\mathbf{t}} \mathbf{R}_1 \end{bmatrix} \tag{3.63}$$

This completes the proposed method, that calculates the keyframe covariance $\Sigma_{\mathbf{D}_0}$ using the IMU gyroscope and linear acceleration measurements and the odometry estimates in the keyframe window.

Chapter 4

SuperLoop

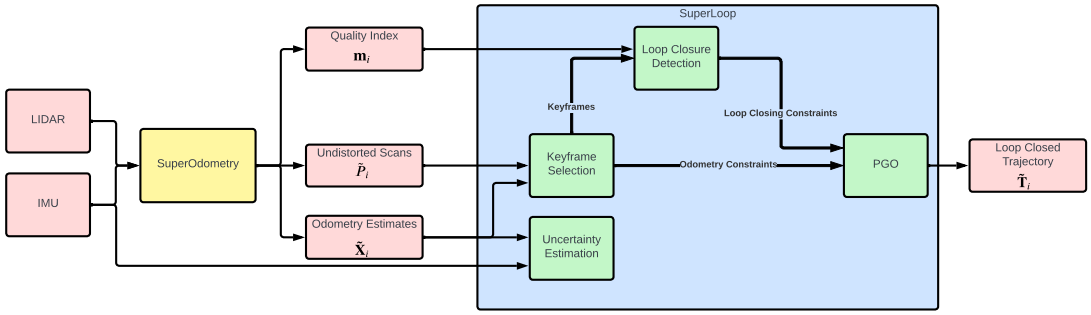


Figure 4.1: Flowchart of SuperLoop components. SuperOdometry takes in the sensor data and produces odometry estimates, quality metric and undistorted pointclouds. These are consumed by SuperLoop, which generates a loop closed trajectory.

A complete modern SLAM system is typically comprised of two components: the front and the back-end. The former is responsible for interpreting the sensor data and extracting useful estimation constraints. The latter uses these constraints to produce the final trajectory and/or map of the environment[6]. The back-end typically is able to refine its own estimates as it receives more constraints (such as in a *loop-closure*), whilst the front-end will likely process each data point only once, providing an initialization point for the back-end.

The aim of this work is to design a SLAM back-end that works in tandem with SuperOdometry as the front-end. This back-end will use a PoseSLAM formulation

[15]. It means the algorithm will only optimize the robot poses. Therefore, no landmark information (lidar or visual) is passed over from SuperOdometry to the back-end. This design choice is done to maintain the principle of abstracting away the exteroceptive data type and using only an IMU sensor to evaluate the information quality. The proposed back-end algorithm has been named SuperLoop.

In the DARPA Subterranean Challenge, Team Explorer SLAM subsystem used SuperOdometry as a front-end and a LOAM-based back-end. That back-end was also formulated as a PoseSLAM. The main contributions of SuperLoop when compared to that algorithm are:

- Use of uncertainty estimation to determine the covariance matrix used in the PoseSLAM factor-graph.
- Ability to perform lidar-based place recognition, via a modified ScanContext algorithm.
- Ability to detect an odometry failure and recovery mechanism.

4.1 Overview

SuperLoop is an extension of SC-LiDAR-SLAM[21], a front-end agnostic LIDAR SLAM system that uses the ScanContext++ algorithm[20] as the main place recognition mechanism. The components of SuperLoop, as shown in Fig. 4.1, are:

1. Keyframe Selection and Uncertainty Estimation.
2. Pose Graph Construction and Optimization (PGO).
3. Loop Closure Detection.
4. Failure Detection.

SuperOdometry provides the following information as input to SuperLoop: a) undistorted LIDAR Scans at 5 Hz, b) pose estimates for each LIDAR scan. c) Geometric degeneracy quality metric for the pose estimates. Additionally, the IMU measurements at 200 Hz are also fed to SuperLoop, since they play an important part in the Uncertainty Estimation.

The main output of SuperLoop is a loop-closed trajectory composed of keyframes, from which a loop-closed pointcloud can be built. A finer trajectory may also be

obtained via interpolation from the 5 Hz odometry estimates.

4.2 KeyFrame Selection and Uncertainty Estimation

At this stage, the 5Hz outputs from SuperOdometry are processed with the goal of obtaining *keyframes*, coarser representations of the data. As explained in Section 3.2 SO provides an odometry estimate $\tilde{\mathbf{X}}_i = \tilde{\mathbf{X}}(t = 0.2i) \in SE(3)$ with a 5Hz frequency. This result is obtained by registering a raw LIDAR scan $\hat{P}_i$ with an internally accumulated map. This raw pointcloud is first undistorted using SO motion prediction mechanism, generating an undistorted estimate of the raw pointcloud $\tilde{P}_i$. Finally, SO extracts line and surface features from $\tilde{P}_i$, and performs a non-linear optimization to align it to the accumulated map. It is common practice to output the *registered* LIDAR scan also. This is not of interest for SuperLoop, however. SuperLoop instead will use the undistorted pointclouds, since they have been motion corrected. Then, it will refine the registration result obtained by SuperOdometry, replacing it with more up-to-date estimates.

The final piece of information that SO relays to SuperLoop is a quality metric $\mathbf{m}_i \in \mathbb{R}^6$. This is a six dimensional vector where each entry is the quality index of a given dimension in $SE(3)$. To obtain this index, SO counts the number of geometric features that contribute to the estimation of each dimension, according to its own methodology. Then, the quality index is the ratio between the counts for each dimension and the maximum count between them. Therefore, the index ranges between 0 and 1.

The keyframing process consists of selecting a subset of the incoming SO data, turning them into keyframes. The actual process in the current implementation of SuperLoop has two criteria: a new keyframe is selected either after the robot moves 2 meters or 10 seconds have passed since the last one. The latter criteria is used to avoid buffering issues in the uncertainty estimation method. This is an area that can be improved in future versions, specially to utilize the uncertainty information itself to select a new keyframe.

Let us denote the odometry, pointcloud and quality index estimate selected

by the keyframing process as $\mathbf{X}_i^*$, $\tilde{P}_i^*$, $\mathbf{m}_i^*$. Then we obtain the new keyframe $F_{k+1} = \{\tilde{\mathbf{T}}_{k+1} = \tilde{\mathbf{X}}_i^*, S_{k+1} = \tilde{P}_i^*, \mathbf{q}_{k+1} = \mathbf{m}_i^*, t_{k+1} = 0.2i\}$. Once the new keyframe is obtained, the uncertainty estimator algorithm will process the sequence of odometry estimates and imu measurements produced since the last keyframe and obtain an covariance $\Sigma_k \equiv \Sigma_{\mathbf{D}_k}$, corresponding to the uncertainty associated with the relative pose $\tilde{\mathbf{D}}_k = \tilde{\mathbf{T}}_k^{-1}\tilde{\mathbf{T}}_{k+1}$, also referred to as the *keyframe relative pose measurement*. The keyframe sequence $\{F_k\}$ with $k \in \{0, \ldots, K\}$ is stored by SuperLoop for its entire execution, including the pointclouds, as they are important for loop closure detections (see Section 4.4).

4.3 Pose Graph Construction

SuperLoop formulates the SLAM problem as a maximum a posteriori (MAP) estimation problem, and solves it via the formalism of pose graphs, a specialization of factor graphs. The unknown variable X estimated is the trajectory of the robot, defined here as the keypose sequence, such that $X = \{\mathbf{T}_0, \ldots, \mathbf{T}_m\}$. The set of measurements is $Z = \{U, Q\}$. where U contains relative poses estimates from SuperOdometry, and Q contains loop closure constraints added by SuperLoop itself. Therefore, the MAP problem solved by SuperLoop is to determine $\mathcal{X}^*$ such that:

$$X^* = \underset{X}{\operatorname{argmax}}\, p(X|Z) \tag{4.1}$$

$$X^* = \underset{X}{\operatorname{argmax}}\, \frac{p(Z|X)p(X)}{p(X)} \tag{4.2}$$

$$X^* = \underset{X}{\operatorname{argmax}}\, p(Z|X)p(X) \tag{4.3}$$

$$X^* = \underset{X}{\operatorname{argmax}}\, p(U|X)p(Q|X)p(X) \tag{4.4}$$

$$X^* = \underset{X}{\operatorname{argmax}}\, p(X) \prod_{k=0}^{K-1} p(u_k|X) \prod_{l=0}^{L-1} p(q_l|X) \tag{4.5}$$

This derivation used the assumption that measurements are independent. The measurement u_k are in fact the keyframe relative pose measurement $\tilde{\mathbf{D}}_k$ and its likelihood is given by:

$$p(u_k|X) \propto \exp(-\frac{1}{2}\|\tilde{\mathbf{D}}_k \ominus \mathbf{T}_k^{-1}\mathbf{T}_{k+1}\|_{\Sigma_k}^2) \tag{4.6}$$

Note that the term $\mathbf{T}_k^{-1}\mathbf{T}_{k+1}$ uses the true state of the robot, and not the estimates produced by SuperOdometry. When solving this problem, value $\tilde{\mathbf{D}}_k$ is a constant input and the state variables are the optimized outputs. Also note that this is where the covariance Σ_k estimated by the proposed method is used.

The measurements q_l are similar, except they relate non-consecutive pose variables. The next section explains how they are obtained. Also, since they are more likely to have outliers, a robust Cauchy loss function[39] is used. In this case the likelihood function is modified to use the weight w_l:

$$p(q_l|X) \propto \exp\left(-\frac{w_l}{2}\|\mathbf{Q}_l \ominus \mathbf{T}_l^{-1}\mathbf{T}_c\|^2_{\Sigma_{loop}}\right) \tag{4.7}$$

The weight is calculated as in [23]. With these likelihood equations, Eq. 4.5 can be further factorized. Note that the argmax function only cares about maximum values, thus the probabilities can be replaced by the negative log likelihood.

$$\mathcal{X}^* = \underset{\mathcal{X}}{\operatorname{argmax}}\, p(X) \prod_{k=0}^{K-1} p(u_k|X) \prod_{l=0}^{L-1} p(q_l|X) \tag{4.8}$$

$$\mathcal{X}^* = \underset{\mathcal{X}}{\operatorname{argmin}} - \log(p(X) \prod_{k=0}^{K-1} p(u_k|X) \prod_{l=0}^{L-1} p(q_l|X)) \tag{4.9}$$

$$\mathcal{X}^* = \underset{\mathcal{X}}{\operatorname{argmin}} \sum_{k=0}^{K-1} \|\tilde{\mathbf{D}}_k \ominus \mathbf{T}_k^{-1}\mathbf{T}_{k+1}\|^2_{\Sigma_k} + \sum_{l=0}^{L-1} w_l \|\mathbf{Q}_l \ominus \mathbf{T}_l^{-1}\mathbf{T}_c\|^2_{\Sigma_{loop}} \tag{4.10}$$

This is a nonlinear least squares problem, due to the nature of pose compositions in SE(3). The structure of this problem is *sparse*, since there are K state variables, and each one connects, i.e. appears in equations together, to few others. In fact, for index k, odometry connections are made with $k-1$ and $k+1$ and possible additional connections are made via loop closures, but these are usually an order of magnitude less than the number of state variables. This sparse structure lends itself to solution via *factor graphs*[15]. SuperLoop uses GTSAM [14] as its factor graph solver.

4.4 Loop Closures

An important part of any complete SLAM system is *loop closure detection*. It helps the robot understand the topology of its environment. Loop Closures are a form of long term data association. It creates a constraint between non-consecutive pose

variables in the factor graph. SuperLoop creates this constraint in two ways: a) Radius search and b) via ScanContext descriptors.

4.4.1 Radius Search Detection

Consider the most recent keyframe $F_k = \{\tilde{\mathbf{T}}_k, S_k, \mathbf{q}_k, t_k\}$. Radius search detection works by looking at the list of previous keyframes and finding the set R of keyframe poses such that the Euclidian distance between their positions and the most recent one is less than a threshold (20 meters in this method). The time elapsed between must also exceed 30 seconds. This is to avoid creating excessive loop closures. Mathematically, $R = \{F_i \mid \|\tilde{\mathbf{t}}_k - \tilde{\mathbf{t}}_i\| < 20 \ \cap \ t_k - t_i > 30\}$. Then, it chooses the oldest element of this set as a loop closure candidate. This creates a larger loop with the goal of correcting the error among more pose variables.

4.4.2 Scan Context Detection

Before detecting loop closures with ScanContext (SC), there is a descriptor creation step. Every time a new keyframe F_k is created, its point cloud S_k is passed onto the SC place descriptor [20]. It works by partitioning the LIDAR scan into sections using polar coordinates. Given a LIDAR point $s = [x, y, z]^\top$, its polar coordinates are: radius $\rho = \sqrt{x^2 + y^2}$ and azimuth $\theta = atan(x, y)$. Then, bins are created by selecting a number of rings (partitions along the radius) and sectors (partitions along the azimuth). In this project, there are 20 rings and 60 sectors, making a total of 1200 partitions. The maximum radius considered is 40 meter. Therefore, the partitions sizes is 2 meter $\times$ 6 degrees. In the original ScanContext formulation, the maximum z-value of all points in each partition is taken as its representative value. Instead, the proposed method caps the maximum height allowed at 2 meters. This is done to avoid the effect of the ceiling in underground environments, which decreases the descriminating power of the descriptor. This process builds an array of size 1200, which is the SC descriptor. It is added to the SC database for future comparisons and retrieval.

Afterwards, the actual loop closure detection occurs. This step is only allowed to happen if no candidates were generated by the radius search. First, the quality index $\mathbf{q}_k$ is evaluated. If the minimum value $q_{min} = min(\mathbf{q}_k)$ is smaller than 0.5, then no

loop closure is allowed to happen. This is important to avoid loops to be detected in geometrically degenerate environments, such as long featureless corridors. These are very common in subterranean man-made environments scenarios. Next, if the quality index is deemed good enough, the ScanContext algorithm will produce a loop closure candidate index c by comparing the current descriptor against the ones in the database. It will also provide an yaw offset estimate.

4.4.3 Validation and Optimization

If the detection steps generate a candidate, then it is necessary to either estimate the relative transform $\tilde{\mathbf{Q}}_k$ between current frame F_k and the candidate F_c or reject this loop. Two sets of pointclouds are used to estimate the transform. The first $\bar{S}_k$ comes from stacking the pointclouds of the 3 latest keyframes S_k, S_{k-2} and S_{k-1}. The second pointcloud $\bar{S}_c$ comes from stacking the pointclouds of up to 2 neighboring keyframes on each side of F_c, such that $\bar{S}_c = S_{c-2} \cap S_{c-1} \cap S_c \cap S_{c+1} \cap S_{c+2}$.

Then, the FastGICP[22] algorithm is used to estimate $\tilde{\mathbf{Q}}_k$ between $\bar{S}_k$ and $\bar{S}_c$. Two tests are applied to validate the result and accept the $k \to c$ loop: if the FastGICP optimization has converged and if the fitness score is lower than a threshold of $0.3m$. The fitness score is the average distance of inlier nearest neighbors between the registered pointclouds.

An additional check is performed when the loop candidate is generated by the ScanContext algorithm. It uses the marginal translational covariance $\Sigma_{\mathbf{t}_k}$ of current frame k as estimated by GTSAM. Then, it calculates the Mahalanobis distance $d = \sqrt{(\mathbf{t}_k - \mathbf{t}_c)^\top \Sigma_{\mathbf{t}_k}^{-1} (\mathbf{t}_k - \mathbf{t}_c)}$. If $d > 3$, the loop candidate is rejected.

If the loop candidate is accepted, then a new factor is added to the factor graph between state variables $\tilde{\mathbf{T}}_k$ and $\tilde{\mathbf{T}}_c$ using the relative transform $\tilde{\mathbf{Q}}_k$ and a robust noise model with the Cauchy M-estimator and a pre-defined covariance $\Sigma_{loop} = diag(0.01, 0.01, 0.01, 0.01, 0.01, 0.01)$. Finally, GTSAM optimizes the updated factor graph and new estimates for the sequence $\{\tilde{\mathbf{T}}_i\}$ are obtained.

4.5 Odometry Failure Handling

SuperLoop proposes a mechanism to handle LIDAR odometry failures. It starts when the uncertainty estimation algorithm signals a failure. In this case, the keyframe relative pose measurement $\tilde{\mathbf{D}}_k$ is disregarded. A replacement measurement is generated by FastGICP considering only S_k and S_{k+1}. Note that no cloud stacking occurs here. The covariance estimate Σ_k is kept. However, the Cauchy M-estimator is used to create the new factor in the graph. This allows the GTSAM optimizer to assign a larger correction to this transform in the future, when it has more information.

Chapter 5

Datasets

This section introduces the datasets used in our experiments. The focus here is on datasets that contain pose estimates and IMU data. These are our uncertainty estimation inputs. We also try to use datasets that contain ground-truth trajectories derived from motion capture devices whenever possible. For long-term operations, the dataset is captured in a representative disaster scenario that we have previously mapped multiple times with LIDAR sensors and obtained control points with Total Stations. In this case, the map itself serves the function of ground-truth.

5.1 EUROC Dataset

The EUROC dataset [5] was generated by ETH-Zurich and contains visual and inertial measurements collected from micro aerial vehicles (MAV). Ground Truth is also obtained for the trajectories using a VICON motion capture system. The data is post-processed to achieve temporal alignment. It also provides accurate ground-truth for the structures, but that is not used in this work. The specific dataset used is the **VICON Room 1-01**, shown in Fig. 5.1.

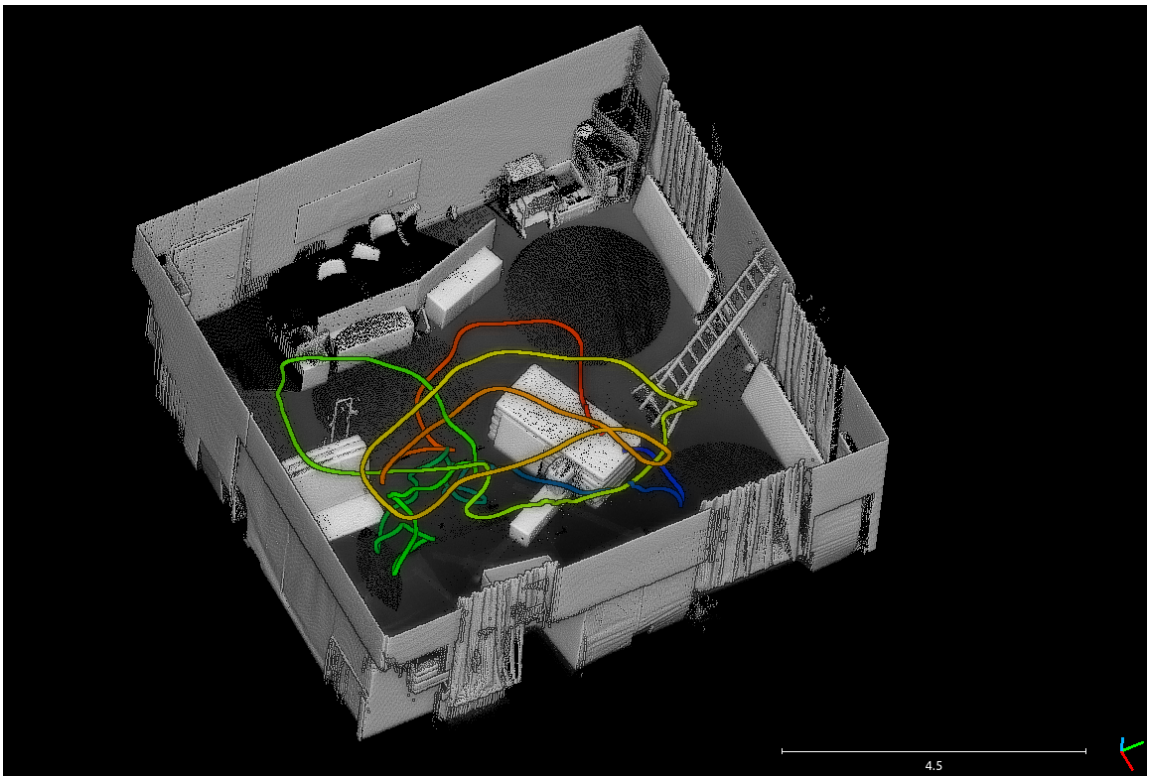
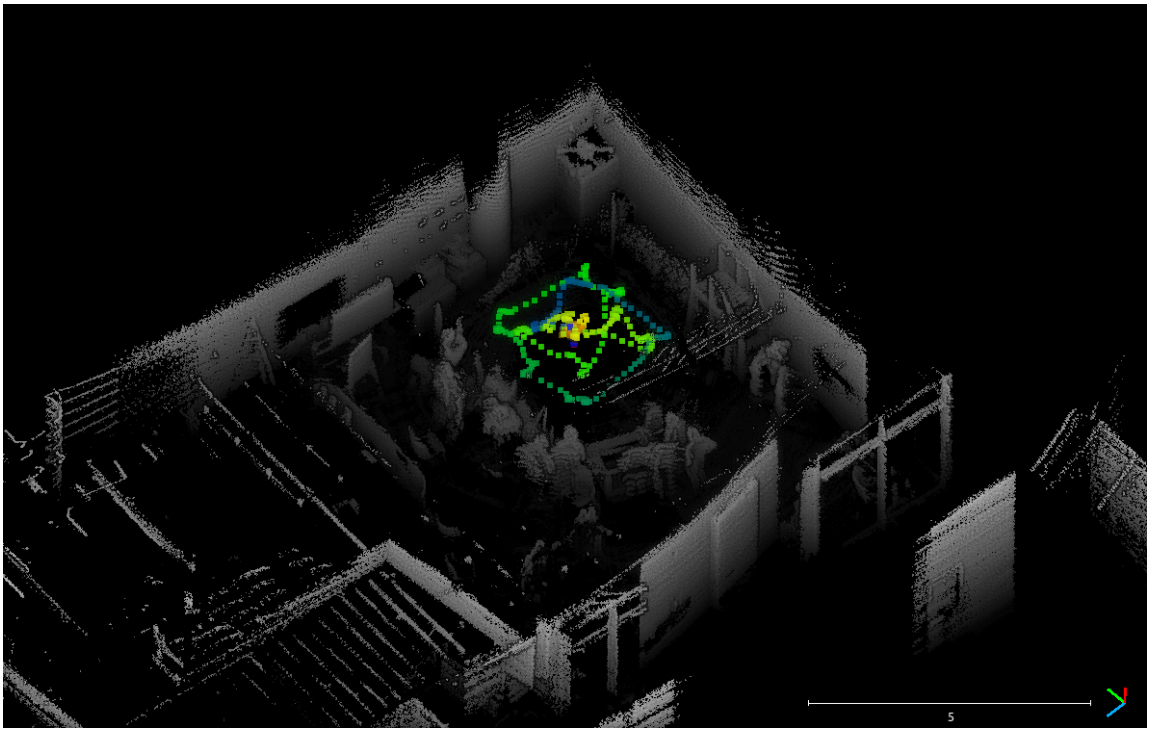


Figure 5.1: Trajectory and structure ground-truth from the VICON Room 1-01 sequence from the Euroc dataset. Trajectory color indicates time.

5.2 Lidar-Visual-Inertial Dataset with Motion Capture Ground-Truth

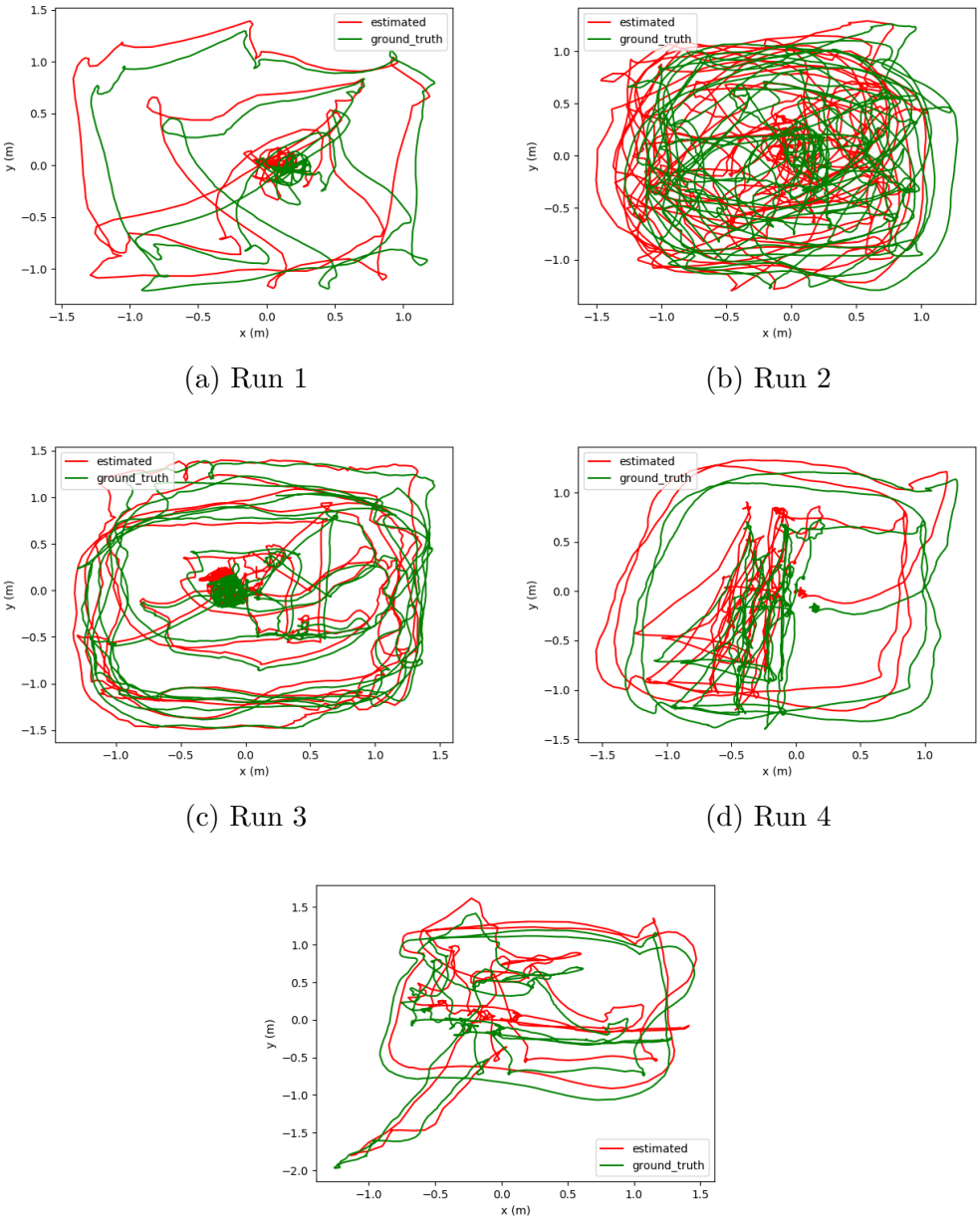
For this dataset, we mounted our lidar-visual-inertial payload on a Boston Dynamics' Spot robot and used an OptiTrack motion capture environment to obtain ground-truth data, as seen in Fig. 5.2a. Then, we were able to generate 5 runs in this setting. The sensor data allows us to run an odometry algorithm and generate an estimated trajectory. Indeed, we used SuperOdometry while capturing the data to generate an online pointcloud reconstruction and trajectory estimation. An example of the pointcloud obtained is shown in Fig. 5.2b. Figure 5.3 shows the trajectories generated from the ground truth motion capture system and the online odometry estimation from SuperOdometry.

(a) Boston Dynamics Spot used in OptiTrack motion capture environment to generate a dataset with ground-truth data.



(b) Pointcloud reconstruction of the motion capture environment, with the robot trajectory colorized by time.

Figure 5.2: Lidar-Visual-Inertial Dataset obtained in a OptiTrack motion capture arena from a payload mounted on a Boston Dynamics' Spot robot dog.



(a) Run 1

(b) Run 2

(c) Run 3

(d) Run 4

(e) Run 5

Figure 5.3: XY trajectories for the motion capture datasets.

5.3 Abandoned Hospital Lidar-Visual-Inertial Datasets

These datasets were obtained by mounting LIDAR-inertial payloads on ground robots and driving them around an abandoned hospital in Pittsburgh, PA. Two of these robots (R1 and R3) were used in the Subterranean Challenge [7]. Another robot used a radio-controller car-like platform. These datasets do not contain ground truth, but they are more representative of actual applications of the complete SLAM system. The topology of the area is characterized by long and featureless corridors that do not intersect often. This implicates that the front-end algorithm may drift by a significant amount before having the chance to close the loop. Perceptual aliasing is another important component of the dataset, since the corridors and rooms of the hospital are very similar to one another, specially after it has been emptied.

5.3.1 R1 Loop Around

This dataset was acquired with the R1 robot platform, used in the Subterranean Challenge. We generate an initial map using SuperOdometry. This is shown in Fig. 5.4. The robot starts in a relatively large room, simulating the Challenge staging area. It then traverses the whole hospital complex in a loop, returning to the same room through another door. The odometry drift is relatively small, considering the traversed distance, but still significant for the purposes of the Challenge.

5.3.2 R3 Loop Around

This dataset was acquired with the R3 robot platform. It notable for the much larger drift it incurs, compared to the other platforms used by Team Explorer. In fact, R3 drift in the final Subterranean Challenge was responsible for most of the map deviation of the team. Fig. 5.5 shows the map generated by SuperOdometry for this dataset.

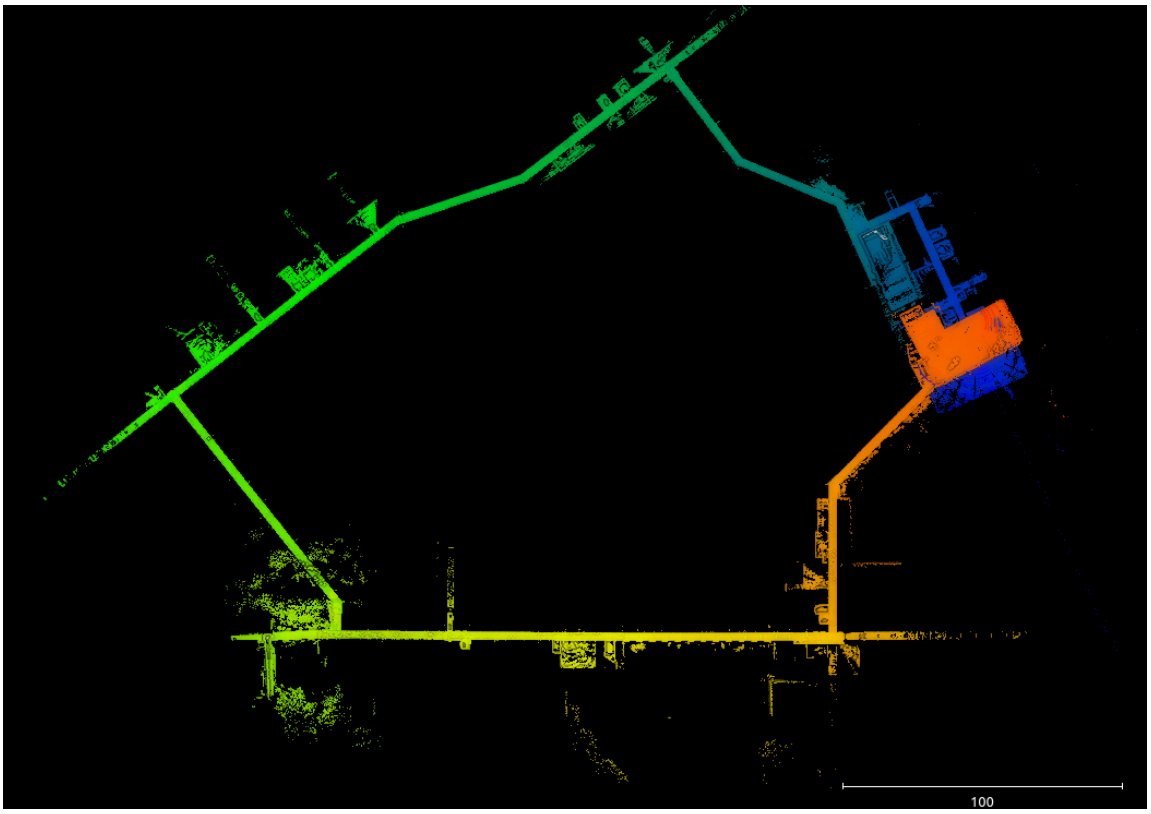


Figure 5.4: R1 pointcloud map created by the front-end odometry algorithm. Colorized by acquisition time (blue to red). Notice the drift at the map endpoints.

5.3.3 Indoor-Outdoor with Failure

This dataset was acquired with an RC car platform. It was teleoperated so that its motion did not depend on having a good position estimate. The trajectory was such that it purposefully goes through an outdoor area with few geometrical landmarks. This causes a singular failure in the odometry algorithm, thus creating at least two separate sections of the map that do not agree with each other. This dataset is used to evaluate if our algorithm can correctly detect the estimation failure and use that information to create a coherent map via loop closing. Fig. 5.6 shows the result of the odometry algorithm after processing the sensor data.

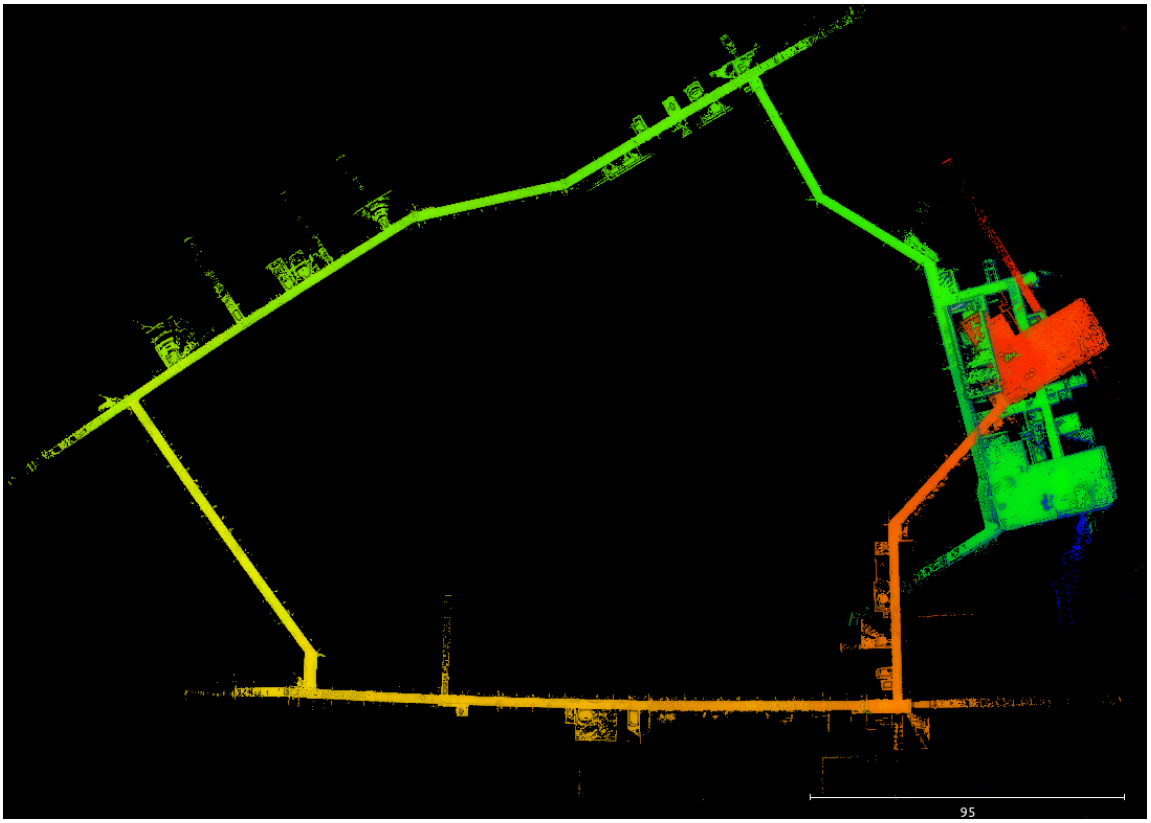


Figure 5.5: R3 pointcloud map created by the front-end odometry algorithm. Colorized by acquisition time (blue to red). Notice both the odometry drift and the map bending effect on the corridors.

Figure 5.6: Indoor-Outdoor With Failure dataset. Pointcloud is colorized by point acquisition time. Robot trajectory is shown in white. Notice the section where the trajectory becomes discontinuous, which leads to drift and failure in mapping.

Chapter 6

Results

6.1 Uncertainty Estimation Using Simulated Data

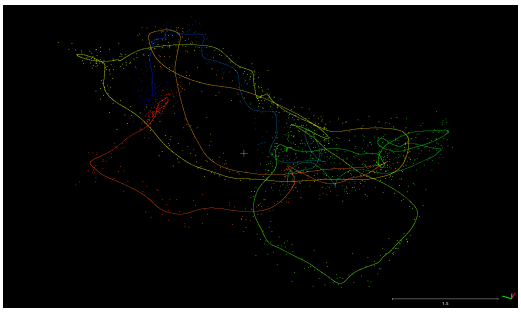
This experiment aims to validate the proposed method by generating a synthetic pose estimation that follows our noise models. The EuRoC dataset provides just the ground-truth pose and imu sequences for this experiment. First, we subsample the ground-truth poses in regular intervals of 0.1 seconds, obtaining a simulated odometry sequence $\mathbf{X}_i \in SE(3)$. Then, for each sampled pose, we perturb it with a noise vector $\xi_{X_i} = [\rho^\top \quad \theta^\top]^\top \in \mathbb{R}^6$. This noise vector is sampled from a multivariate normal distribution with mean $\mu = \mathbf{0}$ and a covariance $\Sigma_i = F(t = 0.1i)$, where F is constructed manually, as we will see. Therefore, we have $\xi_i \sim \mathcal{N}(\mathbf{0}, F(0.1i))$. The synthetic pose estimate $\tilde{\mathbf{X}}_i$ is then obtained by:

$$\tilde{\mathbf{X}}_i = \mathbf{X}_i \oplus \xi_{X_i} \tag{6.1}$$

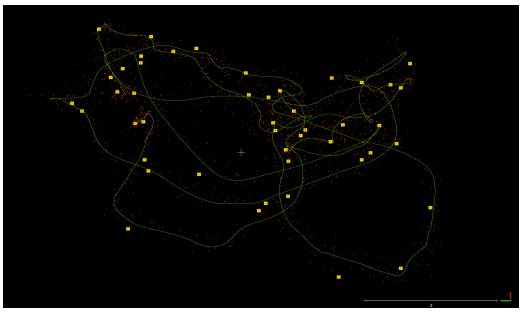
$$\tilde{\mathbf{X}}_i = \mathbf{X}_i \exp\left(\begin{bmatrix} [\theta_i]_\times & \rho_i \\ \mathbf{0} & 0 \end{bmatrix}\right) \tag{6.2}$$

The distribution of the noise is set by adjusting the covariance through the function $F(t)$. This function generates a diagonal covariance matrix. The elements of the diagonal are $\sigma_n^2(t)$, with $n \in \{0, \ldots, 5\}$. The formula used to obtain these variances is:

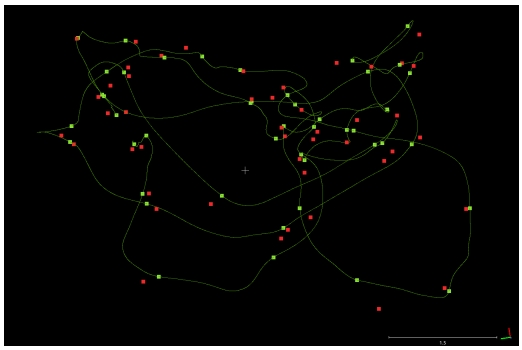
$$\sigma_n(t) = K(n)\left[1 + \sin\left(\frac{t + 100n}{50}\right)\right] \tag{6.3}$$



(a) Scattered points indicate simulated noisy odometry. Smooth line is ground truth trajectory. Color indicates time progression from blue to red.



(b) Keyposes (yellow) selected from the noisy pose estimates (red), using a 3 second interval. Ground truth trajectory (green) shown for reference.



(c) Ground truth keyposes (green dots) compared to estimated keyposes (red). Ground truth trajectory (green line) shown for reference.

Figure 6.1: Trajectories generated from the EuRoC dataset.

where $K(n)$ is a scalar used to set the magnitude of the sine wave for each dimension independently. The $100n$ term inside the sine ensures that each dimension achieves its peak separately from the others, so we can better observe the uncertainty estimation properties of our method. Fig. 6.1a shows the synthetic trajectory obtained for $K(n) = 0.1$.

The next step in the trajectory pre-processing is the *keyframe extraction*. From the estimated trajectory, we select one keyframe after every 3 seconds, generating a sequence of poses $\tilde{\mathbf{T}}_k \in SE(3)$. This is shown in Fig. 6.1b. Then, for each keyframe pair, we obtain the relative transformation $\tilde{\mathbf{D}}_k = \tilde{\mathbf{T}}_k^{-1}\tilde{\mathbf{T}}_{k+1}$. Similarly, we obtain a sequence of ground-truth poses $\mathbf{T}_k$. We use interpolation on the ground truth

trajectory such that the timestamp of $\mathbf{T}_k$ is the same as that of $\tilde{\mathbf{T}}_k$. Again, we obtain the ground-truth relative transformation $\mathbf{D}_k = \mathbf{T}_k^{-1}\mathbf{T}_{k+1}$. These two sequences generated by the keyframe extraction can be seen in Fig. 6.1c.

Now, we calculate the *relative keypose tranformation error* of the estimation algorithm by comparing $\tilde{\mathbf{D}}_k$ and $\mathbf{D}_k$ such that

$$\xi_k = \tilde{\mathbf{D}}_k \ominus \mathbf{D}_k = \ \mathrm{Log}(\mathbf{D}_k^{-1}\tilde{\mathbf{D}}_k) \in \mathbb{R}^6 \tag{6.4}$$

The main goal of our proposed method is to estimate a covariance matrix Σ_k that represents the distribution of ξ_k, such that $xi_k \sim \mathcal{N}(\mathbf{0}, \Sigma_k)$. Knowledge about its distribution can be used to solve a pose-graph created from the keyposes. However, our method also internally estimates the distribution of the local odometry noise ξ_{X_i}, which is the estimation error at any given time instant, when compared to the ground truth pose.

We visualize well how our algorithm estimates both of these quantities, by plotting the estimated errors against the estimated standard deviations. In practice, we plot 3 times the standard deviation of each dimension, which should contain the error 99.7% of the time. Figure 6.2 shows the result of the local noise $\boldsymbol{\epsilon}$. From this graph, we can see the shape of the function $F(t)$ that defines the covariance of the noise. Our estimates of the covariance match reasonably with the real uncertainty most of the time.

In Figure 6.3a we observe the uncertainty estimate final result: the covariance of the relative transform. We compare it with the relative transform estimation error. The goal here is that our estimate can provide a tight bound on the possible range of error values, such that we can feed this onto a pose graph solver and it can adequately distribute the error along the trajectory. Here, we see that the 3 standard deviations bound encompasses all the errors observed, while not by a large margin. In effect, if we estimated a really large covariance, the errors would almost be guaranteed to be inside it, but the result would not actually be useful.

Given the zero-mean error ξ_k and its true covariance Σ_k, the Mahalanobis distance can be calculated as $d = \sqrt{\xi_k^\top \Sigma_k^{-1} \xi_k}$. In this case, d^2 will follow a 6 DOF Chi-Squared distribution. Therefore, one way of evaluating the quality of our covariance estimation is to calculate this distance for every relative transform error-covariance pair in the trajectory, and visualize how well does it track the theoretical distribution. This

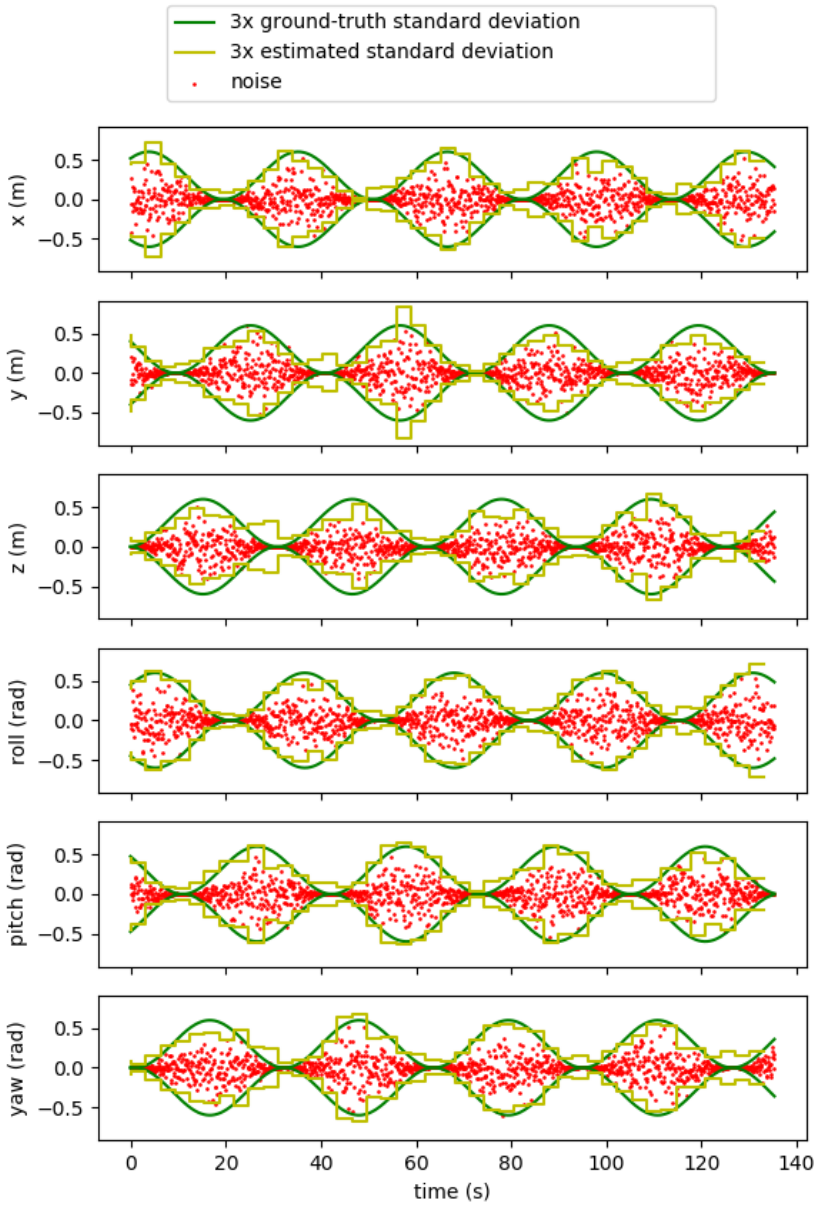
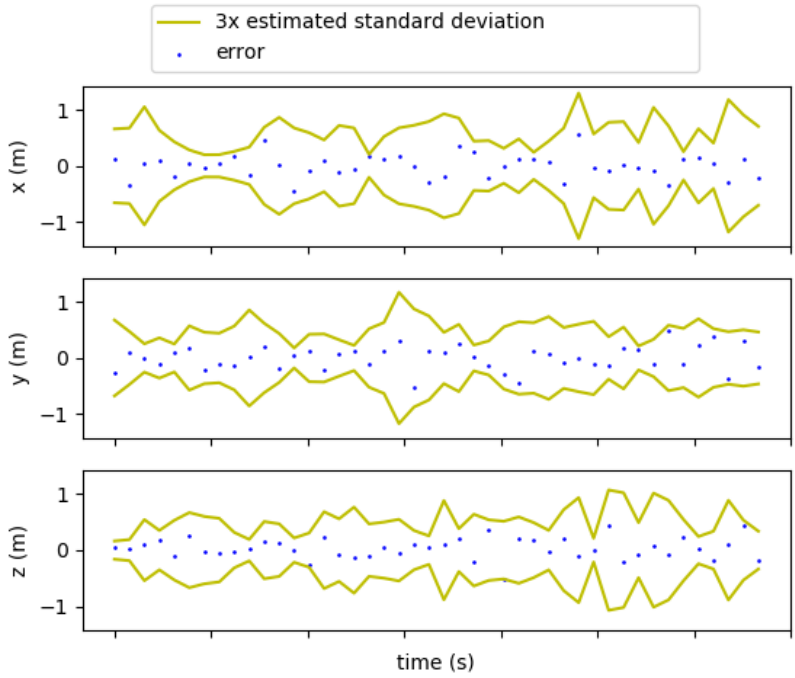
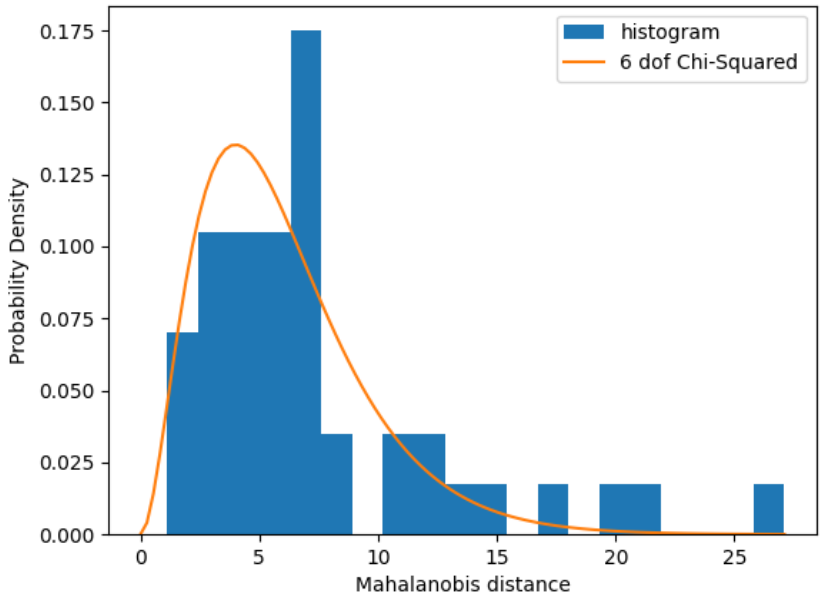


Figure 6.2: Estimation of the local noise ξ_{X_i}. The noise injected artificially into the pose estimates is shown in red. The covariance function $F(t)$ noise is shown in green as the 3-standard-deviation level. The estimates of the uncertainty made by the proposed method are shown in yellow.

comparison is shown in Fig. 6.3b. It can be seen that it mostly tracks the desired curve, except for two details: the peak at around a Mahalanobis distance of 6.5 and the existence of outliers above 20. This could be an indication that the model used is insufficient in specific parts of the trajectory.



(a) Estimation of the covariance of the relative position error ξ_k. The error (blue) is obtained by comparing the true relative transform and its estimate. Notice that the sine wave we insert in the local pose is not directly observable here, since the relative transform composes two noisy estimates.



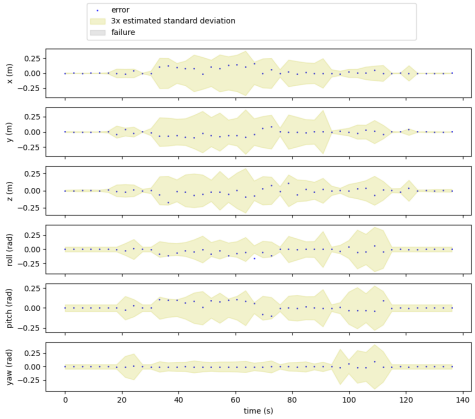
(b) Histogram of the Mahalanobis distances of the relative transformation errors, using our estimated covariances. Ideally, it approaches a Chi-Squared distribution with 6 DOF, also shown for comparison.

Figure 6.3: Covariance Estimation results from the simulated EuRoC data

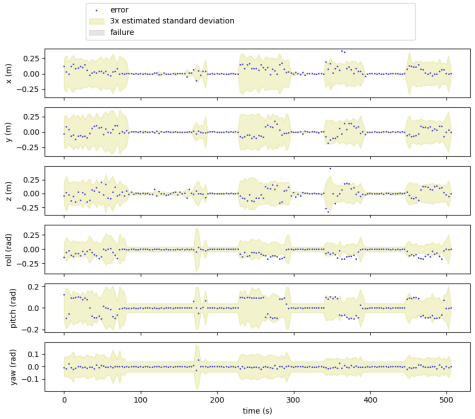
6.2 Uncertainty Estimation Using Motion Capture Ground Truth

The goal of this experiment is to analyse if the proposed method produces acceptable predictions on the expected size of the error made by the estimation algorithm. In order to accomplish this, the data generated at the motion capture arena is used. For each of these datasets, the inputs used are a) the 200Hz imu sensor data; b) the 5 Hz estimated laser odometry results and c) the ground truth position. Fig. 6.4 shows the 3 standard deviation bounds estimated by our proposed methods and the actual error ξ_k incurred for each dimension. Ideally, this graph would show 99.7% the errors inside the bounds. Fig. 6.5 shows the same bounds around the true relative translational motion of $\mathbf{D}_k$. This allows us to compare the scale of the variance, the error and the motion pattern of the robot.

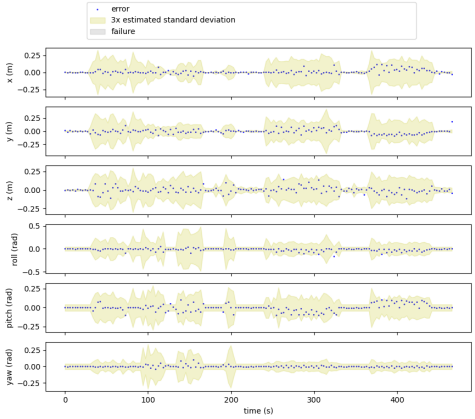
Again, we calculate the Mahalanobis distances $d^2 = \xi_k^\top \tilde{\Sigma}_k^{-1} \xi_k$ of the relative keypose tranformation error for each keyframe k in each run. We create histograms of the values of d^2 and compare it to the graph of the 6 DOF Chi-squared distribution. The results are shown in Fig. 6.6.



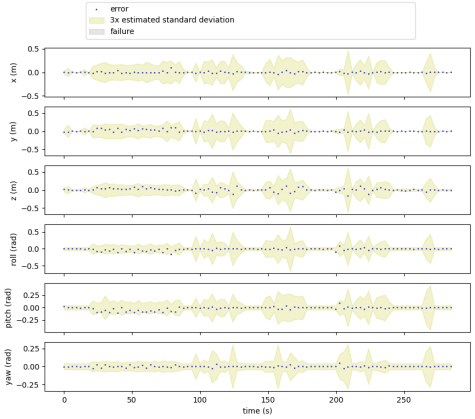
(a) Run 1



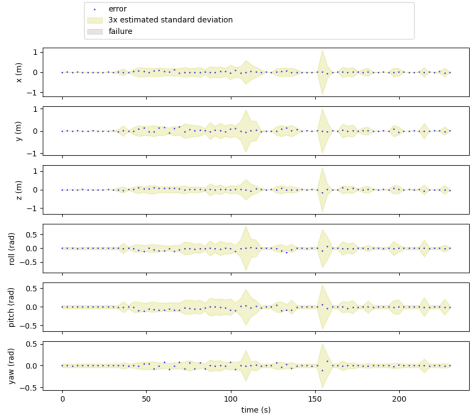
(b) Run 2



(c) Run 3

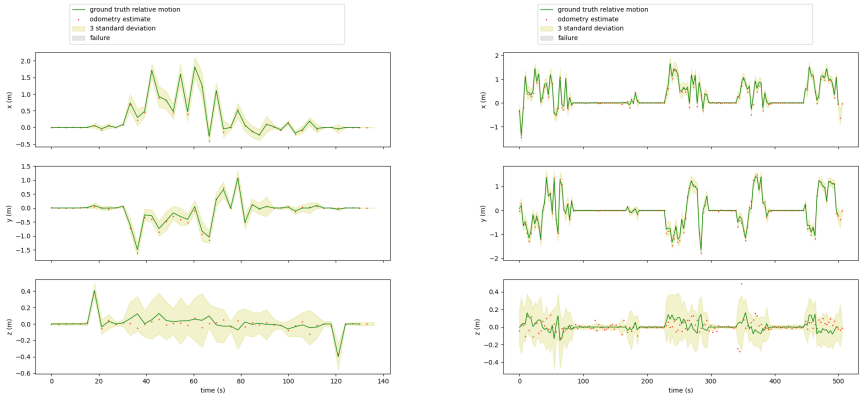


(d) Run 4



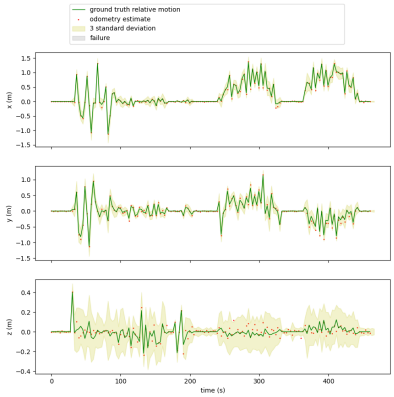
(e) Run 5

Figure 6.4: Uncertainty Estimation Results from motion capture datasets.

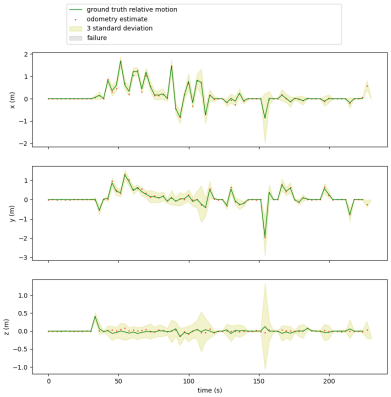


(a) Run 1

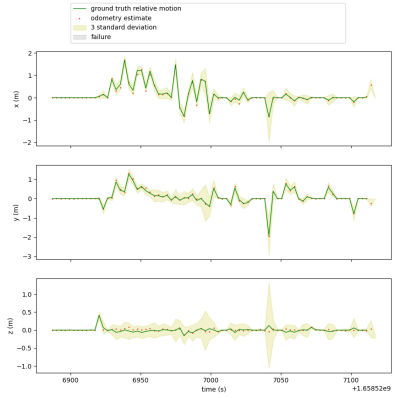
(b) Run 2



(c) Run 3

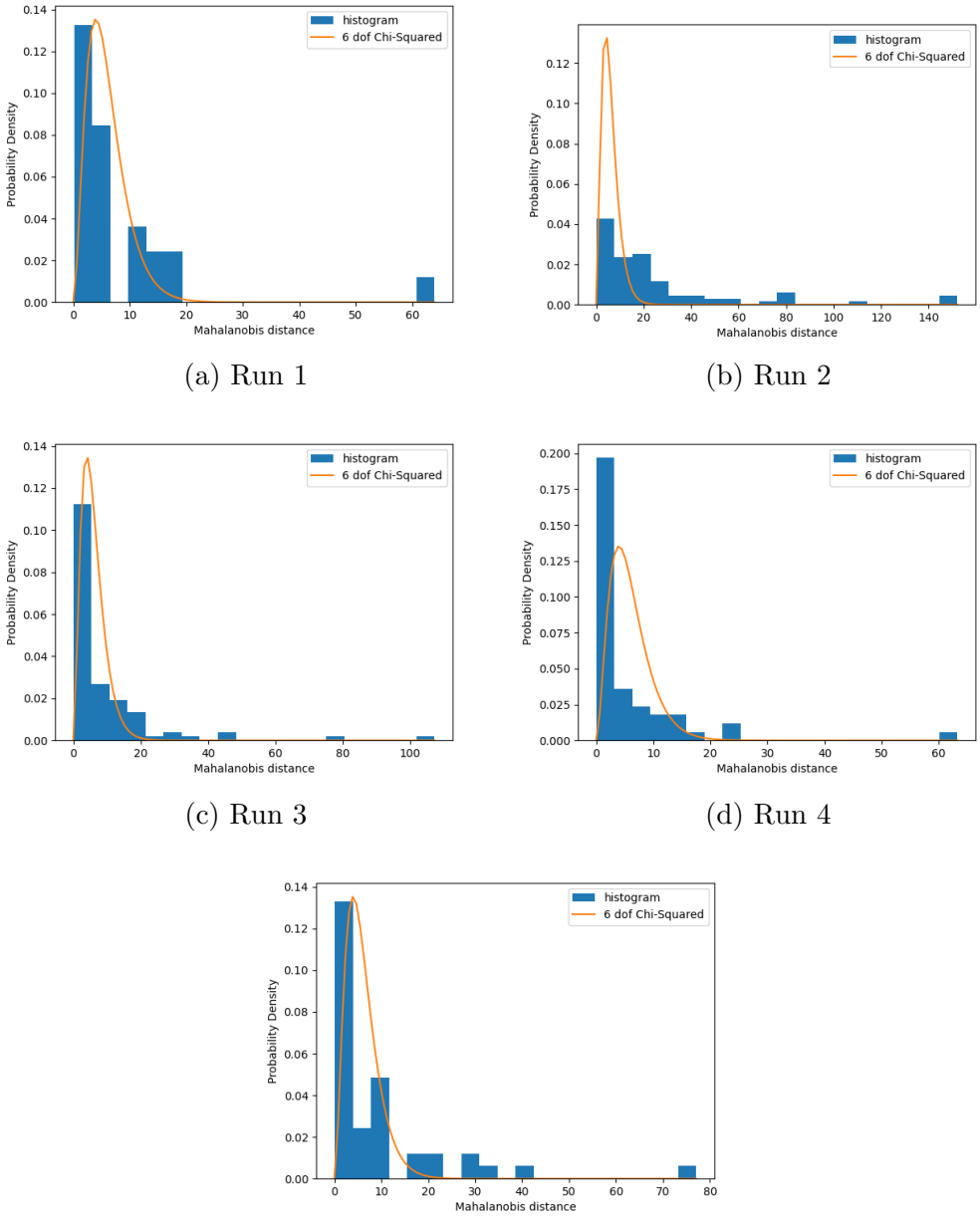


(d) Run 4



(e) Run 5

Figure 6.5: Estimated error bounds around the relative motion.



(a) Run 1

(b) Run 2

(c) Run 3

(d) Run 4

(e) Run 5

Figure 6.6: Comparison of histogram of Mahalanobis Distance and 6 degrees-of-freedom Chi-squared distribution for each motion capture run.

6.3 SuperLoop Results

6.3.1 R1 Loop

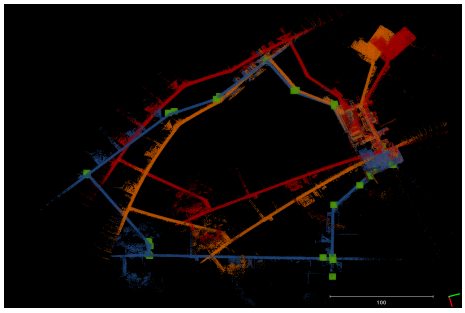
In this experiment, we perform an ablation study to evaluate the impact of using the proposed uncertainty estimation (UE) algorithm versus using a constant diagonal covariance matrix in the odometry constraints in SuperLoop's pose graph. The constant covariance matrices are built such that $\Sigma_k = diag([\sigma^2, \sigma^2, \sigma^2, \sigma^2, \sigma^2, \sigma^2])$. The values chosen for σ^2 are: 10^{-2}, 10^{-3}, 10^{-4} and 10^{-5}. Fig. 6.7 shows the resulting maps. In all cases, the map generated using the uncertainty estimation method is shown in blue. Fig. 6.7a shows the maps generated by the first two covariances values (10^{-2}, 10^{-3}) alongside our method. From this figure, it can be observed that these values are actually too large. This causes the algorithm to be more permissive with its loop closure possibilities and a false positive loop is created, distorting the entire map.

Fig. 6.7b shows the comparison against a constant covariance of 10^{-4}. The results are nearly identical, although there are some discrepancies in the bottom left section. Finally, Fig. 6.7c compares our method against a constant covariance of 10^{-5}. This result shows that this is an underestimation of the covariance, and that when the loop closure does occur at the end, the factor graph does not distribute the error correctly, since it believes the odometry to be more correct than it actually is. This causes the initial area to be duplicated and a clearly wrong result is generated also.

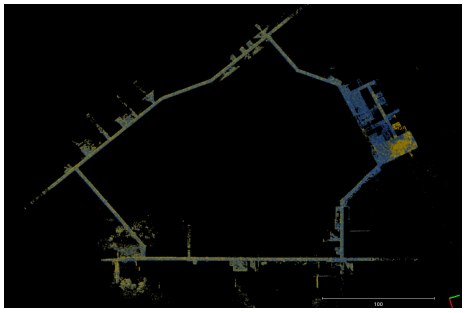
Additionally, the loop-closed maps are compared with a ground truth map. For this purpose, the *deviation metric* is used, as popularized by DARPA during the Subterranean Challenge. The deviation metric considers two pointclouds: a test and a reference one. The deviation is the number of points in the test cloud whose distance to *any* point in the reference cloud is less than a threshold, divided by the total number of points in the test cloud. Therefore, the deviation metric is a percentage of inliers points in the test cloud. The ground truth maps only encompass the main corridor loop that connects the hospital buildings and therefore does not contain other areas that the robots visited. Because of this, we use the ground truth cloud as test cloud and the clouds produced by each version of the algorithm as references. The loop-closed maps are trimmed to contain only the main corridor also. The threshold

used is $20cm$.

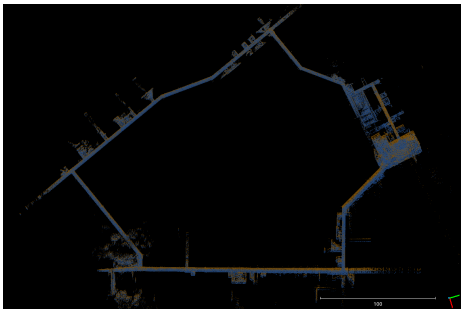
The results are summarized in Table 6.1, and the pointcloud comparison is shown in Fig. 6.8. First, note that there is a section of the loop that is not covered in this experiment by R1. Therefore, it is expected that we get a relatively high deviation as the baseline. Next, we observe that the best performing version of the algorithm is the one with constant diagonal covariance with $\sigma^2 = 0.0001$. The version of SuperLoop using the proposed UE method is the second-best. These results shows that the our UE method is capable of reasonably assessing the correct covariance levels of this dataset. In controlled scenarios, the SLAM engineer can post-process the data with different covariance values until a reasonable result is achieved. However, in a real life search-and-rescue scenario, this cannot be done, and a dynamic method for estimating the uncertainty can increase the SLAM solution robustness.



(a) (Blue): Ours
(Red): $\sigma^2 = 0.01$
(Orange): $\sigma^2 = 0.001$



(b) (Blue): Ours
(Yellow): $\sigma^2 = 0.0001$

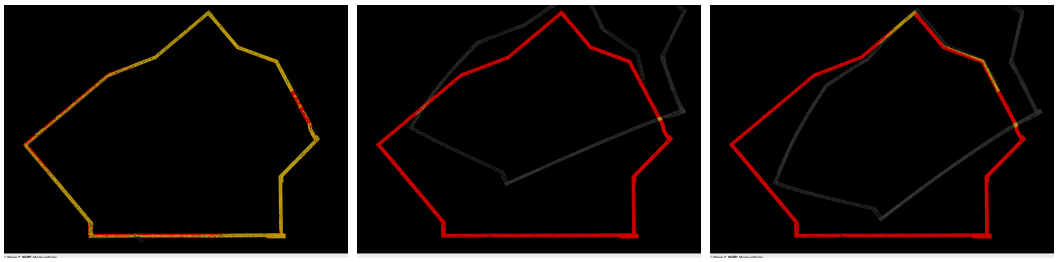


(c) (Blue): Ours
(Brown): $\sigma^2 = 0.00001$

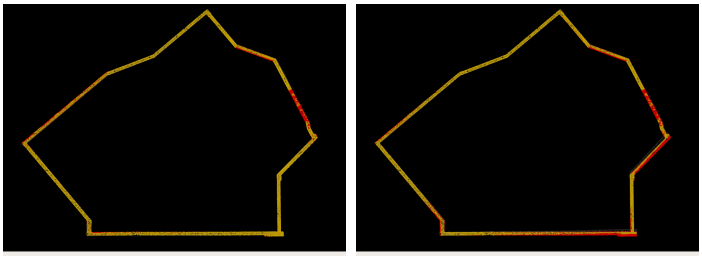
Figure 6.7: Comparison of loop closing results using the proposed uncertainty estimation method versus used a constant diagonal covariance matrix.

Covariance Type	Deviation (%)
Our	19.38
$\sigma^2 = 0.01$	98.52
$\sigma^2 = 0.001$	90.53
$\sigma^2 = 0.0001$	**12.93**
$\sigma^2 = 0.00001$	26.85

Table 6.1: Deviation metric comparison for the R1 Loop dataset. Deviation is calculated as the percentage of points in the test pointcloud that are farther than 0.2 meters from any point in the reference cloud.



(a) Our method (b) $\sigma^2 = 0.01$ (c) $\sigma^2 = 0.001$



(d) $\sigma^2 = 0.0001$ (e) $\sigma^2 = 0.00001$

Figure 6.8: Deviation Metric Visualization: red points indicate outliers, and yellow points indicate indicate inliers. The reference cloud is shown in white, containing the loop-closed map generated by the algorithms. It is only noticeable in some cases where the map makes large mistakes.

6.3.2 R3 Loop

The same experiment described in the previous section is repeated for the R3 loop dataset. The most noticeable difference between the datasets is that the R3 robot had the largest drift rate of all the platforms used by Team Explorer. So, this experiment aims to evaluate if the proposed uncertainty estimation method can detect this different behavior correctly.

Fig. 6.9 shows the results obtained. The map in blue is generated by the proposed method. Then, we compare it with the same algorithm but using a constant diagonal matrix with variances σ^2 valued 10^{-2}, 10^{-3}, 10^{-4} and 10^{-5}. Fig. 6.9a shows the comparison with a $\sigma^2 = 10^{-5}$. In this case, the algorithm becomes overconfident and rejects possible loop-closures, basically maintaining the initial odometry result.

In Fig. 6.9b the results of the comparison with $\sigma^2 = 10^{-3}$ and $\sigma^2 = 10^{-4}$ are shown. In this case, both values are capable of generating a loop closure at the end of the trajectory. There are minor differences in the result shown in the detail of Fig. 6.9c. Lastly, the result of using $\sigma^2 = 10^{-2}$ is not shown because the large covariance caused an erroneous loop-closure to be accepted which ended up completely distorting the final map. The deviation metric comparison in shown in Table 6.2. In this experiment, the proposed method was the best performing one. Notice from the other results that the best result for constant covariance is 10^{-3}, which is a different behavior than the R1 test. Even in this case, the algorithm is able to estimate the covariance correctly. The deviation itself can be viewed in Fig. 6.10

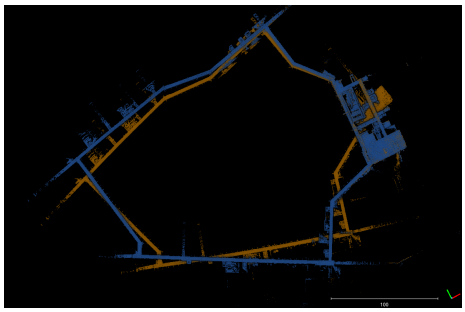
Covariance Type	Deviation (%)
Our	**13.14**
$\sigma^2 = 0.001$	25.14
$\sigma^2 = 0.0001$	27.63
$\sigma^2 = 0.00001$	83.13

Table 6.2: Deviation metric comparison for the R3 Loop dataset. Deviation is calculated as the percentage of points in the test pointcloud that are farther than 0.2 meters from any point in the reference cloud.

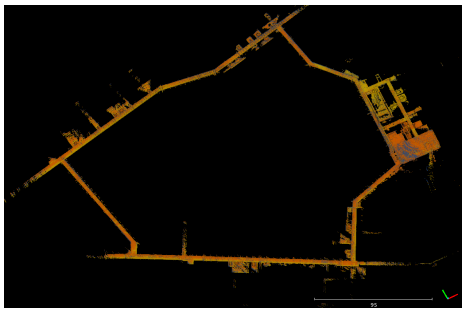
6.3.3 Failure Detection

This experiments evaluates the capability of the proposed method for odometry failure detection. The dataset presented in Sec. 5.3.3 is used. In this dataset, there is a clear single-point failure when the RC car visits the outdoor area and turns rapidly. This causes a large drift error in the mapping result, which is most visible in the mismatch of the large room observed by the robot at the start and end of its trajectory.

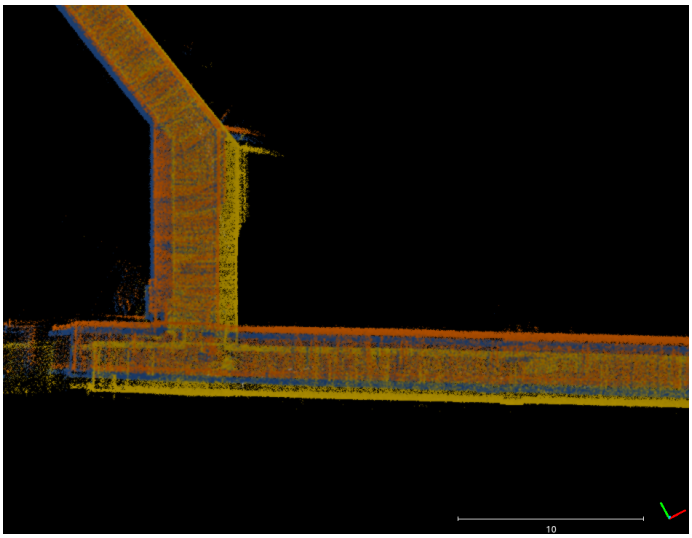
Fig. 6.11 shows the result of an ablation test of the failure detection mechanism. The original pointcloud produced by the odometry method is shown in red. This



(a) (Blue): Ours
(Brown): $\sigma^2 = 0.00001$



(b) (Blue): Ours
(Orange): $\sigma^2 = 0.001$
(Yellow): $\sigma^2 = 0.0001$

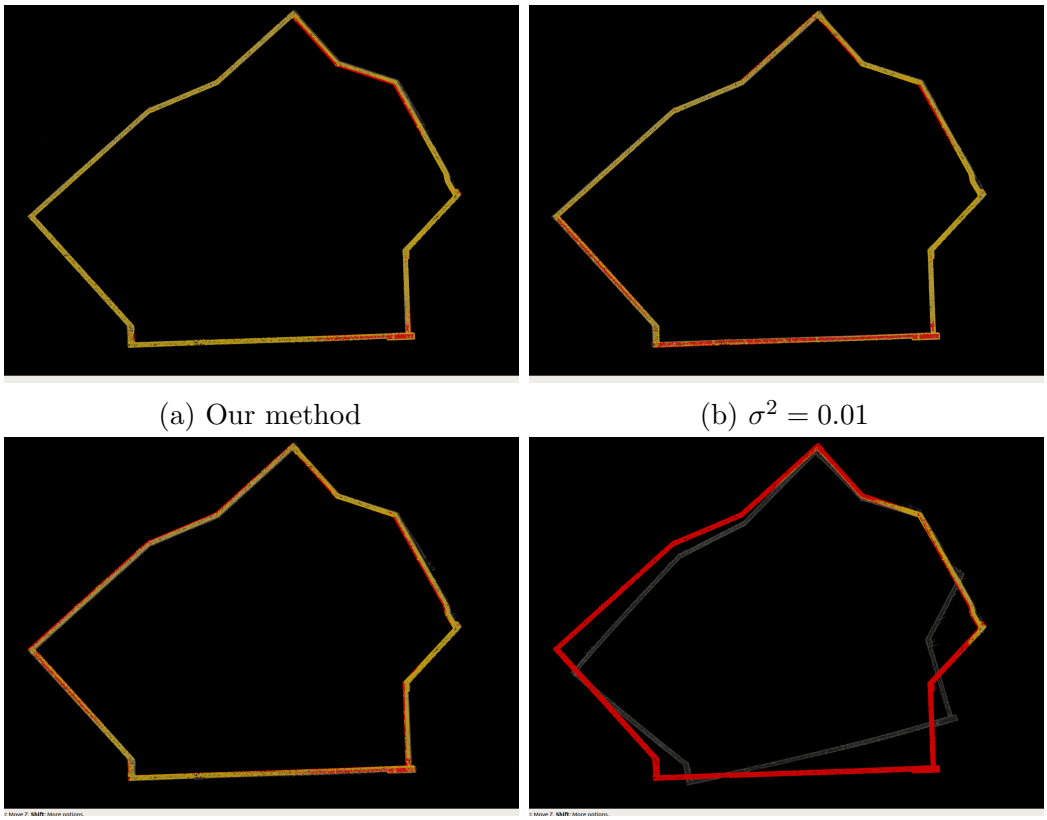


(c) Detail showing the mapping differences for each uncertainty level.

Figure 6.9: Comparison of loop closing results using the proposed uncertainty estimation method versus used a constant diagonal covariance matrix for R3 loop around the hospital complex.

result is used as an input to the proposed loop closure method. The yellow result is produced by disabling the failure detection feature, while the green result does use this feature. It can be observed that the green result produces the best result, as it retains continuity in the trajectory and properly closes the loop at the end points.

Fig. 6.12 gives some insight into how the algorithm works. The failure detection works on each edge of the factor graph, analyzing the mean of the acceleration residual of the measurements in that interval. When that mean is larger than a threshold, we



(a) Our method (b) $\sigma^2 = 0.01$

(c) $\sigma^2 = 0.001$ (d) $\sigma^2 = 0.0001$

Figure 6.10: Deviation Metric Visualization: red points indicate outliers, and yellow points indicate indicate inliers. The reference cloud is shown in white, containing the loop-closed map generated by the algorithms. It is only visible in some cases where the map makes large mistakes.

consider our model to be violated, and we trigger a failure flag. Then, the relative transform from the odometry is thrown away, and replaced with a FastGICP method between the keyposes. The covariance from the uncertainty estimation method is still used, but a robust Cauchy kernel is used to indicate to the algorithm that this is likely to be an outlier. The factor graph solver can them better distribute the errors to the failure edges.

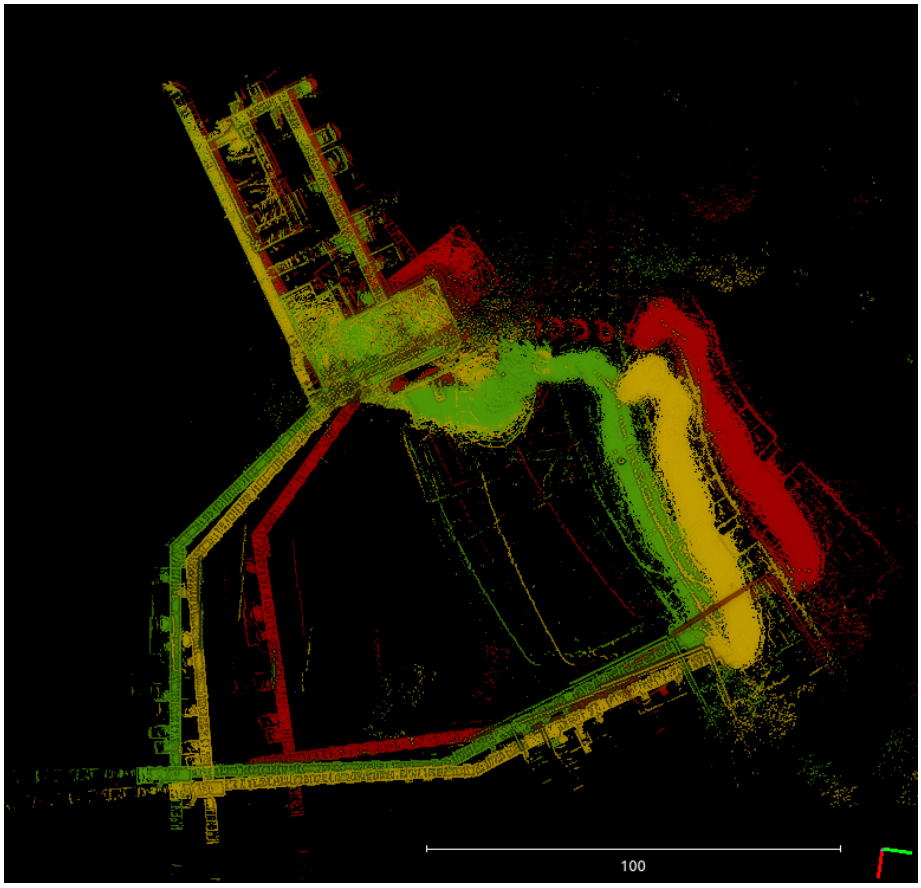


Figure 6.11: Effect of using the proposed failure detection. (red): Odometry result. (yellow): Loop Closure without failure detection. (green): Loop Closure with failure detection enabled.

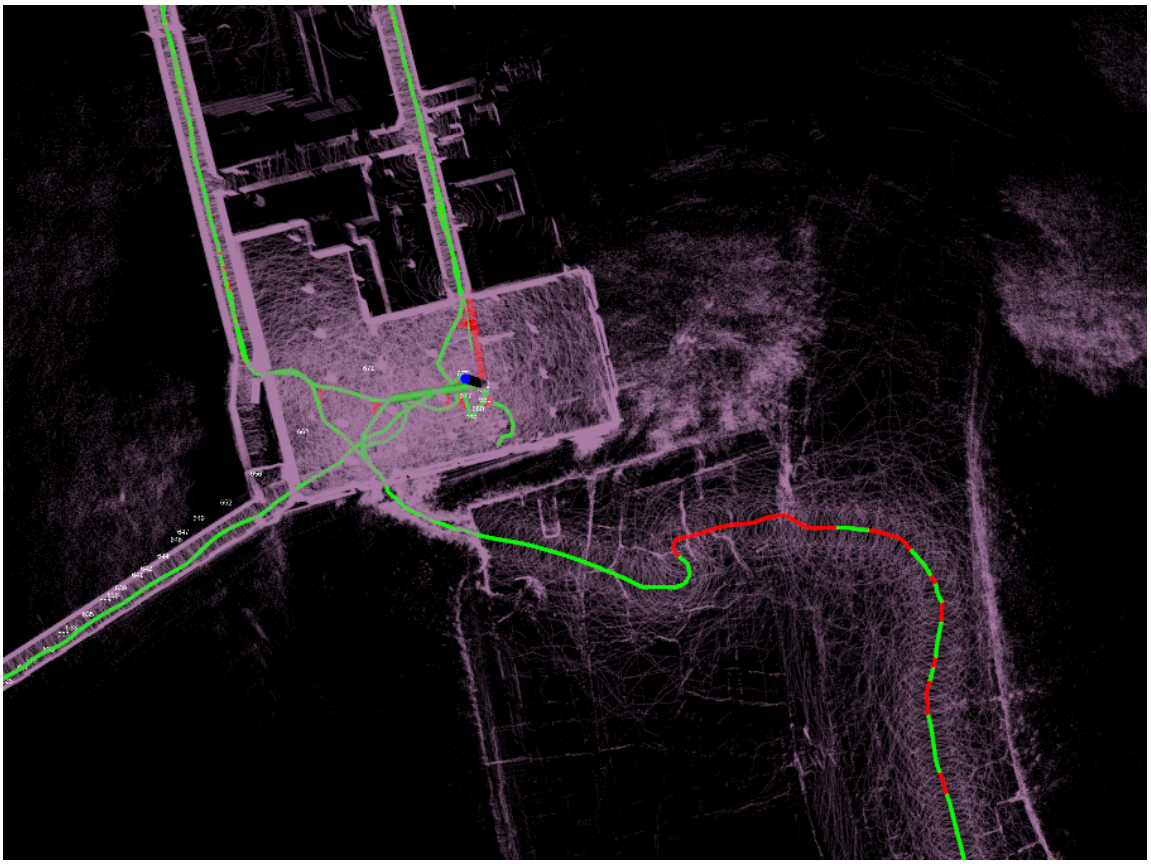


Figure 6.12: The failure detection algorithm indicates edges of the factor graph that are likely to be erroneous and applies a simple ICP procedure to obtain a new transform between the nodes, and applies a robust kernel model for the uncertainty.

Chapter 7

Conclusions and Future Work

This work introduced a SLAM back-end that is compatible with SuperOdometry. It improves upon the one used during the Subterranean Challenge by providing place recognition via a modified ScanContext descriptor. Its main contribution is a method for calculating the keyframe relative pose uncertainty using the sequence of odometry pose estimates and the IMU as an arbitrator. It is shown that using this uncertainty estimation may work better than a *one-size-fits-all* constant and diagonal covariance matrix when building the SLAM factor graph, as is the current practice.

However, since the method compares the derivatives of pose estimates with IMU measurements, it will be overly optimistic in cases where the pose estimates are generated with algorithms that are tightly-coupled with the same IMU. Therefore, it is more suitable to methods such as LOAM and SuperOdometry, that generate its pose estimates using a LIDAR scan matching process, and IMU integration is used only for initialization. One possible way to circumvent this issue is to use separate IMUs or separate a subset of the IMU data for the uncertainty estimation process.

Some ideas to further improve SuperLoop and the Uncertainty Estimation algorithm include:

Diverse Testing Environments The results presented in this work pertain to the man-made subdomain of underground environments. If we use the DARPA Subterranean Challenge as a reference, there still remains at least the cave and tunnel environments where SuperLoop needs to be tested. The challenge in those cases lies

in obtaining good ground truth for the whole trajectory or the entire map, which is quite challenging to do in these scenarios.

Relative Noise Model We presented in Section 3.3 the absolute noise model for the local odometry covariance. This model is similar to a GPS sensor, and we applied it here with the assumption that the underlying LIDAR-based odometry holds a local pointcloud map in its memory, and therefore it is closer to a localization algorithm in the short term. However, when the robot is constantly exploring new environments, this assumption may not hold. In these cases, a relative noise model might be needed, where we consider that the noise is added to the relative odometry transformations. This is a more orthodox model, but the derivation of an Uncertainty Estimation method in this case becomes more complex due to the integration of noise that arises.

Improving Repeatabilty of the Failure Recover Mechanism The failure detection and recovering mechanism presented in this work is still very experimental, and although promising, it is fragile and will clearly not recover from any given odometry failure. Therefore, more work is needed to ensure results are repeatable and the recover mechanism more reliable. In the very least, we should learn more about what are the scenarios when the recover mechanism should work, and when it cannot.

Characterizing the Effects of Different Interpolation Schemes In this work, we used B-splines of order 4 to create continuous trajectories from the discrete odometry estimates. The guiding principle was to use the lowest order number that still gives out satisfactory results. This is due to the fact that more poses are needed when the order is higher, which increases the minimum window size also. However, other interpolation schemes are possible, and analyzing the impact of the choice for B-spline interpolation with odometry poses as knots is important to understand if we are making the best choice here.

Application to Sensor Fusion This work was initially developed with the intention of devising an algorithm that could arbitrate between different odometry sources such as LIDAR-, visual- or wheel-odometry. This algorithm would take in the latest

estimates and decide which of these algorithms are more likely to be performing well. However, we ran into the limitation explained above, where an estimation algorithm that is tightly-coupled with the IMU will almost always produce a lower uncertainty estimate than loosely-coupled ones. Therefore, it would not be possible to compare them using this algorithm. However, it may still be possible to compare different loosely-coupled algorithms, or even two versions of the same LIDAR-odometry algorithm with different parameters, and then carefully selecting the best performing one.

Application to Multi-Robot SLAM Similarly to the previous paragraph, one possible future research direction is the application of the Uncertainty Estimation method in Multi-Robot SLAM. In this application, all robots are running the same underlying odometry algorithm, and UE may be used to appropriately weigh the information from each robot when fusing that information. This is a common process in Multi-Robot SLAM that requires a lot of attention, as different robots have traversed the environment using different pathways and thus might have accrued different levels of SLAM drift.

Bibliography

[1] Timothy D. Barfoot and Paul T. Furgale. Associating uncertainty with three-dimensional poses for use in estimation problems. *IEEE Transactions on Robotics*, 30(3):679–693, 2014. ISSN 1552-3098. doi: 10.1109/tro.2014.2298059. 1.2.2

[2] Michael Bosse and Robert Zlot. Map matching and data association for large-scale two-dimensional laser scan-based slam. *The International Journal of Robotics Research*, 27(6):667–691, 2008. 1.2.2

[3] Martin Brossard, Silvére Bonnabel, and Axel Barrau. A new approach to 3D ICP covariance estimation. *IEEE Robotics and Automation Letters*, 5(2):744–751, 2020. doi: 10.1109/LRA.2020.2965391. 1.2.2

[4] Anders Glent Buch, Dirk Kraft, et al. Prediction of ICP pose uncertainties using Monte Carlo simulation with synthetic depth images. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4640–4647. IEEE, 2017. 1.2.2

[5] Michael Burri, Janosch Nikolic, Pascal Gohl, Thomas Schneider, Joern Rehder, Sammy Omari, Markus W Achtelik, and Roland Siegwart. The EuRoC micro aerial vehicle datasets. *The International Journal of Robotics Research*, 35(10): 1157–1163, 2016. 5.1

[6] Cesar Cadena, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, Jos Neira, Ian Reid, and John J. Leonard. Past, present, and future of simultaneous localization and mapping: toward the robust-perception age. *IEEE Transactions on Robotics*, 32(6):1309–1332, 2016. ISSN 1552-3098. doi: 10.1109/tro.2016.2624754. 4

[7] Chao Cao, Lucas Nogueira, Hongbiao Zhu, John Keller, Graeme Best, Rohit Garg, David Kohanbash, Jay Maier, Shibo Zhao, Fan Yang, Katarina Cujic, Ryan Darnley, Robert DeBortoli, Bill Drozd, Peigen Sun, Ian Higgins, Steven Willits, Greg Armstrong, Ji Zhang, Geoffrey Hollinger, Matthew Travers, and Sebastian Scherer. Exploring the most sectors at the DARPA Subterranean Challenge finals. *Field Robotics*, 3:801–836, 01 2023. doi: 10.55417/fr.2023025. 1.1, 5.3

[8] Andrea Censi. An accurate closed-form estimate of ICP covariance. *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 3167–3172, 2007. ISSN 1050-4729. doi: 10.1109/robot.2007.363961. 1.2.2

[9] Timothy H. Chung, Viktor Orekhov, and Angela Maio. Into the robotic depths: Analysis and insights from the DARPA Subterranean Challenge. *Annual Review of Control, Robotics, and Autonomous Systems*, 6(1):477–502, 2023. doi: 10.1146/annurev-control-062722-100728. URL https://doi.org/10.1146/annurev-control-062722-100728. 1.1

[10] David M Cole and Paul M Newman. Using laser range data for 3D SLAM in outdoor environments. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pages 1556–1563. IEEE, 2006. 1.2.2

[11] Maurice G Cox. The numerical evaluation of B-splines. *IMA Journal of Applied mathematics*, 10(2):134–149, 1972. 2.2

[12] Carl De Boor. On calculating with B-splines. *Journal of Approximation theory*, 6(1):50–62, 1972. 2.2

[13] Frank Dellaert. Factor graphs: Exploiting structure in robotics. *Annual Review of Control, Robotics, and Autonomous Systems*, 4(1):141–166, 2021. doi: 10.1146/annurev-control-061520-010504. URL https://doi.org/10.1146/annurev-control-061520-010504. 2.3.2

[14] Frank Dellaert and GTSAM Contributors. borglab/gtsam, May 2022. URL https://github.com/borglab/gtsam). 2.3.2, 4.3

[15] Frank Dellaert and Michael Kaess. Factor graphs for robot perception. *Foundations and Trends® in Robotics*, 6(1-2):1–139, 2017. 2.3.2, 4, 4.3

[16] Kamak Ebadi, Lukas Bernreiter, Harel Biggie, Gavin Catt, Yun Chang, Arghya Chatterjee, Christopher E. Denniston, Simon-Pierre Deschênes, Kyle Harlow, Shehryar Khattak, Lucas Nogueira, Matteo Palieri, Pavel Petráček, Matěj Petrlík, Andrzej Reinke, Vít Krátký, Shibo Zhao, Ali akbar Agha-mohammadi, Kostas Alexis, Christoffer Heckman, Kasra Khosoussi, Navinda Kottege, Benjamin Morrell, Marco Hutter, Fred Pauling, François Pomerleau, Martin Saska, Sebastian Scherer, Roland Siegwart, Jason L. Williams, and Luca Carlone. Present and future of SLAM in extreme underground environments, 2022. 1.1, 1.1, 1.2.1

[17] Christian Forster, Luca Carlone, Frank Dellaert, and Davide Scaramuzza. IMU preintegration on manifold for efficient visual-inertial maximum-a-posteriori estimation. Technical report, 2015. 1.2.1

[18] Shehryar Khattak, Huan Nguyen, Frank Mascarich, Tung Dang, and Kostas Alexis. Complementary multi–modal sensor fusion for resilient robot pose estimation in subterranean environments. In *2020 International Conference on*

Unmanned Aircraft Systems (ICUAS), pages 1024–1029. IEEE, 2020. 1.2.1

[19] Giseop Kim and Ayoung Kim. Scan context: Egocentric spatial descriptor for place recognition within 3d point cloud map. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4802–4809. IEEE, 2018. (document), 1.2.1, 2.2, 2.4.2

[20] Giseop Kim, Sunwook Choi, and Ayoung Kim. Scan Context++: structural place recognition robust to rotation and lateral variations in urban environments. *arXiv*, 2021. 2.4, 2.4.2, 4.1, 4.4.2

[21] Giseop Kim, Seungsang Yun, Jeongyun Kim, and Ayoung Kim. SC-LiDAR-SLAM: A front-end agnostic versatile lidar SLAM system. *2022 International Conference on Electronics, Information, and Communication (ICEIC)*, 00:1–6, 2022. doi: 10.1109/iceic54506.2022.9748644. 2.4.2, 4.1

[22] Kenji Koide, Masashi Yokozuka, Shuji Oishi, and Atsuhiko Banno. Voxelized GICP for fast and accurate 3D point cloud registration. *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 00:11054–11059, 6 2021. doi: 10.1109/ICRA48506.2021.9560835. 4.4.3

[23] Gim Hee Lee, Friedrich Fraundorfer, and Marc Pollefeys. Robust pose-graph loop-closures with expectation-maximization. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 556–563. IEEE, 2013. 2.3.2, 4.3

[24] Andrew W Long, Kevin C Wolfe, Michael J Mashner, Gregory S Chirikjian, et al. The banana distribution is Gaussian: A localization study with exponential coordinates. *Robotics: Science and Systems VIII*, 265:1, 2013. 1.2.2

[25] Fahira Afzal Maken, Fabio Ramos, and Lionel Ott. Estimating motion uncertainty with bayesian ICP. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 8602–8608. IEEE, 2020. 1.2.2

[26] Fahira Afzal Maken, Fabio Ramos, and Lionel Ott. Stein ICP for uncertainty estimation in point cloud matching. *IEEE Robotics and Automation Letters*, 7 (2):1063–1070, 2022. doi: 10.1109/LRA.2021.3137503. 1.2.2

[27] Joshua G. Mangelson, Maani Ghaffari, Ram Vasudevan, and Ryan M. Eustice. Characterizing the uncertainty of jointly distributed poses in the lie algebra. *IEEE Transactions on Robotics*, 36(5):1371–1388, 2020. doi: 10.1109/TRO.2020.2994457. 1.2.2, 3.3

[28] Ellon Mendes, Pierrick Koch, and Simon Lacroix. ICP-based pose-graph SLAM. In *2016 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pages 195–200, 2016. doi: 10.1109/SSRR.2016.7784298. 1.2.1, 1.2.2, 3.1

[29] Robin R Murphy. How robots helped out after the Surfside Condo collapse.

IEEE Spectrum, 2021. 1.1

[30] Milad Ramezani, Kasra Khosoussi, Gavin Catt, Peyman Moghadam, Jason Williams, Paulo Borges, Fred Pauling, and Navinda Kottege. Wildcat: Online continuous-time 3D lidar-inertial SLAM, 2022. 1.1, 1.2.1

[31] Sebastian Scherer, Vasu Agrawal, Graeme Best, Chao Cao, Katarina Cujic, R Darnley, R DeBortoli, E Dexheimer, B Drozd, R Garg, et al. Resilient and modular subterranean exploration with a team of roving and flying robots. *Field Robotics*, 2022. 1.1

[32] Tixiao Shan, Brendan Englot, Drew Meyers, Wei Wang, Carlo Ratti, and Rus Daniela. 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. 1.2.1, 1.2.2

[33] Joan Sola, Jeremie Deray, and Dinesh Atchuthan. A micro Lie theory for state estimation in robotics. *arXiv preprint arXiv:1812.01537*, 2018. 1.2.2, 2.1, 2.1.2

[34] Christiane Sommer, Vladyslav Usenko, David Schubert, Nikolaus Demmel, and Daniel Cremers. Efficient derivative computation for cumulative B-splines on Lie groups. *CoRR*, abs/1911.08860, 2019. URL http://arxiv.org/abs/1911.08860. 2.2

[35] Zhan Wang and Gamini Dissanayake. Observability analysis of SLAM using Fisher information matrix. In *2008 10th International Conference on Control, Automation, Robotics and Vision*, pages 1242–1247. IEEE, 2008. 3.1

[36] Wei Xu, Yixi Cai, Dongjiao He, Jiarong Lin, Fu Zhang, and Fu Zhang. FAST-LIO2: Fast direct lidar-inertial odometry. *IEEE Transactions on Robotics*, 38 (4):2053–2073, 2022. ISSN 1552-3098. doi: 10.1109/tro.2022.3141876. 3.1

[37] Ji Zhang and Sanjiv Singh. LOAM: Lidar odometry and mapping in real-time. In *Proc. Robotics: Science and Systems*, volume 2, 2014. 1.1, 1.2.1, 1.2.2

[38] 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)*, pages 809–816, 2016. doi: 10.1109/ICRA.2016.7487211. 1.2.2

[39] Zhengyou Zhang. Parameter estimation techniques: a tutorial with application to conic fitting. *Image and Vision Computing*, 15(1):59–76, 1997. ISSN 0262-8856. doi: https://doi.org/10.1016/S0262-8856(96)01112-2. URL https://www.sciencedirect.com/science/article/pii/S0262885696011122. 2.3.2, 4.3

[40] Shibo Zhao, Hengrui Zhang, Peng Wang, Lucas Nogueira, and Sebastian Scherer. Super Odometry: IMU-centric LiDAR-visual-inertial estimator for challenging environments, 2021. 1.1, 1.2.1