Sigurd Vatn Totland

Resilient graph-based multi-modal SLAM for sensor-degraded environments

Specialization Project Report in Cybernetics and Robotics
Supervisor: Kostas Alexis
December 2020

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

NTNU

Norwegian University of
Science and Technology

Abstract

Simultaneous Localization and Mapping (SLAM) is a vital capability for robots that is required to accurately understand and navigate their surroundings. While the last 30 years of research efforts have pushed state-of-the-art SLAM systems to perform superbly in many situations, there is still a long way to go before these systems are resilient enough for general use. Current SLAM systems are particularly challenged in industrial, marine and subterranean environments, where vision can be severely degraded. In these cases, robustness can be increased by combining vision with other sensor modalities such as LiDAR or thermal cameras, so that when one is degraded, the other(s) offer the necessary resilience through redundancy and resourcefulness. This project surveys some of the most important single-modality methods and then explores how they can be extended for multi-modal operation. The primary focus of this work is on the SLAM back-end, where the general framework of factor graphs can be used to fuse constraints from several sensor-specific front-ends into a common optimization problem. First, we provide a survey overview of the inner workings of fundamental graph optimization methods, including the frameworks GTSAM and iSAM2. Provided these frameworks, a preliminary loosely coupled fusion system is implemented for LiDAR-visual configuration. The implementation is benchmarked on a dataset gathered in an underground tunnel, where geometric self-similarity severely degrades the LiDAR odometry estimates. Results of these preliminary experiments show that fusing several modalities improves localization accuracy and map consistency compared to single-modality versions, motivating the need for further research on multi-modal perception. The work presented in this report stands as a prior and preliminary effort to help develop a new tight-fusion paradigm for multi-modal SLAM.

Summary

This project report presents a preliminary inquiry into the use of multiple exteroceptive sensor modalities in Simultaneous Localization and Mapping (SLAM) systems using factor-graphs for the purposes of increased resilience to sensor degradation, especially in challenging conditions such as those faced in subterranean, marine or industrial environments.

The first part of the report consists of a survey of some of the currently important components of a modern SLAM system. This starts of from the perspective of the traditional EKF-SLAM approach, and proceeds thereafter to discuss solving the problem using maximum a posteriori optimization with factor graphs. Lastly, the software libraries and algorithms of GTSAM, iSAM and iSAM2 are presented.

The second part of the report is a preliminary investigation into using factor-graphs for sensor fusion of LiDAR and visual data. Here, GTSAM and iSAM2 are used to implement a loosely coupled fusion of a LiDAR-odometry and a visual-odometry method. This is followed with an experimental evaluation on a dataset gathered in an underground tunnel. The results of this show that the LiDAR-only odometry is unable to successfully reconstruct the map and trajectory, whereas the LiDAR-visual fusion is.

Preface

This project report is the product of the TTK4550 specialization project in engineering cybernetics for final-year students of Cybernetics and Robotics at NTNU. The goal of the project is for the student to perform a larger, independent project work of scientific character under supervision. This includes learning specific specialized tools and techniques required for achieving the intended outcomes, obtaining general scientific skills like literature search and scientific writing, and ultimately preparing the student for writing their master thesis.

I would like to thank my supervisor Kostas Alexis for his highly valued feedback and guidance throughout the project. I also want to thank Nikhil Khedekar for all his help on practical matters even when the time zones were not on our side. Finally, my thanks goes to Shehryar Khattak for giving me access to much needed datasets and calibration files.

Contents

1 Introduction

For robots to become truly useful in difficult real-life tasks, they require accurate and robust SLAM systems that work in several different environments and conditions. State-of-the-art systems show that highly accurate localization and mapping is possible with sensors like visual cameras and LiDARs [28, 42]. Yet, most methods focus on a single one of these modalities, thus forgoing the robustness and accuracy gains available through their combination, especially in challenging conditions like those in subterranean, marine or industrial environments where sensors can significantly degrade.

Many exteroceptive sensor modalities are available for SLAM and several successful implementations exist for each one, but they are rarely combined to take advantage of their complementary properties. Visual camera data is a modality highly prevalent in the literature and a wide range of visual SLAM and visual odometry (VO) algorithms exists that are both robust and accurate [1, 12, 25, 28]. This is rightly deserved as visual cameras are cheap, lightweight and energy efficient, yet provide a lot of information about the scene. However, visual data streams can quickly degrade in more challenging situations, such as in the presence of obscurants like dust and smoke or in environments with self-similar texture. LiDAR is an alternative modality that has proven able to accurately solve the SLAM problem [23, 42], but this modality too has weaknesses. Since LiDARs are also light-based, obscurants can deteriorate point cloud measurements, and while LiDARs do not suffer from textural self-similarity, they will fail in case of *geometric* self-similarity. In addition to LiDAR and visual sensors, other modalities have proven useful for the SLAM problem including RGB-D cameras [11, 29, 30], thermal cameras [21] and recently, event cameras [22, 31, 38], each with their own strengths and weaknesses. The complementary properties of all these different sensors make them good candidates for fusion, yet most SLAM literature has only focused on a single exteroceptive modality alone.

While there are existing methods that fuse multiple exteroceptive modalities, they typically rely on a "main" modality that can become a single point of failure. For visual-LiDAR fusion, vision is often used primarily for refinement of the LiDAR estimate. The LiDAR-visual fusion method V-LOAM [41] uses the visual odometry only as a motion prior for the LiDAR odometry. The goal of this fusion is mainly to improve the tracking, and successfully does so, as V-LOAM currently ranks highest on the KITTI odometry benchmark [14]. It does not however add any robustness to LiDAR failure, at least in the current configuration. On a similar note, the recent work [20] also uses visual odometry as a prior for LiDAR odometry, but can additionally detect degeneracy in the LiDAR estimate and perform pass-through of the visual-odometry. While this improves robustness in severely degraded LiDAR conditions, the scheme is boolean, treating the LiDAR odometry as either degenerate or healthy, with no middle ground. This rules out any

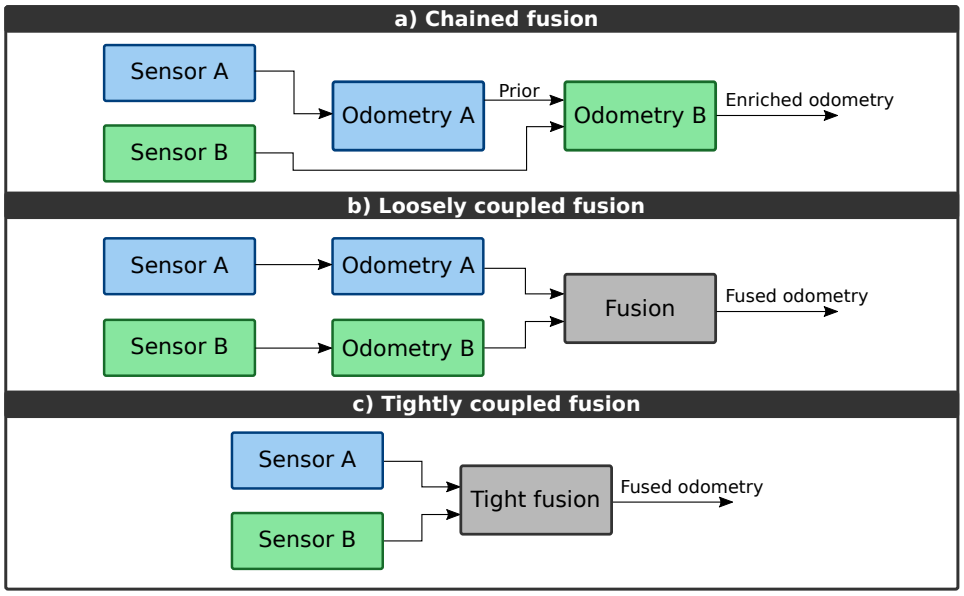

Figure 1: Comparison of odometry fusion schemes

handling of partial degradation. Moreover, as these methods only *chain* the two modalities (as depicted in figure 1 a), using the output of one as the prior of another, they make an implicit decision to trust the final, LiDAR modality the most, rather than basing this decision on uncertainty metrics.

Factor graphs provide a general framework for fusing several, arbitrary modalities and accounting for their uncertainty in changing environments. Many successful current SLAM methods are based on factor graphs, such as the visual method ORB-SLAM3 [3] or the LiDAR method LIO-SAM [32]. Factor graphs are particularly well suited for multi-modal fusion as they can either include constraints from multiple odometry front-ends in a loose coupling (figure 1 b), thus taking advantage of existing high-performing single-modality systems, or include the sensor data *directly* in a tight coupling (figure 1 c), thereby considering all correlations between variables and obtaining a more precise solution [25]. For practical implementations, libraries such as GTSAM [7] and g2o [24] make construction of the factor graphs easier by abstracting away common concerns. The algorithms iSAM [18] and iSAM2 [19] provide an incremental, modality-agnostic solution to factor-graph-based SLAM by exploiting the sparsity of the SLAM problem.

In this report, we survey different SLAM methodologies, with particular focus on factor-graph-based SLAM. We then present a preliminary implementation of a loosely coupled factor-graph-based LiDAR-visual sensor fusion system that can account for LiDAR degradation. We test this system on a dataset with severe self-similar geometry and show that our system successfully retrieves the trajectory and map regardless of LiDAR degradation.

2 Solutions to the SLAM Back-end

Traditionally, the SLAM back-end was solved using filtering methods, but in recent years, the literature has focused more on nonlinear optimization of a factor graph. Factor graphs offer two big benefits. Firstly, they enable simple ways of managing the problem size as the environment is explored to keep the problem tractable. The graph structure clearly mirrors the natural sparsity of the SLAM problem, and we can hence optimize over only a small section of the graph for real time performance. This gives rise to a family of methods referred to as fixed-lag or sliding-window estimators. A second benefit of the factor-graph-based approach is that it makes it easy to formulate different optimization problems, SLAM being one of them. Formulating a problem boils down to just defining the states and the constraints, i.e. factors, between them. Sensor fusion problems like multi-modal SLAM in particular are simply modeled by adding several factors between the same states.

This section aims to provide a brief introduction to the SLAM problem and present the theory underlying the methods used in this project. In addition, some background on the previous state-of-the-art methods is presented, in particular EKF-SLAM since it has historically been such a central part of the literature.

2.1 The SLAM Problem

Simultaneous Localization and Mapping is the problem of simultaneously localizing a robot and at the same time building a map of its environment. It is typically formulated as an online estimation problem where we try to estimate the robot's trajectory and the location of several detected *landmarks* in its environment, given all the measurements received up until now.

Landmark measurements can come from a variety of different sensors. Cameras and LiDARs for instance are some of the most popular sensors for SLAM. With cameras, we obtain a whole image at each timestep containing potentially millions of pixels, so to reduce the computational burden, most methods extract only a handful of *features* and treat these as the measurements instead. Tracking these features as 3D landmarks however, is not straight forward as their 2D pixel locations correspond to an infinite number of valid 3D points. Only by observing the feature from multiple different camera positions, can a 3D estimate of the landmark be estimated, in a process known as *Bundle adjustment* [37].

Due to the need for several measurements of the same feature in consecutive frames, a successful SLAM system needs a way to determine which feature measurements correspond to which landmarks – a process known as data association. There is no one way of doing data association in SLAM, and many different techniques exist. Some SLAM methods known as *di-*

rect methods reformulate the data association step entirely and rely only on projecting the predicted 3D landmark positions into the new image and computing the error in intensity [12].

When given data-associated landmark measurements, the SLAM problem boils down to an online optimization problem, where we seek to obtain a robot trajectory and a set of landmark locations that best describe the received measurements. There are many ways of finding this optimal state, including Kalman filtering and non-linear *maximum a posteriori* (MAP) optimization. Here too, there are many concerns to consider, such as how to keep the problem tractible as the map grows ever larger, how to update the map and trajectory in the event of a detected loop closure and how to balance the tradeoff between accuracy and timeliness needed for real time robotic systems.

Because there are so many different concerns needed to be adressed in a SLAM system, it typically gets broken down into discrete parts. The biggest and most common partition is into a front-end and a back-end. The front-end sits at the front of the pipeline, taking in the raw sensor measurements directly. It has the responsibility of performing feature detection and data association. It will typically also perform landmark initialization when unseen landmarks appear. The back-end on the other hand receives landmark measurements from the frontend and jointly estimates a a robot trajectory and map.

The focus of this report is on the SLAM back-end and in particular the fusion of constraints from multiple frontends of different modalities, such as LiDAR and visual measurements. The remainder of this section will therefore present the theory and frameworks needed to support such a back-end implementation, without focusing much on data association, landmark initialization and other front-end concerns.

2.2 Solving SLAM With Filtering

Kalman filtering is a powerful algorithm for state estimation used across many fields of engineering. Given a linear system model and a linear measurement function corrupted only by Gaussian additive white noise, the Kalman filter is the optimal state estimator and has a closed-form solution. But such nice linear Gaussian conditions are not common in robot localization. As an example, if the robot spins in place around its axis, its landmark measurements will appear to move around it in a circular fashion, clearly following a non-linear motion model. To relax this linear constraint, the Kalman filter is often extended by linearizing the system and measurement models. This gives rise to the Extended Kalman filter (EKF) which has, due to its simplicity and efficiency, enjoyed the status as the default choice for SLAM for a long time. This section presents the workings of the EKF adapted for SLAM. The derivations mostly follow that of [36].

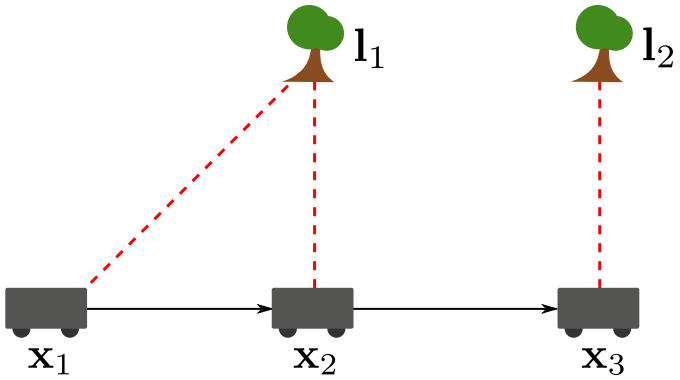

Figure 2: A toy planar SLAM problem showing three consecutive robot poses $\mathbf{x}_1$, $\mathbf{x}_2$ and $\mathbf{x}_3$ along with two landmarks $\mathbf{l}_1$ and $\mathbf{l}_2$. The solid arrows between the robot poses correspond to odometry measurements gathered from e.g. an IMU or wheel encoders, or a markov motion model. The dashed red lines between landmarks and poses correspond to bearing-range measurements, say from a laser scanner. Figure adapted from [9].

In the SLAM problem as described in section 2.1, we seek to find a robot trajectory and a set of landmarks that best explain the received measurements. Due to the Markov property of the Kalman filter however, we are restricted to only keep the newest robot pose in the state, instead of its whole sequence. We can of course store a copy of the robot trajectory outside the EKF, adding the newest pose to it at every timestep, but then the older parts of the sequence will not be a part of the estimation problem and cannot be corrected upon. As we will see later, methods based on nonlinear maximum a posteriori estimation can in contrast optimize over all poses, or a sliding window of poses, enabling them to make such corrections backwards in time and get a more consistent trajectory. For real time robotics operations however, this is not too much of a problem, as an estimate of the current robot pose is typically all that is needed. Hence, the goal for EKF-SLAM is to estimate a joint state of landmarks and the current pose, from measurements of a subset of the landmarks coming in at every timestep.

To facilitate the discussion, we will consider a toy SLAM problem adapted from [9] that is shown in figure 2. In this 2D problem, a robot is traversing a terrain for three timesteps and collecting bearing-range measurements of two trees. It receives odometry measurements from wheel encoders enabling it to predict its motion between the poses. The SLAM objective is to determine the pose of the robot for all three timesteps as well as the location of the two trees. As mentioned before, we assume that a front-end performs data association, so that we receive the range-bearing measurements labeled with the landmarks they correspond to.

2.2.1 State representation

For our discussion of EKF-SLAM, we will denote the state as $\mathbf{x}_t = [\mathbf{x}_t^x, \mathbf{x}_t^l]^\top$, where $\mathbf{x}_t^x$ denotes the part of the state corresponding to the robot pose and $\mathbf{x}_t^l$ corresponds to the vector of landmarks. The term *pose* has in the SLAM and navigation literature come to mean the combined position and orientation of the robot. In particular, for the 2D toy SLAM example, this corresponds to a three-element vector $\mathbf{x}_t^x = [x_t, y_t, \theta_t]^\top$ where x_t and y_t are the 2D coordinates of the robot position and θ_t is the orientation. The landmarks are represented by their 2D locations in the world, meaning a vector of m landmarks will be $\mathbf{x}_t^l = [l_{x,t}^1, l_{y,t}^1, ..., l_{x,t}^m, l_{y,t}^m]^\top$. This of course generalizes easily to 3D as well, by adding a third $l_{z,t}^j$ term to each landmark. The control vector $\mathbf{u}_t$ describes how the robot moves both in position and orientation, so that too will be a pose vector $\mathbf{u}_t = [u_t, v_t, \phi_t]^\top$. Finally, for the measurements, we assume as mentioned known correspondences to their respective landmarks, meaning we can write them in a vector corresponding to $\mathbf{x}_t^l$, i.e $\mathbf{z}_t = [\mathbf{z}_t^1, ..., \mathbf{z}_t^m]^\top$. For a range-bearing sensor, each measurement contains a range r_t and a bearing ρ_t, so $\mathbf{z}_t^j = [r_t^j, \rho_t^j]^\top$. Keep in mind that we do not typically receive measurements for all landmarks, so special care must be taken when updating the state with measurements. The definitions given here are of course specific to planar SLAM with bearing-range measurements, but can be easily generalized to 3D SLAM as well and include other types of sensors. For instance, when using IMUs to obtain the odometry vector, it is common to augment the state vector $\mathbf{x}_t$ with the IMU bias terms so they can be estimated online.

2.2.2 State estimation

The EKF is a recursive state estimator that calculates a new state x_t, from the previous state x_{t-1} and the measurements z_t received. The state and the measurements are respectively given by nonlinear functions g and h and assumed corrupted by zero-mean Gaussian distributed noise vectors, $\mathbf{w}_t \sim \mathcal{N}(0, R_t)$ and $\mathbf{v} \sim \mathcal{N}(0, Q_t)$ so that the state propagates according to the system model

$$\begin{aligned} \mathbf{x}_t &= g(\mathbf{u}_t, \mathbf{x}_{t-1}) + \mathbf{w}_t \\ \mathbf{z}_t &= h(\mathbf{x}_t) + \mathbf{v}_t. \end{aligned} \tag{1}$$

Since $\mathbf{x}_t$ has a generic state representation depending on the problem formulation, the function g can have several different formulations. Even so, all state representations have one property in common, namely that the landmarks are assumed to be rigidly fixed in the world. As a result, they are unchanged by the Markov update, allowing us to write it as

$$g(\mathbf{u}_t, \mathbf{x}_{t-1}) = \begin{bmatrix} g^x(\mathbf{u}_t, \mathbf{x}_{t-1}^x) \\ \mathbf{x}_{t-1}^l \end{bmatrix} \tag{2}$$

where g^x is a generic update function for the robot states only. As will be explained later, this split is beneficial since it manifests itself in a sparsity pattern in the Jacobian. For the toy 2D problem in figure 2 the state prediction becomes

$$g^x(\mathbf{u}_t, \mathbf{x}^x_{t-1}) = \begin{bmatrix} x_{t-1} + u_t \cos(\theta_{t-1}) - v_t \sin(\theta_{t-1}) \\ y_{t-1} + u_t \sin(\theta_{t-1}) + v_t \cos(\theta_{t-1}) \\ \theta_{t-1} + \phi_t \end{bmatrix}. \tag{3}$$

The measurement prediction h predicts the value of landmark measurements $\mathbf{z}_t$ from the current state $\mathbf{x}_t$. We will split this up into functions h^j for each landmark l^j, so that $h(\mathbf{x}_t) = [h^1(\mathbf{x}^x_t, \mathbf{l}^1), ..., h^m(\mathbf{x}^x_t, \mathbf{l}^m_t)]^\top$. For the 2D range-bearing measurements of the toy SLAM problem, this becomes

$$h^j(\mathbf{x}^x_t, \mathbf{l}^j_t) = \begin{bmatrix} r^j_t \\ \rho^j_t \end{bmatrix} = \begin{bmatrix} \|\boldsymbol{\delta}_t\| \\ \text{atan2}(\delta_{y,t}, \delta_{x,t}) - \theta_t \end{bmatrix}, \tag{4}$$

where we have defined the $\boldsymbol{\delta} = [\delta_{x,t}, \delta_{y,t}]^\top$ as the displacement between the robot and the landmark

$$\boldsymbol{\delta}_t = \begin{bmatrix} \delta_{x,t} \\ \delta_{y,t} \end{bmatrix} = \begin{bmatrix} l^j_{x,t} - x_t \\ l^j_{y,t} - y_t \end{bmatrix}. \tag{5}$$

It is important to realize that the KF and EKF are probabilistic state estimators, meaning $\mathbf{x}_t$ represents the current estimate of the robot pose, and not the actual robot pose itself. They are both Gaussian filters meaning they maintain a Gaussian distribution around the mean $\mathbf{x}_t$ with covariance Σ_t. This probabilistic representation is natural as we can never know the exact state of the robot, only a distribution around the current belief. For a problem with n robot states and $d \cdot m$ landmark states, where d is the dimension of the landmark representation, Σ_t becomes a $(n+dm) \times (n+dm)$ matrix. Because $\mathbf{x}_t$ is divided into robot states and landmark states, it is useful to define notation for the specific parts of the covariance matrix corresponding to these parts of $\mathbf{x}_t$. We hence write

$$\Sigma_t = \begin{bmatrix} \Sigma^{x,x}_t & \Sigma^{x,l}_t \\ \Sigma^{x,l}_t & \Sigma^{l,l}_t \end{bmatrix}, \tag{6}$$

where $\Sigma^{x,x}_t$ is $d \times d$, $\Sigma^{x,l}_t = \Sigma^{l,x\top}_t$ is $d \times dm$ and $\Sigma^{l,l}_t$ is $dm \times dm$.

The EKF filter algorithm is divided into two steps, a *predict* step computed entirely from the system model, the previous state $\mathbf{x}_{t-1}$ and optional control inputs or odometry measurements $\mathbf{u_t}$, as well as an *update* step based on the incoming measurement $\mathbf{z}_t$. In the prediction step, we compute an intermediate state with modes $\bar{\mathbf{x}}_t$ and $\bar{\Sigma}_t$ according to

$$\begin{aligned} \bar{\mathbf{x}}_t &= g(\mathbf{u}_t, \mathbf{x}_{t-1}) \\ \bar{\Sigma}_t &= G_t \Sigma_{t-1} G_t^\top + R_t. \end{aligned} \tag{7}$$

Here, R_t is the covariance of the random noise component $\mathbf{w}_t$, while G_t is the Jacobian of g evaluated at $\mathbf{x}_{t-1}$. When a new measurement $\mathbf{z}_t$ arrives, the filter updates the state with the new information resulting in the next mean and covariance $\mathbf{x}_t$ and Σ_t. These are computed according to

$$\begin{aligned} K_t &= \bar{\Sigma}_t H_t^\top (H_t \bar{\Sigma} H_t^\top + Q_t)^{-1} \\ \mathbf{x}_t &= \bar{\mathbf{x}}_t + K_t(\mathbf{z}_t - h(\bar{\mathbf{x}}_t)) \\ \Sigma_t &= (I - K_t H_t)\bar{\Sigma}_t, \end{aligned} \tag{8}$$

where Q_t is the covariance of the random measurement noise component $\mathbf{v}_t$ and H_t is the Jacobian of h computed at $\bar{\mathbf{x}}_t$. The matrix K_t is known as the *Kalman gain* and is the optimal gain in the linear Gaussian case, but for the linearized EKF is not. The residual $\mathbf{z}_t - h(\bar{\mathbf{x}}_t)$ is known as the *innovation* term and can be thought of as a correction, that gets weighted with the Kalman gain and added to the predicted mean.

2.2.3 Linearization

The EKF differs from the regular KF in that the functions g and h can be non-linear. For this reason, it needs to use the Jacobians G_t and H_t in place of what in the linear case would be the system and measurement matrices. The linearization is computed as a first-order Taylor approximation at the most recent mean estimate, i.e.

$$\begin{aligned} g(\mathbf{u}_t, \mathbf{x}_{t-1}) &\approx g(\mathbf{u}_t, \mathbf{x}_{t-1}) + G_t(\mathbf{x}_{t-1} - \mathbf{x}_{t-1}), \text{and} \\ h(\mathbf{x}_t) &\approx h(\bar{\mathbf{x}}_t) + H_t(\mathbf{x}_t \bar{\mathbf{x}}_t), \end{aligned} \tag{9}$$

where the Jacobians are given as

$$G_t = \frac{\partial g(\mathbf{u}_t, \mathbf{x}_{t-1})}{\partial \mathbf{x}_{t-1}}, \quad \text{and} \quad H_t = \frac{\partial g(\mathbf{x}_t)}{\partial \mathbf{x}_t}. \tag{10}$$

From the decomposition of the state prediction function g given in (2), we see that the Jacobian G_t has the sparse structure

$$G_t = \begin{bmatrix} \frac{\partial g^x(\mathbf{u}_t, \mathbf{x}^x_{t-1})}{\partial \mathbf{x}^x_{t-1}} & 0 \\ 0 & I \end{bmatrix}. \tag{11}$$

While G_t will always be sparse, the measurement update Jacobian H_t can potentially be fully dense. However, a sparsity pattern typically arises here as well since in most cases, only a subset of the landmarks in x_t^m are observed and hence only their respective terms in the Jacobian are non-zero. The sparsity patterns in these matrices means that the computations involved in (7) and (8) can be computed efficiently.

2.2.4 Landmark initialization

In any real-life SLAM scenario, the landmarks will not be known up-front and so the EKF-SLAM implementation must be able to initialize new landmarks. Any measurements the front-end cannot associate with an existing measurement becomes a candidate for a new landmark. After that, it can be added as a newly initialized landmark to the state vector and the covariance matrix. For that, the new landmarks are appended to the state vector $\mathbf{x}_t$ and the covariance matrix Σ_t augmented with an initial (high) variance for the landmarks.

2.2.5 Benefits and limitations of filtering approaches

The simplicity and efficiency of the EKF has made the it a popular choice for SLAM systems in the last 30 years. Since the filter only retains the latest robot pose, the size of the problem is kept relatively tractable, and hence enables longer operation. The computational complexity is quite low at only $O(m^2)$ where m is the number of features in the map [36].

The EKF does however have several drawbacks, so as advancements in hardware and algorithms have made more resource intensive methods feasible, EKF-based SLAM has fallen increasingly out of favor [2]. One of the primary drawbacks of the EKF comes from the linearization. Because the problem is linearized at every timestep, linearization errors accumulate. Another drawback is due to the Markov assumption that underlies the KF, namely that the current state can only depend on the previous. This means all old robot poses get marginalized out of the problem and instead baked into the joint probability distribution. This assumption makes it impossible to update old states with new information, as is for instance needed to add a loop closure constraint, without expensively rolling the KF back and forth again. This marginalization also has the effect of adding correlations between existing landmarks, hence making the covariance matrix increasingly dense [34]. In fact, EKF-SLAM in the limit makes all the landmarks fully correlated [36]. This slows down the EKF over time and makes it less suitable for long term operation.

Due to some of these issues, authors like [34] argue methods based on non-linear optimization are better suited than filtering approaches for solving the SLAM problem. This is further backed up by the fact that most recent state-of-the-art systems also happen to be based on non-linear optimization [3]. Yet, there are recent examples of EKF-based methods that show great performance, such as [1] and [27].

2.3 Solving SLAM with Factor Graphs and Nonlinear Optimization

The EKF has enjoyed a long history as the de-facto solution to SLAM regardless of its downsides, but as computers have become more powerful, solutions based on more computationally demanding but also more accurate non-linear optimization have become feasible. Most methods using non-linear optimization represent the SLAM problem as a graphical model, the most common of which being the factor graph [2].

The section closely follows that of the excellent tutorial on factor graphs for robot perception by Daellart and Kaess [9]. Another great introduction to the theory is given in [15] although it does not explicitly refer to the graphical model as a factor graph.

2.3.1 Maximum a posteriori estimation

To see how the factor graph representation arises as a natural model for the back-end, we must look at how to solve SLAM using *maximum a posteriori* (MAP) estimation. For SLAM, the states we are interested in are the robot's trajectory of poses and the locations of landmarks it observes. The state could in addition include other variables of interest such as IMU bias terms or camera extrinsics and intrinsics parameters. By defining X as the collection of these states, and Z as measurements, the *posterior density* $p(X|Z)$ is the probability density over X given the measurements Z. Naturally, to obtain a state that is as close as possible to the actual ground truth, we should find the X that maximizes this posterior, leading to the MAP estimate

$$X^{MAP} = \arg\max_X p(X|Z). \tag{12}$$

Using Bayes' rule we can rewrite the posterior as

$$p(X|Z) = \frac{p(Z|X)p(X)}{p(Z)}. \tag{13}$$

The measurements Z are given, so the density $p(Z)$ can be treated as a constant, letting us instead express the posterior in the simpler form

$$p(X|Z) \propto p(Z|X)p(X). \tag{14}$$

Due to the proportionality, the maxima of these expressions coincide, so

$$X^{MAP} = \arg\max_X \frac{p(Z|X)p(X)}{p(Z)} = \arg\max_X p(Z|X)p(X). \tag{15}$$

As is done in [9], we will also here define the likelihood $l(X;Z)$ as *any function proportional to* $p(Z|X)$. Again, by proportionality, this does not

change the argmax of the objective. The likelihood expresses the probability of observing the measurements Z, given the state X, but since Z is given, it is really a function of X. This notation helps emphasize this by more clearly showing which parameters are known and which are not. With this, (14) can be rewritten as

$$p(X|Z) \propto l(X;Z)p(X). \tag{16}$$

These likelihood functions as we have defined them, do not need to be properly normalized probability distributions that sum to one. This is part of the reason why factor graphs are such suitable graphical models for MAP estimation, compared to e.g. Bayes nets where each function must be a proper probability density [9].

2.3.2 Factorizing the MAP objective

The SLAM posterior is a probability density over all robot poses and landmarks in the set X, given all measurements Z. If we assume the noise statistics of the measurements are independent of one another, the posterior factorizes into a product of densities. Consider for instance the toy SLAM problem used throughout [9] that we also made use of in our discussion of the EKF. For the MAP estimation problem to be valid, we need an initial absolute measurement $\mathbf{z}_1$ on the first pose $\mathbf{x}_t$. We denote the three bearing-range measurements as $\mathbf{z}_2$, $\mathbf{z}_3$ and $\mathbf{z}_4$. We can then visualize the resulting network of states and measurements in a graphical model known as a Bayes' net. Figure 3 shows this network. Notice how the directed edges denote the dependencies between variables. The measurement $\mathbf{z}_2$ for instance depends both on the location of the landmark it is measuring, and the pose the robot was in when the measurement was received. The joint probability density for this network is

$$\begin{aligned} p(X,Z) = & \, p(\mathbf{x}_1)p(\mathbf{x}_2|\mathbf{x}_1)p(\mathbf{x}_3|\mathbf{x}_2) \\ & \times p(\mathbf{l}_1)p(\mathbf{l}_2) \\ & \times p(\mathbf{z}_1|\mathbf{x}_1) \\ & \times p(\mathbf{z}_2|\mathbf{x}_1,\mathbf{l}_1)p(\mathbf{z}_3|\mathbf{x}_2,\mathbf{l}_1)p(\mathbf{z}_4|\mathbf{x}_3,\mathbf{l}_2). \end{aligned} \tag{17}$$

The first line of the product is the priors corresponding to the markov motion model of the robot movement and the second line priors on the landmark locations (something often unavailable in SLAM with no prior map). The conditional distributions on the third and fourth line correspond respectively to the initial absolute pose measurement $\mathbf{z}_1$ and the three landmark measurements in the poses they were observed.

Conditioning (17) on the measurements Z, we obtain the posterior

$$p(X|Z) = \frac{p(X,Z)}{p(Z)} \tag{18}$$

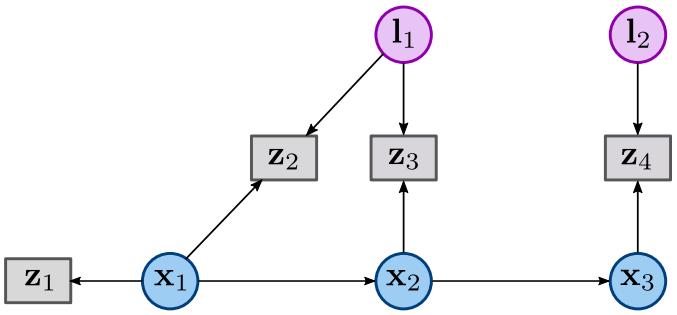

Figure 3: Bayes net for the toy SLAM problem showing robot poses $\mathbf{x}_{1:3}$, landmark locations $\mathbf{l}_{1:2}$, the initial pose measurement $\mathbf{z}_1$ and landmark measurements $\mathbf{z}_{2:4}$. Circular colored nodes denote variables, whereas rectangular grey nodes denote measurements. The directed edges denote the dependence relationships. Figure adapted from [9].

and as we saw in section 2.3.1, the factors in the posterior can be expressed as proportional to a product of priors and likelihoods to eliminate the irrelevant prior on the measurements $p(Z)$. This allows us to formulate the posterior as

$$\begin{aligned} p(X|Z) \propto\ & p(\mathbf{x}_1)p(\mathbf{x}_2|\mathbf{x}_1)p(\mathbf{x}_3|\mathbf{x}_2) \\ & \times p(\mathbf{l}_1)p(\mathbf{l}_2) \\ & \times l(\mathbf{x}_1;\mathbf{z}_1) \\ & \times l(\mathbf{x}_1,\mathbf{l}_1;\mathbf{z}_2)l(\mathbf{x}_2,\mathbf{l}_1;\mathbf{z}_3)l(\mathbf{x}_3,\mathbf{l}_2;\mathbf{z}_4). \end{aligned} \tag{19}$$

This factorization motivates the need for a new graphical modeling framework, namely the factor graph. Bayesian networks are great at modeling networks of Gaussian probability distributions but in contrast to factor graphs, they can not model the generally non-linear likelihood functions [9]. Indeed, the need to treat Gaussian densities and likelihood functions alike in the same MAP estimation problem is the main motivation for the move to factor graphs in favor of Bayes nets [9]. To see how such non-linear likelihood functions appear, even under assumptions of Gaussian measurement noise, consider for instance a robot positioned at the origin receiving a measurement $z \in (-\pi, \pi]$ of a single landmark $\mathbf{l} = [l_x, l_y]^\top$ with a bearing-only sensor. The measurement model conditioned on the landmark is

$$p(z|\mathbf{l}) = \frac{1}{\sqrt{|2\pi R|}} \exp\left\{ -\frac{1}{2} \| \operatorname{atan2}(l_x, l_y) - z \|_R^2 \right\}, \tag{20}$$

where R is the covariance of the noise model of bearing-only sensor. While this function is Gaussian when seen as a probability density over the measurement z, we are in the MAP optimization scenario interested in the likelihood function $l(\mathbf{l}; z) \propto p(z|\mathbf{l})$ over the variable $\mathbf{l}$, where z is considered fixed. Taken as a function over $\mathbf{l}$ instead of z, (20) is clearly non-linear and non-Gaussian and so a Bayes net cannot be used.

To graphically model non-linear likelihoods and probability densities alike, we use a factor graph instead of a Bayes net. With factor graphs, we no longer treat measurements as variables, but rather as the immutable quantities they really are. We do so by defining a factor ϕ_i for each received measurement $\mathbf{z}_i$, that is a function of the variables conditioned on that measurement. As an example, the likelihood on the bearing-only measurement $l(\mathbf{l}, z)$ would correspond to a factor $\phi(\mathbf{l})$. Likewise, the posterior of the toy slam problem in (19) can be expressed as

$$\begin{aligned} p(X|Z) \propto \phi(X) = \phi_1(\mathbf{x}_1)\phi_2(\mathbf{x}_2, \mathbf{x}_1)\phi_3(\mathbf{x}_3, \mathbf{x}_2) \\ \times \phi_4(\mathbf{l}_1)\phi_5(\mathbf{l}_2) \\ \times \phi_6(\mathbf{x}_1) \\ \times \phi_7(\mathbf{x}_1, \mathbf{l}_1)\phi_8(\mathbf{x}_2, \mathbf{l}_1)\phi_9(\mathbf{x}_3, \mathbf{l}_2). \end{aligned} \tag{21}$$

The factor graph for this posterior is shown in figure 4.

Formally, a factor graph is a bipartite graph $F = (\Omega, X, E)$ with factor nodes $\phi_i \in \Omega$ and variable nodes $\mathbf{x}_j \in X$ connected by edges $e_{ij} \in E$. Every factor ϕ_i is a function of the variables $X_i \subset X$ adjacent to it. The factor graph thus defines a factorization of a function into the product [9]

$$\phi(X) = \prod_i \phi(X_i). \tag{22}$$

Since each factor ϕ_i in a factor graph is a function of only its adjacent variables with the corresponding measurement $\mathbf{z}_i$ "baked in", it allows for abstracting out the concept of measurements entirely, enabling us instead to conceptualize measurements as constraints between variables. This lets us phrase the MAP optimization problem as the problem of smoothing a factor graph. This perspective has given rise to a family of algorithms for SLAM known as smoothing and mapping solutions to SLAM [7] [8] [18] [19]. Figure 4 shows the factor graph for the toy SLAM problem.

2.3.3 Non-linear least squares

Under the assumption of Gaussian measurement noise, the MAP objective corresponds to a least squares optimization problem. Gaussian measurement noise means all the factors, both priors and likelihoods, can be written on the form

$$\phi_i(X_i) \propto \exp\left\{\frac{1}{2}\|h_i(X_i) - \mathbf{z}_i\|^2_{\Sigma_i}\right\} \tag{23}$$

where h_i is an arbitrary, possibly nonlinear measurement function and $\|.\|^2_{\Sigma_i}$ denotes the squared Mahalanobis distance[1] for a measurement with covariance Σ_i [9]. Taking the logarithm of this turns the product of exponentials

[1]Informally, the Mahalanobis distance is a measure of the distance from a point to a probability distribution. It can be seen as an extension to higher dimensions of counting the number of standard deviations from the mean.

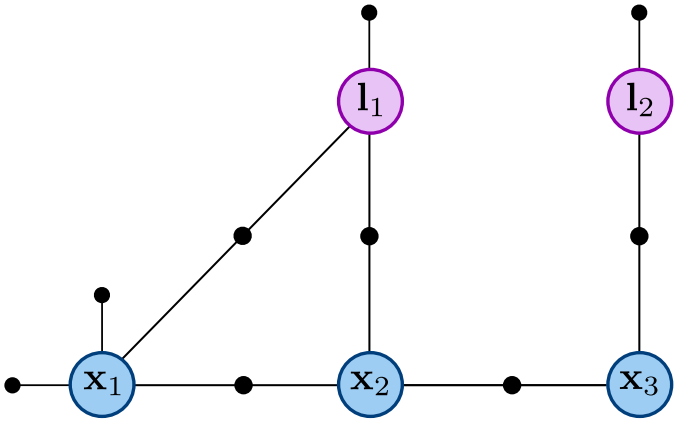

Figure 4: The factor graph for the toy SLAM problem corresponding to the posterior in (19). Variable nodes are shown as big colored circles and factor nodes as small black circles. Notice that $\mathbf{x}_1$ has a factor ϕ_6 corresponding to the initial absolute pose measurement and a regular prior factor ϕ_1. Figure adapted from [9].

into a sum of the exponents, and since the logarithm is a monotonically increasing function, applying it does not change the location of the maximum. Scaling by $\frac{1}{2}$ and flipping the sign turns this into a nonlinear least squares minimization problem

$$X^{MAP} = \arg\min_{X} \sum_{i} \|h_i(X_i) - \mathbf{z}_i\|^2_{\Sigma_i}. \tag{24}$$

In the following, we will see how this can be solved using nonlinear optimization methods such as Gauss Newton or Levenberg-Marquardt.

The least squares objective in (34) is nonlinear due to the nonlinear measurement functions h_i, meaning a nonlinear optimization method is needed to optimize it. One often used non-linear optimization solver is Gauss-Newton (GN) however most smoothing based SLAM methods use an extension of this method called Levenberg-Marquardt (LM) that can additionally reject bad steps and fine-tune its step size to ensure convergence [25] [28]. Both these algorithm iterate a basic three step process of 1) linearizing the objective function at the current estimate, 2) solving the resulting linear least squares problem with the normal equations and 3) updating the estimate with the solution obtained from the linear problem.

For values in Euclidean vector spaces, the linearization performed in GN and LM is a first order Taylor expansion of the measurement functions,

$$h_i(X_i) = h_i(X_i^0 + \Delta_i) \approx h_i(X_i^0) + H_i\Delta_i \tag{25}$$

where H_i is the measurement Jacobian calculated at the current estimate X_i^0 as

$$H_i \triangleq \left.\frac{\partial h_i(X_i)}{\partial X_i}\right|_{X_i^0} \tag{26}$$

and Δ_i is the update vector GN or LM will make. In general however, values do not lie in vector spaces and special care must be taken to compute the Jacobian. In particular, the rotational component of a robot pose lies on the non-linear manifold $SO(3)$. Some SLAM algorithms handle this specially, but as we will see in section 2.4, using Lie theory, $SO(3)$ and vector spaces can be treated with the same general framework.

Inserting (25) into (34), we get a linear least squares problem

$$\begin{aligned}\Delta^* &= \arg\min_{\Delta} \sum_i \|h_i(X_i^0) + H_i\Delta_i - \mathbf{z}_i\|^2_{\Sigma_i} \\ &= \arg\min_{\Delta} \sum_i \|H_i\Delta_i - (\mathbf{z}_i - h_i(X_i^0))\|^2_{\Sigma_i}\end{aligned} \tag{27}$$

which can be solved for the update step Δ^* by solving the normal equations.

To solve (27) using the normal equations, we first formulate it as a standard least-squares problem. The Mahalanobis norm $\|\cdot\|^2_{\Sigma_i}$ for an error term $\mathbf{e}$ can be written as

$$\|\mathbf{e}\|^2_{\Sigma_i} = \mathbf{e}^\top\Sigma_i^{-1}\mathbf{e} = \left(\Sigma_i^{-1/2}\mathbf{e}\right)^\top\left(\Sigma_i^{-1/2}\mathbf{e}\right) = \|\Sigma_i^{-1/2}\mathbf{e}\|_2^2. \tag{28}$$

Hence, by defining the substitutions

$$\begin{aligned}A_i &= \Sigma_i^{-1/2}H_i, \\ \mathbf{b}_i &= \Sigma_i^{-1/2}\Big(\mathbf{z}_i - h_i(X_i^0)\Big)\end{aligned} \tag{29}$$

we obtain the standard least-squares problem

$$\begin{aligned}\Delta^* &= \arg\min_{\Delta} \Sigma_i\|A_i\Delta_i - \mathbf{b}_i\|_2^2 \\ &= \arg\min_{\Delta} \|A\Delta - \mathbf{b}\|_2^2.\end{aligned} \tag{30}$$

This lets us solve the problem by solving the normal equations,

$$(A^\top A)\Delta^* = A^\top\mathbf{b}. \tag{31}$$

2.3.4 Gauss-Newton

The Gauss-Newton (GN) algorithm for solving non-linear least-squares problems that takes curvature into account by approximating the Hessian with the squared Jacobian matrix $A^\top A$. It iteratively updates its solution by repeatedly linearizing the objective to obtain the linear least squares problem (27), and then solve the normal equations (31). That is, the update vector is obtained as

$$\Delta^* = (A^\top A)^{-1}A^\top\mathbf{b}. \tag{32}$$

There are however some downsides to the basic GN scheme. Firstly, for particularly non-linear problems, the update step can perturb the estimate to a worse estimate than before. Secondly, inverting the $A^\top A$ matrix can fail if $A^\top A$ is singular, or close to singular. This situation corresponds to a degenerate problem formulation in which parts of the solution space is unobservable [17].

2.3.5 Levenberg-Marquardt

Levenberg-Marquardt is a damped extension of Gauss-Newton that will reject steps if they increase the objective. It defines a damping factor $\lambda > 0$ that decides how far the algorithm will step in a single iteration. This is achieved by instead solving the augmented version of the normal equations,

$$(A^\top A + \lambda \text{diag}(A^\top A))\Delta^* = A^\top \mathbf{b}. \tag{33}$$

Large values of λ result in small steps, and small values of λ in large. If a step increases the value of the objective function, λ is increased (say by a factor of 10 [9]) to make the algorithm more cautious. Conversely, if a step reduces the objective, λ is decreased. Adding the $\text{diag}(A^\top A)$ term has the effect of making the step size larger in flat directions and more cautious in steep directions [9]. Finally, the $\text{diag}(A^\top A)$ term also ensures the system can be solved for the update step Δ^*, even in the case of a singular $A^\top A$.

2.3.6 MAP estimation vs the EKF

As with the EKF, GN and LM rely on linearization to approximate the optimum, but unlike the EKF, they allow repeatedly linearizing and updating their estimates to find a better approximation. Smoothing solutions that use these iterative non-linear optimization schemes also do not marginalize out old states every iteration, meaning that bad linearizations can get corrected in future iterations as more data becomes available. A second effect of this longer-horizon smoothing approach is that factors can be added with latency, such as loop closures or slowly computed measurements. These benefits are some of the reasons why non-linear optimization of factor graphs has become such a popular approach to solving the SLAM problem.

2.4 GTSAM

Georgia Tech Smoothing and Mapping (GTSAM) is a general factor graph optimization framework developed by F. Daellert, M. Kaess, et. al. at Georgia Tech. The framework is general in its nature and can optimize a variety of problems provided the necessary Lie theoretic group operations are defined.

2.4.1 Motivation: The $SO(3)$ manifold

The generality of GTSAM owns to the efficacy of the underlying Lie theoretic machinery it builds upon. We will, as is done in [9], begin our discussion of Lie theory with a motivating example central to the SLAM problem, namely how to deal with the rotational part of the robot state lying on the non-linear manifold $SO(3)$. In the iterative non-linear optimization procedures of GN and LM as discussed in section 2.3, we must linearize and solve the objective

$$\sum_i \|h_i(X_i) - \mathbf{z}_i\|^2_{\Sigma_i} \tag{34}$$

to find an update vector Δ^*, and then increment our current estimate with it, according to

$$X^{t+1} = X^t + \Delta^*. \tag{35}$$

Recall however, that the state X_i^t contains a set of robot poses, and thus a set of robot orientations. Let us denote one such orientation $R_0 \in SO(3)$ and consider the single square error term corresponding to it in (34):

$$\|h_0(R_0) - \mathbf{z}_0\|^2_{\Sigma_0}. \tag{36}$$

We have defined $z_0 \in \mathbb{R}^3$, $h_0 : SO(3) \to \mathbb{R}^3$ and $\Sigma_0 \in \mathbb{R}^{3\times3}$ respectively to be the corresponding measurement, measurement function and covariance of R_0.

There are different ways of expressing the orientation. One representation is that of Euler angles, where the orientation is expressed by three angles, φ, θ and ψ. This representation has the benefit of being minimal, but has the major downside of multiple singularities, e.g. at $\theta = \pi/2$. This makes it impractical to use in robot navigation. An often used alternative to Euler angles is rotation matrices, commonly defined as the set

$$SO(3) = \{R \in \mathbb{R}^{3\times3} | RR^\top = I, \det R = 1\}. \tag{37}$$

This representation is overparametrised: 3×3 rotation matrices require 9 parameters to represent a state with only 3 degrees of freedom. This redundancy means the actual valid rotation matrices, forming the $SO(3)$ rotation group, lie on a non-linear manifold within the 9-dimensional Cartesian space of all 3×3 matrices. To add an increment $\xi^\wedge \in \mathbb{R}^{3\times3}$, we might be tempted to simply add with the common addition operator, to obtain

$$R = R_0 + \xi^\wedge. \tag{38}$$

However, this result is very unlikely to lie on the $SO(3)$ manifold. In fact, it will in general not, because the group of $SO(3)$ rotation matrices is not closed under addition. We instead seek a more general "addition" operator $\oplus$ that we can use to add perturbations to orientations.

A minimal representation often used for this purpose is the axis-angle parametrization where the rotation is described with an angle θ and a unit vector $\bar{\omega}$ along the axis of rotation. Since $\bar{\omega}$ is constrained to unit length, it can be expressed as only two parameters, meaning a three parameter vector is enough to represent the rotation. One such three parameter vector is obtained as the product of angle and axis, namely

$$\xi = \theta\bar{\omega} = \begin{bmatrix} \xi_x \\ \xi_y \\ \xi_z \end{bmatrix} \tag{39}$$

This vector corresponds to the skew symmetric matrix, which we denote with the hat $(\cdot)^\wedge$ operator,

$$\xi^\wedge = [\xi]_\times = \begin{bmatrix} 0 & -\xi_z & \xi_y \\ \xi_z & 0 & -\xi_x \\ -\xi_y & \xi_x & 0 \end{bmatrix} \tag{40}$$

For small θ, computing the perturbation as $R \approx R_0 + \xi^\wedge$ is a reasonable approximation, however R will still not be on the manifold, only close to it [9]. To obtain a valid estimate, we can use Rodrigues' formula to obtain the rotation matrix corresponding to the angle-axis representation

$$R(\xi) = I + \frac{\sin\theta}{\theta}\xi^\wedge + \frac{1-\cos\theta}{\theta^2}\xi^{\wedge 2}. \tag{41}$$

Multiplying this with the original rotation matrix perturbs it to a valid new rotation matrix, and is hence the generalized $\oplus$ operator we are after:

$$R = R_0 \oplus \xi = R_0 R(\xi). \tag{42}$$

This will let us add the increment in (35) even when the state contains orientations.

The new $\oplus$ operator also enables defining a linearization of the error term in (36), according to

$$h_0(R_0 \oplus \xi) \approx h_i(R_0) + H_0\xi, \tag{43}$$

where H_0 is the Jacobian of h_0 calculated at R_0 [9]. For details on how the Jacobian can be calculated, refer to [33]. With this linearization in hand, we can solve the linear least squares problem in the tangent space, i.e. we can formulate and solve the linear least squares problem as in (27) even when rotations are involved. As [9] explains, a simple scheme for using GN and LM with rotation matrices is to iteratively solve the linear least squares problem according to (27) and then update the estimate according to (42).

2.4.2 Lie theory

What we have presented up until now is only the specific way to handle the particularities of the rotation group $SO(3)$. It turns out however that these definitions fit into the more general framework of Lie groups. Expressing it in the general language of Lie theory allows us to work with several categories of values with a common framework.

A Lie group is a smooth manifold that satisfies the group axioms and locally resembles a linear space [33]. As described in [33], a group $(\mathcal{G}, \circ)$ is a set $\mathcal{G}$ together with an operator $\circ$ that for elements $\mathcal{X}, \mathcal{Y}, \mathcal{Z}, \mathcal{E} \in \mathcal{G}$ satisfy the axioms

$$\begin{aligned} \text{Closure under } \circ: &\quad \mathcal{X} \circ \mathcal{Y} \in \mathcal{G} && (44)\\ \text{Identity } \mathcal{E}: &\quad \mathcal{E} \circ \mathcal{X} = \mathcal{X} \circ \mathcal{E} = \mathcal{X} && (45)\\ \text{Inverse } \mathcal{X}^{-1}: &\quad \mathcal{X}^{-1} \circ \mathcal{X} = \mathcal{X} \circ \mathcal{X}^{-1} = \mathcal{E} && (46)\\ \text{Associativity}: &\quad (\mathcal{X} \circ \mathcal{Y}) \circ \mathcal{X} = \mathcal{X} \circ (\mathcal{Y} \circ \mathcal{Z}). && (47) \end{aligned}$$

Special to Lie groups is that they define an action on members of other sets that transforms them in some way. For instance 3×3 rotation matrices transform vectors of length three by left matrix multiplication. Formally, given a Lie group $\mathcal{M}$ and a set $\mathcal{V}$, we denote the action of $\mathcal{X} \in \mathcal{M}$ on $v \in \mathcal{V}$ as $\mathcal{X} \cdot v$. The action must in addition satisfy the axioms

$$\begin{aligned} \text{Identity}: &\quad \mathcal{E} \cdot v = v && (48)\\ \text{Compatibility}: &\quad (\mathcal{X} \circ \mathcal{Y}) \cdot v = \mathcal{X} \cdot (\mathcal{Y} \cdot v). && (49)\\ & && (50) \end{aligned}$$

Another special property of Lie groups is the Lie algebra $\mathfrak{m}$, defined as the tangent space at the identity,

$$\mathfrak{m} \triangleq T_{\mathcal{E}}\mathcal{M}, \tag{51}$$

with the notation $T_{\mathcal{X}}$ denoting the tangent space around the element X. We have already seen an example of a Lie algebra in our discussion of rotation matrices, namely the family of skew symmetric matrices $\hat{\xi}$ corresponding to axis-angle vectors ξ. Members of the Lie algebra $\hat{\tau} \in \mathfrak{m}$ relate back to the manifold through a mapping known as the *exponential map* $\exp : \mathfrak{m} \to \mathcal{M}$. Conversely, the *logarithmic map* $\log : \mathcal{M} \to \mathfrak{m}$ maps members of the Lie group $\mathcal{M}$ into the Lie algebra $\mathfrak{m}$. The exponential map is typically defined as the Taylor series

$$\exp: \quad \exp(\tau^{\wedge}) \triangleq \mathcal{E} + \tau^{\wedge} + \frac{1}{2!}\tau^{\wedge 2} + \frac{1}{3!}\tau^{\wedge 3} + \ldots \quad . \tag{52}$$

For rotation matrices, this develops into the familiar Rodrigues' formula:

$$\begin{aligned} R(\xi) = \exp(\xi^{\wedge}) &= I + \xi^{\wedge} + \frac{1}{2!}\xi^{\wedge 2} + \frac{1}{3!}\xi^{\wedge 3} + \ldots \\ &= I + \sin\theta[\bar{\omega}]_{\times} + (1 - \cos\theta)[\bar{\omega}]_{\times}^{2}. \end{aligned} \tag{53}$$

The logarithmic map is found by inverting (52). For $SO(3)$ rotation matrices, this has the closed form expression

$$\xi^\wedge = \log(R) = \frac{\theta}{2\sin\theta}(R - R^\top), \tag{54}$$

where we find θ as

$$\theta = \arccos\left(\frac{\mathrm{tr}(R) - 1}{2}\right). \tag{55}$$

It is recommended by e.g. [10] that practical implementations use a Taylor approximation for $\theta/(2\sin\theta)$ for small values of θ. In addition, when θ is small, the approximation

$$R = \exp(\xi^\wedge) \approx I + \xi^\wedge \tag{56}$$

holds. GTSAM [7] makes use of both these approximations.

The elements $\tau^\wedge \in \mathfrak{m}$ of the Lie algebra can have non-trivial structures, like skew-symmetric matrices for $SO(3)$, but they can always be expressed as a linear combination of some generator elements E_i. This defines a linear Cartesian space $\mathbb{R}^m$ with $m = \dim \mathfrak{m}$ in which we can do linear algebra. We will denote members of this space as τ (without the $^\wedge$). Transforming between the Lie algebra and $\mathbb{R}^m$ is done with the *hat* and *vee* operators:

$$\text{Hat}: \quad \mathbb{R}^m \to \mathfrak{m}; \quad \tau \mapsto \tau^\wedge = \sum_{i=1}^{m} \tau_i E_i \tag{57}$$

$$\text{Vee}: \quad \mathfrak{m} \to \mathbb{R}^m; \quad \tau^\wedge \mapsto (\tau^\wedge)^\vee = \tau = \sum_{i=1}^{m} \tau_i e_i, \tag{58}$$

where e_i are the basis vectors of $\mathbb{R}^m$. For $SO(3)$, the axis-angle vectors $\xi \in \mathbb{R}^3$ form this linear Cartesian space. As mentioned previously, this formulation allows using linear algebra, such as for solving the linear least-squares problem from (27) using the normal equations.

Equipped with the general Lie theoretical framework, we can define a general $\oplus$ operator, just like we did for $SO(3)$ matrices, so that any Lie group variable types can be treated by the same general framework. We define this as

$$\oplus: \quad \mathcal{Y} = \mathcal{X} \oplus \boldsymbol{\tau} = \mathcal{X} \circ \exp(\boldsymbol{\tau}^\wedge), \tag{59}$$

where $\mathcal{Y} \in \mathcal{M}$, $\mathcal{X} \in \mathcal{M}$ and $\boldsymbol{\tau} \in T_{\mathcal{X}}\mathcal{M}$. Formally, this operator is known as the right-$\oplus$ operator and we can similarily define a left version of this, as well as $\ominus$ counterparts. We refer the reader to [33] for these definitions however, as texts such as [24] and [15] do just fine by defining only a single $\oplus$ operator.

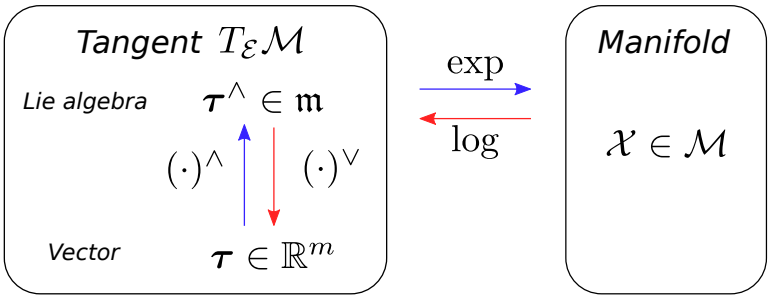

Figure 5: Overview of the spaces relevant for Lie groups with mappings between them. The exp-map maps from the manifold $\mathcal{M}$ to the Lie algebra $\mathfrak{m}$, whereas the log-map maps in the reverse direction. The hat $(\cdot)^\wedge$ operator maps from the $\mathbb{R}^m$ vector representation of the tangent space to the Lie algebra $\mathfrak{m}$ and the vee operator $(\cdot)^\vee$ back again. Figure adapted from [33] and licenced under CC BY-NC-SA 4.0.

2.4.3 Implementing new variable types in GTSAM

GTSAM is built on top of the Lie theoretic framework discussed previously for generality and extensibility. By implementing variables according to their Lie group structure, they can be treated by the same general optimization framework, eliminating the need for special handling of rotations and poses, or any other Lie group for that matter. To implement a new type of variable, say for example $Sim(3)$ transforms, all that is needed is to implement the operator $\circ$, the action $\cdot$ and their axioms along with the exp- and log-maps. Jacobians should also be defined: Even though GTSAM supports numeric differentiation, having the Jacobians explicitly defined will increase performance.

Variable types on non-Lie manifolds, such as the set S^2 of all unit vectors in $\mathbb{R}^3$, can also be implemented in GTSAM, provided a *retraction* can be defined [9]. For a manifold $\mathcal{M}$ and a point $a \in \mathcal{M}$, a retraction defines a mapping from a point ξ on the tangent space around a, $T_a\mathcal{M}$ back onto a corresponding point on $\mathcal{M}$, i.e.

$$\mathcal{R}_a : \quad \mathcal{M} \times \mathbb{R}^m \to \mathcal{M}; \quad a \oplus \xi \triangleq \mathcal{R}_a(\xi). \tag{60}$$

The retraction hence replaces the need for the exponential map in defining the $\oplus$ operator. A retraction needs to be smooth and map the origin of $T_a\mathcal{M}$ back onto a, but can otherwise be defined in several ways [9]. Even for proper Lie groups, defining a retraction can sometimes be computationally advantageous compared to the regular Lie exp-map. The $SE(3)$ manifold is one such example [9].

One of the reasons the Lie theoric framework is so popular for SLAM is that several commonly used value types in various SLAM problems are Lie groups!. 3D SLAM for instance require use of both Lie groups $SO(3)$ rotations and $SE(3)$ poses. 2D SLAM based on e.g. range-bearing sensors,

or simple 2D navigation using IMU and GPS similarily require the Lie groups $SO(2)$ and $SE(2)$. Monocular-only visual SLAM systems such as [28] that attempt to estimate the (unobservable) drift in scale often make use of the Lie group $Sim(3)$. [10] presents an overview of the expressions needed to implement all these Lie groups.

2.4.4 Implementing new factors in GTSAM

For many SLAM implementations, the built in variable types GTSAM offers will be enough, but the implementation requires defining a new factor instead. Defining such a factor requires significantly less work than a new variable type as the only method that must be implemented is the `evaluateError` method. This method computes the error term used in computing the sum in (34),

$$h_i(X_i) - \mathbf{z}_i. \tag{61}$$

The method must also calculate the Jacobian H_i at the current estimate X_i.

2.5 iSAM

The Incremental Smoothing and Mapping (iSAM) algorithm is an incremental version of the smoothing and mapping solution to SLAM, developed by Michael Kaess et al. [18]. While the algorithm was developed in 2008, it is still relevant mainly due to its successor iSAM2 [19]. Both algorithms are available in the GTSAM library [7] and the interface makes it easy to solve a SLAM problem with iSAM provided the factor graph is formulated in GTSAM.

The main idea behind iSAM's incremental updates is taking advantage of the sparsity of the SLAM problem to update only the parts of the *square root factor matrix* for the problem that are significantly changed by new measurements. To see how this square root factor matrix is formed, we begin by discussing iSAM's predecessor algorithm, square root SAM [8], in which this square root factor was initially proposed.

2.5.1 Square root SAM

Square root smoothing and mapping, or simply SAM, is an algorithm for solving SLAM with factor-graphs developed by Daellert and Kaess in 2006 [8]. The algorithm solves a SLAM problem by forming its corresponding linear least squares problem that we defined in (27),

$$\Delta^* = \arg\min_{\Delta} \sum_i \|H_i\Delta_i - (\mathbf{z}_i - h_i(X_i^0))\|^2_{\Sigma_i}, \tag{62}$$

converting it to a standard least-squares problem through variable substitution and solving it with Cholesky decomposition or QR factorization. After

converting the problem to a standard least-squares problem, we can solve it directly by solving the normal equations,

$$(A^\top A)\Delta^* = A^\top \mathbf{b}. \tag{63}$$

Typically, (63) is solved by means of *Cholesky factorization* of the *information matrix* $A^\top A = R^\top R$ into the upper-triangular *square root matrix* R. However, SAM also allows using *QR-factorization* which, while being slower than Cholesky, offers increased accuracy and numerical stability. We will focus on QR-factorization here, although the approach can be derived with Cholesky factorization as well. With QR-factorization, instead of computing the information matrix, we obtain R by factorizing A itself,

$$A = Q \begin{bmatrix} R \\ 0 \end{bmatrix}. \tag{64}$$

Here, Q is an $m \times m$ orthogonal matrix. By also defining vectors $\mathbf{d} \in \mathbb{R}^n$ and $\mathbf{e} \in \mathbb{R}^{m-n}$ as

$$\begin{bmatrix} \mathbf{d} \\ \mathbf{e} \end{bmatrix} = Q^\top \mathbf{b}, \tag{65}$$

and exploiting the orthogonality of Q, we can rewrite the least squares problem as

$$\|A\Delta - \mathbf{b}\|_2^2 = \|Q^\top A\Delta - Q^\top \mathbf{b}\|_2^2 = \|R\Delta - \mathbf{d}\|_2^2 + \|\mathbf{e}\|_2^2. \tag{66}$$

This means we can solve it by solving the triangular system

$$R\Delta^* = \mathbf{d} \tag{67}$$

by back-substitution.

In SAM, the QR-factorization is computed with n successive *Householder transformations* [16]. These transformations dominate the computational cost of the QR-based least squares scheme, leading to a total cost of $2(m - n/3)n^2$. The Cholesky scheme on the other hand, which includes calculating the information matrix $A^\top A$, requires only $(m+n/3)n^2$. In other words, the Cholesky scheme is around twice as fast. It is also noted in [9] that Cholesky factorization significantly outperforms QR-factorization for sparse matrices.

2.5.2 Incremental QR-updating with Givens rotations

The SLAM problem is fundamentally of an incremental nature. As new measurements arrive, they must be added to the factor graph and the corresponding matrices in the least squares problem augmented with new rows. In regular SAM, when receiving new measurements, the A matrix must be augmented with new rows and the square root factor R subsequently recomputed. The iSAM algorithm however, addresses the incremental nature of

the SLAM problem by performing incremental updates of R by means of QR-updating.

The QR-factorization is updated by means of Givens rotations. When receiving a measurement, the measurement Jacobian A must be augmented with a new row $\mathbf{w}^\top \in \mathbb{R}^n$ to form an updated matrix

$$A' = \begin{bmatrix} A \\ \mathbf{w}^\top \end{bmatrix}. \tag{68}$$

We must then update the existing QR-factorization $A = Q[R, 0]^\top$. However, instead of recomputing the entire R matrix again, we can append the new row directly to R, forming a Hessenberg matrix $[R, \mathbf{w}^\top]^\top$. We then apply a series of at most n Givens rotations $J_n, ..., J_1$ to move any new non-zero elements above the main diagonal, yielding an updated R matrix

$$R' = J_n ... J_1 \begin{bmatrix} R \\ \mathbf{w}^\top \end{bmatrix}. \tag{69}$$

Each of the J_i rotations move a single non-zero element either rightwards within the same row, or rightwards within the first row above it so that it ends up above the main diagonal. A graphical illustration of this can be found in figure 2 of [18]. The RHS vector d is similarly augmented with the right hand side value γ corresponding to the new measurement. The same Givens rotations are applied to this vector to form the updated RHS vector d' [18].

As [18] notes, the number of Givens rotations needed to add a measurement row is maximally n, but typically much fewer. This stems from the fact that the Givens rotations only need to be applied from the leftmost non-zero element in $\mathbf{w}^\top$ and then move rightwards until the right end of the matrix. Furthermore, in linear exploration tasks, new measurements only result in factors involving the few most recent values. The new measurement row $\mathbf{w}^\top$ will therefore be mostly zero except for a few values at the right end. The Givens rotations need only start at the leftmost of these non-zero values, greatly reducing the number needed.

2.5.3 Variable reordering and relinearization

As explained in the previous section, updating the square root factor matrix R with new measurements in a linear exploration task requires little computation as only a few Givens rotations must be applied. Real robot localization tasks however rarely contain only exploration, but also occasionally returning to previously visited locations. Such "loops" in the trajectory introduce correlations between old states and new measurements [18]. New measurement rows $\mathbf{w}^\top$ to be appended to A will therefore have non-zero values placed arbitrarily far to the left, requiring a number of Givens rotations proportional to n.

In addition to requiring more computation time, applying the extra Givens rotations makes R less sparse. The information matrix of the SLAM problem typically has a sparse structure, even in the case of loop closures, and this manifests itself in R as well. However, incrementally updating R with non-local measurements leads to non-zero elements outside this sparsity pattern [18]. In [18], this is referred to as "fill-in". Fill-in is disadvantageous since it makes all operations on R more computationally demanding.

To avoid fill-in, iSAM periodically reorders the states and recomputes the QR-factorization, leading to a R matrix that is sparse. To efficiently find a good ordering, they use the column approximate minimum degree (COLAMD) ordering heuristic from [6]. As loops are often not common, reordering must not be done with every timestep, but can instead be done only occasionally. In [18], reordering every 100th timestep has shown to yield good results.

Since all SLAM problems are usually of a non-linear nature, containing e.g. non-linear rotation components or bearing measurement functions, the problem needs to be linearized before it can be solved with the linear approaches discussed in this section. QR-updating the R matrix does not account for this linearization and will hence lead to accumulation of error. This calls for periodically relinearizing the system and recomputing R from scratch. Since the reordering step already recomputes R periodically, iSAM combines these two operations into one, forming a combined periodic reordering and relinearization step [18]. It is noted in [19] however that relinearizing only periodically is sub-optimal. In the following section, we will therefore look at how iSAM2 circumvents this by instead performing fluid relinearization at every timestep.

2.6 iSAM2

The iSAM2 algorithm [19] extends on iSAM by representing the square root factor matrix R as a novel datastructure called a *Bayes tree*. This tree structure allows both variable reordering and relinearization to be done as part of the incremental update and not in a separate batch step as was done in regular iSAM.

2.6.1 The Bayes tree

Using sparse QR-factorization to find the upper triangular square root factor matrix R used in iSAM is equivalent to converting the factor graph into a Bayes tree using the elimination algorithm [19]. The first step of this conversion procedure is to convert the factor-graph to a Bayes net. For this, the non-linear factor graph must first be linearized into a Gaussian factor graph. The graph is then gradually converted from a factor graph into a Bayes net by considering one variable at a time and eliminating the factors

adjacent to it. This process gets repeated for the next variable in the variable ordering until the entire factor-graph has been converted to a Bayes net. For the specifics of this procedure, see algorithm 2 of [19].

After constructing a Bayes net, iSAM2 goes on to convert it into a Bayes tree. This can be done because the Bayes net created by the elimination algorithm is *chordal*. A chordal graph is a graph where each cycle of more than 4 edges has a *chord*, that is, an edge between two nodes in the cycle that is not itself part of the cycle. Another way to see this is that every induced sub-cycle has exactly three edges. For that reason, the machine learning community sometimes refer to these graphs as *triangulated* [9]. The procedure for creating the Bayes tree involves finding its cliques. In iSAM2, this is done by means of the maximum cardinality search algorithm [35]. The resulting Bayes tree has the same structure as a *clique tree*, where each tree node corresponds to a clique in the Bayes net, but the edges are directed and preserves the elimination ordering [9]. The variables at the end of the ordering, which are typically the most recent states, are placed at the root of the tree, and the variables that preceded them during elimination, follow downwards into the branches in reverse elimination ordering.

Since constructing the Bayes tree is equivalent to QR-factorization, the least squares solution Δ^* can be obtained by back-substitution. For the Bayes tree, this amounts to one pass from the leaf nodes up to the root of the tree to construct the functions, and one pass down again to obtain the values [19].

The reason the Bayes tree is so well applicable to SLAM is due to its structure capturing the sparsity characteristics of the SLAM problem well. The tree is structured with the most recently added variables at the top, the most recent being the root. As an example, consider the Bayes tree shown in Fig. 3 (c) in [19]. This Bayes tree corresponds to the toy SLAM problem we have considered throughout this text, whose factor graph is shown in figure 4. The tree follows directly downwards with directed edges pointing to variables those above depend on. Junctions in the tree appear where a variable depends on two separate parts of the graph, such as in the toy example where landmark l_2 depends on the pose x_3, and l_1 and x_1 both depend on the pose x_2. Another reason for junctions in the Bayes tree common to SLAM occurs in case of a loop, where the trajectory makes a T junction. See e.g. Fig 5 in [19] for an example of this. For any Bayes tree, going down either one of the branches leads you to variables that are correlated with one another due to their local structure, however the variables in either branch will not be correlated with the variables of any other branch. This mirroring of the local structure of the SLAM problem is what makes the Bayes tree so useful.

It is important to notice that the order the variables are eliminated in, determines the structure of the resulting Bayes tree. Analogous to how variable ordering in regular iSAM determined whether or not the R matrix would

get fill-in [18], the elimination order in iSAM2 determines how many nodes of the Bayes tree must be changed during inference. For linear exploration, a good ordering is obtained by simply adding new states to the end of it. When the trajectory features loops however, this naive ordering would result in the equivalent of fill-in in the Bayes tree.

2.6.2 Incremental inference with the Bayes tree

During inference, we wish to add new factors and variables to the factor graph, connecting existing states to new. This requires eliminating the parts of the factor graph involved in the update with the elimination algorithm, and adding them to the Bayes tree. As in regular iSAM, where incremental inference could be done without refactoring the entire R matrix, inference with a Bayes tree also only requires modifying the variables involved in the factor and those above it in the tree. In other words, the remaining tree, extending downwards into other branches, remains unchanged.

One special situation occurs when factors are placed between variables that form part of two different branches, such as in case of a loop closure. In that case, both branches must be re-eliminated to form a new structure that accounts for their dependence.

2.6.3 Incremental reordering

As mentioned previously, the elimination ordering has a big impact on the performance of iSAM2. As the robot explores its environment, the optimal ordering will change, prompting the need for reordering. In iSAM2, variables can be reordered incrementally, as opposed to the periodic batch-reordering of regular iSAM. This is achieved by reordering during re-elimination in the incremental inference steps. The variables that are re-eliminated are at the same time reordered to better reflect new dependencies between them. By reordering at every increment instead of in batches, fill-in can be reduced without incurring the significant overhead of a complete reordering.

In [19], they show that reordering the variables with COLAMD before re-eliminating them yields faster inference for later steps. The reordering is especially important in case of loop closures, which can otherwise lead to significant fill-in [18], [19].

However, naively using COLAMD for variable reordering has another downside. During linear exploration, new variables are usually correlated with other recently added variables that form part of the local surroundings of the robot. This suggests that the most recent variables should be placed at the end of the ordering. The COLAMD heuristic however, does not does not account for this. In order to force new variables to be added to the end of the ordering, iSAM2 uses a constrained version of the heuristic, that forces the most recently accessed variables to the end of the ordering. Their

experiments show that this leads to slightly more fill-in, but with the benefit of less variables being affected in subsequent inference steps [19].

2.6.4 Fluid relinearization

Recall from section 2.3 that in order to optimize the non-linear least squares objective (34), it had to be linearized at the current estimate X^t. We could then solve a linear least squares problem for an update vector Δ and add it to the estimate, obtaining a new linearization point $X^{t+1} = X^t \oplus \Delta$. However, as in the regular iSAM algorithm, where we keep incrementally predicting the state without relinearizing the objective, Δ can grow quite large and consequently bring the updated state far from the linearization point. In regular iSAM, this leads to large linearization errors between the periodic relinearization steps. One optimization in iSAM2 however, stems from the observation that only some parts of Δ tend to grow between iterations, namely those parts affected by new measurements [19]. For factors ϕ_i where Δ_i is still small, the previous linearization point of the corresponding variables X_i^t is still valid. This calls for only relinearizing variables with large Δ_i, while omitting those with small Δ_i.

Using the Bayes tree, iSAM2 can treat differently those variables that need re-linearization and those that do not. It does so by thresholding Δ_i by a value $\beta > 0$. Those variables with $|\Delta_i| \geq \beta$ are replaced with re-linearized counterparts, along with all variables above them up to the root. Variables further down in the branches however, are kept as they were.

2.6.5 Partial variable updates

As mentioned, we can compute the update vector Δ from the Bayes tree by propagating upwards to the root to obtain the functions, then downwards again to compute the new values in Δ. This Δ vector is then compounded with the current estimate X^t, forming the new estimate and concluding one step of the iSAM2 algorithm. However, it is important to notice that large parts of Δ go unchanged between iterations. In particular, due to the structure of the SLAM problem, new measurements are likely to only affect local states, leaving far-away states mostly unchanged. This structure is reflected in the Bayes tree, and iSAM2 takes advantage of it to perform partial state updates and save computational costs.

In order to only update those values that significantly change, we update the values at the top of the tree affected by the new factors, then propagate downwards updating only those that contribute a large change to Δ. The change in Δ is thresholded by a small value α (values of 0.005 and 0.05 have shown good results). When a node is reached where the change in Δ is less than the α threshold, the propagation stops and the previous values for the remaining nodes in that branch are used instead.

3 Back-end for Resilient Multi-Modal SLAM

We will now see how we can fuse constraints from multiple modalities in a single factor graph, to produce a SLAM system more resilient to sensor degradation. In particular, we present a preliminary loosely coupled factor-graph odometry fusion system capable of fusing visual and LiDAR odometries. The system can handle sensor-degradation by preventing constraints of one modality from being added to the factor graph if it becomes unhealthy.

3.1 Loosely coupled odometry fusion

The system we present in this section is a loosely coupled sensor-fusion system. In the context of visual or LiDAR odometry, this means that the quantities to be fused in the joint optimization is the odometry measurements from the individual single-modality methods. These odometry estimates are in themselves valid solutions to the localization problem, but a more robust solution can be obtained by fusing them, especially when the sensor modalities offer complementary properties such as is the case for vision and LiDAR. This loosely coupled scheme stands in contrast to tightly coupled schemes, where the sensor measurements from both modalities are directly included in the common optimization problem.

While we note that the approach is in principle modality-agnostic, we will for clarity present the system formulated with the visual-inertial-odometry method ROVIO [1] and the LiDAR-odometry method LOAM [42] as odometry front ends. LOAM receives point clouds from a LiDAR device, whereas ROVIO receives images from one or more cameras and inertial measurements from an inertial measurement unit (IMU). Provided these inputs, they both produce odometry estimates in the global frame that we subsequently incorporate in a GTSAM-implemented factor-graph on which we can readily perform pose-graph optimization to obtain a fused odometry. To provide the fused odometries incrementally as new measurements become available, we use the iSAM2 [19] implementation found in the GTSAM library. A block diagram overview of the described system that shows this information flow is provided in figure 6.

Adding odometry constraints to the factor graph from multiple front-ends requires some special care. Firstly, the odometry measurements received are $SE(3)$ poses in the global frame, representing the newest estimated pose of the robot. Since neither ROVIO nor LOAM has any mechanism for globally anchoring their trajectories with e.g. GNSS, these $SE(3)$ odometry estimates will inevitably drift, and do so in different directions. Consequently, fusing the odometries directly will yield increasingly wrong results. A better approach is to compute and add to the factor graph the relative transformations between each robot state. The GTSAM library provides a handy *between-factor* for this purpose that adds an $SE(3)$ transformation constraint

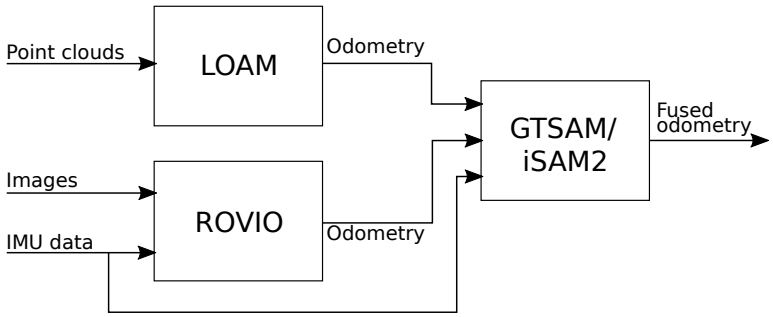

Figure 6: A system overview block diagram showing the information flow between the two single-modality front-ends LOAM and ROVIO and the GTSAM/iSAM2 factor-graph based back-end. Raw sensor data in the form of point clouds and images, as well as IMU data, is fed into the front-ends and they produce odometry estimates which are subsequently processed by the back-end to produce an optimized trajectory of odometries. The IMU data is also utilized directly in the back-end.

between two pose variables [7].

The second complicating factor is the different rates at which the two odometry streams arrive. LiDAR odometry is typically significantly slower than visual due to the lower rates of point cloud gathering and the computational cost of processing such large point clouds. The different rates also imply that we must treat the measurements as asynchronous. We must hence ensure that between-factors are placed between pose variables of correct type, i.e. variables created from the same type of measurement modalities. This means we can think of the pose variables $\mathbf{x}_{1:k}$ as separated into two interleaved chains connected with between factors. In order to connect the two chains, each consecutive pose pair $(\mathbf{x}_t, \mathbf{x}_{t+1})$ needs a factor between them from either a Markov motion model or, in our case, from IMU measurements. Figure 7 shows the resulting factor graph.

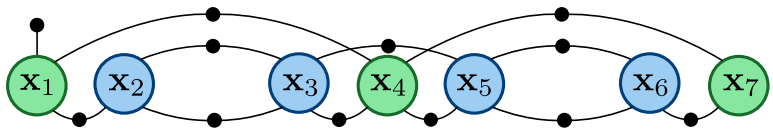

Figure 7: Factor graph for loosely coupled odometry fusion with two front-ends. Colored circles indicate variables whereas small black nodes indicate factors. Pose variables are connected with between-factors to form two chains corresponding to the two modalities. The two modalities are indicated with blue and green colors. Consecutive poses are also connected with IMU factors (below), although they are not shown in their full form. The first variable $\mathbf{x}_1$ is initialized with a prior factor.

3.2 Integrating IMU data

Our system incorporates inertial data in the factor-graph optimization by adding IMU factors between consecutive poses. An IMU measures linear and rotational acceleration so in order to use it for position estimation, it must hence be doubly integrated. The acceleration measurements of IMUs typically suffer from a slowly varying bias which, when integrated twice, can lead to large errors in the position estimate [39]. For this reason, most visual-inertial odometry systems estimate the bias as part of the optimization problem [1] [3] [25].

IMUs typically operate at much higher frequencies than either visual or LiDAR based odometry systems can keep up with. Hence, including the IMU measurements directly in the factor graph is not ideal due to the sheer number of factors needed. Modern inertially-aided SLAM methods such as [3] and [25] therefore preintegrate the IMU measurements to form a single IMU-factor between robot poses. GTSAM includes a state-of-the-art preintegration scheme based on [26] and extended in [13] that performs preintegration on the nonlinear manifold. In GTSAM 4.0 however, which is used for our system, preintegration is instead done in the tangent space because this increases efficiency [7].

GTSAM provides a *combined IMU factor* together with the preintegration scheme, which adds a single combined factor between pose $\mathbf{x}$, velocity $\mathbf{v}$ and bias $\mathbf{b}$ terms. With this, we can jointly estimate the velocity and bias terms along with the robot poses. As new IMU measurements come in, we store them in a buffer such that when receiving an odometry, we can integrate measurements from the buffer between the newly arrived odometry and the previous. The resulting factor graph then has a velocity and bias variable for every pose, as can be seen in figure 8.

One common situation with asynchronous odometry measurements occurs when they arrive so close in time that no IMU measurements have ar-

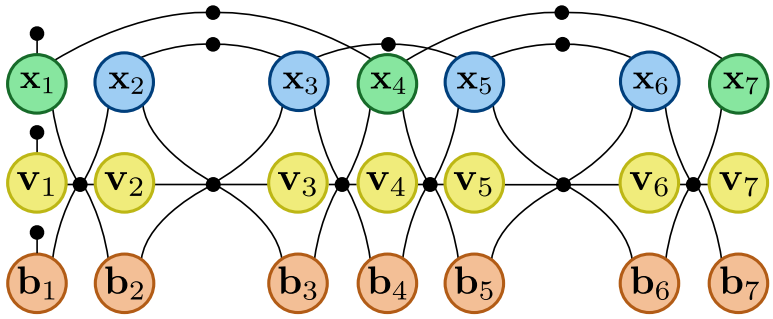

Figure 8: The same factor graph as in figure 7, this time showing the velocity and bias variables involved in each combined IMU factor. As with the first pose $\mathbf{x}_1$, the first velocity and bias terms $\mathbf{v}_1$ and $\mathbf{b}_1$ are initialized with prior factors.

rived in between. With no IMU measurements, we can not add a combined IMU factor. Hence, in that case, we do not add a new pose, velocity and bias value for the latest measurement, but instead add the between factor on the latest existing pose. This means we effectively disregard the negligible time difference between the two measurements and instead treat them as having arrived exactly simultaneously. Figure 9 shows the factor graph resulting from this.

3.3 Detecting and handling sensor failure

If one modality becomes degraded, the odometry estimates based on it will inevitably be incorrect. Including them without special handling in the sensor fusion would therefore deteriorate the fused result. Our approach avoids this by excluding unhealthy measurements from the factor graph. There are many health metrics that can be used for detecting this, e.g. the *D-optimality* criterion described in [5] or the degeneracy factor as described in [40]. In our preliminary implementation however, we rely on this health metric being provided by the odometry front-ends. That is, we assume the front-ends provide a second stream of boolean health messages corresponding to each odometry measurement.

To account for the healthy criterion of the received odometries, we check the newest health message before adding anything to the graph. If the odometry is considered unhealthy, the measurement is skipped and nothing is added to the graph. To additionally ensure that the odometry front-end does not wrongly consider itself healthy when it is not, we buffer the last n_h health messages and declare the odometry *degenerate* if the number of unhealthy messages in the buffer exceeds a certain threshold n_u. When a modality is declared degenerate, all its incoming odometry messages will be considered unhealthy, even if they are reported as healthy. This degeneracy state persists until the buffer only contains healthy messages.

After a period of degeneracy for a certain modality, its chain in the factor graph must be reinitialized. Indeed, because the odometry measurements received are the estimated global position of the robot, adding a between-factor between the last pose before degeneracy and the newest pose after degeneracy, will be incorrect. As an example of this, consider a situation

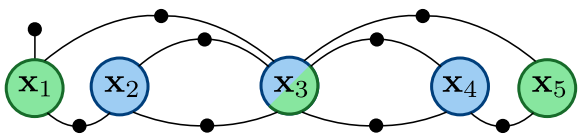

Figure 9: A factor graph illustrating the situation where no IMU measurements have arrived between subsequent odometry measurements. In this case, no new pose variable is added and between factors are instead added on the existing pose.

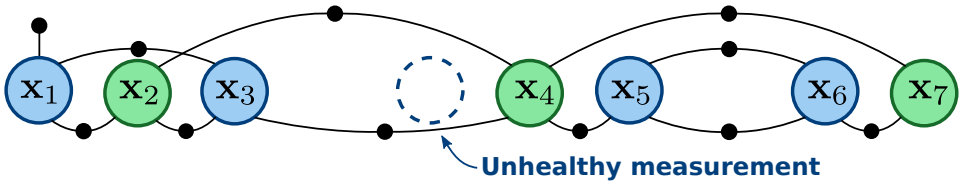

Figure 10: A factor graph illustrating a degenerate odometry measurement. IMU velocity and bias terms are omitted for clarity. The chain of the blue modality is broken due to the missing variable. No between-factor is thus placed between the poses $\mathbf{x}_3$ and $\mathbf{x}_5$. Instead, $\mathbf{x}_5$ is reinitialized with a IMU-factor between $\mathbf{x}_4$ and $\mathbf{x}_5$.

where LOAM has been degenerate for a longer period, while ROVIO has been healthy. When LOAM becomes healthy again, we must wait for two subsequent LOAM odometry measurements to arrive so we can calculate a delta for the between-factor. Thus, no between-factor is added between the LOAM poses just before and just after degeneracy. However, in order to connect the new chain of LOAM odometries to the rest of the graph, we add an IMU factor connecting the first healthy LOAM message to the newest ROVIO message. This effectively reinitializes the LOAM chain. A factor graph showing this situation is shown in figure 10. Note that the same behavior applies for handling a single unhealthy measurement.

4 Experimental Evaluation

4.1 Dataset and experimental setup

We test our system on a sequence of visual, LiDAR and IMU data recorded in a highway underpass tunnel, where severe geometric homogeneity causes the LiDAR odometry to degenerate. As illustrated in figure 11, the LiDAR produces accurate point clouds representing the structure of the tunnel, but its tubular shape causes consecutive point clouds to have multiple valid alignment solutions along the direction of motion. This consequently makes the odometry degenerate along the tunnel, producing an incorrect trajectory and map. However, while the tunnel walls are geometrically self-similar, they have enough texture for a visual-odometry method like ROVIO to work sufficiently well. The tunnel sequence is therefore a good real-life illustrative example for the efficacy of the proposed multi-modal method .

The sequence was recorded using a lightweight sensing platform mounted on a research quadcopter. A Velodyne PuckLITE LiDAR running at 10Hz was used for point cloud measurements, a FLIR BlackFly camera running at 20Hz for images and a VectorNav VN-100 IMU running at 200Hz for inertial measurements. The recording was done in handheld configuration to avoid dust being whipped up by the propellers and obscuring the camera. There is no ground truth available for this sequence, but we can still evaluate the results qualitatively and compare with satellite photos of the tunnel from Google Earth. For further details on the dataset, see [20].

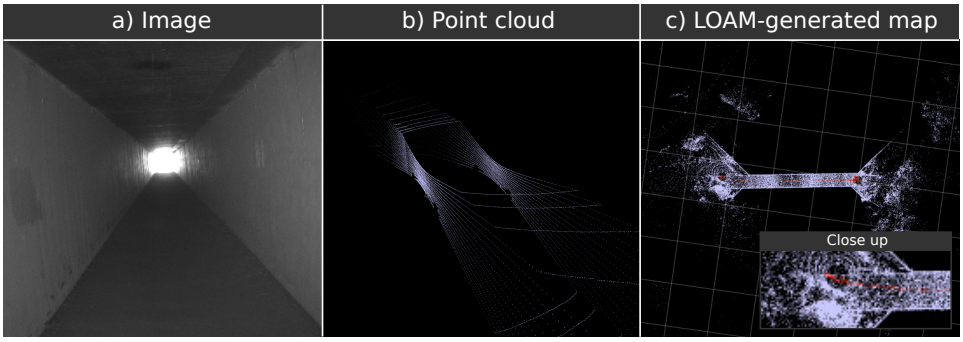

Figure 11: An example image (**a**) and registered point cloud (**b**) from one time step in the tunnel sequence showing the challenging conditions LOAM faces due to the geometrically homogeneous environment. The point cloud has no distinct geometric features along the direction of the tunnel that could constrain the alignment between consecutive frames to a single unique solution. This leads to degeneracy in the estimated odometry and the resulting map generated by LOAM (**c**) is hence much shorter than in reality and features geometric inconsistencies at the edges of the tunnel.

4.2 Odometry front-ends

For these preliminary experiments, we used slightly modified versions of ROVIO [1] and LOAM [42] as single-modality odometry front-ends. The LOAM implementation is a research-implementation that uses a similar degeneracy detection mechanism to that described in [40] to detect when it enters a degenerate state. It provides this along with its odometry estimates so that we can integrate them as health indicators like described in section 3.3. The ROVIO version used has been slightly modified to correct for a time-alignment error between the IMU and camera streams. See table 2 in appendix A for links to the individual software repositories where available.

The LOAM implementation we use has been extended in [20] for degeneracy awareness. To see why explicit degeneracy awareness is needed to handle LiDAR-odometry degeneracy, we must understand how LOAM's pointcloud alignment procedure works. LOAM uses a Levenberg-Marquardt-based optimization scheme for finding the best alignment between consecutive point clouds [42]. LM, just like Gauss-Newton, approximates the Hessian with the squared Jacobian matrix $A^\top A$. This matrix becomes rank-deficient in case of degeneracy, which would make the regular GN system unsolvable. LM, however ensures solvability, but will not step in unobservable directions [25], hence causing LOAM to predict no forward motion. However, predicting no motion is still only one of an infinite number of solutions to the ill-conditioned point cloud alignment problem, meaning this solution should not be trusted. LOAM was therefore extended in [20] to heuristically detect this degeneracy and pass through a camera-based odometry estimate instead of the LiDAR-odometry. This scheme detects degeneracy by checking the smallest eigenvalue λ_{min} of the $A^\top A$ matrix: If λ_{min} falls below a certain threshold, it indicates $A^\top A$ is close to singular and the odometry is hence treated as degenerate. In our system, we use the implementation from [20], but do not configure it for camera-odometry pass-through since we implement multi-modality in our GTSAM back-end instead. We use the degeneracy messages it publishes to decide if LiDAR factors should be added to the factor graph.

The complete system under evaluation is shown in figure 12 as a block-diagram, complete with the data flow of the health messages and all sensors used for the data collection. It is worth noting that we use the same IMU data both for ROVIO's visual-inertial odometry, and in the factor graph. This kind of double inclusion of sensor data introduces correlations into the estimation problem not reflected in the factor graph, which can produce incorrect results. For the purposes of illustration however, this effect can be neglected.

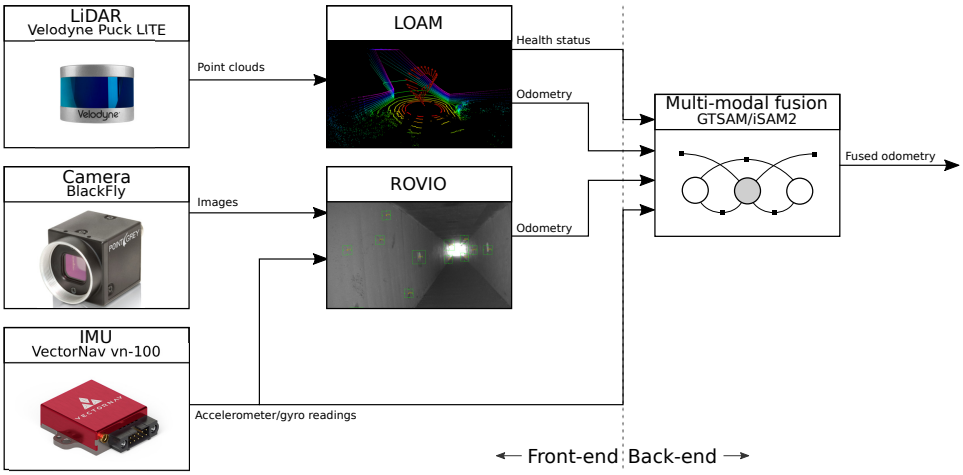

Figure 12: Full block diagram showing the sensors used for recording the tunnel sequence, along with odometry front-ends and the GTMSAM/iSAM2 back-end.

4.3 Single-modality performance on the sequence

To illustrate the need for multi-modality on this sequence, we show in figure 13 the trajectories generated from each single modality alone. The LOAM odometry shows no forward movement between $t = 40$s and $t = 80$s. This is reflected in the smallest eigenvalue of the $A^\top A$ matrix dropping below the degeneracy threshold as shown in figure 14. The figure also exemplifies how the homogeneous geometry of the tunnel provokes this degeneracy, whereas the more structure-rich geometry outside the tunnel does not. In contrast to LOAM, the ROVIO odometry does well throughout the sequence aside from drifting slightly.

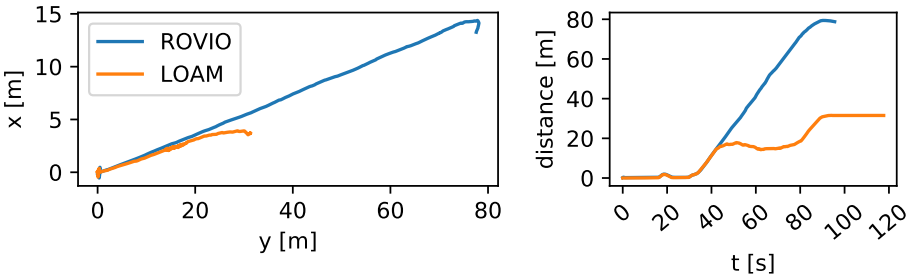

Figure 13: ROVIO vs LOAM generated trajectories on the tunnel sequence (left) and the how the corresponding distance from starting position develops over time (right). ROVIO tracks consistently through the entire tunnel sequence whereas LOAM has a period of degeneracy between $t = 40$s and $t = 80$s where it registers no forward movement.

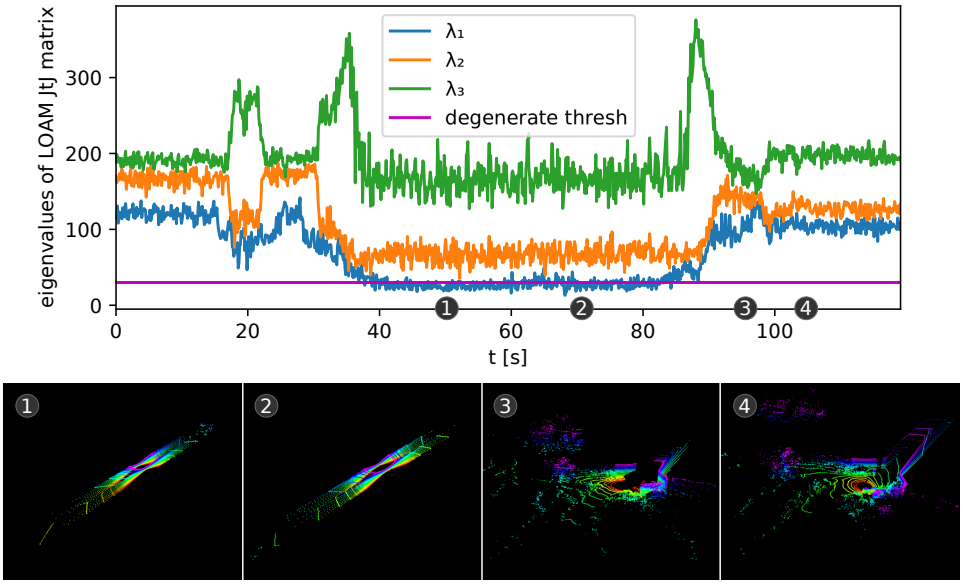

Figure 14: Development of the first three eigenvalues of the $A^\top A$ matrix in LOAM's LM procedure showing a period of degeneracy in the 40–80s interval (top) and selected point clouds extracted at indicated timestamps, showcasing examples of degeneracy and non-degeneracy (bottom). The pointclouds during degeneracy (1-2) have multiple valid alignment solutions due to the tubular environment, whereas the pointclouds during non-degenerate operation (3-4) are geometrically richer and hence constrain the alignment problem better. This fact is reflected in the eigenvalues of $A^\top A$ and if the smallest of these falls below a certain threshold (30 in this case), the odometry is considered degenerate. The process is detailed further in [20] and [40].

4.4 Multi-modal results

As indicated in figure 15, the LOAM/ROVIO fusion produces a more realistic length estimate of the tunnel and effectively eliminates the inconsistent geometry induced by the degenerate LiDAR odometry. This suggests that a loose factor-graph based coupling of estimated odometries can add robustness to situations of similar degenerate geometry provided the front-ends have a means of detecting degeneracy.

4.5 Discussion

As is clear from figure 13, the ROVIO odometry is able to perform adequately on the tunnel sequence, whereas LOAM is not. The point of illustrating this is not to show that either modality is better than the other, but to demonstrate that including multiple complementary modalities in a SLAM system increases the overall robustness of the system. Our presented approach does however require a way to detect degeneracy of each individual modality, so

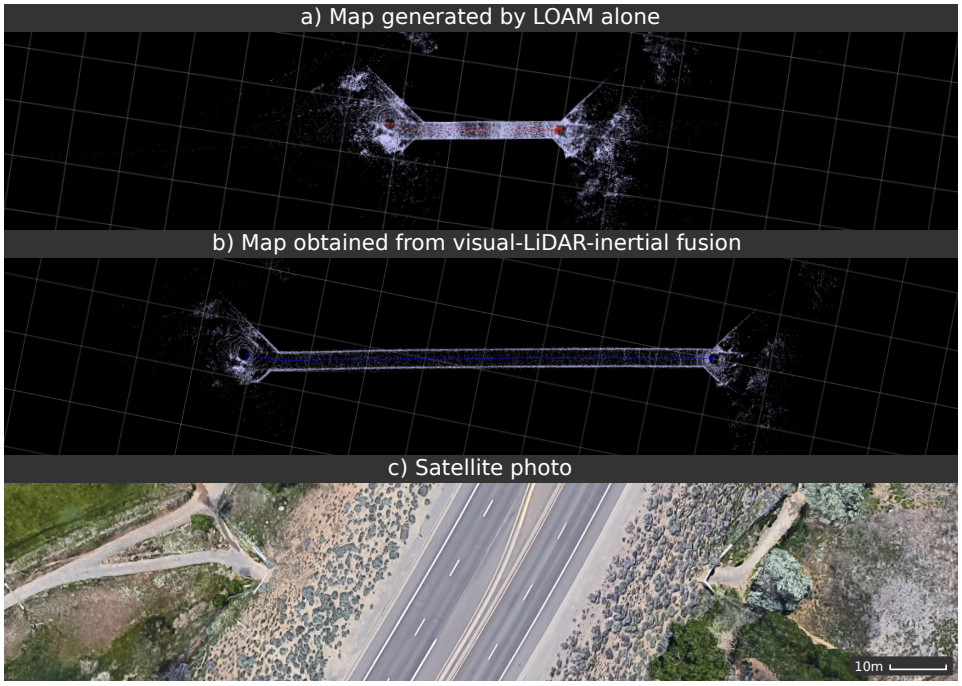

Figure 15: To-scale comparison of maps generated from the tunnel sequence compared to satellite images of the actual tunnel. The map generated by fusing LOAM and ROVIO (b) does not have the geometrical inconsistencies as in the LOAM-only map (a). We can also qualitatively verify that the scale is correctly estimated by comparing with the same-scale satellite image of the tunnel (c) collected from Google Earth. Grid square sizes in (a) and (b) are 10×10 meters.

that degenerate measurements can be excluded from the optimization.

The accurate results in figure 15 demonstrates that the factor graph framework enables gaining robustness even with a simple implementation. No major modifications to the LOAM or ROVIO implementations were necessary, and the factor graph facilitating the loose coupling is also relatively simple. This approach hence makes the front-ends interchangeable and different off-the-shelf SLAM systems can be mixed and matched to obtain best results. Moreover, the modality-agnostic nature of this approach opens up for including other types of sensors as well, such as RGB-D cameras or thermal cameras which also offer complementary properties to traditional sensors.

5 Conclusions and Discussion

5.1 Conclusions

In this reporrt we surveyed some of the currently important frameworks for SLAM, and presented a preliminary implementation based on these for multi-modal operation. We have presented a loosely coupled fusion approach based on factor-graphs that can combine odometry constraints from several front-ends into a single joint pose-graph optimization problem. The system can handle degeneracy in one of the modalities by excluding constraints made by the degenerate modality from the factor graph. We have tested our implementation on a dataset provoking a specific failure case of LiDAR odometry and the results suggests that this approach can handle such degenerate scenarios and hence increase overall robustness.

5.2 Loosely vs tightly coupled sensor fusion

While our method of loosely coupling two odometry streams allows handling degeneracy in either sensor modality by falling back on the other, our solution is far from optimal. Our approach of excluding degenerate constraints from the factor graph has the adverse effect of throwing away useful information produced by the degenerate modality. For instance, in the tunnel scenario we analysed, the movement in the x-direction (forward through the tunnel) is not observable from the LiDAR measurements alone, but movement in the y- and z-directions in relation to the tunnel walls is. The LiDAR measurements hence constrain the solution space well in the y-z-plane, but not along the x-axis. If a second modality such as visual camera data could constrain the problem in the x-direction, the joint problem would be well-conditioned.

The above discussion motivates using a tight coupling of the sensor streams rather than our loosely coupled odometry fusion approach. In a tight fusion, the map features used by the front-ends would be directly included in the factor graph and all correlations between them would therefore be considered [25]. That way, estimates of visual landmark positions could influence the alignment of LiDAR point clouds, and vice versa. If done correctly, problem configurations such as the tunnel sequence, where a single modality degenerates, would be implicitly handled by the factor-graph framework, as the non-degenerate modality would provide constraints where the degenerate modality does not. That is not to say however that the tightly coupled factor-graph formulation makes trivial the detection and handling of degeneracy. One particularly challenging aspect of degeneracy handling in factor-graphs stems from how linearization is performed. Degenerate variable configurations, together with bad variable initialization can make the linearization invalid. As an example, consider the initialization of a landmark in a monocular vision system wherein the robot moves directly towards the landmark. This problem is ill-conditioned since the depth of the landmark is

unobservable. A bad initial guess for the depth could cause the linearization point to be chosen far off the actual value, and subsequently throw off the optimization. To prevent this, GTSAM implements a special type of factor called *smart-factors*, described in [4]. These factors allow incorporating domain-specific knowledge for detecting and dealing with different types of degeneracy. Using these factors means degeneracy will be handled within the factors themselves, rather than on the factor-graph level, something [4] argues makes the handling much simpler.

5.3 Boolean degeneracy vs partial degradation

The degeneracy handling we described in section 3 treats degeneracy as a boolean state. That is, the modality in question is either degenerate or healthy. The method defines an arbitrary threshold for the smallest eigenvalue involved in the optimization problem and flips the boolean degenerate state only when that threshold is passed. This scheme, although simple to implement, therefore disregards any notion of partial degradation, or a close-to-degenerate system. Yet, figure 14 clearly shows the eigenvalues develop in a smooth manner as the system transitions into degeneracy. This suggests that a notion of partial degradation and degeneration could be formulated and used as a weighting heuristic for how much a particular modality should be trusted, similar to what is done in [40].

5.4 Future work

We believe a tight coupling of the sensor modalities will offer superior performance to the loosely coupled approach presented here. In doing so, it would be beneficial to investigate the effectiveness of the smart projection factors included in GTSAM for handling visual landmarks. Additionally, it could be useful to formalize novel smart-factors for representing LiDAR measurements that can detect and handle different forms of degeneracy.

Another approach we are currently looking into for handling degeneracy in a tightly coupled system is to rigorously or heuristically estimate the covariance matrices underlying the problem. Having a notion of the covariances involved would make the fusion of the different modalities more meaningful. If the covariances could also account for partial degeneracy, say by increasing their magnitude in unobservable directions, they would in effect be able to provide a smooth transition into degeneracy instead of the boolean state we use. One of the key challenges with this approach is making the covariances of different modalities correspond metrically, so they can be compared without bias.

Finally, while this project has focused on LiDAR and visual modalities only, for our future work we want to investigate including other modalities as well. Thermal vision is one such modality that has shown promising results

in subterranean environments [21]. Its ability to penetrate both obscurants and darkness makes it directly complementary to vision and LiDAR. It would therefore be of great interest to explore the merits of including this in the joint optimization.

6 Abbreviations

Table 1: Abbreviations

Abbreviation	Meaning
COLAMD	Column Approximate Minimum Degree
EKF	Extended Kalman Filter
GNSS	Global Navigation Satellite System
GTSAM	Georgia Tech Smoothing and Mapping
IMU	Inertial Measurement Unit
iSAM	Incremental Smoothing and Mapping
KF	Kalman Filter
LiDAR	Light Detection and Ranging
MAP	Maximum a posteriori
RHS	Right Hand Side
SAM	Smoothing and Mapping
SE(n)	Special Euclidean Group n
SLAM	Simultaneous Localization and Mapping
SO(n)	Special Orthogonal Group n
VIO	Visual-Inertial Odometry
VO	Visual Odometry

A Software Repositories

Table 2: Links to software repositories used for experiments

Repository	Link	Last commit at time of writing
Odometry fusion	github.com/sigtot/odometry-fusion	`ae5c1658238b5a79fcde0e99c886145cc83b3b67`
ROVIO	github.com/sigtot/rovio (Fork of github.com/ethz-asl/rovio)	`3792ebb81434d9b3ad87d6ff4cba1fb8b1405033`
LOAM	Private repository	`N/A`

References

[1] M. Bloesch et al. “Robust visual inertial odometry using a direct EKF-based approach.” In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2015, pp. 298–304.

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

[3] Carlos Campos et al. “ORB-SLAM3: An accurate open-source library for visual, visual-inertial and multi-map SLAM.” In: *arXiv preprint arXiv:2007.11898* (2020).

[4] L. Carlone et al. “Eliminating conditionally independent sets in factor graphs: A unifying perspective based on smart factors.” In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. 2014, pp. 4290–4297.

[5] H. Carrillo, I. Reid, and J. A. Castellanos. “On the comparison of uncertainty criteria for active SLAM.” In: *2012 IEEE International Conference on Robotics and Automation*. 2012, pp. 2080–2087.

[6] Tim Davis et al. “A column approximate minimum degree ordering algorithm.” In: *ACM Trans. Math. Softw.* 30 (Sept. 2004), pp. 353–376.

[7] Frank Dellaert. *Factor graphs and GTSAM: A hands-on introduction*. Tech. rep. Georgia Institute of Technology, 2012.

[8] Frank Dellaert and Michael Kaess. “Square Root SAM: Simultaneous localization and mapping via square root information smoothing.” In: *The International Journal of Robotics Research* 25.12 (2006), pp. 1181–1203.

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

[10] Ethan Eade. “Lie groups for 2d and 3d transformations.” In: ().

[11] Felix Endres et al. “An evaluation of the RGB-D SLAM system.” In: *2012 IEEE International Conference on Robotics and Automation*. IEEE. 2012, pp. 1691–1696.

[12] Jakob Engel, Vladlen Koltun, and Daniel Cremers. “Direct sparse odometry.” In: *IEEE transactions on pattern analysis and machine intelligence* 40.3 (2017), pp. 611–625.

[13] C. Forster et al. “On-Manifold Preintegration for Real-Time Visual–Inertial Odometry.” In: *IEEE Transactions on Robotics* 33.1 (2017), pp. 1–21.

[14] Andreas Geiger, Philip Lenz, and Raquel Urtasun. "Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite." In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2012.

[15] Giorgio Grisetti et al. "A tutorial on graph-based SLAM." In: *IEEE Intelligent Transportation Systems Magazine* 2.4 (2010), pp. 31–43.

[16] Alston S Householder. "Unitary triangularization of a nonsymmetric matrix." In: *Journal of the ACM (JACM)* 5.4 (1958), pp. 339–342.

[17] C. Jauffret. "Observability and fisher information matrix in nonlinear regression." In: *IEEE Transactions on Aerospace and Electronic Systems* 43.2 (2007), pp. 756–759.

[18] M. Kaess, A. Ranganathan, and F. Dellaert. "iSAM: Incremental Smoothing and Mapping." In: *IEEE Transactions on Robotics* 24.6 (2008), pp. 1365–1378.

[19] Michael Kaess et al. "iSAM2: Incremental smoothing and mapping using the Bayes tree." In: *The International Journal of Robotics Research* 31.2 (2012), pp. 216–235. eprint: https://doi.org/10.1177/0278364911430419. URL: https://doi.org/10.1177/0278364911430419.

[20] S. Khattak et al. "Complementary Multi–Modal Sensor Fusion for Resilient Robot Pose Estimation in Subterranean Environments." In: *2020 International Conference on Unmanned Aircraft Systems (ICUAS)*. 2020, pp. 1024–1029.

[21] Shehryar Khattak, Christos Papachristos, and Kostas Alexis. "Keyframe-based thermal–inertial odometry." In: *Journal of Field Robotics* 37.4 (2020), pp. 552–579.

[22] Hanme Kim, Stefan Leutenegger, and Andrew J Davison. "Real-time 3D reconstruction and 6-DoF tracking with an event camera." In: *European Conference on Computer Vision*. Springer. 2016, pp. 349–364.

[23] S. Kohlbrecher et al. "A flexible and scalable SLAM system with full 3D motion estimation." In: *2011 IEEE International Symposium on Safety, Security, and Rescue Robotics*. 2011, pp. 155–160.

[24] R. Kümmerle et al. "G2o: A general framework for graph optimization." In: *2011 IEEE International Conference on Robotics and Automation*. 2011, pp. 3607–3613.

[25] Stefan Leutenegger et al. "Keyframe-based visual–inertial odometry using nonlinear optimization." In: *The International Journal of Robotics Research* 34.3 (2015), pp. 314–334.

[26] Todd Lupton and Salah Sukkarieh. "Visual-inertial-aided navigation for high-dynamic motion in built environments without initial conditions." In: *IEEE Transactions on Robotics* 28.1 (2011), pp. 61–76.

[27] Anastasios I Mourikis and Stergios I Roumeliotis. "A multi-state constraint Kalman filter for vision-aided inertial navigation." In: *Proceedings 2007 IEEE International Conference on Robotics and Automation*. IEEE. 2007, pp. 3565–3572.

[28] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. "ORB-SLAM: a versatile and accurate monocular SLAM system." In: *IEEE transactions on robotics* 31.5 (2015), pp. 1147–1163.

[29] Raúl Mur-Artal and Juan D. Tardós. "ORB-SLAM2: an Open-Source SLAM System for Monocular, Stereo and RGB-D Cameras." In: *IEEE Transactions on Robotics* 33.5 (2017), pp. 1255–1262.

[30] R. A. Newcombe et al. "KinectFusion: Real-time dense surface mapping and tracking." In: *2011 10th IEEE International Symposium on Mixed and Augmented Reality*. 2011, pp. 127–136.

[31] H. Rebecq et al. "EVO: A Geometric Approach to Event-Based 6-DOF Parallel Tracking and Mapping in Real Time." In: *IEEE Robotics and Automation Letters* 2.2 (2017), pp. 593–600.

[32] Tixiao Shan et al. "LIO-SAM: Tightly-coupled Lidar Inertial Odometry via Smoothing and Mapping." In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2020, pp. 5135–5142.

[33] Joan Solà, Jeremie Deray, and Dinesh Atchuthan. *A micro Lie theory for state estimation in robotics*. Tech. rep. IRI-TR-18-01. Barcelona: Institut de Robòtica i Informàtica Industrial, 2018.

[34] Hauke Strasdat, J.M.M. Montiel, and Andrew J. Davison. "Visual SLAM: Why filter?" In: *Image and Vision Computing* 30.2 (2012), pp. 65–77. URL: http://www.sciencedirect.com/science/article/pii/S0262885612000248.

[35] Robert E Tarjan and Mihalis Yannakakis. "Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs." In: *SIAM Journal on computing* 13.3 (1984), pp. 566–579.

[36] Sebastian Thrun. "Probabilistic Robotics." In: *Commun. ACM* 45.3 (Mar. 2002), pp. 52–57. URL: https://doi.org/10.1145/504729.504754.

[37] Bill Triggs et al. "Bundle adjustment—a modern synthesis." In: *International workshop on vision algorithms*. Springer. 1999, pp. 298–372.

[38] Antoni Rosinol Vidal et al. "Ultimate SLAM? Combining events, images, and IMU for robust visual SLAM in HDR and high-speed scenarios." In: *IEEE Robotics and Automation Letters* 3.2 (2018), pp. 994–1001.

[39] Oliver J Woodman. *An introduction to inertial navigation*. Tech. rep. University of Cambridge, Computer Laboratory, 2007.

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

[41] J. Zhang and S. Singh. "Visual-lidar odometry and mapping: low-drift, robust, and fast." In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. 2015, pp. 2174–2181.

[42] Ji Zhang and Sanjiv Singh. "LOAM: Lidar Odometry and Mapping in Real-time." In: *Robotics: Science and Systems*. Vol. 2. 9.