Foundations and Trends® in Robotics
Vol. 6, No. 1-2 (2017) 1–139

DOI: 10.1561/2300000043

now
the essence of knowledge

Factor Graphs for Robot Perception

Frank Dellaert
Georgia Institute of Technology
dellaert@cc.gatech.edu

Michael Kaess
Carnegie Mellon University
kaess@cmu.edu

Contents

Abstract

We review the use of factor graphs for the modeling and solving of large-scale inference problems in robotics. Factor graphs are a family of probabilistic graphical models, other examples of which are Bayesian networks and Markov random fields, well known from the statistical modeling and machine learning literature. They provide a powerful abstraction that gives insight into particular inference problems, making it easier to think about and design solutions, and write modular software to perform the actual inference. We illustrate their use in the simultaneous localization and mapping problem and other important problems associated with deploying robots in the real world. We introduce factor graphs as an economical representation within which to formulate the different inference problems, setting the stage for the subsequent sections on practical methods to solve them. We explain the nonlinear optimization techniques for solving arbitrary nonlinear factor graphs, which requires repeatedly solving large sparse linear systems.

The sparse structure of the factor graph is the key to understanding this more general algorithm, and hence also understanding (and improving) sparse factorization methods. We provide insight into the graphs underlying robotics inference, and how their sparsity is affected by the implementation choices we make, crucial for achieving highly performant algorithms. As many inference problems in robotics are incremental, we also discuss the iSAM class of algorithms that can reuse previous computations, re-interpreting incremental matrix factorization methods as operations on graphical models, introducing the Bayes tree in the process. Because in most practical situations we will have to deal with 3D rotations and other nonlinear manifolds**,** we also introduce the more sophisticated machinery to perform optimization on nonlinear manifolds. Finally, we provide an overview of applications of factor graphs for robot perception, showing the broad impact factor graphs had in robot perception.

F. Dellaert and M. Kaess. *Factor Graphs for Robot Perception.* Foundations and Trends® in Robotics, vol. 6, no. 1-2, pp. 1–139, 2017.

DOI: 10.1561/2300000043.

1

Introduction

This article reviews the use of factor graphs for the modeling and solving of large-scale inference problems in robotics, including the simultaneous localization and mapping (SLAM) problem. Factor graphs are a family of probabilistic graphical models, other examples of which are Bayesian networks and Markov random fields, which are well known from the statistical modeling and machine learning literature. They provide a powerful abstraction to give insight into particular inference problems, making it easier to think about and design solutions, and write modular, flexible software to perform the actual inference. Below we illustrate their use in SLAM, one of the key problems in mobile robotics. Other important problems associated with deploying robots in the real world are localization, tracking, and calibration, all of which can be phrased in terms of factor graphs, as well.

In this first section we introduce Bayesian networks and factor graphs in the context of robotics problems. We start with Bayesian networks as they are probably the most familiar to the reader, and show how they are useful to *model* problems in robotics. However, since sensor data is typically given to us, we introduce factor graphs as a more relevant and economical representation. We show Bayesian

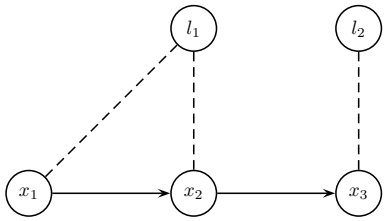


Figure 1.1: A toy SLAM (simultaneous localization and mapping) example with three robot poses and two landmarks. Above we schematically indicate the robot motion with arrows, while the dotted lines indicate bearing measurements.

networks can be effortlessly converted to factor graphs by conditioning on the sensor data. We then formulate the different inference problems as optimization problems on factor graphs, setting the stage for the subsequent sections on practical methods to solve them.

1.1 Inference Problems in Robotics

To act sensibly in the world, robots need to infer knowledge about the world from their sensors, while drawing on a priori knowledge. There are many different such inference problems in robotics, but none of them have received as much attention as simultaneous localization and mapping (SLAM). We discuss SLAM in detail and use it as a motivating example below. Other inference problems include localization in a *known* environment, tracking other actors in the environment, and multi-robot versions of all of the above. More specialized problems are also of interest, e.g., calibration or long-term inertial navigation.

In the SLAM problem the goal is to localize a robot using the information coming from the robot's sensors. In a simple case this could be a set of bearing measurements to a set of landmarks. If the landmarks' positions are known, this comes down to a triangulation problem reminiscent of how ships navigate at sea. However, the additional wrinkle in SLAM is that we do *not* know the landmark map a priori, and hence we have to infer the unknown map simultaneously with localization with respect to the evolving map.

Figure 1.1 shows a simple toy example illustrating the structure of the problem graphically. A robot located at three successive poses x_1, x_2, and x_3 makes bearing observations on two landmarks l_1 and l_2. To anchor the solution in space, let us also assume there is an absolute position/orientation measurement on the first pose x_1. Without this there would be no information about absolute position, as bearing measurements are all relative.

1.2 Probabilistic Modeling

Because of measurement uncertainty, we cannot hope to recover the true state of the world, but we can obtain a probabilistic description of what can be inferred from the measurements. In the Bayesian probability framework, we use the language of probability theory to assign a subjective degree of belief to uncertain events. Many excellent texts are available and listed at the end of this section that treat this subject in depth, which we do not have space for here.

In robotics we typically need to model a belief over continuous, multivariate random variables $x \in \mathbb{R}^n$. We do this using **probability density functions** (PDFs) $p(x)$ over the variables x, satisfying

$$\int p(x)dx = 1. \tag{1.1}$$

In terms of notation, we use lowercase letters for random variables, and uppercase letters to denote sets of them.

In SLAM we want to characterize our knowledge about the unknowns X, in this case robot poses and the unknown landmark positions, when given a set of *observed* measurements Z. Using the language of Bayesian probability, this is simply the conditional density

$$p(X|Z), \tag{1.2}$$

and obtaining a description like this is called **probabilistic inference**. A prerequisite is to first specify a probabilistic model for the variables of interest and how they give rise to (uncertain) measurements. This is where probabilistic graphical models enter the picture.

Probabilistic graphical models provide a mechanism to compactly describe complex probability densities by exploiting the struc-

ture in them [121]. In particular, high-dimensional probability densities can often be factorized as a product of many factors, each of which is a probability density over a much smaller domain. This will be explicitly modeled when we introduce factor graphs, later in this section. However, below we first introduce a different and perhaps more familiar graphical model, Bayesian networks, as they provide a gentler introduction into generative modeling.

1.3 Bayesian Networks for Generative Modeling

Bayesian networks are an expedient graphical language for modeling inference problems in robotics. This is because it is often easy to think about how measurements are generated by sensors. For example, if someone tells us the exact location of a landmark and the pose of a robot, as well as the geometry of its sensor configuration, it is not hard to *predict* what the measurement should be. And we can either assume or learn a *noise model* for a particular sensor. Measurement predictions and noise models are the core elements of a generative model, which is well matched with the Bayesian network framework.

Formally, a Bayesian network [163] or **Bayes net** is a directed graphical model where the nodes represent variables θ_j. We denote the entire set of random variables of interest as $\Theta = \{\theta_1 \dots \theta_n\}$. A Bayes net then defines a joint probability density $p(\Theta)$ over all variables Θ as the product of conditional densities associated with each of the nodes:

$$p(\Theta) \triangleq \prod_j p(\theta_j|\pi_j). \tag{1.3}$$

In the equation above $p(\theta_j|\pi_j)$ is the conditional density associated with node θ_j, and π_j is an assignment of values to the *parents* of θ_j. Hence, in a Bayes net, the factorization of the joint density is dictated by its graph structure, in particular the node-parent relationships.

As an example, let us consider the Bayes net associated with the toy SLAM example from Figure 1.1. In this case the random variables of interest are $\Theta = \{X, Z\}$, i.e., the unknown poses and landmarks X, *as well as* the measurements Z. The corresponding Bayes net for this toy example is shown in Figure 1.2, with the measurements shown in

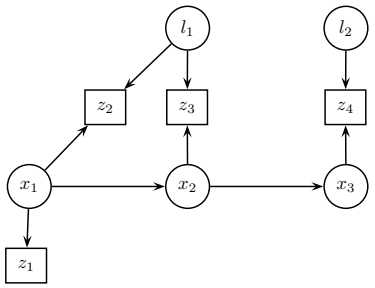


Figure 1.2: Bayes net for the toy SLAM example from Figure 1.1. Above we showed measurements with square nodes, as these variables are typically observed.

boxes as they are *observed*. Per the general definition of Bayes nets, the joint density $p(X, Z) = p(x_1, x_2, x_3, l_1, l_2, z_1, z_2, z_3, z_4)$ is obtained as a product of conditional densities:

$$\begin{aligned} p(X, Z) &= p(x_1)p(x_2|x_1)p(x_3|x_2) && (1.4)\\ &\times p(l_1)p(l_2) && (1.5)\\ &\times p(z_1|x_1) && (1.6)\\ &\times p(z_2|x_1, l_1)p(z_3|x_2, l_1)p(z_4|x_3, l_2). && (1.7) \end{aligned}$$

One can see that the joint density in this case consists of four qualitatively different sets of factors:

- A "Markov chain" $p(x_1)p(x_2|x_1)p(x_3|x_2)$ on the poses x_1, x_2, and x_3 [Eq. 1.4]. The conditional densities $p(x_{t+1}|x_t)$ might represent prior knowledge or can be derived from known control inputs.

- "Prior densities" $p(l_1)$ and $p(l_2)$ on the landmarks l_1 and l_2 (often omitted in SLAM settings when there is no prior map) [Eq. 1.5].

- A conditional density $p(z_1|x_1)$ corresponding to the absolute pose measurement on the first pose x_1 [Eq. 1.6].

- Last but not least, a product of three conditional densities, $p(z_2|x_1, l_1)p(z_3|x_2, l_1)p(z_4|x_3, l_2)$, corresponding to the three bearing measurements on the landmarks l_1 and l_2 from the poses x_1, x_2, and x_3 [Eq. 1.7].

Note that the graph structure makes an explicit statement about data association, i.e., for every measurement z_k we know which landmark it is a measurement of. While it is possible to model unknown data association in a graphical model context, in this text we assume that data association is given to us as the result of a pre-processing step.

1.4 Specifying Probability Densities

The exact form of the densities above depends very much on the application and the sensors used. The most often-used densities involve the **multivariate Gaussian distribution**, with probability density

$$\mathcal{N}(\theta; \mu, \Sigma) = \frac{1}{\sqrt{|2\pi\Sigma|}} \exp\left\{-\frac{1}{2} \|\theta - \mu\|_\Sigma^2\right\}, \tag{1.8}$$

where $\mu \in \mathbb{R}^n$ is the mean, Σ is an $n \times n$ covariance matrix, and

$$\|\theta - \mu\|_\Sigma^2 \triangleq (\theta - \mu)^\top \Sigma^{-1} (\theta - \mu) \tag{1.9}$$

denotes the squared Mahalanobis distance. For example, priors on unknown quantities are often specified using a Gaussian density.

In many cases it is both justified and convenient to model measurements as corrupted by zero-mean Gaussian noise. For example, a bearing measurement from a given pose x to a given landmark l would be modeled as

$$z = h(x, l) + \eta, \tag{1.10}$$

where $h(.)$ is a **measurement prediction function**, and the noise η is drawn from a zero-mean Gaussian density with measurement covariance R. This yields the following conditional density $p(z|x, l)$ on the measurement z:

$$p(z|x, l) = \mathcal{N}(z; h(x, l), R) = \frac{1}{\sqrt{|2\pi R|}} \exp\left\{-\frac{1}{2} \|h(x, l) - z\|_R^2\right\}. \tag{1.11}$$

The measurement functions $h(.)$ are often nonlinear in practical robotics applications. Still, while they depend on the actual sensor used, they are typically not difficult to reason about or write down. The measurement function for a 2D bearing measurement is simply

$$h(x, l) = \text{atan2}(l_y - x_y, l_x - x_x), \tag{1.12}$$

where atan2 is the well-known two-argument arctangent variant. Hence, the final **probabilistic measurement model** $p(z|x,l)$ is obtained as

$$p(z|x,l) = \frac{1}{\sqrt{|2\pi R|}} \exp\left\{-\frac{1}{2}\left\|\text{atan2}(l_y - x_y, l_x - x_x) - z\right\|_R^2\right\}. \quad (1.13)$$

Note that we will not *always* assume Gaussian measurement noise: to cope with the occasional data association mistake, for example, many authors have proposed the use of robust measurement densities, with heavier tails than a Gaussian density.

Not all probability densities involved are derived from measurements. For example, in the toy SLAM problem we have densities of the form $p(x_{t+1}|x_t)$, specifying a **probabilistic motion model** which the robot is assumed to obey. This *could* be derived from odometry measurements, in which case we would proceed exactly as described above. Alternatively, such a motion model could arise from known control inputs u_t. In practice, we often use a conditional Gaussian assumption,

$$p(x_{t+1}|x_t,u_t) = \frac{1}{\sqrt{|2\pi Q|}} \exp\left\{-\frac{1}{2}\left\|g(x_t,u_t) - x_{t+1}\right\|_Q^2\right\}, \quad (1.14)$$

where $g(.)$ is a motion model, and Q a covariance matrix of the appropriate dimensionality, e.g., 3×3 in the case of robots operating in the plane. Note that for robots operating in three-dimensional space, we will need slightly more sophisticated machinery to specify densities on nonlinear manifolds such as $SE(3)$, as discussed in Section 6.

1.5 Simulating from a Bayes Net Model

As an aside, once a probability model is specified as a Bayes net, it is easy to simulate from it. This is the reason why Bayes nets are the language of choice for generative modeling, and we mention it here because it is often beneficial to think about this when building models.

In particular, to simulate from $P(\Theta) \stackrel{\Delta}{=} \prod_j P(\theta_j|\pi_j)$, one simply has to topologically sort the nodes in the graph and sample in such a way that all parent values π_j are generated before sampling θ_j from the conditional $P(\theta_j|\pi_j)$, which can always be done. This technique is called *ancestral sampling* [16].

As an example, let us again consider the SLAM toy problem. Even in this tiny problem it is easy to see how the factorization of the joint density affords us to think *locally* rather than having to think globally. Indeed, we can use the Bayes net from Figure 1.2 as a guide to simulate from the joint density $p(x_1, x_2, x_3, l_1, l_2, z_1, z_2, z_3, z_4)$ by respectively

1. sampling the poses x_1, x_2, and x_3 from $p(x_1)p(x_2|x_1)p(x_3|x_2)$, i.e., simulate a robot trajectory;

2. sampling l_1 and l_2 from $p(l_1)$ and $p(l_2)$, i.e., generate some plausible landmarks;

3. sampling the measurements from the conditional densities $p(z_1|x_1)$, $p(z_2|x_1, l_1)$, $p(z_3|x_2, l_1)$, and $p(z_4|x_3, l_2)$, i.e., simulate the robot's sensors.

Many other topological orderings are possible. For example, steps 1 and 2 above can be switched without consequence. Also, we can generate the pose measurement z_1 at any time after x_1 is generated, etc.

1.6 Maximum a Posteriori Inference

Now that we have the means to model the world, we can infer knowledge about the world when given information about it. Above we saw how to fully specify a joint density $P(\Theta)$ in terms of a Bayes net: its factorization is given by its graphical structure, and its exact computational form by specifying the associated priors and conditional densities.

In robotics we are typically interested in the **unknown state variables** X, such as poses and/or landmarks, *given* the measurements Z. The most often used *estimator* for these unknown state variables X is the maximum a posteriori or **MAP estimate**, so named because it maximizes the posterior density $p(X|Z)$ of the states X given the measurements Z:

$$X^{MAP} = \operatorname*{argmax}_X p(X|Z) \tag{1.15}$$

$$= \operatorname*{argmax}_X \frac{p(Z|X)p(X)}{p(Z)}. \tag{1.16}$$

The second equation above is Bayes' law, and expresses the posterior as the product of the measurement density $p(Z|X)$ and the prior $p(X)$ over the states, appropriately normalized by the factor $p(Z)$.

However, a different expression of Bayes law is the key to understanding the true computation underlying MAP inference. Indeed, all of the quantities in Bayes' law as stated in (1.16) can in theory be computed from the Bayes net. However, as the measurements Z are *given*, the normalization factor $p(Z)$ is irrelevant to the maximization and can be dropped. In addition, while the conditional density $p(Z|X)$ is a properly normalized Gaussian density in Z, we are only concerned with it as a function in the unknown states X. Hence the second and more important form of Bayes' law:

$$X^{MAP} = \underset{X}{\operatorname{argmax}}\, l(X; Z)p(X). \tag{1.17}$$

Here $l(X; Z)$ is the **likelihood of the states X given the measurements** Z, and is defined as any function proportional to $p(Z|X)$:

$$l(X; Z) \propto p(Z|X). \tag{1.18}$$

The notation $l(X; Z)$ emphasizes the fact that the likelihood is a function of X and *not* Z, which acts merely as a parameter in this context.

It is important to realize that conditioning on the measurements yields likelihood functions that *do not look like Gaussian densities*, in general. To see this, consider again the 2D bearing measurement density in Equation 1.13. When written as a likelihood function we obtain

$$l(x, l; z) \propto \exp\left\{-\frac{1}{2}\left\|\text{atan2}(l_y - x_y, l_x - x_x) - z\right\|_R^2\right\}, \tag{1.19}$$

which is Gaussian in z (after normalization), but decidedly not so in any other variable. Even in the case of a *linear* measurement function, the measurement z is often of lower dimensionality than the unknown variables it depends on. Hence, conditioning on it results in a degenerate Gaussian density on the unknowns, at best; it is only when we fuse the information from several measurements that the density on the unknowns becomes a proper probability density. In the case that not enough measurements are available to fully constrain all variables,

MAP inference will fail, because a unique maximizer of the posterior (1.17) is not available.

All of the above motivates the introduction of factor graphs in the next section. The reasons for introducing a new graphical modeling language are (a) the distinct division between states X and measurements Z, and (b) the fact that we are more interested in the non-Gaussian likelihood functions, which are not proper probability densities. Hence, the Bayes net language is rather mismatched with the actual optimization problem that we are concerned with. Finally, we will see in Section 3 that the structure of factor graphs is intimately connected with the computational strategies to solve large-scale inference problems.

1.7 Factor Graphs for Inference

While Bayes nets are a great language for modeling, factor graphs are better suited to perform inference. Like Bayes nets, factor graphs allow us to specify a joint density as a product of factors. However, they are more general in that they can be used to specify *any* factored function $\phi(X)$ over a set of variables X, not just probability densities.

To motivate this, consider performing MAP inference for the toy SLAM example. After conditioning on the observed measurements Z, the posterior $p(X|Z)$ can be re-written using Bayes' law (1.16) as

$$\begin{aligned} p(X|Z) &\propto p(x_1)p(x_2|x_1)p(x_3|x_2) && (1.20)\\ &\quad \times p(l_1)p(l_2) && (1.21)\\ &\quad \times l(x_1; z_1) && (1.22)\\ &\quad \times l(x_1, l_1; z_2)l(x_2, l_1; z_3)l(x_3, l_2; z_4). && (1.23) \end{aligned}$$

It is clear that the above represents a factored probability density on the unknowns only, albeit unnormalized.

To make this factorization explicit, we use a **factor graph**. Figure 1.3 introduces the corresponding factor graph by example: all unknown states X, both poses and landmarks, have a node associated with them, as in the Bayes net. However, unlike the Bayes net case, measurements are *not* represented explicitly as they are given, and hence not of interest. Rather than associating each node with a conditional density, in

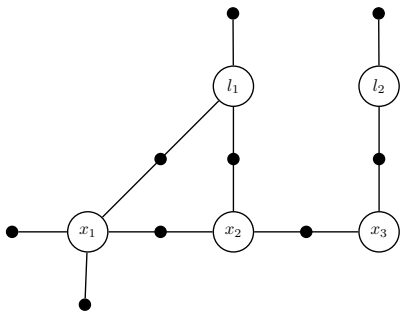


Figure 1.3: Factor graph resulting from the Bayes net in Figure 1.2 on page 6 after conditioning on the measurements Z.

factor graphs we explicitly introduce an additional node type to represent every *factor* in the posterior $p(X|Z)$. In the figure, each small black node represents a factor, and—importantly—is connected to only those state variables it is a function of. For example, the likelihood factor $l(x_3, l_2; z_4)$ is connected only to the variable nodes x_3 and l_2. Using this as a guide, it should be easy to associate each of the 9 factor nodes in the graph with the 9 factors in the posterior $p(X|Z)$.

Formally a factor graph is a bipartite graph $F = (\mathcal{U}, \mathcal{V}, \mathcal{E})$ with two types of nodes: **factors** $\phi_i \in \mathcal{U}$ and **variables** $x_j \in \mathcal{V}$. Edges $e_{ij} \in \mathcal{E}$ are always between factor nodes and variables nodes. The set of variable nodes adjacent to a factor ϕ_i is written as $\mathcal{N}(\phi_i)$, and we write X_i for an assignment to this set. With these definitions, a factor graph F defines the factorization of a global function $\phi(X)$ as

$$\phi(X) = \prod_i \phi_i(X_i). \tag{1.24}$$

In other words, the independence relationships are encoded by the edges e_{ij} of the factor graph, with each factor ϕ_i a function of *only* the variables X_i in its adjacency set $\mathcal{N}(\phi_i)$.

Every Bayes net can be trivially converted to a factor graph. Recall that every node in a Bayes net denotes a conditional density on the corresponding variable and its parent nodes. Hence, the conversion is quite simple: every Bayes net node splits in *both* a variable node and a factor node in the corresponding factor graph. The factor is connected

to the variable node, as well as the variable nodes corresponding to the parent nodes in the Bayes net. If some nodes in the Bayes net are evidence nodes, i.e., they are given as known variables, we omit the corresponding variable nodes: the known variable simply becomes a fixed parameter in the corresponding factor.

Following this recipe, in the simple SLAM example we obtain the following factor graph factorization,

$$\begin{aligned}\phi(l_1, l_2, x_1, x_2, x_3) &= \phi_1(x_1)\phi_2(x_2, x_1)\phi_3(x_3, x_2) && (1.25)\\ &\times \phi_4(l_1)\phi_5(l_2) && (1.26)\\ &\times \phi_6(x_1) && (1.27)\\ &\times \phi_7(x_1, l_1)\phi_8(x_2, l_1)\phi_9(x_3, l_2), && (1.28)\end{aligned}$$

where the correspondence between the factors and the original probability densities and/or likelihood factors in Equations 1.20-1.23 should be obvious, e.g., $\phi_7(x_1, l_1) = l(x_1, l_1; z_2) \propto p(z_2|x_1, l_1)$.

1.8 Computations Supported by Factor Graphs

While in the remainder of this document we concentrate on fast optimization methods for SLAM, it is of interest to ask what types of computations are supported by factor graphs *in general*. Converting a Bayes net $p(X, Z)$ to a factor graph (by conditioning on the evidence Z) yields a representation of the posterior $\phi(X) \propto p(X|Z)$, and it is natural to ask what we can do with this. While in SLAM we will be able to fully exploit the specific form of the factors to perform very fast inference, some domain-agnostic operations that are supported are *evaluation*, several *optimization* methods, and *sampling*.

Given any factor graph defining an unnormalized density $\phi(X)$, we can easily **evaluate** it for any given value, by simply evaluating every factor and multiplying the results. Often it is easier to work in log or negative log-space because of the small numbers involved, in which case we have to sum as many numbers as there are factors. Evaluation opens up the way to **optimization**, and nearly all gradient-agnostic optimization methods can be applied. If the factors are differentiable functions in continuous variables, gradient-based methods can quickly

find local maxima of the posterior. In the case of discrete variables, graph search methods can be applied, but they can often be quite costly. The hardest problems involve both discrete and continuous variables.

While local or global maxima of the posterior are often of most interest, **sampling** from a probability density can be used to visualize, explore, and compute statistics and expected values associated with the posterior. However, the ancestral sampling method from Section 1.5 only applies to directed acyclic graphs. The general sampling algorithms that are most useful for factor graphs are Markov chain Monte Carlo (MCMC) methods. One such method is Gibbs sampling, which proceeds by sampling one variable at a time from its conditional density given all other variables it is connected to via factors. This assumes that this conditional density can be easily obtained, however, which is true for discrete variables but far from obvious in the general case.

Below we use factor graphs as the organizing principle for all sections on specific inference algorithms. They aptly describe the independence assumptions and sparse nature of the large nonlinear least-squares problems arising in robotics, and that is where we start in the next section. But their usefulness extends far beyond that: they are at the core of the sparse linear solvers we use as building blocks, they clearly show the nature of filtering and incremental inference, and lead naturally to distributed and/or parallel versions of robotics. Before we dive in, we first lay out the roadmap for the remainder of the document.

1.9 Roadmap

In the next section, Section 2, we discuss **nonlinear optimization** techniques for solving the map inference problem in SLAM. Doing so requires repeatedly solving large sparse linear systems, but we do not go into detail on how this is done. The resulting graph-based optimization methods are now the most popular methods for the SLAM problem, at least when solved offline or in batch.

In Section 3 we make the connection between factor graphs and **sparse linear algebra** more explicit. While there exist efficient software libraries to solve sparse linear systems, these are but instantiations of a much more general algorithm: the elimination algorithm.

In Section 4 we discuss elimination **ordering** strategies and their effect on performance. This will also allow us to understand, in Section 5, the effects of marginalizing out variables, and its possibly deleterious effect on sparsity, especially in the SLAM case. Other inference problems in robotics do benefit from only keeping track of the most recent state estimate, which leads to filtering and/or fixed-lag smoothing algorithms.

In Section 5 we discuss **incremental factorization** and reinterpret it in terms of graphical models. We introduce the Bayes tree to establish a connection between sparse matrix factorization and graphical models, based on which incremental smoothing and mapping algorithms are developed.

While in many robotics problems we can get away with vector-valued unknowns, 3D rotations and other nonlinear **manifolds** need slightly more sophisticated machinery. Hence, in Section 6 we discuss optimization on manifolds.

1.10 Bibliographic Remarks

The SLAM problem [174, 129, 186] has received considerable attention in mobile robotics as it is one way to enable a robot to explore and navigate previously unknown environments. In addition, in many applications the map of the environment itself is the artifact of interest, e.g., in urban reconstruction, search-and-rescue operations, and battlefield reconnaissance. As such, it is one of the core competencies of autonomous robots [187]. A comprehensive review was done by Durrant-Whyte and Bailey in 2006 [59, 6] and more recently by Cadena et al. [19], but the field is still generating a steady stream of contributions at the top-tier robotics conferences.

The foundational book by Pearl [163] is still one of the best places to read about Bayesian probability and Bayesian networks, as is the tome by Koller and Friedman [121], and the book by Darwiche [38]. Although in these works the emphasis is (mostly) on problems with discrete-valued unknowns, they can just as easily be applied to continuous estimation problems like SLAM.

Because of their ability to represent the unnormalized posterior for MAP inference problems, factor graphs are an ideal graphical model for probabilistic robotics. However, factor graphs are also used extensively in a variety of other computer science fields, including Boolean satisfiability, constraint satisfaction, and machine learning. Excellent overviews of factor graphs and their applications are given by Kschischang et al. [125], and Loeliger [139].

Markov chain Monte Carlo (MCMC) and Gibbs sampling provide a way to sample over high-dimensional state-spaces as described by factor graphs, and are discussed in [151, 82, 55].

2

Smoothing and Mapping

Below we discuss the smoothing and mapping (SAM) algorithm, which is representative of the state of the art in batch solutions for SLAM. We explain the nonlinear optimization techniques for solving arbitrary nonlinear factor graphs, which requires repeatedly solving large sparse linear systems. In this section we will not go into detail on sparse linear algebra, but defer that to the next section.

2.1 Factor Graphs in SLAM

The factor graph for a more realistic SLAM problem than the toy example from the previous section could look something like Figure 2.1. This graph was created by simulating a 2D robot, moving in the plane for about 100 time steps, as it observes landmarks. For visualization purposes each robot pose and landmark is rendered at its ground truth position in 2D. With this, we see that the odometry factors form a prominent, chain-like backbone, whereas off to the sides binary likelihood factors are connected to the 20 or so landmarks. All factors in such SLAM problems are typically nonlinear, except for priors.

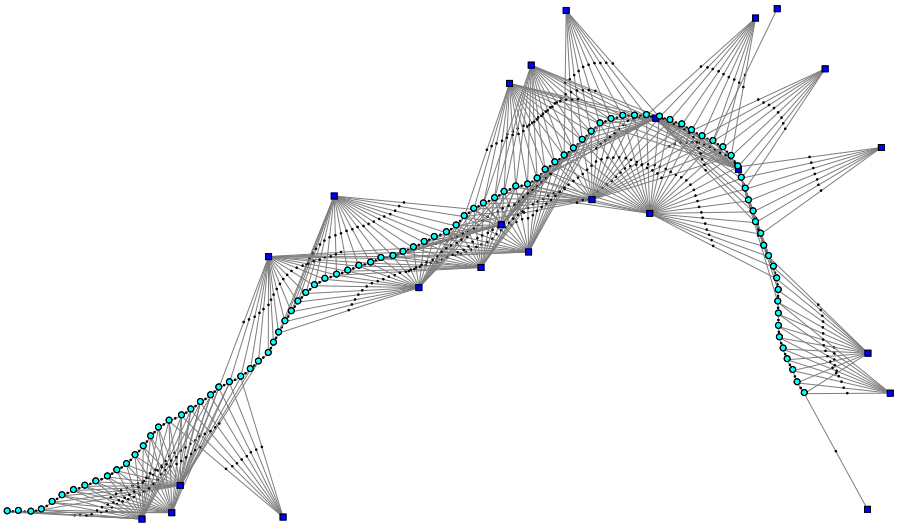


Figure 2.1: Factor graph for a larger, simulated SLAM example.

Simply examining the factor graph reveals a great deal of structure by which we can gain insight into a particular instance of the SLAM problem. First, there are landmarks with a large number of measurements, which we expect to be pinned down very well. Others have only a tenuous connection to the graph, and hence we expect them to be less well determined. For example, the lone landmark near the bottom-right has only a single measurement associated with it: if this is a bearing-only measurement, many assignments of a 2D location to the landmark will be equally "correct". This is the same as saying that we have infinite uncertainty in some subset of the domain of the unknowns, which is where prior knowledge should come to the rescue.

MAP inference in SLAM is exactly the process of determining those values for the unknowns that maximally agree with the information present in the uncertain measurements. In real life we are *not* given the ground truth locations for the landmarks, nor the time-varying pose of the robot, although in many practical cases we might have a good initial estimate. Below we show how to find an optimal assignment, the MAP estimate, through nonlinear optimization over the unknown variables in the factor graph.

2.2 MAP Inference for Nonlinear Factor Graphs

We now show that MAP inference for SLAM problems with Gaussian noise models is equivalent to solving a nonlinear least-squares problem. Indeed, for an arbitrary factor graph, MAP inference comes down to maximizing the product (1.24) of all factor graph potentials:

$$X^{MAP} = \operatorname*{argmax}_X \phi(X) \tag{2.1}$$

$$= \operatorname*{argmax}_X \prod_i \phi_i(X_i). \tag{2.2}$$

Let us for now assume that all factors are of the form

$$\phi_i(X_i) \propto \exp\left\{-\frac{1}{2}\left\|h_i(X_i) - z_i\right\|^2_{\Sigma_i}\right\}, \tag{2.3}$$

which include both simple Gaussian priors and likelihood factors derived from measurements corrupted by zero-mean, normally distributed noise. Taking the negative log of (2.2) and dropping the factor $1/2$ allows us to instead minimize a sum of **nonlinear least-squares**:

$$X^{MAP} = \operatorname*{argmin}_X \sum_i \left\|h_i(X_i) - z_i\right\|^2_{\Sigma_i}. \tag{2.4}$$

Minimizing this objective function performs sensor fusion through the process of combining several measurement-derived likelihood factors, and possibly several priors, to uniquely determine the MAP solution for the unknowns.

An important and non-obvious observation is that the factors in (2.4) typically represent rather *uninformed* densities on the involved unknown variables X_i. Indeed, except for simple prior factors, the measurements z_i are typically of lower dimension than the unknowns X_i. In those cases, the factor by itself accords the same likelihood to an infinite subset of the domain of X_i. For example, a 2D measurement in a camera image is consistent with an entire ray of 3D points that project to the same image location. Only when multiple measurements are combined can we hope to recover a unique solution for the variables.

Even though the functions h_i are nonlinear, *if* we have a decent initial guess available, then nonlinear optimization methods such as Gauss-Newton iterations or the Levenberg-Marquardt (LM) algorithm will be able to converge to the global minimum of (2.4). They do so by solving a succession of linear approximations to (2.4) in order to approach the minimum [50]. Hence, in the following we first consider how to build a linearized version of the nonlinear problem.

2.3 Linearization

We can linearize all measurement functions $h_i(\cdot)$ in the nonlinear least-squares objective function (2.4) using a simple Taylor expansion,

$$h_i(X_i) = h_i(X_i^0 + \Delta_i) \approx h_i(X_i^0) + H_i\Delta_i, \tag{2.5}$$

where the **measurement Jacobian** H_i is defined as the (multivariate) partial derivative of $h_i(.)$ at a given linearization point X_i^0,

$$H_i \triangleq \frac{\partial h_i(X_i)}{\partial X_i}\bigg|_{X_i^0}, \tag{2.6}$$

and $\Delta_i \triangleq X_i - X_i^0$ is the **state update vector**. Note that we make an assumption that X_i lives in a vector-space or, equivalently, can be represented by a *vector*. This is not always the case, e.g., when some of the unknown states in X represent 3D rotations or other more complex manifold types. We will revisit this issue in Section 6.

Substituting the Taylor expansion (2.5) into the nonlinear least-squares expression (2.4) we obtain a *linear* least-squares problem in the state update vector Δ,

$$\Delta^* = \underset{\Delta}{\operatorname{argmin}} \sum_i \left\| h_i(X_i^0) + H_i\Delta_i - z_i \right\|_{\Sigma_i}^2 \tag{2.7}$$

$$= \underset{\Delta}{\operatorname{argmin}} \sum_i \left\| H_i\Delta_i - \left\{ z_i - h_i(X_i^0) \right\} \right\|_{\Sigma_i}^2, \tag{2.8}$$

where $z_i - h_i(X_i^0)$ is the **prediction error** at the linearization point, i.e. the difference between actual and predicted measurement. Above Δ^* denotes the solution to the locally linearized problem.

By a simple change of variables we can drop the covariance matrices Σ_i from this point forward: with $\Sigma^{1/2}$ the matrix square root of Σ we can rewrite the Mahalanobis norm of some term e as follows:

$$\|e\|_\Sigma^2 \triangleq e^\top \Sigma^{-1} e = \left(\Sigma^{-1/2} e\right)^\top \left(\Sigma^{-1/2} e\right) = \left\|\Sigma^{-1/2} e\right\|_2^2. \tag{2.9}$$

Hence, we can eliminate the covariances Σ_i by pre-multiplying the Jacobian H_i and the prediction error in each term in (2.8) with $\Sigma_i^{-1/2}$:

$$A_i = \Sigma_i^{-1/2} H_i \tag{2.10}$$

$$b_i = \Sigma_i^{-1/2} \left(z_i - h_i(X_i^0)\right). \tag{2.11}$$

This process is a form of *whitening*. For example, in the case of scalar measurements it simply means dividing each term by the measurement standard deviation σ_i. Note that this eliminates the units of the measurements (e.g. length, angles) so that the different rows can be combined into a single cost function.

We finally obtain the following standard least-squares problem,

$$\Delta^* = \underset{\Delta}{\operatorname{argmin}} \sum_i \|A_i \Delta_i - b_i\|_2^2 \tag{2.12}$$

$$= \underset{\Delta}{\operatorname{argmin}} \|A\Delta - b\|_2^2, \tag{2.13}$$

where A and b are obtained by collecting all whitened Jacobian matrices A_i and whitened prediction errors b_i into one large matrix A and right-hand-side (RHS) vector b, respectively.

The Jacobian A is a large but sparse matrix, with a block structure that mirrors the structure of the underlying factor graph. We will examine this sparsity structure in detail in Section 3. First, however, we review the more classical linear algebra approach below.

2.4 Direct Methods for Least-Squares

For a full-rank $m \times n$ matrix A, with $m \geq n$, the unique least-squares solution to (2.13) can be found by solving the **normal equations**:

$$\left(A^\top A\right) \Delta^* = A^\top b. \tag{2.14}$$

This is normally done by factoring the **information matrix** Λ, defined and factored as follows:

$$\Lambda \triangleq A^\top A = R^\top R. \tag{2.15}$$

Above, the **Cholesky triangle** R is an upper-triangular $n \times n$ matrix[1] and is computed using **Cholesky factorization**, a variant of LU factorization for symmetric positive definite matrices. After this, Δ^* can be found by solving first

$$R^\top y = A^\top b \tag{2.16}$$

and then

$$R\Delta^* = y \tag{2.17}$$

by forward and back-substitution. For dense matrices Cholesky factorization requires $n^3/3$ flops, and the entire algorithm, including computing half of the symmetric $A^\top A$, requires $(m + n/3)n^2$ flops. One could also use LDL factorization, a variant of Cholesky decomposition that avoids the computation of square roots.

An alternative to Cholesky factorization that is more accurate and more numerically stable is to proceed via **QR-factorization**, which works *without* computing the information matrix Λ. Instead, we compute the QR-factorization of A itself along with its corresponding RHS:

$$A = Q \begin{bmatrix} R \\ 0 \end{bmatrix} \quad \begin{bmatrix} d \\ e \end{bmatrix} = Q^\top b. \tag{2.18}$$

Here Q is an $m \times m$ orthogonal matrix, $d \in \mathbb{R}^n$, $e \in \mathbb{R}^{m-n}$, and R is the *same* upper-triangular Cholesky triangle. The preferred method for factorizing a dense matrix A is to compute R column by column, proceeding from left to right. For each column j, all non-zero elements below the diagonal are zeroed out by multiplying A on the left with a **Householder reflection matrix** H_j. After n iterations A is completely factorized:

$$H_n \dots H_2 H_1 A = Q^\top A = \begin{bmatrix} R \\ 0 \end{bmatrix}. \tag{2.19}$$

[1] Some treatments, including [84], define the Cholesky triangle as the lower-triangular matrix $L = R^\top$, but the other convention is more convenient here.

The orthogonal matrix Q is not usually formed: instead, the transformed RHS $Q^\top b$ is computed by appending b as an extra column to A. Because the Q factor is orthogonal, we have

$$\|A\Delta - b\|_2^2 = \left\|Q^\top A\Delta - Q^\top b\right\|_2^2 = \|R\Delta - d\|_2^2 + \|e\|_2^2\,, \tag{2.20}$$

where we made use of the equalities from Equation 2.18. Clearly, $\|e\|_2^2$ will be the least-squares sum of squared residuals, and the least-squares solution Δ^* can be obtained by solving the triangular system

$$R\ \Delta^* = d \tag{2.21}$$

via back-substitution. The cost of QR is dominated by the cost of the Householder reflections, which is $2(m - n/3)n^2$. Comparing this with Cholesky, we see that both algorithms require $O(mn^2)$ operations when $m \gg n$, but that QR-factorization is slower by a factor of 2.

Note that the upper-triangular factor R obtained using QR factorization is the same (up to possible sign changes on the diagonal) as would be obtained by Cholesky factorization, as

$$A^\top A = \begin{bmatrix} R \\ 0 \end{bmatrix}^\top Q^\top Q \begin{bmatrix} R \\ 0 \end{bmatrix} = R^\top R, \tag{2.22}$$

where we again made use of the fact that Q is orthogonal.

There are efficient algorithms for factorizing large *sparse* matrices, for both QR and Cholesky variants. Depending on the amount of nonzeros and on the sparsity structure, the cost of a sparse factorization can be *far* lower than its dense equivalent. Efficient software implementations are available, e.g., CHOLMOD [28] and SuiteSparseQR [39], which are also used under the hood by MATLAB. In practice sparse Cholesky and LDL factorization outperform QR factorization on sparse problems as well, and not just by a constant factor.

In summary, the optimization problem associated with SLAM can be concisely stated in terms of sparse linear algebra. It comes down to factorizing either the information matrix Λ or the measurement Jacobian A into square root form. Because they are based on matrix square roots derived from the smoothing and mapping (SAM) problem, we have referred to this family of approaches as **square root SAM**, or $\sqrt{\text{SAM}}$ for short [46, 48].

2.5 Nonlinear Optimization for MAP Inference

Nonlinear least-squares problems cannot be solved directly, but require an iterative solution starting from a suitable initial estimate. A variety of algorithms exist that differ in how they locally approximate the cost function, and in how they find an improved estimate based on that local approximation. As a reminder, in our case the cost function is

$$g(X) \stackrel{\Delta}{=} \sum_i \|h_i(X_i) - z_i\|^2_{\Sigma_i} \tag{2.23}$$

and corresponds to a nonlinear factor graph derived from the measurements along with prior densities on some or all unknowns.

All of the algorithms share the following basic structure: They start from an initial estimate X^0. In each iteration, an update step Δ is calculated and applied to obtain the next estimate $X^{t+1} = X^t + \Delta$. This process ends when certain convergence criteria are reached, such as the change Δ falling below a small threshold.

2.5.1 Steepest Descent

Steepest descent (SD) or gradient descent uses the direction of steepest descent at the current estimate to calculate the following update step:

$$\Delta_{sd} = -\alpha \, \nabla g\,(X)|_{X=X^t} \,. \tag{2.24}$$

Here the negative gradient is used to identify the direction of steepest descent. For the nonlinear least-squares objective function (2.23), we compute the Jacobian A as in Section 2.3 to locally approximate $g(X) \approx \|A(X - X^t) - b\|_2^2$ and obtain the exact gradient $\nabla g\,(X)|_{X=X^t} = -2A^\top b$ at the linearization point X^t.

The step size α needs to be carefully chosen to balance between safe updates and reasonable convergence speed. An explicit line search can be performed to find a minimum in the given direction. SD is a simple algorithm, but suffers from slow convergence near the minimum.

2.5.2 Gauss-Newton

Gauss-Newton (GN) provides faster convergence by using a second order update. GN exploits the special structure of the nonlinear least-

squares problem to approximate the Hessian by the square of the Jacobian as $A^\top A$. The GN update step is obtained by solving the normal equations (2.14)

$$A^\top A \Delta_{gn} = A^\top b \tag{2.25}$$

by any of the methods in Section 2.4. For a well-behaved (i.e. nearly quadratic) objective function and a good initial estimate, Gauss-Newton exhibits nearly quadratic convergence. If the quadratic fit is poor, a GN step can lead to a new estimate that is further from the minimum and subsequent divergence.

2.5.3 Levenberg-Marquardt

The Levenberg-Marquardt (LM) algorithm allows for iterating multiple times to convergence while controlling in which region one is willing to trust the quadratic approximation made by Gauss-Newton. Hence, such a method is often called a **trust region method**.

To combine the advantages of both the SD and GN methods, Levenberg [133] proposed to modify the normal equations (2.14) by adding a non-negative constant $\lambda \in \mathbb{R}^+ \cup \{0\}$ to the diagonal

$$\left(A^\top A + \lambda I\right) \Delta_{lb} = A^\top b. \tag{2.26}$$

Note that for $\lambda = 0$ we obtain GN, and for large λ we approximately obtain $\Delta^* \approx \frac{1}{\lambda} A^\top b$, an update in the negative gradient direction of the cost function g (2.23). Hence, LM can be seen to blend naturally between the Gauss-Newton and Steepest Descent methods.

Marquardt [144] later proposed to take into account the scaling of the diagonal entries to provide faster convergence:

$$\left(A^\top A + \lambda \operatorname{diag}(A^\top A)\right) \Delta_{lm} = A^\top b. \tag{2.27}$$

This modification causes larger steps in the steepest descent direction if the gradient is small (nearly flat directions of the objective function) because there the inverse of the diagonal entries will be large. Conversely, in steep directions of the objective function the algorithm becomes more cautious and takes smaller steps. Both modifications of the normal equations can be interpreted in Bayesian terms as adding a zero-mean prior to the system.

Algorithm 2.1 The Levenberg-Marquardt algorithm

function LM($g(), X^0$) ▷ quadratic cost function $g()$, ▷ initial estimate X^0
  $\lambda = 10^{-4}$
  $t = 0$
  repeat
    $A, b \leftarrow$ linearize $g(X)$ at X^t
    $\Delta \leftarrow$ solve $\left(A^\top A + \lambda \operatorname{diag}(A^\top A)\right) \Delta = A^\top b$
    if g($X^t + \Delta$)<g(X^t) **then**
      $X^{t+1} = X^t + \Delta$ ▷ accept update
      $\lambda \leftarrow \lambda/10$
    else
      $X^{t+1} = X^t$ ▷ reject update
      $\lambda \leftarrow \lambda * 10$
    $t \leftarrow t + 1$
  until convergence
  return X^t ▷ return latest estimate

The LM algorithm is given in Algorithm 2.1. A key difference between GN and LM is that the latter rejects updates that would lead to a higher sum of squared residuals. A rejected update means that the nonlinear function is locally not well-behaved, and smaller steps are needed. This is achieved by heuristically increasing the value of λ, for example by multiplying its current value by a factor of 10, and resolving the modified normal equations. On the other hand, if a step leads to a reduction of the sum of squared residuals, it is accepted, and the state estimate is updated accordingly. In this case, λ is reduced (by dividing by a factor of 10), and the algorithm repeats with a new linearization point, until convergence.

2.5.4 Dogleg Minimization

Powell's dogleg (PDL) algorithm [167] can be a more efficient alternative to LM [140]. A major disadvantage of the Levenberg-Marquardt algorithm is that in case a step gets rejected, the modified information

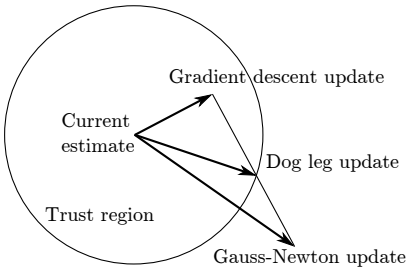


Figure 2.2: Powell's dogleg algorithm combines the separately computed Gauss-Newton and gradient descent update steps.

matrix has to be refactored, which is the most expensive component of the algorithm. Hence, the key idea behind PDL is to separately compute the GN and SD steps, and then combine appropriately. If the step gets rejected, the directions of the steps are still valid, and they can be combined in a different way until a reduction in the cost is achieved. Hence, each update of the state estimate only involves one matrix factorization, as opposed to several.

Figure 2.2 shows how the GN and SD steps are combined. The combined step starts with the SD update, followed by a sharp bend (hence the term dogleg) towards the GN update, but stopping at the trust region boundary. Unlike LM, PDL maintains an explicit trust region Δ within which we trust the linear assumption. The appropriateness of the linear approximation is determined by the gain ratio

$$\rho = \frac{g(X^t) - g(X^t + \Delta)}{L(0) - L(\Delta)}, \tag{2.28}$$

where $L(\Delta) = A^\top A\Delta - A^\top b$ is the linearization of the nonlinear quadratic cost function g from Equation 2.23 at the current estimate X^t. If ρ is small, i.e. $\rho < 0.25$, then the cost has not reduced as predicted by the linearization and the trust region is reduced. On the other hand, if the reduction is as predicted (or better), i.e. $\rho > 0.75$, then the trust region is increased depending on the magnitude of the update vector, and the step is accepted.

Both algorithms, GN and PDL, require the measurement Jacobian to be full rank so that $A^\top A$ is invertible. When encountering

under-constrained systems (insufficient measurements) or for numerically poorly constrained systems, the LM algorithm can be used instead, even though its convergence speed might be impacted.

2.6 Bibliographic Remarks

There is a large body of literature on the field of robot localization and mapping. A general overview of the area of SLAM can be found in [19, 59, 6, 187, 186]. Initial work on probabilistic SLAM was based on the extended Kalman filter (EKF) and is due to Smith et al. [176], building on earlier work [174, 175, 58]. In Section 4 we will treat the EKF more thoroughly, but in essence it recursively estimates a Gaussian density over the current pose of the robot and the position of all landmarks. However, as we will see, the computational complexity of the EKF becomes intractable fairly quickly. Many attempts were made to extending the filtering approach to cope with larger-scale environments [26, 52, 130, 91, 184, 188, 92], but *filtering* itself was shown to be inconsistent [105] when applied to the inherently nonlinear SLAM problem. This is mainly due to linearization choices that cannot be undone in a filtering framework. Later work [91, 120] focuses on reducing the effect of nonlinearities and providing more efficient, but typically approximate solutions to deal with larger environments.

A *smoothing* approach to SLAM involves not just the most current robot location, but the entire robot trajectory up to the current time. A number of authors consider the problem of smoothing the robot trajectory only [27, 141, 142, 94, 122, 61], which is particularly suited to sensors such as laser-range finders that easily yield pairwise constraints between nearby robot poses. More generally, one can consider the *full SLAM problem* [187], i.e., the problem of optimally estimating the entire set of sensor poses along with the parameters of all features in the environment. In fact, this problem has a long history in surveying [85], photogrammetry [18, 87, 173, 31], where it is known as "bundle adjustment", and computer vision [64, 181, 182, 191, 95], where it is referred to as "structure from motion". These then led to a flurry of work between 2000 and 2005 where these ideas were applied in the context of SLAM [56, 71, 70, 187].

Square root SAM was introduced in [46, 48] as a fundamentally better approach to the problem of SLAM than the EKF, based on the realization that,

- in contrast to the filtering-based covariance or information matrices, which *both* become fully dense over time [160, 188], the information matrix associated with smoothing is and stays sparse;
- in typical mapping scenarios (i.e., not repeatedly traversing a small environment) this matrix is a much more compact representation of the map covariance structure;
- the information matrix or measurement Jacobian can be factorized efficiently using sparse Cholesky or QR factorization, respectively, yielding a square root information matrix that can be used to immediately obtain the optimal robot trajectory and map.

Factoring the information matrix is known in the sequential estimation literature as square root information filtering (SRIF), and was developed in 1969 for use in JPL's Mariner 10 missions to Venus (as recounted by Bierman [14]). The use of square roots results in more accurate and stable algorithms, and, quoting Maybeck [145] "a number of practitioners have argued, with considerable logic, that square root filters should *always* be adopted in preference to the standard Kalman filter recursion". Maybeck briefly discusses the SRIF in a chapter on square root filtering, and it and other square root type algorithms are the subject of a book by Bierman [14].

Suitable nonlinear solvers are needed to apply the smoothing approach to measurement functions. A general in depth treatment of nonlinear solvers is provided by [155], while [84] focuses on the linear algebra perspective. The most basic nonlinear solver applicable to smoothing is the well-known Gauss-Newton algorithm. A more advanced and frequently used algorithm is Levenberg-Marquardt [133, 144]—this is also the algorithm used for square root SAM. Powell's dog leg [167, 140] can provide improved efficiency, and, as we will later see, is essential when incrementally updating matrix factorizations.

3

Exploiting Sparsity

As we saw in the previous section, performing MAP inference in *nonlinear* SLAM requires repeatedly solving large (but sparse) linear systems. While there exist efficient software libraries to solve these, these are but instantiations of a much more general algorithm. Sparse linear algebra is just the special case for linear-Gaussian factors, i.e., where all priors and measurements are assumed Gaussian, and only linear measurement functions are involved. The sparse structure of the factor graph is the key to understanding this more general algorithm, and hence also understanding (and improving) sparse factorization methods.

3.1 On Sparsity

3.1.1 Motivating Example

Dense methods will not scale to realistic problem sizes in SLAM. In the introduction we looked at a small toy problem to explain Bayes nets and factor graph formulations, for which a dense method will work fine. The larger simulation example, with its factor graph shown in Figure 2.1 on page 18, is more representative of real-world problems. However, it is still relatively small as real SLAM problems go, where problems with

thousands or even millions of unknowns are not unheard of. Yet, we are able to handle these without a problem because of sparsity.

The sparsity can be appreciated directly from looking at the factor graph. It is clear from Figure 2.1 that the graph is **sparse**, i.e., it is by no means a fully connected graph. The odometry chain linking the 100 unknown poses is a linear structure of 100 binary factors, instead of the possible 100^2 (binary) factors. In addition, with 20 landmarks we could have up to 2000 likelihood factors linking each landmark to each pose: the true number is closer to 400. And finally, there are no factors between landmarks at all. This reflects that we have not been given any information about their relative position. This structure is typical of most SLAM problems.

Below we will try to fully understand the sparse structure of these problems. We use the toy problem from the introduction as an illustration throughout. Then, at the end of this section we show how these concepts translate to the larger example, and to real-world problems.

3.1.2 The Sparse Jacobian and its Factor Graph

The key to modern SLAM algorithms is exploiting sparsity, and an important property of factor graphs in SLAM is that they represent the sparse block structure in the resulting sparse Jacobian matrix A. To see this, let us revisit the least-squares problem that is the key computation in the inner loop of the nonlinear SLAM problem:

$$\Delta^* \quad = \quad \underset{\Delta}{\operatorname{argmin}} \sum_i \|A_i \Delta_i - b_i\|_2^2 \,. \tag{3.1}$$

Each term above is derived from a factor in the original, nonlinear SLAM problem, linearized around the current linearization point (Equation 2.8). The matrices A_i can be broken up in blocks corresponding to each variable, and collected in a large, block-sparse Jacobian whose sparsity structure is given exactly by the factor graph.

Even though these linear problems typically arise as inner iterations in nonlinear optimization, we drop the Δ notation below, as everything holds for general linear problems regardless of their origin.

Example. Consider the factor graph for the small toy example, shown again for convenience in Figure 3.1. After linearization, we ob-

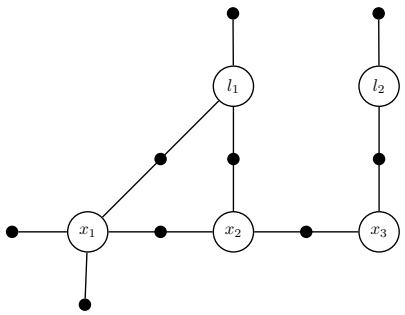


Figure 3.1: Factor graph (again) for the toy SLAM example.

$$
[A|b] = \begin{array}{c} \\ \phi_1 \\ \phi_2 \\ \phi_3 \\ \phi_4 \\ \phi_5 \\ \phi_6 \\ \phi_7 \\ \phi_8 \\ \phi_9 \end{array}
\left[\begin{array}{ccccc|c}
\delta l_1 & \delta l_2 & \delta x_1 & \delta x_2 & \delta x_3 & b \\
\hline
 & & A_{13} & & & b_1 \\
 & & A_{23} & A_{24} & & b_2 \\
 & & & A_{34} & A_{35} & b_3 \\
A_{41} & & & & & b_4 \\
 & A_{52} & & & & b_5 \\
 & & A_{63} & & & b_6 \\
A_{71} & & A_{73} & & & b_7 \\
A_{81} & & & A_{84} & & b_8 \\
 & A_{92} & & & A_{95} & b_9
\end{array}\right]
$$

Figure 3.2: Block structure of the sparse Jacobian A for the toy SLAM example with $\Delta = \left(\delta l_1^\top, \delta l_2^\top, \delta x_1^\top, \delta x_2^\top, \delta x_3^\top\right)^\top$.

tain a sparse system $[A|b]$ with the block structure in Figure 3.2. Comparing this with the factor graph, it is obvious that every factor corresponds to a block row, and every variable corresponds to a block column of A. In total there are nine block-rows, one for every factor in the factorization of $\phi(l_1, l_2, x_1, x_2, x_3)$.

3.1.3 The Sparse Information Matrix and its Graph

When using Cholesky factorization for solving the normal equations, as explained in Section 2.4, we first form the Hessian or information matrix $\Lambda = A^\top A$**.** In general, since the Jacobian A is block-sparse, the

$$\begin{bmatrix} \Lambda_{11} & & \Lambda_{13} & \Lambda_{14} & \\ & \Lambda_{22} & & & \Lambda_{25} \\ \Lambda_{31} & & \Lambda_{33} & \Lambda_{34} & \\ \Lambda_{41} & & \Lambda_{43} & \Lambda_{44} & \Lambda_{45} \\ & \Lambda_{52} & & \Lambda_{54} & \Lambda_{55} \end{bmatrix}$$

Figure 3.3: The Hessian matrix $\Lambda \triangleq A^\top A$ for the toy SLAM problem.

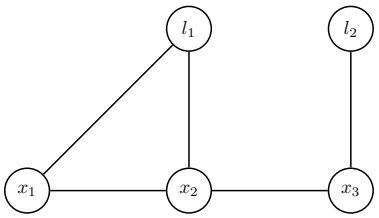


Figure 3.4: The Hessian matrix Λ can be interpreted as the matrix associated with the Markov random field representation for the problem.

Hessian Λ is expected to be sparse as well. By construction, the Hessian is a symmetric matrix, and if a unique MAP solution to the problem exists, it is also positive definite.

The information matrix Λ can be associated with yet another, *undirected* graphical model for the SLAM problem, namely a **Markov random field** or MRF. In contrast to a factor graph, an MRF is a graphical model that involves only the variables, just like a Bayes net. But unlike a Bayes net, the graph G of an MRF is an undirected graph: the edges only indicate that there is *some* interaction between the variables involved. At the block-level, the sparsity pattern of $\Lambda = A^\top A$ is exactly the adjacency matrix of G.

Example. Figure 3.3 shows the information matrix Λ associated with our running toy example. In this case there are five variables that partition the Hessian as shown. The zero blocks indicate which variables do not interact, e.g., l_1 and l_2 have no direct interaction. Figure 3.4 shows the corresponding MRF.

In what follows we will frequently refer to the undirected graph G of the MRF associated with an inference problem. However, we will not

use the MRF graphical model much beyond that. Note that one can develop an equivalent theory of how MRFs represent a different family of factored probability densities, see e.g. Koller and Friedman [121]. In the linear-Gaussian case, for instance, the least-squares error can be re-written as

$$\begin{aligned} \|A\Delta - b\|_2^2 &= \Delta^\top A^\top A\Delta - 2\Delta^\top A^\top b + b^\top b & (3.2)\\ &= b^\top b - 2\sum_j g_j^\top \Delta_j + \sum_{ij} \Delta_i^\top \Lambda_{ij} \Delta_j, & (3.3) \end{aligned}$$

where $g \triangleq A^\top b$. After exponentiating, we see that the induced Gaussian density has the form

$$p(\Delta) \propto \exp\left(-\|A\Delta - b\|_2^2\right) \propto \prod_j \phi_j(\Delta_j) \prod_{ij} \psi_j(\Delta_i, \Delta_j), \qquad (3.4)$$

which is the general form for densities induced by binary MRFs [204].

In what follows, however, factor graphs are better suited to our needs. They are able to express a finer-grained factorization, and are more closely related to the original problem formulation. For example, if there exist ternary (or higher arity) factors in the factor graph, the graph G of the equivalent MRF connects those nodes in an undirected clique (a fully connected subgraph), but the origin of the corresponding clique potential is lost. In linear algebra, this reflects the fact that many matrices A can yield the same $\Lambda = A^\top A$ matrix: important information on the sparsity is lost.

3.2 The Elimination Algorithm

There exists a general algorithm that, given any (preferably sparse) factor graph, can compute the corresponding posterior density $p(X|Z)$ on the unknown variables X in a form that allows easy recovery of the MAP solution to the problem. As we saw, a factor graph represents the unnormalized posterior $\phi(X) \propto P(X|Z)$ as a product of factors, and in SLAM problems this graph is typically generated directly from the measurements. The elimination algorithm is a recipe for converting a factor graph back to a Bayes net, but now *only* on the unknown

Algorithm 3.1 The Variable Elimination Algorithm

function ELIMINATE($\Phi_{1:n}$) ▷ given a factor graph on n variables
  for $j = 1...n$ **do** ▷ for all variables
    $p(x_j|S_j), \Phi_{j+1:n} \leftarrow \text{EliminateOne}(\Phi_{j:n}, x_j)$ ▷ eliminate x_j
  return $p(x_1|S_1)p(x_2|S_2)\dots p(x_n)$ ▷ return Bayes net

Algorithm 3.2 Eliminate variable x_j from a factor graph $\Phi_{j:n}$.

function ELIMINATEONE($\Phi_{j:n}, x_j$) ▷ given reduced graph $\Phi_{j:n}$
  Remove all factors $\phi_i(X_i)$ that are adjacent to x_j
  $\mathcal{S}(x_j) \leftarrow$ all variables involved excluding x_j ▷ the separator
  $\psi(x_j, S_j) \leftarrow \prod_i \phi_i(X_i)$ ▷ create the product factor ψ
  $p(x_j|S_j)\tau(S_j) \leftarrow \psi(x_j, S_j)$ ▷ factorize the product ψ
  Add the new factor $\tau(S_j)$ back into the graph
  return $p(x_j|S_j), \Phi_{j+1:n}$ ▷ Conditional and reduced graph

variables X. This then allows for easy MAP inference, and even other operations such as sampling (as we saw before) and/or marginalization.

In particular, the **variable elimination** algorithm is a way to factorize any factor graph of the form

$$\phi(X) = \phi(x_1, \dots, x_n) \tag{3.5}$$

into a factored Bayes net probability density of the form

$$p(X) = p(x_1|S_1)p(x_2|S_2)\dots p(x_n) = \prod_j p(x_j|S_j), \tag{3.6}$$

where S_j denotes an assignment to the **separator** $\mathcal{S}(x_j)$ associated with variable x_j under the chosen variable ordering $x_1, \dots, x_n$. The separator is defined as the set of variables on which x_j is conditioned, after elimination. While this factorization is akin to the chain rule, eliminating a sparse factor graph will typically lead to small separators.

The elimination algorithm is listed as Algorithm 3.1, where we used the shorthand notation $\Phi_{j:n} \triangleq \phi(x_j, \dots, x_n)$ to denote a partially eliminated factor graph. The algorithm proceeds by eliminating one variable x_j at a time, starting with the complete factor graph $\Phi_{1:n}$. As we eliminate each variable x_j, the function ELIMINATEONE produces a single

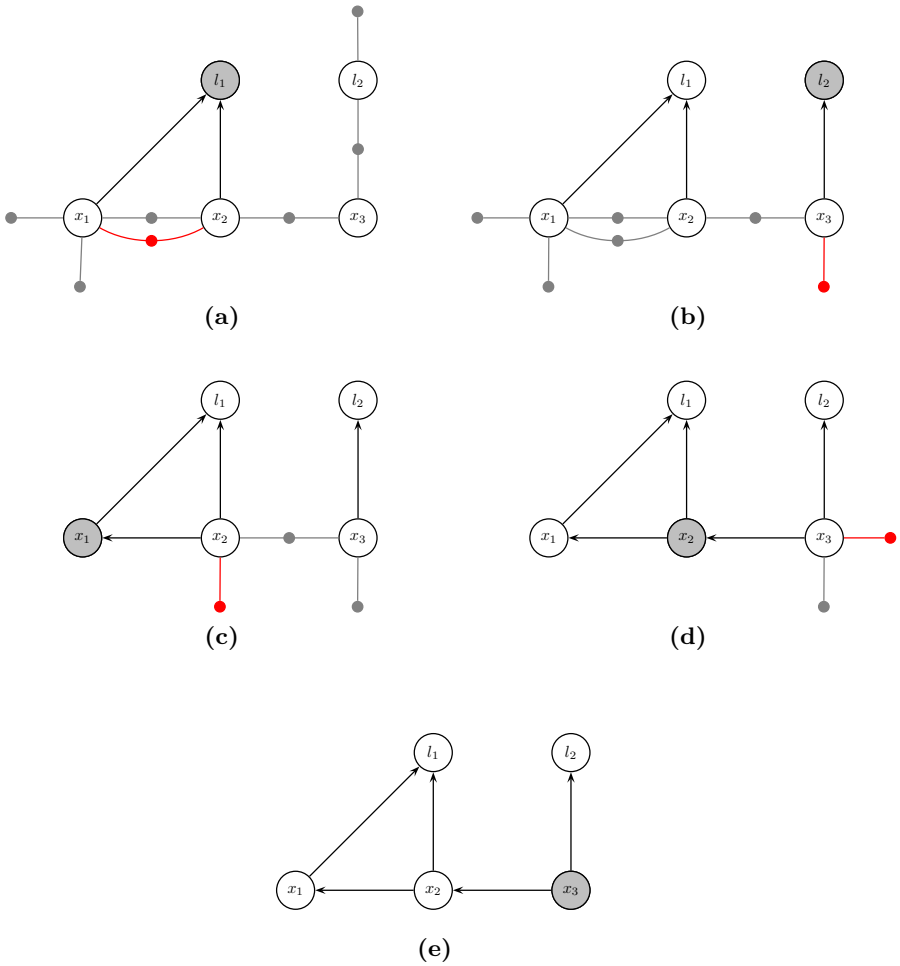


Figure 3.5: Variable elimination for the toy SLAM example, transforming the factor graph from Figure 3.1 into a Bayes net, using the ordering l_1, l_2, x_1, x_2, x_3.

conditional $p(x_j|S_j)$, as well as a reduced factor graph $\Phi_{j+1:n}$ on the remaining variables. After all variables have been eliminated, the algorithm returns the resulting Bayes net with the desired factorization.

The pseudo-code for eliminating a single variable x_j is listed as Algorithm 3.2. Given a partially eliminated factor graph $\Phi_{j:n}$, we first remove all factors $\phi_i(x_i)$ that are adjacent to x_j and multiply them

into the product factor $\psi(x_j, S_j)$. We then factorize $\psi(x_j, S_j)$ into a conditional distribution $p(x_j|S_j)$ on the eliminated variable x_j, and a new factor $\tau(S_j)$ on the separator $\mathcal{S}(x_j)$:

$$\psi(x_j, S_j) = p(x_j|S_j)\tau(S_j). \tag{3.7}$$

Hence, *the entire factorization from $\phi(X)$ to $p(X)$ is seen to be a succession of n local factorization steps.* When eliminating the last variable x_n the separator $\mathcal{S}(x_n)$ will be empty, and the conditional produced will simply be a prior $p(x_n)$ on x_n.

Example. One possible elimination sequence for the toy example is shown in Figure 3.5, for the ordering l_1, l_2, x_1, x_2, x_3. In each step, the variable being eliminated is shaded gray, and the new factor $\tau(S_j)$ on the separator S_j is shown in red. Taken as a whole, the variable elimination algorithm factorizes the factor graph $\phi(l_1, l_2, x_1, x_2, x_3)$ into the Bayes net in Figure 3.5, corresponding to the factorization

$$\begin{aligned} p(l_1, l_2, x_1, x_2, x_3) &= p(l_1|x_1, x_2)p(l_2|x_3) \\ &\quad p(x_1|x_2)p(x_2|x_3)p(x_3). \end{aligned} \tag{3.8}$$

3.3 Sparse Matrix Factorization as Variable Elimination

In the case of linear measurement functions and additive normally distributed noise, the *elimination algorithm is equivalent to sparse matrix factorization.* Both sparse Cholesky and QR factorization are a special case of the general algorithm.

3.3.1 Sparse Gaussian Factors

Let us consider the elimination of a single variable x_j, as outlined in Algorithm 3.2 on page 35. In the least-squares problem (3.1), all factors are of the form

$$\phi_i(X_i) = \exp\left\{-\frac{1}{2}\left\|A_i X_i - b_i\right\|_2^2\right\}, \tag{3.9}$$

where X_i are all the variables involved in factor ϕ_i, with A_i composed of smaller sub-blocks corresponding to each variable.

Example. The linearized factor ϕ_7 between l_1 and x_1 in the toy SLAM example is equal to

$$\phi_7(l_1, x_1) = \exp\left\{-\frac{1}{2}\left\|A_{71}\ l_1 + A_{73}\ x_1 - b_7\right\|_2^2\right\}, \tag{3.10}$$

which corresponds to

$$A_7 \triangleq [A_{71}|A_{73}] \tag{3.11}$$

$$X_7 \triangleq [l_1; x_1], \tag{3.12}$$

where we use the semicolon to indicate concatenation of column vectors.

3.3.2 Forming the Product Factor

As explained before, the elimination algorithm proceeds one variable at a time. Following Algorithm 3.2, for every variable x_j we remove all factors $\phi_i(X_i)$ adjacent to x_j, and form the intermediate product factor $\psi(x_j, S_j)$. This can be done by accumulating all the matrices A_i into a new, larger block-matrix $\bar{A}_j$, as we can write

$$\psi(x_j, S_j) \leftarrow \prod_i \phi_i(X_i) \tag{3.13}$$

$$= \exp\left\{-\frac{1}{2}\sum_i \left\|A_i X_i - b_i\right\|_2^2\right\} \tag{3.14}$$

$$= \exp\left\{-\frac{1}{2}\left\|\bar{A}_j[x_j; S_j] - \bar{b}_j\right\|_2^2\right\}, \tag{3.15}$$

where the new RHS vector $\bar{b}_j$ stacks all b_i.

Example. Consider eliminating the variable l_1 in the toy example. The adjacent factors are ϕ_4, ϕ_7 and ϕ_8, in turn inducing the separator $S_1 = [x_1; x_2]$. The product factor is then equal to

$$\psi\left(l_1, x_1, x_2\right) = \exp\left\{-\frac{1}{2}\left\|\bar{A}_1[l_1; x_1; x_2] - \bar{b}_1\right\|_2^2\right\}, \tag{3.16}$$

with

$$\bar{A}_1 \triangleq \begin{bmatrix} A_{41} & & \\ A_{71} & A_{73} & \\ A_{81} & & A_{84} \end{bmatrix}, \qquad \bar{b}_1 \triangleq \begin{bmatrix} b_4 \\ b_7 \\ b_8 \end{bmatrix}. \tag{3.17}$$

Looking at the sparse Jacobian in Figure 3.2 on page 32, this simply boils down to taking out the block rows with non-zero blocks in the first column, corresponding to the three factors adjacent to l_1.

3.3.3 Eliminating a Variable using Partial QR

Factorizing the product $\psi(x_j, S_j)$ can be done in several different ways. We first discuss the QR variant, as it more directly connects to the linearized factors. In particular, the augmented matrix $[\bar{A}_j|\bar{b}_j]$ corresponding to the product factor $\psi(x_j, S_j)$ can be rewritten using partial QR-factorization [84] as follows:

$$[\bar{A}_j|\bar{b}_j] = Q \begin{bmatrix} R_j & T_j & d_j \\ & \tilde{A}_\tau & \tilde{b}_\tau \end{bmatrix}, \tag{3.18}$$

where R_j is an upper-triangular matrix. This allows us to factor $\psi(x_j, S_j)$ as follows:

$$\begin{aligned} \psi(x_j, S_j) &= \exp\left\{-\frac{1}{2}\left\|\bar{A}_j[x_j; S_j] - \bar{b}_j\right\|_2^2\right\} && (3.19) \\ &= \exp\left\{-\frac{1}{2}\left\|R_j x_j + T_j S_j - d_j\right\|_2^2\right\} \exp\left\{-\frac{1}{2}\left\|\tilde{A}_\tau S_j - \tilde{b}_\tau\right\|_2^2\right\} \\ &= p(x_j|S_j)\tau(S_j), && (3.20) \end{aligned}$$

where we used the fact that the rotation matrix Q does not alter the value of the norms involved.

Example. In Figure 3.6 we show the result of eliminating the first variable in the example, the landmark l_1 with separator $\{x_1, x_2\}$. We show the operation on the factor graph *and* the corresponding effect on the sparse Jacobian from Figure 3.2, omitting the RHS. The partition above the line corresponds to a sparse, upper-triangular matrix R that is being formed. New contributions to the matrix are shown in boldface: blue for the contributions to R, and red for newly created factors.

3.3.4 Multifrontal QR Factorization

The entire elimination algorithm, using partial QR to eliminate a single variable, is equivalent to **sparse QR factorization**. As the treatment above considers multi-dimensional variables $x_j \in \mathbb{R}^{n_j}$, this is in fact an

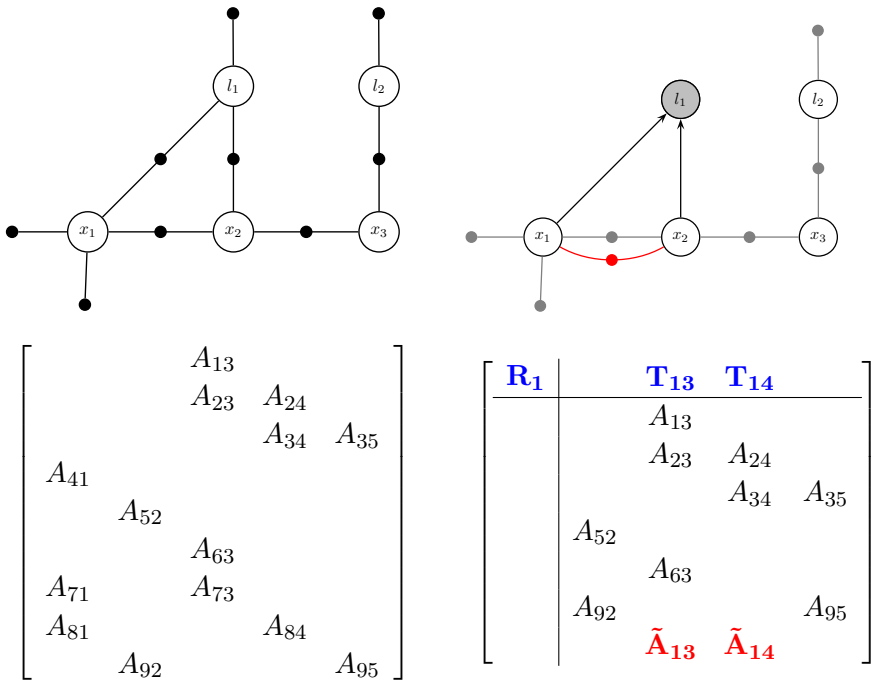


Figure 3.6: Eliminating the variable l_1 as a partial sparse factorization step.

instance of multi-frontal QR factorization [57], as we eliminate several scalar variables at a time, which is beneficial for processor utilization. While in our case the scalar variables are grouped because of their semantic meaning in the inference problem, sparse linear algebra codes typically analyze the problem to group for maximum computational efficiency. In many cases these two strategies are closely aligned.

Example. For completeness, we show the four remaining variable elimination steps in Figure 3.7, showing an end-to-end example of how multifrontal QR factorization proceeds on a small example. The final step shows the equivalence between the resulting Bayes net and the sparse upper-triangular factor R.

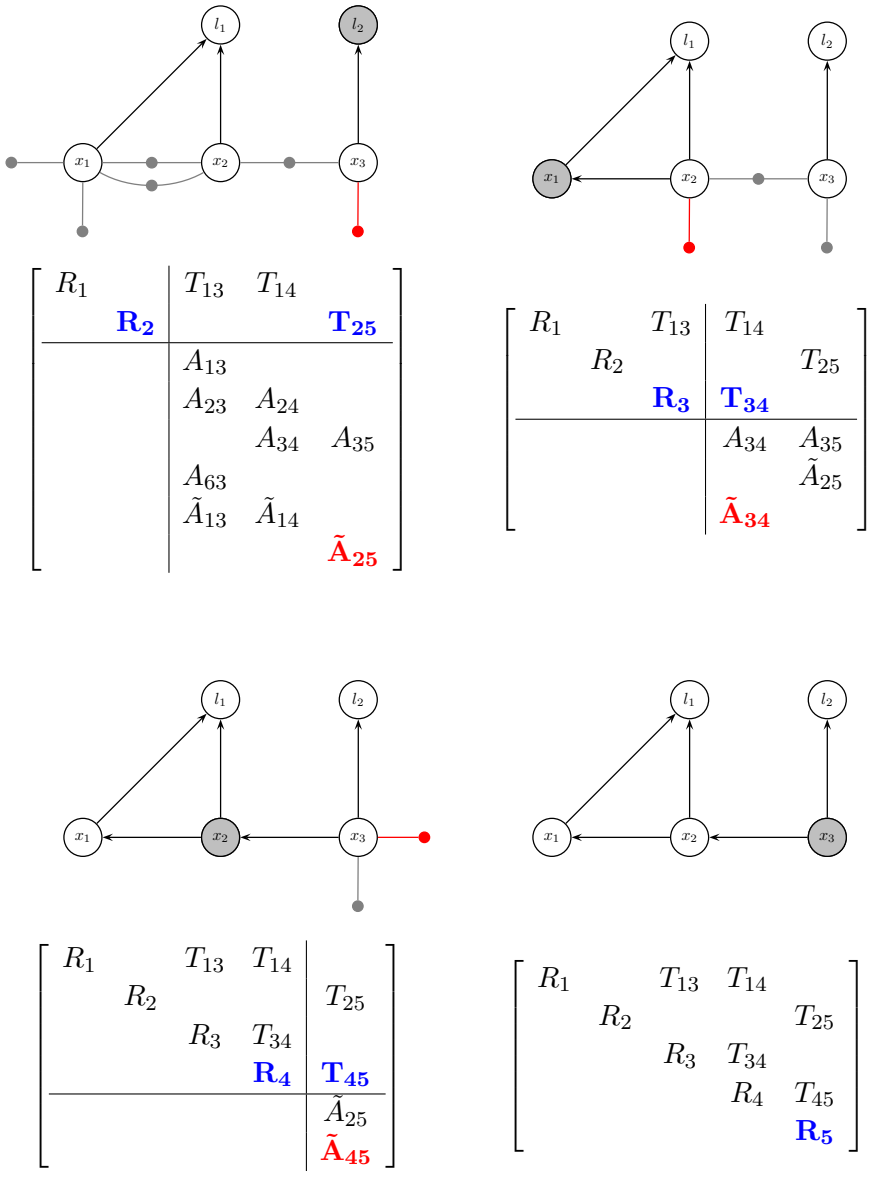


Figure 3.7: The remaining elimination steps for the toy example, completing a full multifrontal QR factorization.

3.4 The Sparse Cholesky Factor as a Bayes Net

The equivalence between variable elimination and sparse matrix factorization reveals that the graphical model associated with an upper triangular matrix is a Bayes net! Just like a factor graph is the graphical embodiment of a sparse Jacobian, and an MRF can be associated with the Hessian, a Bayes net reveals the sparsity structure of a Cholesky factor. In hindsight, this perhaps is not too surprising: a Bayes net is a directed acyclic graph (DAG), and that is exactly the "upper-triangular" property for matrices.

What's more, the Cholesky factor corresponds to a **Gaussian Bayes net**, which we defined as one made up of linear-Gaussian conditionals. The variable elimination algorithm holds for general densities, but in case the factor graph only contains linear measurement functions and Gaussian additive noise, the resulting Bayes net has a very specific form. We discuss the details below, as well as how to solve for the MAP estimate in the linear case.

3.4.1 Linear-Gaussian Conditionals

As we discussed in Section 3.2 on page 34 on the elimination algorithm in general, the Gaussian factor graph corresponding to the linearized nonlinear problem is transformed by elimination into the density $P(X)$ given by the now familiar Bayes net factorization:

$$P(X) = \prod_j p(x_j|S_j). \tag{3.21}$$

In both QR and Cholesky variants, the conditional densities $p(x_j|S_j)$ are given by

$$p(x_j|S_j) = k \exp\left\{-\frac{1}{2}\left\|R_j x_j + T_j S_j - d_j\right\|_2^2\right\}, \tag{3.22}$$

which is a linear-Gaussian density on the eliminated variable x_j. Indeed, we have

$$\|R_j x_j + T_j S_j - d_j\|_2^2 = (x_j - \mu_j)^\top R_j^\top R_j (x_j - \mu_j) \stackrel{\Delta}{=} \|x_j - \mu_j\|_{\Sigma_j}^2, \tag{3.23}$$

where the mean $\mu_j = R_j^{-1}(d_j - T_j S_j)$ depends linearly on the separator S_j, and the covariance matrix is given by $\Sigma_j = (R_j^\top R_j)^{-1}$. Hence, the normalization constant $k = |2\pi\Sigma_j|^{-1/2}$.

3.4.2 Solving a Bayes Net is Back-substitution

After the elimination step is complete, back-substitution is used to obtain the MAP estimate of each variable. As seen in Figure 3.7, the last variable eliminated does not depend on any other variables. Thus, the MAP estimate of the last variable can be directly extracted from the Bayes net. By proceeding in reverse elimination order, the values of all the separator variables for each conditional will always be available from the previous steps, allowing the estimate for the current frontal variable to be computed.

Algorithm 3.3 Back-substitution in Bayes Net

1: **function** SOLVE($p(X)$) ▷ given Gaussian Bayes net on n variables
2: **for** $j = n...1$ **do** ▷ reverse elimination order
3: $x_j^* \leftarrow R_j^{-1}(d_j - T_j S_j^*)$ ▷ solve for x_j^* given separator S_j^*

The solving procedure is summarized in Algorithm 3.3. At every step, the MAP estimate for the variable x_j is the conditional mean,

$$x_j^* = R_j^{-1}(d_j - T_j S_j^*), \tag{3.24}$$

since by construction the MAP estimate for the separator S_j^* is fully known by this point.

3.5 Discussion

Above we show how the elimination algorithm can be used to efficiently solve the linear systems created as part of MAP inference in SLAM, and this generalizes to other applications. In particular, we show that using partial QR factorization, when eliminating a single variable, leads to a well-known sparse matrix factorization method. One could then rightfully ask why all this matters, since efficient codes exist to solve

sparse linear systems. The reason *we* think it matters is because it provides *insight*, and because the elimination approach is more general than just linear algebra.

The interpretation in terms of probabilistic graphical models is not something that is appreciated by the sparse linear algebra community. There, researchers are concerned with linear systems regardless of their provenance. They could derive from applications as diverse as fluid dynamics, airplane design, or weather forecasting, and hence the software packages have to treat them in a generic way. However, in robotics, and indeed other continuous estimation problems, it is advantageous to reason in terms of Bayesian probabilities and MAP inference. The explanation above makes this connection explicit, and highlights how sparse linear algebra can be used as the computational engine.

In a very real sense, sparse linear algebra factorization methods are just a special case of a much more general algorithm, and this opens the door to algorithmic innovation and/or judicious, informed approximations. When stating the algorithm, we did *not* specify that the densities involved needed to be Gaussian, or even that the variables need to be continuous. The very same algorithm can perform MAP inference and/or marginalization in discrete problems, or even mixed discrete-continuous problems.

In the next sections, we will deepen this connection and describe both old and new algorithms in this new light.

3.6 Bibliographic Remarks

The variable elimination algorithm originated in order to solve systems of linear equations. It was first applied in modern times by Gauss in the early 1800s [75, see article 180 on page 262 of the English translation]. He was interested in solving least-squares problems related to astronomy, in particular, computing the orbit of the "planets" Ceres and Pallas [75, 76]. The method he discovered is now known as Cholesky factorization, an elimination variant for least-squares problems. However, the algorithm which we now commonly refer to as *Gaussian elimination* was already known to the Chinese in the 2^{nd} century B.C.

A graphical view of the elimination process first appeared in the analysis of large systems of *sparse* linear equations. Modern linear algebra originated after WWII with the advent of digital computers, and matrix factorization methods revolutionized matrix computations (see Stewart [178] and the references therein). Parter [159] seems to have been the first to analyze sparse matrices using graphs and study different triangulations of a graph. He describes Cholesky factorization on this graph by repeatedly choosing a vertex v, adding edges to make the neighborhood of v into a clique, and then removing v from the graph.

Whereas sparse symmetric matrices are represented by undirected graphs, non-symmetric matrices and their factorizations (most notably QR factorization) can be analyzed by means of *bipartite* graphs [79, 81]. In bipartite graphs, elimination is done via a *bipartite elimination game*, as described by Heggernes and Matstoms [97]. These graphs, of course, are the symbolic equivalent of the factor graphs we discuss in this article.

The view of elimination as an algorithm that operates on a graph allows one to generalize away from linear equations. Carré [25] shows that all operations can be specified in terms of a semiring $(S, \oplus, \otimes)$, and that quite a few shortest path and network flow methods can be seen as variations of known matrix computations, e.g. the Jacobi and Gauss-Seidel methods, Jordan elimination, etc.

In fact, the elimination algorithm has popped up in a surprising variety of fields, making it one of the most important algorithms in science and engineering. In the early 70s Bertele and Brioschi [10, 11, 12] started using vertex elimination to solve combinatorial optimization problems via dynamic programming [9]. In relation database theory, similar methods are used to improve the efficiency of query processing, see Beeri et al. [8], Fagin et al. [62], Goodman and Shmueli [86].

In parallel, many interesting developments happened in the constraint-satisfaction literature. In particular, Montanari [148] and [73] pointed out that a constraint-satisfaction problems (CSP) involving only binary constraints could be represented by a graph. The connection with linear algebra was made by Seidel [172], who derived an elimination procedure for binary constraint networks which he called

the "invasion procedure". The same algorithm was later rediscovered by Dechter and Pearl [45] under the name "adaptive consistency".

In the AI community the development of expert systems spurred the development of probabilistic reasoning systems, and, in a landmark paper, Pearl [162] developed belief propagation in trees. The tractability of probabilistic inference in trees was earlier noted by Kelly and Barclay [115] and also by Cannings et al. [20]. The latter paper also developed the *peeling algorithm*, which is essentially variable elimination on an arbitrary graph. Pure elimination algorithms for belief nets, i.e., without a belief propagation metaphor, were developed by D'Ambrosio [37] and Zhang and Poole [207], and unified with similar algorithms in the CSP literature under the name *bucket elimination* by Dechter [42, 43, 44].

Tanner [183] introduced the use of bipartite graphs to describe low-density parity-check codes, which were subsequently named *Tanner graphs* in the literature. Later, Wiberg et al. [199, 198] rediscovered Tanner's work and extended it to include (hidden) state variables [66]. Frey et al. then later built upon the work by Wiberg [198] and introduced *factor graphs* [74, 125, 124], a generalization of Tanner graphs where the "factors" can now be arbitrary functions. Kschischang et al. [125] show how the sum-product algorithm on factor graphs can be applied to a wide variety of other settings, including behavioral modeling, linear codes, trellises and state-space models, Markov chains, and hidden Markov models; the latter yielding both the Viterbi algorithm and the Kalman filter as special cases.

In robotics, it was noted by Thrun et al. [188] and others that the information matrix Λ is the matrix of a Markov random field associated with the SLAM problem. The objective function in SLAM corresponds to a pairwise Markov random field (MRF) [201, 204] through the Hammersley-Clifford theorem [201], which associates cliques in the MRF with potentials. Many more connections between factor graphs, MRFs and inference problems in robotics were made by us in [49, 46, 123, 48] and expanded upon here.

4

Elimination Ordering

Insight into the graphs underlying robotics inference, and how their sparsity is affected by the implementation choices we make, is crucial for achieving highly performant algorithms. In this section we discuss elimination ordering strategies and their effect on performance. This will also allows us, in Section 5, to understand the effects of marginalizing out variables, and its possibly deleterious effect on sparsity, especially in the SLAM case. Other inference problems in robotics benefit from only keeping track of the most recent state estimate; this leads to filtering and/or fixed-lag smoothing algorithms.

4.1 Complexity of Elimination

Let us examine the computational complexity of the elimination algorithm 3.1 on page 35. Since we eliminate n variables, the cost is

$$f(\Phi_{1:n}) = \sum_{j=1}^{n} g(\Phi_{j:n}, x_j), \tag{4.1}$$

where $g(\Phi_{j:n}, x_j)$ is the cost of eliminating variable x_j from the remaining graph $\Phi_{j:n}$. Elimination is a general algorithm to transform a factor graph into a Bayes net that encodes the posterior, and hence

this cost heavily depends on the application. However, for the case of sparse matrix factorization, the cost of Algorithm 3.2 is dominated by the partial QR factorization step (Section 3.3.3). The main cost in QR is incurred by the Householder reflections, which—without going into detail—take $4m_k n_k$ flops when applied to an $m_k \times n_k$ matrix. We need one Householder reflection for every dimension k in x_j, hence

$$g(\Phi_{j:n}, x_j) \approx \sum_{k=1}^{n_j} 4m_k n_k = \sum_{k=1}^{n_j} 4(m_j - k)(n_j + s_j + 1 - k), \qquad (4.2)$$

with n_j the dimension of the variable x_j to be eliminated, s_j the size of the separator S_j, and m_j the number of rows in the augmented matrix $[\bar{A}_j|\bar{b}_j]$ of the product factor $\psi(x_j, S_j)$.

The rather involved calculation above simplifies for a dense, scalar $m \times n$ matrix to the well-known complexity of dense QR [84]:

$$f(\Phi_{1:n}) = \sum_{k=1}^{n-1} 4(m-k)(n+1-k) = 2(m - n/3)n^2 + O(mn). \quad (4.3)$$

However, for the sparse, multifrontal algorithm the flop count will be *much* lower than this number. In addition, the multifrontal method can make use of Level-3 BLAS methods that exploit cache coherence or even thread parallelism for larger elimination steps [39].

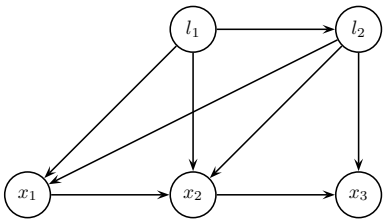
Example. For our running SLAM example, for which the multifrontal QR algorithm is shown in Figures 3.6 and 3.7, we have

$$f(\Phi_{1:5}) = 32 + 20 + 488 + 488 + 128 = 1156 \text{ flops} \qquad (4.4)$$

for the elimination order $\alpha = \{l_1, l_2, x_1, x_2, x_3\}$, and where we assumed $m_{l_1} = m_{l_2} = 2$ (as with, for example, bearing-range measurements). Using the same elimination order for a dense matrix would have yielded

$$f(\Phi_{1:5}) = 1752 + 1304 + 1256 + 608 + 152 = 5072 \text{ flops}. \qquad (4.5)$$

The dramatic improvement in each elimination step is because of the smaller number of rows m_j and columns $n_j + s_j + 1$ involved.



$$\begin{array}{c} \begin{array}{ccccc} x_3 & x_2 & x_1 & l_2 & l_1 \end{array} \\ \left[\begin{array}{ccccc} R_1 & T_{12} & & T_{14} & \\ & R_2 & T_{23} & T_{24} & T_{25} \\ & & R_3 & T_{34} & T_{35} \\ & & & R_4 & T_{45} \\ & & & & R_5 \end{array}\right] \end{array}$$

Figure 4.1: Bayes net resulting from using the ordering x_3, x_2, x_1, l_2, l_1 for the robotics example (the reverse ordering from Figure 3.5 on page 36).

4.2 Variable Ordering Matters

The flop count for sparse factorization will be much lower than for a dense matrix, but can vary dramatically for different elimination orderings. While any order will ultimately produce an identical MAP estimate, the order in which variables are eliminated matters, as different orderings lead to Bayes nets with different topologies. This will in turn affect the computational complexity of the elimination algorithm, as the sizes s_j of the separators at each step are affected. To illustrate, let us look at two examples below.

Example 1. In our toy example, the result of using the reverse ordering from before is shown in Figure 4.1. Note that the resulting Bayes net is almost fully dense: only the first variable eliminated, x_3, is not fully connected to the variables later in the ordering. This figure should be compared with the final elimination stage of Figure 3.7 on page 41.

Example 2. A more realistic example shows what is really at stake. To this end, recall the larger simulation example, with its factor graph shown in Figure 2.1 on page 18. The sparsity patterns for the corre-

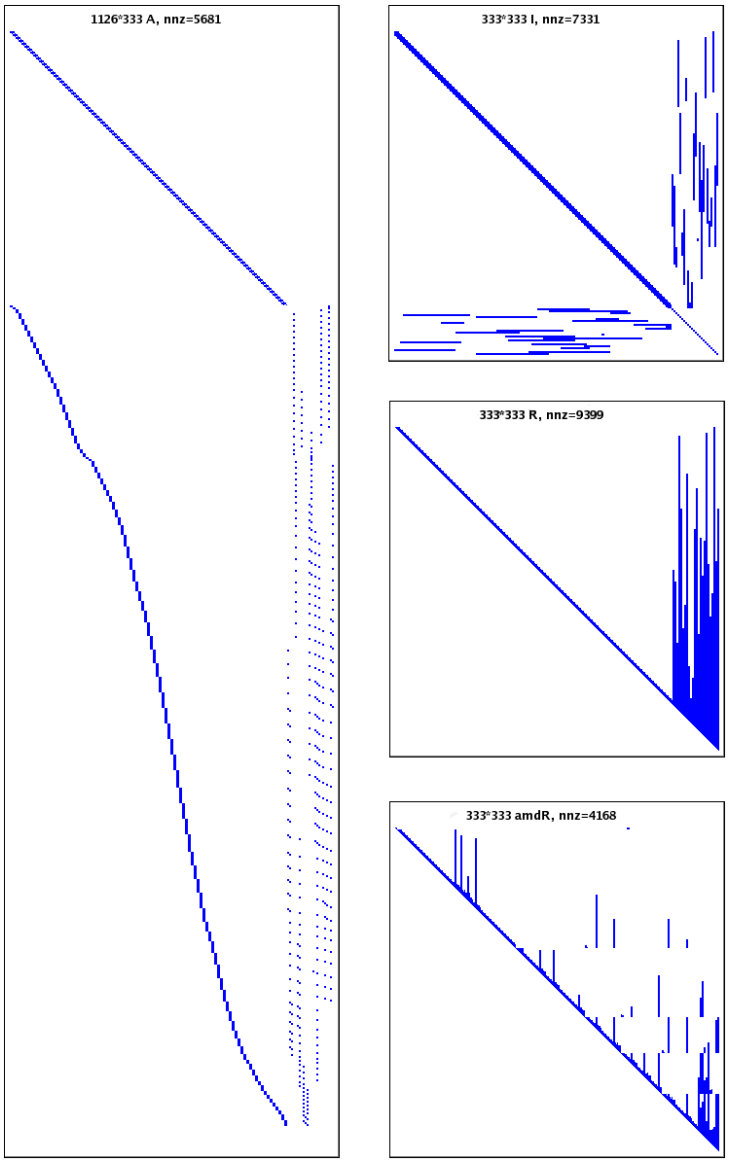


Figure 4.2: On the left, the measurement Jacobian A associated with the problem in Figure 2.1, which has $3 \times 95 + 2 \times 24 = 333$ unknowns. The number of rows, 1126, is equal to the number of (scalar) measurements. Also given is the number of non-zero entries "nnz". On the right: (top) the information matrix $\Lambda \triangleq A^\top A$; (middle) its upper triangular Cholesky triangle R; (bottom) an alternative factor $amdR$ obtained with a better variable ordering (COLAMD).

sponding sparse Jacobian matrix A is shown in Figure 4.2. Also shown is the pattern for the information matrix $\Lambda \triangleq A^\top A$, in the top-right corner.

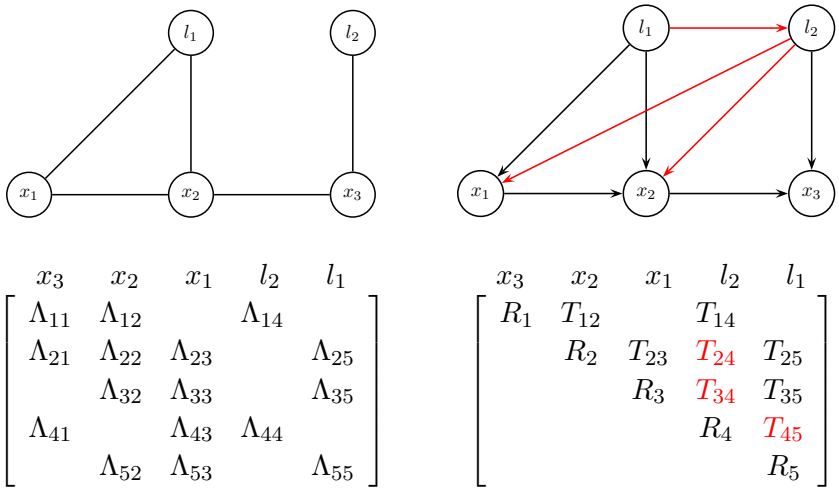
On the right of Figure 4.2, we show the resulting upper triangular Choleksy factor R for two different orderings. Both of them are sparse, and both of them satisfy $R^\top R = A^\top A$ (up to a permutation of the variables), but they differ in the amount of sparsity they exhibit. It is exactly this that will determine how expensive it is to factorize A. The first version of the ordering comes naturally: it eliminates the poses first, and then the landmarks, leading to a sparse R factor with 9399 non-zeros. In contrast, the sparse factor R in the bottom-right was obtained by reordering the variables according to the COLAMD heuristic (Section 4.4.1 below) and only has 4168 non-zeros. Yet back-substitution gives exactly the same solution for both versions.

4.3 The Concept of Fill-in

Different elimination orderings influence complexity by giving rise to different separator sizes throughout the elimination process. Larger separator sizes are the result of **fill-in**, the creation of dependencies in the sparse graphs between variables that were previously independent of each other. Hence, it is natural that we would want to both characterize this phenomenon and find ways to minimize it.

Formally, we define fill-in by referring to the structure of the undirected Markov random field G associated with the information matrix Λ (see Section 3.1.3), and the directed Bayes net associated with the Cholesky factor R. In general, the structure of R above the diagonal is identical to that of $\Lambda = A^\top A = R^\top R$, except for fill-in with non-zeros in some (or all) places. In graphical terms, these are represented by directed edges that were not present as undirected edges in G.

Example. Figure 4.3 shows how an unfavorable elimination order yields three extra edges in the resulting Bayes net on the right, highlighted in red, that were not present in G on the left. Also shown are the information matrix Λ and the Cholesky factor R, with the fill-in blocks colored red, as well. Note that the sparsity of the information



$$\begin{array}{c} \begin{matrix} x_3 & x_2 & x_1 & l_2 & l_1 \end{matrix} \\ \begin{bmatrix} \Lambda_{11} & \Lambda_{12} & & \Lambda_{14} & \\ \Lambda_{21} & \Lambda_{22} & \Lambda_{23} & & \Lambda_{25} \\ & \Lambda_{32} & \Lambda_{33} & & \Lambda_{35} \\ \Lambda_{41} & & \Lambda_{43} & \Lambda_{44} & \\ & \Lambda_{52} & \Lambda_{53} & & \Lambda_{55} \end{bmatrix} \end{array} \qquad \begin{array}{c} \begin{matrix} x_3 & x_2 & x_1 & l_2 & l_1 \end{matrix} \\ \begin{bmatrix} R_1 & T_{12} & & T_{14} & \\ & R_2 & T_{23} & T_{24} & T_{25} \\ & & R_3 & T_{34} & T_{35} \\ & & & R_4 & T_{45} \\ & & & & R_5 \end{bmatrix} \end{array}$$

Figure 4.3: Fill-in occurs when the DAG resulting from elimination into R has extra edges with respect to the Markov random field G corresponding to Λ.

matrix Λ is *not* affected by re-ordering the columns and rows (compare with Figure 3.3 on page 33). In contrast, the matrix R has three extra non-zero blocks as compared to the last step in Figure 3.7 on page 41.

4.4 Ordering Heuristics

An elimination ordering with minimum fill-in minimizes the cost of the elimination/factorization algorithm, but finding it is NP-hard. Fortunately, many useful heuristics have been developed to approximate an optimal ordering.

4.4.1 Minimum Degree Orderings

The two most widely used sparse matrix ordering algorithms for scientific computation on standard desktop machines are based on the heuristic of first eliminating the least constrained variables of G. This family of algorithms is known as the **minimum degree** algorithms.

A first approach is to eliminate all variables of minimal degree in one call of the elimination function, known as multiple elimination or minimum degree MMD. In addition *indistinguishable* nodes are elim-

inated, whose elimination will not introduce additional dependencies as they are subsumed by another elimination performed in the same step. MMD saves time both on updating the graph and determining the next elimination candidates.

The second approach avoids computing the exact vertex degrees when eliminating one or more variables, by collecting nodes into cliques. Only the degrees for the cliques are calculated; an approximate bound on the degree of the remaining vertices can be kept up to date relatively cheaply. This algorithm is known as the **Approximate Minimum Degree** or AMD method.

4.4.2 Nested Dissection Orderings

A fundamentally different approach to reordering is to apply a *divide-and-conquer* paradigm. This is feasible as eliminating a node only induces new constraints for a set of (spatially) direct neighbors. **Nested dissection** (ND) algorithms try to exploit this by recursively partitioning the graph and returning a post-fix notation of the partitioning tree as the ordering.

In nested dissection algorithms the size of the *separators* are of central importance. To bound the complexity of the factorization when using an ND ordering, the $f(n)$-*separator theorem* is key, which is a statement about a *class* of graphs S:

Theorem 1. *($f(n)$-Separator Theorem)* There exist constants $\alpha < 1$ and $\beta > 0$ such that if G is any n-vertex graph in S, the vertices of G can be partitioned into three sets A, B, C in a way that no edge joins a vertex in A with one in B, A nor B contains more than αn vertices, and C contains no more than $\beta f(n)$ vertices.

For all classes of graphs S for which a $f(n)$-theorem holds, an efficient *divide-and-conquer* ordering can be found [136]. Note that one needs to guarantee that the algorithms for the graph partitioning and local subgraph elimination are less complex than the factorization. The two most important results are:

- For chain-like graphs we can recursively find constant separators, and the resulting factorization will cost $O(n)$ flops, i.e., is *linear*

in the number of variables. In linear algebra, this corresponds to the well-known fact that inverting/factorizing a band-diagonal matrix is linear.

- For planar graphs we can recursively find separators of size less than $2\sqrt{2n}$, and the resulting factorization will cost $O(n^{1.5})$ flops.

The latter result immediately leads to the basic nested dissection method—as developed for planar graphs. It is given below in Algorithm *4.1*.

Algorithm 4.1 Nested Dissection for Planar Graphs

Let $G = (V, E)$ be a graph with n vertices V and a set of edges E.

1. Partition G into subgraphs A, B and C, with $|A|, |B| \leq \frac{2}{3}n$ and $|C| \leq 2\sqrt{2}\sqrt{n}$
2. Repeat *Step (1)* until $|A|, |B| \leq \epsilon$ or $|A|, |B| = 1$
3. Obtain the ordering by putting the binary tree of the recursive partitioning in post-order, with the nodes of the separating set C last for every triple of sets.

In practice, a graph partitioning algorithm like METIS [114] is used. Most algorithms use a two step approach for determining the separators in the graph. First, they try to find good areas for a cut that preserve the balance between the induced subgraphs. Second, a refinement algorithm like [116] or [65] is applied. These algorithms can be understood as variants of bipartite graph matching algorithms as they try to find the minimal cut between a set of nodes.

4.5 Ordering Heuristics in Robotics

In SLAM, a naive ordering strategy is to eliminate the landmarks first, and then the poses. This is often called the "Schur-complement trick", because when the elimination is written down at the block level the

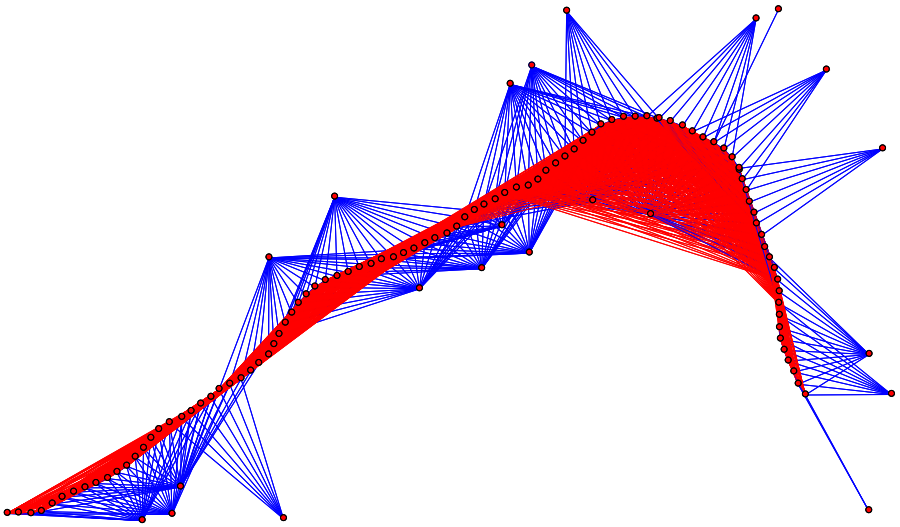


Figure 4.4: The so-called "Schur complement trick" eliminates all landmarks first, but this leads to a very connected graph on the robot poses. Doing the reverse (poses first) would similarly lead to a fully connected clique of landmarks.

corresponding linear algebra operation involves taking the Schur complement of a matrix. Indeed, if we reorder the columns of the Jacobian A such that landmarks come first, we can split the matrix A into a block F corresponding to landmark unknowns, and a block G corresponding to poses (or cameras, in the structure from motion case). Block elimination then yields

$$A = \begin{bmatrix} F & G \end{bmatrix} \tag{4.6}$$

$$\rightarrow \quad A^\top A = \begin{bmatrix} F^\top F & F^\top G \\ G^\top F & G^\top G \end{bmatrix} \tag{4.7}$$

$$\rightarrow \quad R = \begin{bmatrix} F^\top F & F^\top G \\ 0 & G^\top G - G^\top F \left(F^\top F\right)^{-1} F^\top G \end{bmatrix}, \tag{4.8}$$

where the matrix $F^\top F$ is block-diagonal, and the lower-right block in R is known as the Schur complement of $F^\top F$. Its inverse is the covariance matrix on the poses/cameras, which is why it is also known as the "reduced camera matrix".

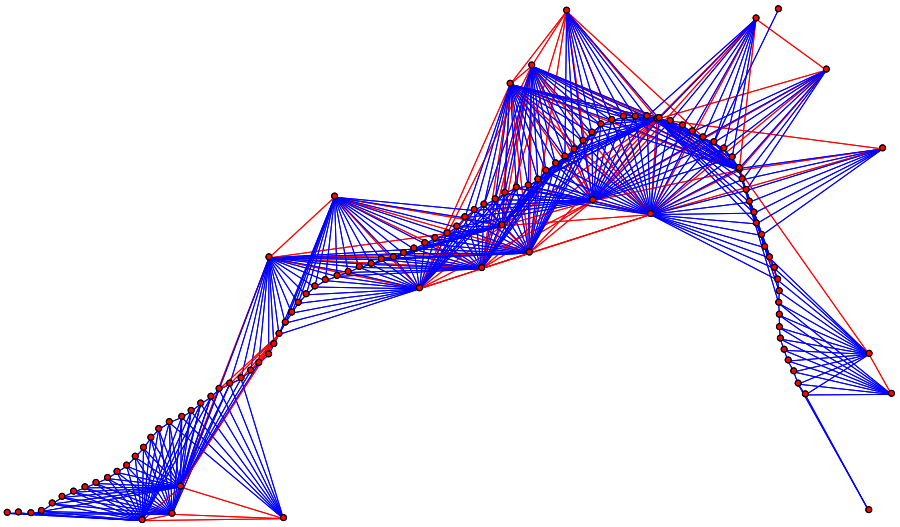
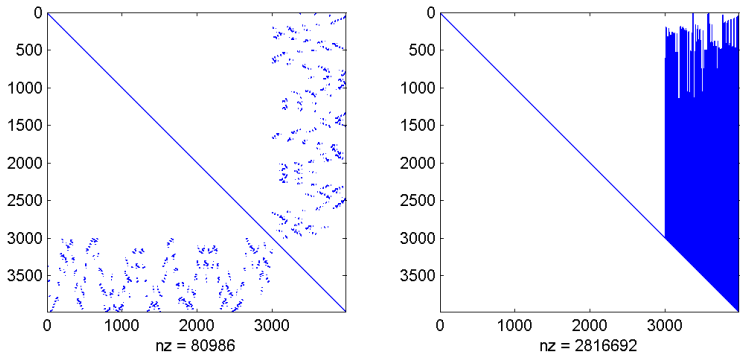


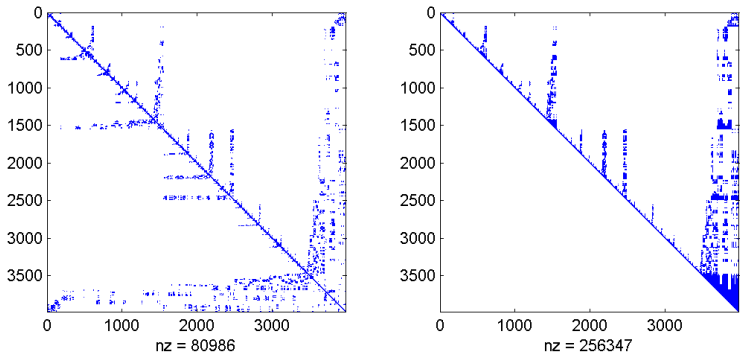
Figure 4.5: In contrast, an approximate minimum degree ordering (using COLAMD) leads to much less fill-in and considerably faster linear solve steps.

However, "natural" orderings such as the Schur complement trick, with either landmarks or poses eliminated first, can be quite expensive as compared to using better heuristics. In structure from motion, with possibly hundreds of thousands of points, it can be advantageous to use the Schur complement trick as it is beneficial in terms of cache coherence and locality. But in typical robotics applications even a domain-independent heuristic such as AMD to eliminate the variables in SLAM yields substantial computational wins.

Example. Figures 4.4 and 4.5 illustrate this for the simulated SLAM example from Figure 2.1 on page 18. In Figure 4.4 the landmarks were eliminated first, leading to a densely connected graph on the robot poses. Doing the reverse (poses first) would lead to a *fully* connected clique of landmarks. In contrast, in Figure 4.5, an approximate minimum degree ordering leads to much less fill-in and considerably faster linear solve steps. Figures 4.6a and 4.6b show the same again, but now in the form of the matrix sparsity patterns. Note the dense fill-in on the right, linking the entire trajectory to all landmarks. Reordering of columns (unknowns) does not affect the sparseness of Λ,



(a) Original information matrix Λ and its Cholesky triangle.



(b) Information matrix Λ after reordering and its Cholesky triangle.

Figure 4.6: Comparing the information matrix and the Cholesky factor for the two different orderings from Figures 4.4 and 4.5; nz = number of non-zeros.

but the number of non-zeros in R has dropped from approximately 2.8 million to about 250 thousand.

Because finding an optimal ordering is NP-complete, any piece of domain-specific information can help a great deal. The above domain agnostic ordering heuristics go a long way towards minimizing fill-in, and these methods are built into packages such as MATLAB. However, when solving a sparse inference problem in MATLAB you are doing so at the *scalar* level. A very simple domain-specific heuristic is the semantic information from the true factor graph at the level of poses and landmarks, *not* their scalar components. Figure 4.7 shows a further

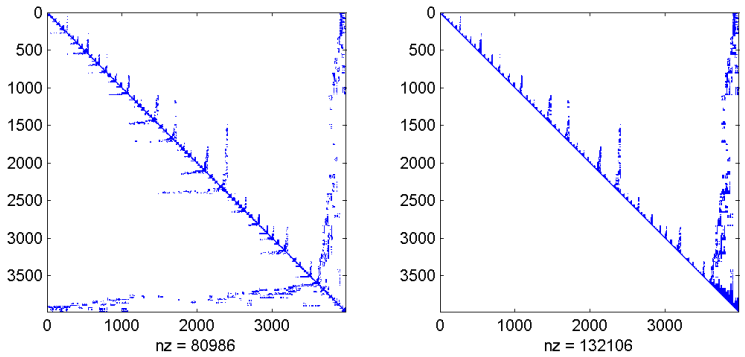


Figure 4.7: By reordering while taking into account the special block structure of the SLAM problem, the non-zero count can be reduced even further, to about 130K, a reduction by a factor of 20 with respect to the original R.

improvement by a factor of two by simply applying AMD at the block level vs. the scalar level.

4.6 Nested Dissection and SLAM

The application of nested dissection orderings is especially relevant to SLAM, as environments mapped by mobile robots often contain parts that are spatially separated. A divide-and-conquer scheme is one of the most promising ways to solve challenging SLAM problems, especially in large-scale environments. One advantage of submap based approaches is that the computation can be done in an out-of-core manner, making it possible to distribute most of the work over multiple computation resources, increasing the scalability in terms of both time and memory.

Another practical advantage of a divide-and-conquer approach is that it also leads to a good initialization scheme for batch optimization approaches, which is one of the most crucial issues in nonlinear optimization. By employing a divide-and-conquer approach, we can recursively compute the initializations from the optimized submaps.

Many of these ideas were implemented in the tectonic smoothing and mapping (TSAM) algorithm [154, 152], which combines nested dissection and careful initialization. Some partitioning results from that

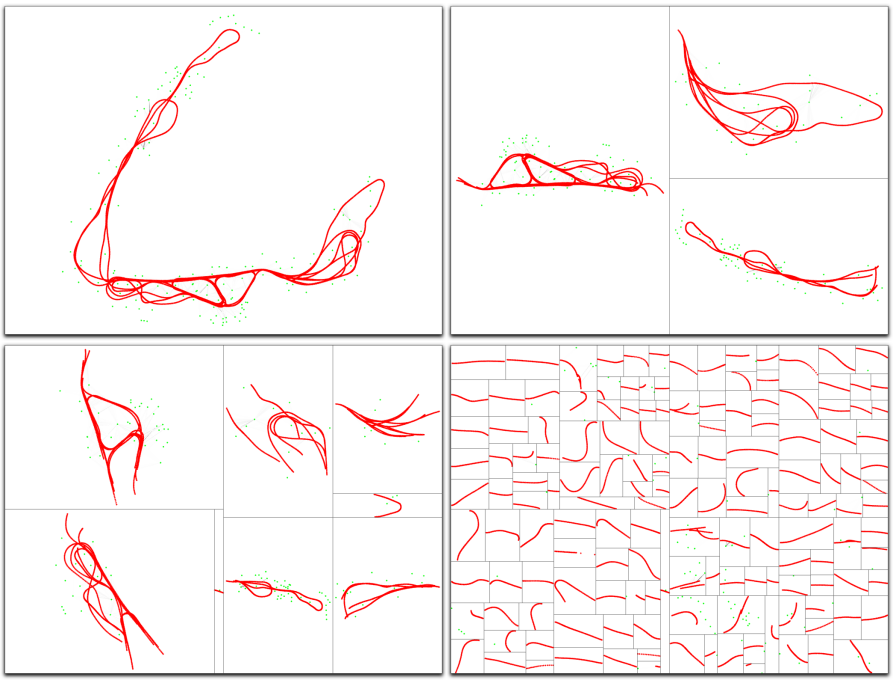
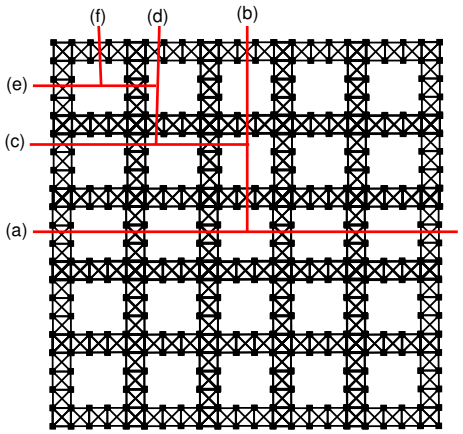


Figure 4.8: A nested dissection ordering leads to a hierarchical decomposition of a SLAM problem, here illustrated on the Victoria park dataset. The figure shows 4 levels of partitioning, leading to an efficient elimination ordering [152].

work are reproduced in Figure 4.8, which shows a recursive partitioning of the well-known Victoria park dataset. TSAM2 uses a combination of the METIS package to find a nested dissection ordering at the global level and then orders the resulting subgraphs locally using AMD.

If the factor graph arising from a SLAM problem can be embedded in a planar graph, nested dissection leads to a provably optimal $O(n^{1.5})$ computation bound, as opposed to the $O(n^3)$ complexity that arises when we repeatedly have to factorize a dense matrix. Figure 4.9 illustrates the partitioning process in a simulated block-world, designed to mimic an indoor or urban scenario with lots of occlusion. The partitionings from (a) to (f) correspond to the cuts of the first depth-first partitioning recursion. For these types of environments, a planar embedding of the graph with $O(\sqrt{n})$ separators can be easily



	n	$\sqrt{n}$	$\lvert A\rvert, \lvert B\rvert$	$\lvert C\rvert$
(a)	25,000	158.1	12446	108
(b)	12446	111.5	6196	54
(c)	6196	78.7	3071	54
(d)	3071	55.4	1518	36
(e)	1518	38.9	741	36
(f)	741	27.2	362	18

Figure 4.9: Simulated block world example that has a global planar graph structure. The table shows the level of the ND recursion, the number of nodes n in the each partition, $\sqrt{n}$, the size of the next partitions and the separator size.

found. To illustrate the $\sqrt{n}$ separator theorem at work in these types of environments, the table in Figure 4.9 shows an example of the relevant partition and separator sizes for a block-world with 25,000 nodes.

4.7 Bibliographic Remarks

Graph-based representations became especially popular in the finite element community, where the focus was on reducing the "bandwidth" of the typically very *sparse* stiffness matrices in order to speed up computations [169, 36]. Finding an elimination ordering with minimum fill is NP-hard, as has long been known in the linear algebra [203, 17] and scientific computation communities [114, 97, 93]. The MMD

ordering heuristic was introduced in [137], and the algorithm known as COLAMD in [3, 40]. Both AMD and MMD produce equally good orderings, but AMD is faster, and it has become the de facto standard.

The nested dissection algorithm was introduced by George [78], but the seminal work on the complexity of the methods was by Lipton and Tarjan [136, 135], who introduces the $f(n)$-*separator theorem.* Improvements focus on finding the separating subgraphs more efficiently, e.g., using spectral analysis [166], partially applied MD-orderings as indicators for good partitions [138], and k-way partitioning methods [114].

The importance of sparsity and variable ordering in robotics was first discussed in [46, 48], and before that in bundle adjustment in [191]. Agarwal and Olson [2] investigated the performance of several ordering heuristics on a sample of SLAM problems, and found that COLAMD and METIS [114] are best overall performers. The Schur complement trick is an often-used technique in computer vision [191, 96].

The divide-and-conquer scheme has been well studied in the SLAM community. As early as 1976, Brown [18] first employed the submap scheme in the aero-triangulation and mapping of city-scale areas. A recursive partitioning is used to exploit the band diagonal structure of the linear system in the project, and no nonlinearity is considered. The submap idea for SLAM problems was also investigated in hierarchical SLAM by Estrada et al. [60] with a filtering-based local map building. Also related is the multi-level relaxation by Frese et al. [71]. Ni et al. [154] introduced tectonic smoothing and mapping (TSAM), which is a two-level submap-based approach based on factor graphs. Paz et al. [161] improved the work in [60] by fusing the local maps in a hierarchical way, which has a nested dissection flavor. A similarly hierarchical approach was taken in HOG-Man [90] and in a generalization by Grisetti et al. [89].

The application of ND orderings in SLAM was introduced in [123], where ND is shown to be optimal for SLAM problems that can be embedded in a planar graph, a feature of many urban datasets. A fully recursive, ND-based approach to large-scale SLAM problems was introduced as TSAM2 by Ni and Dellaert [152]. This work was extended to handle structure from motion problems in [153].

5

Incremental Smoothing and Mapping

In previous sections we have discussed efficient batch optimization algorithms under the assumption that all the data is available in advance. However, many inference problems in robotics are **incremental**, meaning that measurements arrive as a **temporal sequence**, and intermediate solutions are needed. It is natural to question whether we can reuse previous computations or if we have to perform a full batch optimization each time. Below we will see that reuse is possible, leading to efficient incremental algorithms. We first discuss the linear case, where we can incrementally update a matrix factorization, using well-known linear algebra techniques. To extend to nonlinear systems we return to graphical models. Here we describe inference using matrix factorization as operations on graphical models, introducing the Bayes tree. We then use the Bayes tree to obtain a fully incremental nonlinear inference algorithm. Finally, marginalization leads to special cases of incremental inference, namely filtering and fixed-lag smoothing.

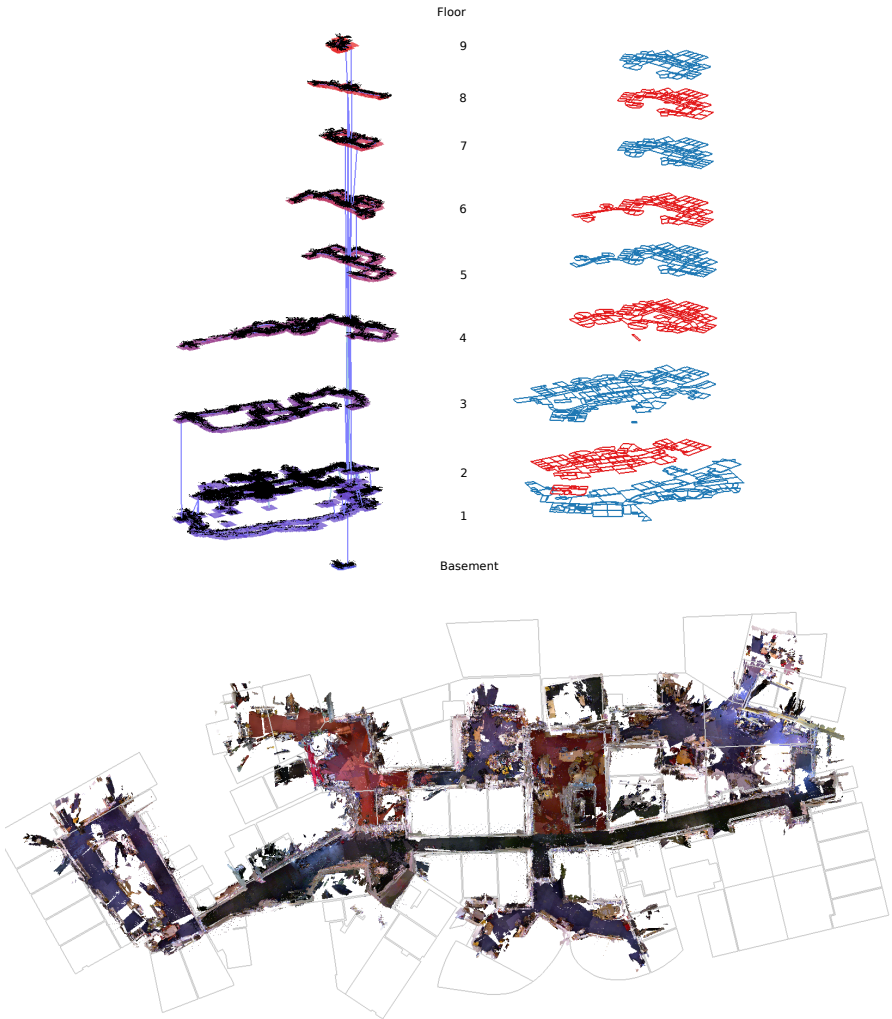


Figure 5.1: (top left) A map of ten floors of the MIT Stata Center created using a reduced pose graph [104] in combination with a real-time visual SLAM system. The data used was collected in 14 sessions spanning a six-month period. The total operation time was nine hours and the distance traveled was 11km. Elevator transitions are shown as vertical blue lines. The view is orthographic and the vertical axis has been exaggerated to make it easier to see each floor. The 2nd floor is approximately 90m across. (top right) Floor plans for each of the floors that were mapped. (bottom) Map for one of the sessions covering the second floor of the MIT Stata Center with ground truth floor plan added for reference. The dense point cloud consists of RGB-D frames rendered at the solution of the pose graph. The pose graph itself is derived from sparse stereo point features.

5.1 Incremental Inference

We use a SLAM example to motivate the need for incremental inference. Imagine a robot operating in a building without a priori map. While the robot is traversing rooms and corridors, a constant stream of sensor data is generated. Processing the sensor data yields a sequence of optimization problems that constantly grow in size: each new measurement adds a new probabilistic constraint, while those induced by previous measurements remain unchanged. Repeatedly solving the optimization problem after adding each measurement is essential: both because the robot needs the best possible knowledge of the world given its sensor measurements, but also because the current solution provides an initialization point for the next optimization. The latter is essential for nonlinear problems to not get stuck in local minima.

While the problem is small initially and can easily be solved repeatedly, the time to solve grows over time to the point that real-time batch processing is no longer feasible. Figure 5.1 serves as an example: a map generated from multiple sessions of a robot operating in a building, totaling nine hours of operation time. Even though an approximation is used to significantly reduce the size of the optimization problem (via reduced pose graphs, see [104]) repeated batch optimization soon becomes too expensive. Instead, incremental updates that make use of previous calculations are essential.

5.2 Updating a Matrix Factorization

In an incremental setting, we want to *update* the most recent matrix factorization with the new measurements, to reuse the computation that already incorporated all previous measurements. In the linear case, this is possible through incremental factorization methods.

Recall that the linearization of a nonlinear objective function yields the least-squares problem of Equation 2.13, repeated below,

$$\Delta^* = \underset{\Delta}{\operatorname{argmin}} \sum_i \|A\Delta - b\|_2^2 \,, \tag{5.1}$$

where $\Delta \triangleq X - X^0$ is the state update vector, and A and b represent

the sparse linearized system in Δ. Using Householder QR factorization, as in Equation 2.19, we can rewrite the objective as:

$$\|A\Delta - b\|_2^2 = \|R\Delta - d\|_2^2 + c, \tag{5.2}$$

where c is a constant that reflects the sum of squared residuals of the least-squares problem.

When a new measurement arrives, instead of updating and refactoring a new system $A'|b'$ from scratch, we have the option to modify the previous factorization directly by **QR-updating**. Let us assume that A' is formed by adding a single new measurement row $a^\top$ with corresponding RHS element β, i.e.,

$$A' = \begin{bmatrix} A \\ a^\top \end{bmatrix} \qquad b' = \begin{bmatrix} b \\ \beta \end{bmatrix}. \tag{5.3}$$

Then a QR update proceeds as follows: adding $a^\top \in \mathbb{R}^n$ to the previous factor $R \in \mathbb{R}^{n \times n}$ and the new scalar element $\beta \in \mathbb{R}$ to the previous RHS d yields a new system $R_a|d_a$ that is not yet in the correct factorized form:

$$R_a = \begin{bmatrix} R \\ a^\top \end{bmatrix} = \begin{bmatrix} Q^\top & \\ & 1 \end{bmatrix} \begin{bmatrix} A \\ a^\top \end{bmatrix} \qquad d_a = \begin{bmatrix} d \\ \beta \end{bmatrix}. \tag{5.4}$$

To bring this new system into the right form, a sequence of **Givens rotations** can be determined that zero out the newly added row on the left. While a full QR factorization uses Householder reflections to zero out *columns*, Givens rotations are more appropriate for updating, as only a sparse set of entries in a single *row* need to be zeroed out.

Updating a matrix factorization using Givens rotations is numerically stable and accurate to machine precision if the rotations are properly formed as described in Golub and Loan [84, Sec. 5.1]. The updating process starts from the left-most non-zero entry below the diagonal (row i and column j with $i > j$) by applying the Givens rotation

$$G \triangleq \begin{bmatrix} \cos\phi & \sin\phi \\ -\sin\phi & \cos\phi \end{bmatrix} \tag{5.5}$$

as shown in Figure 5.2. The parameter ϕ is chosen so that the $(i,j)^{th}$ entry of A becomes 0.

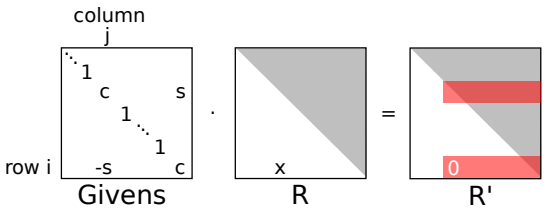


Figure 5.2: Using a Givens rotation as a step in transforming a (sparse) matrix into upper triangular form. The entry marked 'x' is eliminated, changing some of the entries marked in red (dark), depending on sparsity.

A series of Givens rotations are applied to zero out the new row, starting from the left-most nonzero entry (i, j_1) and resulting in an upper triangular matrix that contains the updated factor R'

$$G_{j_k} \dots G_{j_2} G_{j_1} R_a = \begin{bmatrix} R' \\ 0 \end{bmatrix}. \tag{5.6}$$

Note that Givens rotations can introduce additional non-zero entries in the new row so that k might be larger than the number of nonzero entries in a. The RHS vector d_a is updated with the same sequence of rotations to obtain d'.

After all is said and done, the incremental factorization is equivalent to rewriting the updated objective without re-factoring A, as desired:

$$\|R'\Delta - d'\|_2^2 = \|A\Delta - b\|_2^2 + \|a\Delta - \beta\|_2^2 + c', \tag{5.7}$$

where c' is the updated sum of squared residuals from Equation 5.2.

Several steps of this update process for an exploration task are illustrated in Figure 5.3. New variables are added to the QR factorization by expanding the factor R by the appropriate number of empty columns and rows. This expansion is simply done before new measurement rows containing the new variables are added. At the same time, the RHS d is augmented by the same number of new rows. For more general cases, including loop closure, more advanced methods are needed to avoid fill-in as we will later see in Section 5.4.3.

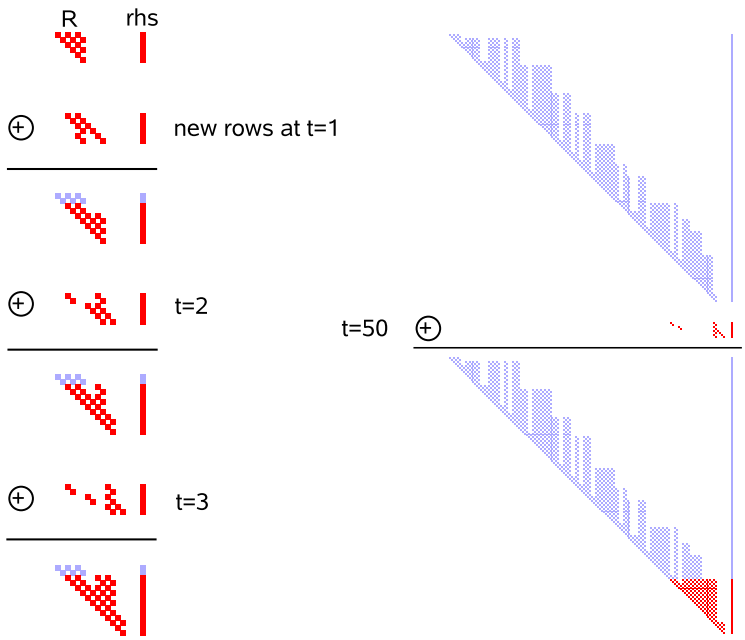


Figure 5.3: Incrementally updating the square-root information matrix for an exploration task: new measurement rows are added to the upper triangular factor R and the right-hand side (RHS). The left column shows the updates for the first three steps, the right column shows the update after 50 steps. The update operation is symbolically denoted by $\oplus$. Entries that remain unchanged are shown in light blue.

5.3 Kalman Filtering and Smoothing

The Kalman filter and smoother [145] are special cases of incrementally updating a linear system. For simplicity, let us consider the simplest possible factor graph structure—a simple chain—which is most often encountered in localization tasks, or for example in pose graph SLAM in the absence of loop-closure detection. To make the connection with the Kalman smoother we first describe the concept of marginalization in factor graphs, and then discuss two popular methods in robotics that are based on marginalization: fixed-lag smoothing and filtering.

5.3.1 Marginalization

Even with incremental updating, memory usage and computation are still unbounded in time; one solution is to remove older variables without removing information, a process that is called **marginalization**. Stated in terms of probability densities, if we have a joint density $p(x, y)$ over two variables x and y, then marginalizing out the variable x corresponds to integrating over x, i.e.,

$$p(y) = \int_x p(x, y), \tag{5.8}$$

resulting in a density $p(y)$ over the remaining variable y.

If the density $p(x, y)$ is Gaussian and given in covariance form with mean μ and covariance Σ, partitioned as follows:

$$p(x, y) = \mathcal{N}(\begin{bmatrix} \mu_x \\ \mu_y \end{bmatrix}, \begin{bmatrix} \Sigma_{xx} & \Sigma_{xy} \\ \Sigma_{xy}^\top & \Sigma_{yy} \end{bmatrix}), \tag{5.9}$$

marginalization is simple, as the corresponding sub-block Σ_{yy} already contains the covariance on y after marginalizing out x, i.e.,

$$p(y) = \mathcal{N}(\mu_y, \Sigma_{yy}). \tag{5.10}$$

In contrast, if the density is given in information form with information vector η and information matrix Λ, partitioned as follows:

$$p(x, y) = \mathcal{N}(\Lambda^{-1} \begin{bmatrix} \eta_x \\ \eta_y \end{bmatrix}, \begin{bmatrix} \Lambda_{xx} & \Lambda_{xy} \\ \Lambda_{xy}^\top & \Lambda_{yy} \end{bmatrix}^{-1}), \tag{5.11}$$

the information matrix for y after marginalization is given by the **Schur complement** of Λ_{xx} in the matrix Λ, i.e., $\Lambda_{yy} - \Lambda_{xy}^\top \Lambda_{xx}^{-1} \Lambda_{xy}$.

Variable elimination, or equivalently matrix factorization, represents an intermediate step between the covariance and information forms: the factored or square-root information form. Hence, the joint density is given as

$$p(x, y) = \mathcal{N}(R^{-1}d, R^{-1}R^{-\top}), \tag{5.12}$$

where

$$R = \begin{bmatrix} R_{xx} & S_{xy} \\ 0 & R_{yy} \end{bmatrix} \qquad d = \begin{bmatrix} d_x \\ d_y \end{bmatrix}. \tag{5.13}$$

In this factored form, marginalization can be as easy as in the covariance form, but *only* for a contiguous block of variables x that was eliminated first. In matrix terms, marginalization is performed by removing the first n_x rows and columns of the square root information matrix, where n_x is the number of variables in x, yielding R_{yy} as the new square-root information matrix on y, along with RHS d_y.

In summary, whether we can easily marginalize or not depends:

- Covariance form: always easy, select rows and columns from Σ.
- Information form: always hard, involves Schur complement on Λ.
- Square-root information form: depends on the variable ordering.

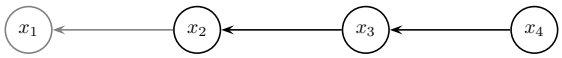
In terms of graphical models, remember that the matrix R corresponds to a directed acyclic graph with a topological order that follows the chosen variable ordering. Hence, any variable that does not point to other variables can always be marginalized out without any computation, and the equivalent also holds for groups of variables. Below we will see how this simple graphical way of thinking explains fixed-lag Kalman smoothing and filtering in an intuitive way.

5.3.2 Fixed-lag Smoothing and Filtering

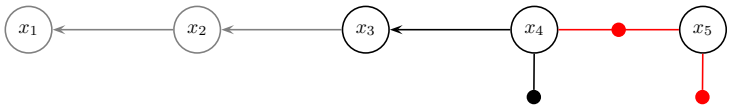
The **fixed-lag smoother** is an iterative algorithm that alternates a measurement update step followed by a marginalization step to recursively maintain a full density on the last n states. While in the literature these algorithms are typically presented in terms of matrix operations, it is instructive to view this in terms of graphical models, while keeping in mind that the marginalization operation can be done quite effectively in the square-root information form, as discussed above.

We explain the general scheme using an example. The graphical models corresponding to a linear fixed-lag smoothing example are shown in Figure 5.4, for a lag of $n = 3$, where the state variables correspond to robot poses. We discuss each step in detail below.

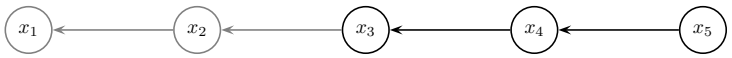
Figure 5.4a shows the Bayes net we would obtain after eliminating a chain-like factor graph with four variables $x_1 \dots x_4$, corresponding to the state of the system at four different moments in time. However,



(a) Fully eliminated Bayes net after having seen four measurements. In a fixed-lag smoothing scenario with $n = 3$, we "forget" about the pose x_1, hence it is grayed out. This is mathematically equivalent to marginalization.



(b) At the next time step, the new pose x_5 is predicted using a motion model factor, and measured via a new unary factor, both colored red. We also convert back the density on pose x_4 to a unary factor, in black.



(c) After eliminating x_4 and then x_5, we arrive back at a fully eliminated Bayes net. In the linear case we do not have to re-eliminate x_3, as it would yield the same result.

Figure 5.4: Fixed-lag smoothing for a localization scenario, which illustrates incremental inference with the language of graphical models.

because the variable x_1 is a leaf, i.e., it does not point to any other variable, it can be marginalized without any further computation. This is indicated in the figure by graying x_1 out.

In Figure 5.4b, a new pose x_5 is added to the system, along with a new relative constraint between x_4 and x_5 and a unary measurement constraint on x_5. These are shown in the figure as a red binary and unary factor, respectively. We now do something unexpected: the root density $p(x_4)$ is converted back to a unary factor $f(x_4)$, indicated in black in the figure. In other words, we build a small "mini factor graph" on the two variables x_4 and x_5 that takes into account the previous information on x_4, as well as the measurements associated with the new pose x_5. Since we added an new variable, we marginalize out the now oldest variable in the lag, the pose x_2, leaving only the three variables $x_3 \dots x_5$ in play (the marginalization step).

Finally, in Figure 5.4c, we apply variable elimination again (the measurement update step) to convert the factor graph portion back into a Bayes net, first eliminating the oldest variable x_4 and then x_5. We end up in a steady state where at any given time the joint density on the three most recent variables is readily available. For example, in the last panel, we can readily read off $p(x_3, x_4, x_5) = p(x_3|x_4, x_5)p(x_4|x_5)p(x_5)$.

The standard **Kalman filter** simply corresponds to using a fixed-lag of $n = 1$ in the above scheme, and hence both smoother and filter variants are readily explained in terms of simple graphical model operations. Note that simply dropping the earliest pose in the *factor graph* (instead of the Bayes net resulting from elimination) is not equivalent to marginalization, but results in a loss of information. Specifically, information gained from all previous measurements would be lost in such an optimization scheme that is not the same as fixed-lag smoothing.

Note that the graphical operations of elimination and marginalization above subsume matrix computations in this linear inference case: eliminating a graph corresponds to matrix factorization, and marginalization corresponds to dropping rows and columns, as explained in the previous section. In addition, as we are adding one measurement at a time, the factorization can be done incrementally using Givens rotations, as explained in Section 5.2. When doing the computations in square-root information form, these versions of the linear fixed-lag filter and smoother are also called a square-root-information filter or SRIF [15], or SRIS [110] in the smoother case.

5.4 Nonlinear Filtering and Smoothing

The desire of generalizing incremental inference to nonlinear problems motivates the introduction of the Bayes tree graphical model. Matrix factorization operates on linear systems, but as we discussed above most inference problems in robotics of practical interest are *nonlinear*, including SLAM. In the incremental matrix factorization story it is far from obvious how re-linearization can be performed incrementally without refactoring the complete matrix. To overcome this problem we investigate inference in graphical models, and introduce a new graphical model, the Bayes tree.

It is well known that inference in a tree-structured graph is efficient, and this includes the simple chain example we discussed above. In contrast, the factor graphs associated with typical robotics problems contain many loops. Still, we can construct a tree-structured graphical model in a two-step process: first, perform variable elimination on the factor graph (see Section 3.3.3) to obtain a Bayes net with a special property. Second, exploit that special property to find a tree structure over cliques in this Bayes net.

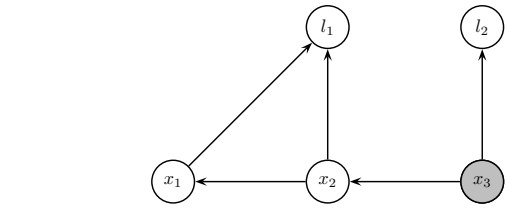
In particular, a Bayes net obtained by running the elimination algorithm (Algorithm 3.1) on a factor graph satisfies a special property: it is **chordal**, meaning that any undirected cycle of length greater than three has a chord. A chord is an edge connecting two non-consecutive vertices on the cycle. In AI and machine learning a chordal graph is more commonly said to be **triangulated**. Because it is still a Bayes net, the corresponding joint density $p(X)$ is given by factorizing over the individual variables x_j,

$$p(X) = \prod_j p(x_j|\pi_j), \tag{5.14}$$

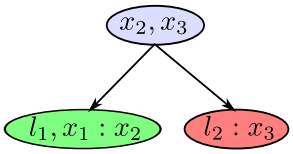
where π_j are the parent nodes of x_j. However, although the Bayes net is chordal, at this variable level it is still a non-trivial graph: neither chain-like nor tree-structured. The chordal Bayes net for our running toy SLAM example is shown in Figure 3.5e on page 36, and it is clear that there is an undirected cycle $x_1 - x_2 - l_1$.

5.4.1 The Bayes Tree

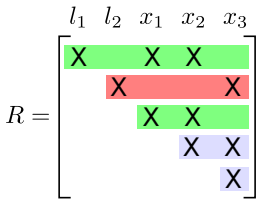
By identifying cliques (groups of fully connected variables) in this chordal graph, the Bayes net may be rewritten as a **Bayes tree.** We introduce this new, tree-structured graphical model to capture the *clique structure* of the Bayes net. It is not obvious that cliques in the Bayes net should form a tree. They do so because of the chordal property, although we will not attempt to prove that here. Listing all these cliques in an undirected tree yields a **clique tree**, also known as a **junction tree** in AI and machine learning. The Bayes tree is just a directed version of this that preserves information about the elimination order.



(a) Chordal Bayes net



(b) Bayes tree



(c) Square root information matrix

Figure 5.5: The Bayes tree (b) and the associated square root information matrix R (c) describing the clique structure in the chordal Bayes net (a) based on our canonical example from Figure 1.3. A Bayes tree is similar to a clique tree, but is better at capturing the formal equivalence between sparse linear algebra and inference in graphical models. The association of cliques with rows in the R factor is indicated by color.

More formally, a Bayes tree is a directed tree where the nodes represent **cliques** C_k of the underlying chordal Bayes net. In particular, we define one conditional density $p(F_k|S_k)$ per node, with the **separator** S_k as the intersection $C_k \cap \Pi_k$ of the clique C_k and its parent clique Π_k. The **frontal variables** F_k are the remaining variables, i.e. $F_k \triangleq C_k \setminus S_k$. We write $C_k = F_k : S_k$. The following expression gives the joint density $p(X)$ on the variables X defined by a Bayes tree:

$$p(X) \quad = \quad \prod_k p(F_k|S_k). \tag{5.15}$$

For the root F_r the separator is empty, i.e., it is a simple prior $p(F_r)$ on the root variables. The way Bayes trees are defined, the separator

S_k for a clique C_k is always a subset of the parent clique Π_k, and hence the directed edges in the graph have the same semantic meaning as in a Bayes net: conditioning.

Example. The Bayes tree associated with our canonical toy SLAM problem (Figure 1.3) is shown in Figure 5.5 on the previous page. The root clique $C_1 = \{x_2, x_3\}$ (shown in blue) comprises of x_2 and x_3, which intersects with two other cliques, $C_2 = \{l_1, x_1, x_2\} = \{l_1, x_1\} : \{x_2\}$ shown in green, and $C_3 = \{l_2, x_3\} = \{l_2\} : \{x_3\}$ shown in red. The colors also indicate how the rows of square-root information matrix R map to the different cliques, and how the Bayes tree captures independence relationships between them. For example, the green and red rows only intersect in variables that belong to the root clique, as predicted.

5.4.2 Updating the Bayes Tree

Incremental inference corresponds to a simple editing of the Bayes tree. This view provides a better explanation and understanding of the otherwise abstract incremental matrix factorization process. It also allows us to store and compute the square root information matrix in the form of a Bayes tree, a deeply meaningful sparse storage scheme.

To incrementally update the Bayes tree we proceed as in the fixed-lag smoothing example in Figure 5.4, by selectively converting part of the Bayes tree back into factor graph form. When a new measurement is added this corresponds to adding a factor, e.g., a measurement involving two variables will induce a new binary factor $f(x_j, x_{j'})$. In this case, *only* the paths in the Bayes tree between the cliques containing x_j and $x_{j'}$ and the root will be affected. The sub-trees below these cliques are unaffected, as are any other sub-trees not containing x_j or $x_{j'}$. Hence, to update the Bayes tree, the affected parts of the tree are converted back into a factor graph, and the new factor associated with the new measurement is added to it. By re-eliminating this temporary factor graph, using whatever elimination ordering is convenient, a new Bayes tree is formed and the unaffected sub-trees can be reattached.

In order to understand why only the top part of the tree is affected, we look at two important properties of the Bayes tree. These directly

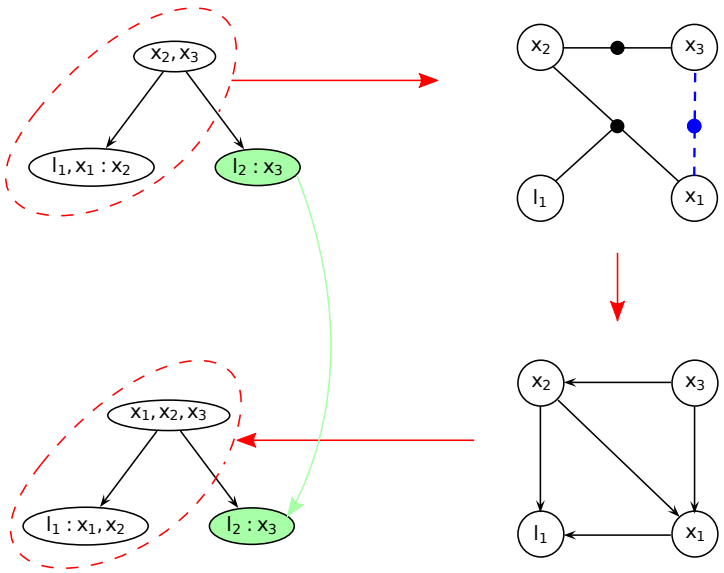


Figure 5.6: Updating a Bayes tree with a new factor, based on the example in Figure 5.5. The affected part of the Bayes tree is highlighted for the case of adding a new factor between x_1 and x_3. Note that the right branch (green) is not affected by the change. (top right) The factor graph generated from the affected part of the Bayes tree with the new factor (dashed blue) inserted. (bottom right) The chordal Bayes net resulting from eliminating the factor graph. (bottom left) The Bayes tree created from the chordal Bayes net, with the unmodified right "orphan" sub-tree from the original Bayes tree added back in.

arise from the fact that it encodes the information flow during elimination. The Bayes tree is formed from the chordal Bayes net following the inverse elimination order. In this way, variables in each clique collect information from their child cliques via the elimination of these children. Thus, information in any clique propagates only upwards to the root. Second, the information from a factor enters elimination only when the first variable connected to that factor is eliminated. Combining these two properties, we see that a new factor cannot influence any other variables that are not successors of the factor's variables. However, a factor on variables having different (i.e., independent) paths to the root means that these paths must now be re-eliminated to express the new dependency between them.

Algorithm 5.1 Updating the Bayes tree by recalculating a new Bayes tree from all affected cliques.

1: **function** UPDATEBAYESTREE(Bayes tree $\mathcal{T}$, new factor $f(\mathcal{J})$)
2: For each affected variable in $\mathcal{J}$:
3: Remove corresponding clique and all parent cliques up to root.
4: Store orphaned sub-trees $\mathcal{T}_{orph}$ of removed cliques.
5: Convert removed cliques to factor graph and add factor $f(\mathcal{J})$.
6: Eliminate the factor graph into a chordal Bayes net (using any convenient ordering).
7: Create a new Bayes tree $\mathcal{T}$' from this new chordal Bayes net.
8: Insert the orphans $\mathcal{T}_{orph}$ back into the new Bayes tree $\mathcal{T}$'.
9: **return** the updated Bayes Tree $\mathcal{T}$'.

Algorithm 5.1 shows the pseudo-code for the Bayes tree updating scheme, and Figure 5.6 shows how these incremental factorization/inference steps are applied to our canonical SLAM example. In this example, we add a new factor between x_1 and x_3, affecting only the left branch of the tree, marked by the red dashed line in to top left figure. We then create the factor graph shown in the top right figure by creating a factor for each of the clique densities, $p(x_2, x_3)$ and $p(l_1, x_1|x_2)$, and add the new factor $f(x_1, x_3)$. The bottom right figure shows the eliminated graph using the ordering l_1, x_1, x_2, x_3. And finally, in the bottom left figure, the reassembled Bayes tree is shown consisting of two parts: the Bayes tree derived from the eliminated graph, and the unaffected clique from the original Bayes tree (shown in green).

Figure 5.7 shows an example of the Bayes tree for a small SLAM sequence. Shown is the tree for step 400 of the well-known Manhattan world simulated sequence by Olson et al. [157]. As a robot explores the environment, new measurements only affect parts of the tree, and only those parts are re-calculated.

5.4.3 Incremental Smoothing and Mapping

Putting all of the above together and addressing some practical consideration about re-linearization yields a state of the art incremental,

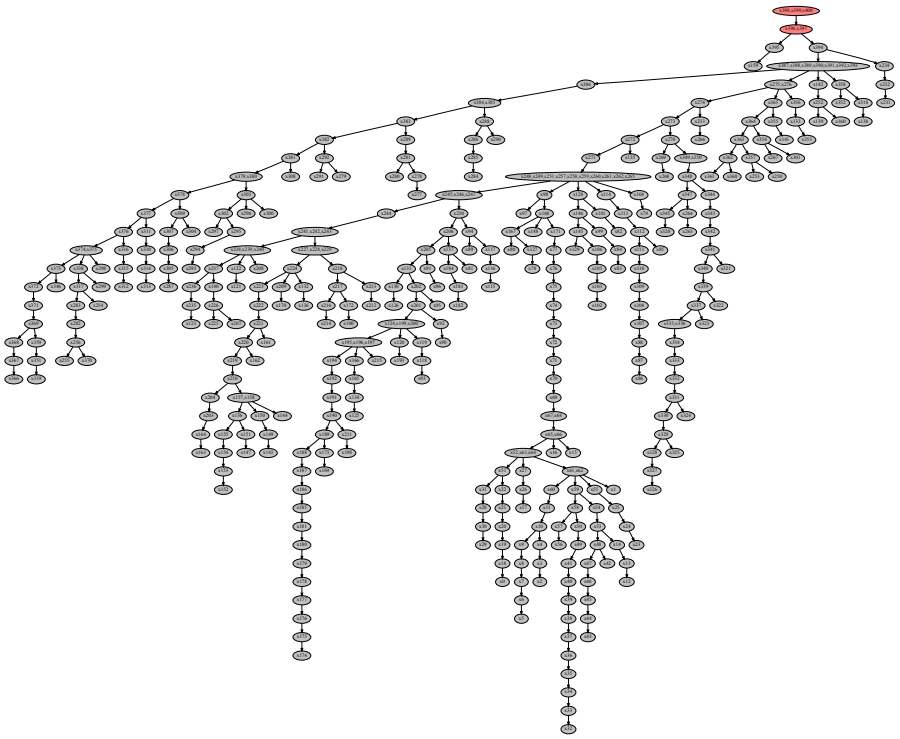


Figure 5.7: An example of the Bayes tree data structure for a small SLAM sequence. The incremental nonlinear least-squares estimation algorithm iSAM2 [109] is based on viewing incremental factorization as editing the graphical model corresponding to the posterior probability of the solution, the Bayes tree. As a robot explores the environment, new measurements often only affect small parts of the tree, and only those parts are re-calculated (shown in red).

nonlinear approach to MAP estimation in robotics, iSAM2, summarized below but described in full detail in [109]. The acronym "iSAM" stands for incremental smoothing and mapping, and the first version of it [111] used the incremental matrix factorization methods from Section 5.2. However, linearization in iSAM1 was handled in a suboptimal way: it was done for the full factor graph at periodic instances and/or when matrix fill-in became unwieldy. The second version of the approach, iSAM2, uses a Bayes tree representation for the posterior density. It then employs Bayes tree incremental updating as each new

measurement comes in, as described above. Below we discuss the practical aspects of variable ordering, solution updating, and dealing with non-linear measurements that together then constitute iSAM2.

What variable ordering should we use in re-eliminating the affected cliques? Only the frontal variables in the affected part of the Bayes tree are updated. One strategy then is to apply COLAMD locally to the affected variables. However, in the incremental case we can do better: To keep the size of the affected part of the tree smaller for future updates, we force recently accessed variables to the end of the ordering, i.e. into the root clique. For this incremental variable ordering strategy one can use the constrained COLAMD (CCOLAMD) algorithm [40]. This both forces the most recently accessed variables to the end and still provides a good overall ordering. Generally, subsequent updates will then only affect a small part of the tree, and can therefore be expected to be efficient in most cases, except for large loop closures. Since only a subset of the variables are ordered independent of the rest of the tree, the overall ordering generates somewhat more fill-in (larger cliques), but in practice the difference is not large.

After updating the tree we also need to *update the solution*. Back-substitution in the Bayes tree proceeds from the root (which does not depend on any other variables) and proceeds to the leaves. However, it is typically not necessary to recompute a solution for all variables: Local updates to the tree often do not affect variables in remote parts of the tree. Instead, at each clique we can check the difference in variable estimates that is propagated downwards and stop when this difference falls below a small threshold.

Our motivation for introducing the Bayes tree was to incrementally solve nonlinear optimization problems. For this we *selectively relinearize* factors that contain variables whose deviation from the linearization point (as obtained by back-substitution) exceeds a small threshold. In contrast to the tree modification caused by adding new factors, we now have to redo all cliques that contain the affected variables, not just as frontal variables, but also as separator variables. This affects larger parts of the tree, but in most cases is still significantly cheaper than recomputing the complete tree. We also have to go back

to the original factors, instead of directly turning the cliques into a factor graph. And that requires caching certain quantities during elimination. The overall incremental nonlinear algorithm, iSAM2, is described in much more detail in [109].

Just as we discussed the *linear* Kalman filter and fixed-lag smoother in terms of graphical models, the well-known Extended Kalman Filter and Smoother [145] can be seen as a special case of the iSAM2 algorithm, where the Bayes tree reduces to a chain rather than a proper tree. Or, another way to state this is that the Bayes tree representation enabled us to generalize the earlier non-linear estimators to general graphical models. Indeed, iSAM and iSAM2 have been applied successfully to many different robotics estimation problems with non-trivial constraints between variables that number into the millions, as will be discussed in depth in Section 7.

5.5 Bibliographic Remarks

Updating of matrix factorizations is a well-known technique in many areas, with applications such as computer vision [191, 117] and signal processing [134]. Golub and Van Loan [84] present general methods for updating matrix factorizations based on [83, 77], including the Givens rotations we use in this work. Davis has done much work in the areas of variable ordering and factorization updates, and provides highly optimized software libraries [40, 41] for various such tasks.

Kaess et al. [111] proposed incremental smoothing and mapping (iSAM), which performs fast incremental updates of the square root information matrix, yet is able to compute the full map and trajectory at any time. New measurements are added using matrix update equations [83, 77, 84], so that previously calculated components of the square root information matrix are reused. However, to remain efficient and consistent, iSAM requires periodic batch steps to allow for variable reordering and re-linearization, which is expensive and detracts from the intended online nature of the algorithm.

It is well known that a chordal Bayes net can be converted into a tree-structured graphical model in which these operations are easy. This

data structure is similar to the clique tree [165, 17, 121], also known as the junction tree in the AI literature [32], which has also been exploited for distributed inference in SLAM by Paskin [160], Dellaert et al. [49], and Pinies et al. [164].

To combine the advantages of the graphical model and sparse linear algebra perspectives, we propose a novel data structure, the Bayes tree, first presented in Kaess et al. [107]. Our approach is based on viewing matrix factorization as eliminating a factor graph into a Bayes net, which is the graphical model equivalent of the square root information matrix. However, the Bayes tree is *directed* and corresponds more naturally to the result of QR and Cholesky factorization in linear algebra, allowing us to analyze it in terms of conditional probability densities in the tree. As trees of cliques in chordal graphs, Bayes trees are similar to clique trees [17] and junction trees [121]. However, a Bayes tree is directed and is semantically closer to a Bayes net in the way it encodes a factored probability density.

As defined in [121], a cluster tree is a directed tree of clusters in which the running intersection property holds, and each factor in the original graph is associated with a cluster. The cluster tree is more general than the junction tree or the clique tree. Ni and Dellaert [152] proposed TSAM 2, a multi-level SLAM approach that combines nested dissection [78, 135] with a cluster tree representation.

Exploiting the Bayes tree and the insights gained, we proposed iSAM2 [108, 109], a novel incremental exact inference method that allows for incremental reordering and just-in-time re-linearization. iSAM2 extends our original iSAM algorithm by leveraging these insights about the connections between graphical model and sparse linear algebra perspectives. To the best of our knowledge this is a completely novel approach to providing an efficient and exact solution to a sparse nonlinear optimization problem in an incremental setting, with general applications beyond SLAM. While standard nonlinear optimization methods repeatedly solve a linear batch problem to update the linearization point, our Bayes tree-based algorithm allows fluid relinearization of a reduced set of variables, which translates into higher efficiency, while retaining sparseness and full accuracy.

Some other SLAM algorithms employ direct equation solvers based on Cholesky or QR factorization. Treemap [72] uses Cholesky factors to represent probability distributions in a tree-based algorithm. However, multiple approximations are employed to reduce the complexity, while iSAM solves the full and exact problem, and therefore allows re-linearization of all variables at any time.

Beyond the use of better inference algorithms, the graph itself can be simplified to further improve efficiency. The reduced pose graph by Johannsson et al. [104] reuses spatially nearby poses to limit growth in the number of variables to the size of the explored space, rather than time. This is a special form of marginalization that is constructed in such as way as not to make the graph more dense. General marginalization can be combined with sparsification techniques as done by Carlevaris-Bianco et al. [23].

The Bayes tree is not limited to inference over Gaussian densities. Segal and Reid [171] performs mixed discrete-continuous inference over junction trees to deal with incorrect data association. The recently proposed multi-modal iSAM (mmiSAM) algorithm by Fourie et al. [69] exploits the Bayes tree structure for efficient inference over general densities, obtaining not just a mean, but a complete posterior density that can be non-Gaussian and multimodal. This is achieved by using non-parametric belief propagation on the cliques of the Bayes tree, where kernel density estimates approximate both intermediate and posterior densities. Following iSAM2, the use of the Bayes tree is expected to allow mmiSAM to be used in an incremental fashion.

6

Optimization on Manifolds

While in some robotics problems we can get away with vector-valued unknowns, in most practical situations we have to deal with 3D rotations and other nonlinear manifolds**.** These need a more sophisticated machinery that takes into account their special structure. In this section we discuss how to perform optimization on nonlinear manifolds, which will build upon the optimization framework for vector spaces from Sections 2, 3, and 4.

6.1 Attitude and Heading Estimation

Let us consider the following practical example: we would like to estimate the unknown orientation $R \in SO(3)$ of a robotic platform, which is known to have three degrees of freedom. Indeed, the most common way to refer to a robot's orientation are the Euler angles: roll, pitch, and yaw. Here roll and pitch together describe the platform's **attitude** with respect to level, and yaw is referred to as the platform's **heading**.

However, the Euler angle representation suffers from singularities, and hence we often prefer to represent R using a quaternion or a 3×3 matrix. While these representations are over-parameterized, i.e., they

use more than 3 numbers, they obey constraints that makes them refer to the same underlying three-dimensional **rotation manifold**. Appendix B.3 defines the 3D rotation manifold and describes the different representations in more detail.

Now, as an example, let us assume that we have access to an accelerometer that is subject to no forces except gravity, and outputs a three-dimensional measurement $z_a \in \mathbb{R}^3$. Furthermore, we assume that we have access to the following measurement function that predicts the accelerometer reading z_a from the rotation R:

$$h^a : SO(3) \to \mathbb{R}^3 : R \mapsto z_a. \tag{6.1}$$

Given what we have seen so far, it is natural to try and estimate the unknown rotation R by minimizing the least-squares error criterion

$$R^* = \underset{R}{\operatorname{argmin}} \left\| h^a(R) - z_a \right\|_\Sigma^2, \tag{6.2}$$

where Σ is the measurement covariance for the accelerometer.

Unfortunately, since R lives on the three-dimensional manifold $SO(3)$, we cannot directly optimize over it. Indeed, suppose we use the 3×3 matrix representation for R, and initialize it to some matrix R_0, say the identity matrix I_3. Then, to use any of the nonlinear optimization schemes we discussed above in Section 2, we need a notion of how to move in the neighborhood of this initial estimate R_0. If rotations were vectors (which they are *not*) we could try to use vector addition $+$ as a candidate for a generalized addition operator $\oplus$:

$$R_0 \oplus \xi \stackrel{\Delta}{=} R_0 + \xi. \tag{6.3}$$

However, this does not quite work for 3D rotations, *even if* we properly vectorize the initial rotation estimate R_0 to add the 9-dimensional update vector ξ. This is because adding an arbitrary 3×3 matrix Ξ obtained from a 9-dimensional vector ξ almost inevitably moves away from the $SO(3)$ manifold. In particular, the matrix $R + \Xi$ is almost surely no longer orthogonal. In addition, we know that rotations have only three degrees of freedom, not nine, so it seems counter-intuitive to work with 9-dimensional increments.

6.1.1 Incremental Rotations

For 3D rotations, a good candidate for three-dimensional, vector-valued increments can be obtained from the axis-angle representation $(\bar{\omega}, \theta)$. As reviewed in appendix B.3, $\bar{\omega} \in S^2$ represents the axis of rotation, and θ the angle by which we rotate. By multiplying both we obtain a three-dimensional vector $\xi \triangleq \bar{\omega}\theta \in \mathbb{R}^3$ that combines both axis and rotation angle. For small θ the corresponding rotation matrix $R(\xi)$ is well approximated by the matrix

$$R(\xi) \approx \begin{bmatrix} 1 & -\xi_z & \xi_y \\ \xi_z & 1 & -\xi_x \\ -\xi_y & \xi_x & 1 \end{bmatrix}. \tag{6.4}$$

The 3×3 matrix above is *not* on the $SO(3)$ manifold, strictly speaking, but it is at least close to the manifold for small ξ. We can write the above more concisely as

$$R(\xi) \approx I + \hat{\xi}. \tag{6.5}$$

Above, the **hat operator** creates a skew-symmetric matrix $\hat{\xi}$ from ξ, and is defined as

$$\hat{\xi} \triangleq \begin{bmatrix} 0 & -\xi_z & \xi_y \\ \xi_z & 0 & -\xi_x \\ -\xi_y & \xi_x & 0 \end{bmatrix}. \tag{6.6}$$

6.1.2 The Exponential Map

While Equation 6.5 is only approximate, the **exponential map** provides an *exact* mapping from 3-dimensional increments ξ onto proper rotations. This is done by taking the **matrix exponential** of the quantity $\hat{\xi}$, as given by the following infinite series,

$$\exp \hat{\xi} \triangleq \sum_{k=0}^{\infty} \frac{1}{k!} \hat{\xi}^k = I + \hat{\xi} + \frac{\hat{\xi}^2}{2!} + \frac{\hat{\xi}^3}{3!} + \ldots, \tag{6.7}$$

where the powers are to be interpreted as *matrix powers*. Note that the first two terms are identical to the approximation in Equation 6.5; the other terms can be seen as increasingly smaller corrections to bring the matrix $R(\xi)$ back to the $SO(3)$ manifold.

While not always fully appreciated, the exponential map is exact *for arbitrarily large* vectors ξ. In addition, the exponential map for $SO(3)$ is available in closed form, through **Rodrigues' formula**:

$$\exp \hat{\xi} = I + \frac{\sin \theta}{\theta} \hat{\xi} + \frac{1 - \cos \theta}{\theta^2} \hat{\xi}^2. \tag{6.8}$$

6.1.3 Local Coordinates

The notion of an exponential map allows us to define a mapping from **local coordinates** ξ back to a neighborhood around the estimate R_0,

$$R_0 \oplus \xi \triangleq R_0 \cdot \exp \hat{\xi}, \tag{6.9}$$

with $\xi \in \mathbb{R}^3$. In other words, we exponentiate the coordinates ξ around the identity to create an incremental rotation that is composed with the base rotation R_0. Local coordinates equal to zero correspond to R_0 itself, and non-zero local coordinates ξ around zero are smoothly mapped to a neighborhood of R_0 on the rotation manifold. Because $\mathbb{R}^3$ is a vector space under addition, this will allow us to use R_0 as the linearization point in a nonlinear optimization scheme.

Indeed, after this re-parameterization, we can minimize for ξ instead, starting from zero. We obtain the following estimator

$$\xi^* = \underset{\xi}{\operatorname{argmin}} \|h^a(R_0 e^{\hat{\xi}}) - z_a\|_\Sigma^2 = \underset{\xi}{\operatorname{argmin}} \|g^a(\xi; R_0) - z_a\|_\Sigma^2, \tag{6.10}$$

where now $g^a(\xi; R_0) \triangleq h^a(R_0 e^{\hat{\xi}})$ is a new measurement function defined over the local coordinates ξ.

To minimize (6.10) we need a notion of how this new prediction function $g^a(\xi; R_0)$ behaves in the neighborhood of zero. Loosely speaking, we need to find the 3×3 Jacobian matrix $G^a_{R_0}$ such that

$$g^a(\xi; R_0) \approx h(R_0) + G^a_{R_0} \xi \tag{6.11}$$

with $\xi \in \mathbb{R}^3$. A simple but computationally expensive way to calculate $G^a_{R_0}$ is numerical differentiation, but more advanced code-bases use symbolic derivatives or some type of automatic differentiation [113, 47].

Once equipped with the approximation (6.11), we can minimize the linear objective function (6.10) with respect to the local coordinates ξ:

$$\xi^* = \underset{\xi}{\operatorname{argmin}} \|h(R_0) + G^a_{R_0}\xi - z_a\|^2_\Sigma, \tag{6.12}$$

using the techniques we discussed in the previous sections.

We can now proceed to using any non-linear optimization scheme discussed in Section 2, but after each iteration of whatever nonlinear optimization scheme we use, we need to update the base rotation R_0 via (6.9). As an example, the Gauss-Newton method would re-compute the Jacobian $G^a_{R_0}$of the measurement function g^a, and solve the approximate linear update step in 6.12 via

$$\xi^* = \left(G^a_{R_0}\right)^\dagger \left(z_a - h(R_0)\right) \tag{6.13}$$

with $.^\dagger$ denoting the pseudo-inverse. But since the measurement function $h(.)$ can be highly nonlinear, a trust-region method such as Levenberg-Marquardt might have to be used instead.

6.1.4 Incorporating Heading Information

An issue with the minimization in (6.10) is that the solution is not well-defined using only a single accelerometer reading. Indeed, an accelerometer does not give *any* information about the rotation around the gravity vector! In practice, this will manifest itself in the fact that the Jacobian $G^a_{R_0}$ will be rank-deficient.

Additional information on R, e.g., using a magnetometer, is needed to fully determine R in $SO(3)$. A magnetometer yields a three-dimensional reading $z_m \in \mathbb{R}^3$ corresponding to the local magnetic field, which will vary predictably when the magnetometer itself is rotated. We can model this in the same way as above, through a measurement function g^m that models the magnetometer in terms of local (incremental rotation) coordinates ξ:

$$g^m(\xi; R_0) \triangleq h^m(R_0 e^{\hat{\xi}}). \tag{6.14}$$

Of course, the detailed expression for g^m and its Jacobian $G^m_{R_0}$ will differ substantially from the accelerometer case.

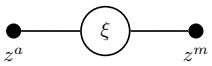


Figure 6.1: Trivial factor graph for estimating attitude and heading by fusing both accelerometer and magnetometer readings z_a and z_m.

A full attitude and heading determination can then be achieved by fusing both accelerometer and magnetometer readings z_a and z_m. This is done by minimizing the sum of squared residuals corresponding to a trivial factor graph, shown in Figure 6.1 with a single unknown ξ and two factors corresponding to z_a and z_m, respectively:

$$\xi^* = \underset{\xi}{\operatorname{argmin}} \|g^a(\xi; R_0) - z_a\|^2_\Sigma + \|g^m(\xi; R_0) - z_m\|^2_\Sigma \,. \tag{6.15}$$

If we add time into this equation and introduce unknown rotation matrices for successive time steps we can then estimate the changing 3D orientation of a robotic platform over time.

While outside the scope of the current document, if in addition we measure the difference between successive orientations through use of a gyroscope, add variables to model the slowly varying accelerometer and gyroscope biases, and integrate all these in a factor graph, we obtain a fully-fledged attitude and heading reference system (AHRS) [63].

6.1.5 Planar Rotations

For completeness, and because this will be useful when we talk about PoseSLAM in the next section, we now show that *planar* rotations can be treated in a similar way, respecting their manifold nature. While it is tempting to simply represent a planar rotation with an angle $\theta \in \mathbb{R}$, this does not accurately reflect the wrapping by 2π that so often introduces bugs in code. Instead, as reviewed in Appendix B.1, the nonlinear manifold $SO(2)$ containing all orthonormal 2×2 matrices accurately reflects this circular topology, using:

$$\theta \rightarrow \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}. \tag{6.16}$$

For planar rotations $R \in SO(2)$ the local parameterization $\xi \in \mathbb{R}$ is one-dimensional, and the hat operator upgrades that to a 2×2 matrix:

$$\hat{\xi} \triangleq \begin{bmatrix} 0 & -\xi \\ \xi & 0 \end{bmatrix}. \tag{6.17}$$

Now, when we apply the exponential map (6.7), and by recognizing the sine and cosine series, we obtain

$$\sum_{k=0}^{\infty} \frac{1}{k!} \hat{\xi}^k = \begin{bmatrix} 1 - \xi^2/2 \dots & -\xi + \xi^3/6 \dots \\ \xi - \xi^3/6 \dots & 1 - \xi^2/2 \dots \end{bmatrix} = \begin{bmatrix} \cos\xi & -\sin\xi \\ \sin\xi & \cos\xi \end{bmatrix}, \tag{6.18}$$

i.e., the closed-form expression that we know to be correct for $SO(2)$.

Putting it all together, the $SO(2)$ equivalent of the local update equation (6.9) becomes

$$\begin{aligned} R_0 \oplus \xi &\triangleq R_0 \cdot \exp \hat{\xi} && (6.19) \\ &= \begin{bmatrix} \cos\theta_0 & -\sin\theta_0 \\ \sin\theta_0 & \cos\theta_0 \end{bmatrix} \begin{bmatrix} \cos\xi & -\sin\xi \\ \sin\xi & \cos\xi \end{bmatrix} && (6.20) \\ &= \begin{bmatrix} \cos(\theta_0 + \xi) & -\sin(\theta_0 + \xi) \\ \sin(\theta_0 + \xi) & \cos(\theta_0 + \xi) \end{bmatrix}, && (6.21) \end{aligned}$$

i.e., the effect is exactly the same as adding ξ to θ_0, but with the wrapping handled correctly.

6.2 PoseSLAM

In many robotics applications the main quantity of interest is the pose of the robot over time, i.e., the robot's **trajectory**. While we have so far been content to talk about this in the abstract, we now need to confront that poses, like rotations, live on a manifold as well. For example, consider again the factor graph from Figure 2.1 in Section 2. What is not apparent from the figure is that even in the planar case the robot poses actually have three degrees of freedom: in addition to a 2D position, they include the robot *orientation*.

PoseSLAM is a variant of simultaneous localization and mapping (SLAM) where we only optimize over robot **poses**, and do not explicitly

create a map of the environment. The goal of SLAM is to simultaneously localize a robot and map the environment given incoming sensor measurements [59]. Besides wheel odometry, one of the most popular sensors for robots moving on a plane is a 2D laser-range finder, which provides both odometry constraints between successive poses, and loop-closure constraints when the robot re-visits a previously explored part of the environment. A LIDAR sensor can generate many thousands to even millions of readings per second: it is impractical to explicitly optimize over a dense 3D map of the environment. Instead, if we can reconstruct the *trajectory* of robot poses over time, a dense 3D map can be generated simply by re-projecting all LIDAR measurements into a coordinate frame aligned with the first pose.

6.2.1 Representing Poses

Planar poses live on a three-dimensional, non-linear manifold. If we know that the robot operates on a planar surface, we have three degrees of freedom: translation (x, y) and heading. In particular, the robot poses x_i live on the Special Euclidean Group $SE(2)$, a manifold obtained by combining position (a two-dimensional vector space) with the nonlinear manifold $SO(2)$ of planar rotations, discussed above.

In many other applications we need to reason about the robot's motion in 3D. In those cases we need to consider the full 6-DOF combined position and orientation. A typical example is a quadrotor UAV, or any autonomous ground-vehicle that needs to operate in uneven terrain. We say that in that case the poses $x_i \in SE(3)$, a 6-dimensional manifold resulting from combining positions in $\mathbb{R}^3$ with a three-dimensional rotation matrix in $SO(3)$. The details on how to represent poses in the planar case and the full 3D case are in Appendices B.2 and B.4.

6.2.2 Local Pose Coordinates

The question then is how we can optimize over a set of 2D or 3D poses. As we did for rotations, the solution is switching to a local parameterization and using the exponential map to convert incremental updates at each iteration back to the pose manifold. In the case of $SE(2)$ and $SE(3)$ it is common to think about the *rate of change* and obtain an

increment ξ by multiplying with a finite time $\Delta\tau$. In particular, we define the **angular velocity** ω and **translational velocity** v, and define ξ as

$$\xi \triangleq \begin{bmatrix} \omega \\ v \end{bmatrix} \Delta\tau \tag{6.22}$$

respectively a 3D or 6D vector. The hat operator is then given by

$$\hat{}: \begin{bmatrix} \omega \\ v \end{bmatrix} \longmapsto \begin{bmatrix} \hat{\omega} & v \\ 0 & 0 \end{bmatrix}, \tag{6.23}$$

with $\hat{\omega}$ the result of applying the hat operator of the corresponding rotation group to ω. In detail, for $SE(2)$ and $SE(3)$, we have respectively

$$\hat{}: \mathbb{R}^3 \to \mathfrak{se}(2): \ \xi \mapsto \left[\begin{array}{cc|c} 0 & -\omega_z & v_x \\ \omega_z & 0 & v_y \\ \hline 0 & 0 & 0 \end{array}\right] \Delta\tau \tag{6.24}$$

$$\hat{}: \mathbb{R}^6 \to \mathfrak{se}(3): \ \xi \mapsto \left[\begin{array}{ccc|c} 0 & -\omega_z & \omega_y & v_x \\ \omega_z & 0 & -\omega_x & v_y \\ -\omega_y & \omega_x & 0 & v_z \\ \hline 0 & 0 & 0 & 0 \end{array}\right] \Delta\tau, \tag{6.25}$$

where one recognizes the skew-symmetric matrices corresponding to an incremental rotation in the top-left. Closed-form solutions of the associated exponential maps exist as well, see for example [150].

In either case, we can again use the exponential map to define a mapping from **local pose coordinates** ξ back to a neighborhood around an initial pose estimate x_0:

$$x_0 \oplus \xi \triangleq x_0 \cdot \exp \hat{\xi}, \tag{6.26}$$

where $\xi \in \mathbb{R}^3$ for planar poses $x_0 \in SE(2)$, and $\xi \in \mathbb{R}^6$ for 3D poses $x_0 \in SE(3)$, and the hat operator is defined by (6.24) and (6.25), respectively.

6.2.3 Optimizing over Poses

We now have the essential ingredients to optimize over poses $x \in SE(2)$ or $x \in SE(3)$. For ease of exposition, let us consider a simple minimiza-

tion problem in a *single* pose $x \in SE(3)$:

$$x^* = \arg\min_x \|h(x) - z\|^2_\Sigma. \tag{6.27}$$

As before, we assume that $z \in \mathbb{R}^m$ is a known, vector-valued measurement. However, since the pose $x \in SE(3)$ is *not* a vector, the measurement function $h : SE(3) \to \mathbb{R}^m$ is now a vector-valued function defined on the $SE(3)$ manifold. Two simple examples that we have already encountered in Section 6.1 are predicting accelerometer and magnetometer readings.

To minimize the objective function (6.27) we again need a notion of how the nonlinear measurement function $h(x)$ behaves in the neighborhood of a base pose x_0, typically the current linearization point. Hence, we need to calculate the $m \times 6$ Jacobian matrix H_0 such that

$$h(x_0 \oplus \xi) \approx h(x_0) + H_0\xi \tag{6.28}$$

with $\xi \in \mathbb{R}^6$. The vector ξ is 6-dimensional, capturing all directions we can move on the 6-dimensional pose manifold $SE(3)$. As discussed before, we can calculate H_0 using numerical, symbolic, or automatic differentiation.

Once equipped with the approximation (6.28), we can minimize the objective function (6.27) with respect to the local coordinates ξ instead:

$$\xi^* = \operatorname*{argmin}_\xi \|h(x_0) + H_0\xi - z\|^2_\Sigma. \tag{6.29}$$

6.2.4 PoseSLAM

In PoseSLAM we do not over a single pose, of course, but over all poses in a robot trajectory that we want to reconstruct. Typically, two types of factors will be involved: unary factors such as pose priors and/or absolute pose measurements (e.g., from GPS) and binary factors, such as relative pose constraints derived from LIDAR.

The factor graph for a simple PoseSLAM example is shown in Figure 6.2. To anchor the graph we add the unary factor $f_0(x_1)$, and as the robot travels through the world, binary factors $f_t(x_t, x_{t+1})$ corresponding to odometry are created. The red factor models a different event:

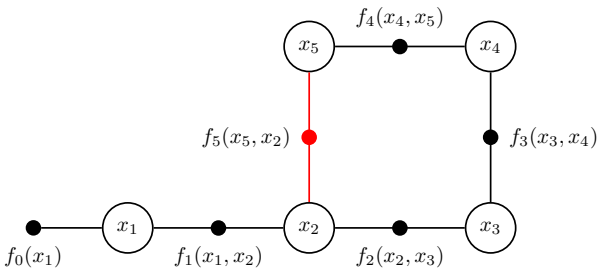


Figure 6.2: Factor graph for PoseSLAM with a loop closure constraint in red.

a **loop closure**. For example, the robot might recognize the same location using vision or a laser range finder, and calculate the geometric pose constraint to when it first visited this location. This is illustrated for poses x_5 and x_2, which generates the loop closing factor $f_5(x_5, x_2)$.

Deriving the detailed expressions for the corresponding measurement functions and their Jacobians is beyond the scope of this document, but the final optimization will—in each iteration—minimize over the local coordinates of all poses by summing over the following linearized measurement factors:

$$\Xi^* = \underset{\Xi}{\operatorname{argmin}} \sum_i \|h(x_i) + H_i\xi_i - z\|^2_\Sigma + \sum_k \|g(x_i, x_j) + F_i\xi_i + G_j\xi_j - z\|^2_\Sigma. \tag{6.30}$$

Above $\Xi \triangleq \{\xi_i\}$, the set of all incremental pose coordinates, h and g are the unary and binary measurement functions, respectively, and H_i, F_i and G_j their respective Jacobians.

6.3 Optimization over Lie Groups and Arbitrary Manifolds

While the rotation and rigid transformation groups are by far the most important in robotics, the local parameterization technique above can be generalized to any matrix Lie group and even to non-group nonlinear manifolds. In fact, sometimes it is computationally advantageous to forget about the Lie group structure and simply use local mappings other than the exponential map. We discuss all of this below.

6.3.1 Matrix Lie Groups

For any n-dimensional matrix Lie group $G \subset GL(n)$, the quantity $\hat{\xi}$ generated by the hat operator is an $n \times n$ matrix. These matrices, together with a composition operator known as the Lie bracket, form a separate algebraic structure known as the **Lie algebra** $\mathfrak{g}$ associated with G. The matrices $\hat{\xi}$ are referred to as *elements of the Lie algebra*.

Generalizing the cases for $SO(n)$ and $SE(n)$ above, the hat operator maps a vector ξ from $\mathbb{R}^n$ to $\mathfrak{g}$:

$$\hat{}: \mathbb{R}^n \to \mathfrak{g},\ \xi \mapsto \hat{\xi}. \tag{6.31}$$

The inverse map also exists, and is known as the **vee operator**:

$$\check{}: \mathfrak{g} \to \mathbb{R}^{\mathfrak{n}},\ \hat{\xi} \mapsto \xi. \tag{6.32}$$

Hence, there is a one-to-one mapping between $\mathbb{R}^n$ and the Lie algebra $\mathfrak{g}$ associated with an n-dimensional manifold.

The exponential map defined by (6.7) allows us to define a mapping from **local coordinates** ξ back to a neighborhood around any initial estimate $a \in G$:

$$a \oplus \xi \triangleq a \cdot \exp \hat{\xi}. \tag{6.33}$$

This generalizes the notion of exponentiating canonical coordinates ξ around the identity to create an incremental transformation that is composed with the base transformation a. Note that zero local coordinates correspond to a itself, i.e.,

$$a \oplus 0 = a \cdot \exp \hat{0} = a, \tag{6.34}$$

and non-zero local coordinates ξ around the identity are smoothly mapped to a neighborhood of a on the manifold. Because $\mathbb{R}^n$ is a vector space under addition, this allows us to use a as the linearization point in a nonlinear optimization scheme.

6.3.2 General Manifolds and Retractions

General manifolds $\mathcal{M}$ that do not possess a group structure can still be handled by defining a **retraction** $\mathcal{R}_a : \mathcal{M} \times \mathbb{R}^n \to \mathcal{M}$, such that

$$a \oplus \xi \triangleq \mathcal{R}_a(\xi). \tag{6.35}$$

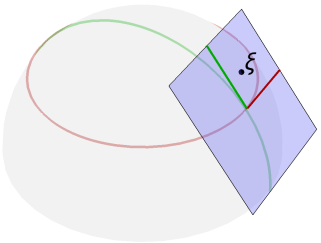


Figure 6.3: For the sphere manifold, the local tangent plane with a local basis provides the notion of local coordinates.

Here $\xi \in \mathbb{R}^n$ are the local coordinates, which have to be uniquely defined for each point a on the manifold, and the retraction $\mathcal{R}_a$ maps these back onto the manifold. A rigorous definition for retractions can be found in [1], but informally we want it to be smooth and map the identity in $\mathbb{R}^n$ back to a, i.e., $\mathcal{R}_a(0) = a$.

Example. As discussed in Section B.5, an important two-dimensional manifold that is *not* a group is the set of all directions in 3D space, i.e., the sphere S^2 of all unit vectors in $\mathbb{R}^3$:

$$S^2 = \{p \in \mathbb{R}^3 | \, \|p\| = 1\}. \tag{6.36}$$

We can define local coordinates and a retraction for the sphere by making use of tangent planes. Figure 6.3 shows the sphere S^2 with a local tangent plane at a point p. We write T_pS^2, and define it as all three-vectors $\hat{\xi}$ tangent to S^2 at p, i.e.,

$$T_pS^2 \triangleq \left\{\hat{\xi} \in \mathbb{R}^3 | p^\top \hat{\xi} = 0\right\}. \tag{6.37}$$

To uniquely define local coordinates at a given point p we need to define a basis B_p for the local tangent space T_pS^2. One way to choose the local basis B_p is by using QR decomposition to write $p = QR$, where Q is orthonormal and R is of the form $[1\,0\,0]^\top$. It follows that $Q_1 = p$, and we can use the two last columns of Q as the basis $B_p = [Q_2|Q_3]$. We can then write $\hat{\xi} = B_p\xi$ with $\xi \in R^2$ for any tangent vector $\hat{\xi}$.

As a retraction $\mathcal{R}_p(\xi)$, we can simply add $\hat{\xi} = B_p\xi$ to p and renormalize to get a new point q on the sphere:

$$q = \mathcal{R}_p(\xi) = \frac{p + B_p\xi}{\|p + B_p\xi\|}. \tag{6.38}$$

6.3.3 Retractions and Lie Groups

We can define many retractions for a given manifold $\mathcal{M}$, even for those with group structure. For the vector space $\mathbb{R}^n$ the retraction is just vector addition, and for Lie groups the obvious retraction is simply the exponential map, i.e., $\mathcal{R}_a(\xi) = a \cdot \exp \hat{\xi}$. However, one can choose other, possibly more computationally attractive retractions, as long as around a they agree with the geodesic induced by the exponential map, i.e.,

$$\lim_{\xi \to 0} \frac{\left| a \cdot \exp \hat{\xi} - \mathcal{R}_a(\xi) \right|}{|\xi|} = 0. \tag{6.39}$$

Example. For $SE(3)$, instead of using the true exponential map it is computationally more efficient to define a retraction which uses the simpler expression $v\Delta\tau$ for the translation update:

$$\mathcal{R}_T(\xi) = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} e^{\hat{\omega}\Delta\tau} & v\Delta\tau \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} Re^{\hat{\omega}\Delta\tau} & t + Rv\Delta\tau \\ 0 & 1 \end{bmatrix}. \tag{6.40}$$

In effect, both rotation and translation are updated separately, although still in the original coordinate frame. In contrast to the above, the exponential map takes into account the rotating coordinate frame when updating the translation, and the corresponding trajectory is a *screw* motion when plotted as a function of $\Delta\tau$. This correspond to a rigid body with constant angular and translational velocity in the body frame. However, for the purposes of optimization the exact mapping away from zero does not matter all that much.

6.4 Bibliographic Remarks

The book by Murray, Li, and Sastry [150] is an excellent text on Lie groups in robotics, albeit focused mainly on manipulation tasks. Optimization on manifolds using retractions is discussed in great detail in [1]. Our exposition on local function behavior and the definition of Jacobian matrices is taken from [177].

7

Applications

In this section we provide an overview of applications of factor graphs for robot perception. This overview is aimed at showing the broad impact factor graphs had in robot perception and is by no means intended to represent a complete survey of the literature.

7.1 Inertial Navigation

An inertial measurement unit (IMU) is an important sensor in robotics, because it provides high frequency information about a platform's motion, which is crucial in several important applications such as autonomous driving or autonomous flight. Unfortunately, IMUs are not suitable for long-term navigation by themselves, because they quickly accumulate error over time and, even more difficult to deal with, suffer from measurement bias that also drifts over time. However, IMUs are perfect to fill in the gaps between lower-frequency measurements such as LIDARs, cameras, or GPS, and those sensors can in turn be used to correct IMU drift. Factor graphs provide a very flexible framework in which to fuse these two complimentary sources of information.

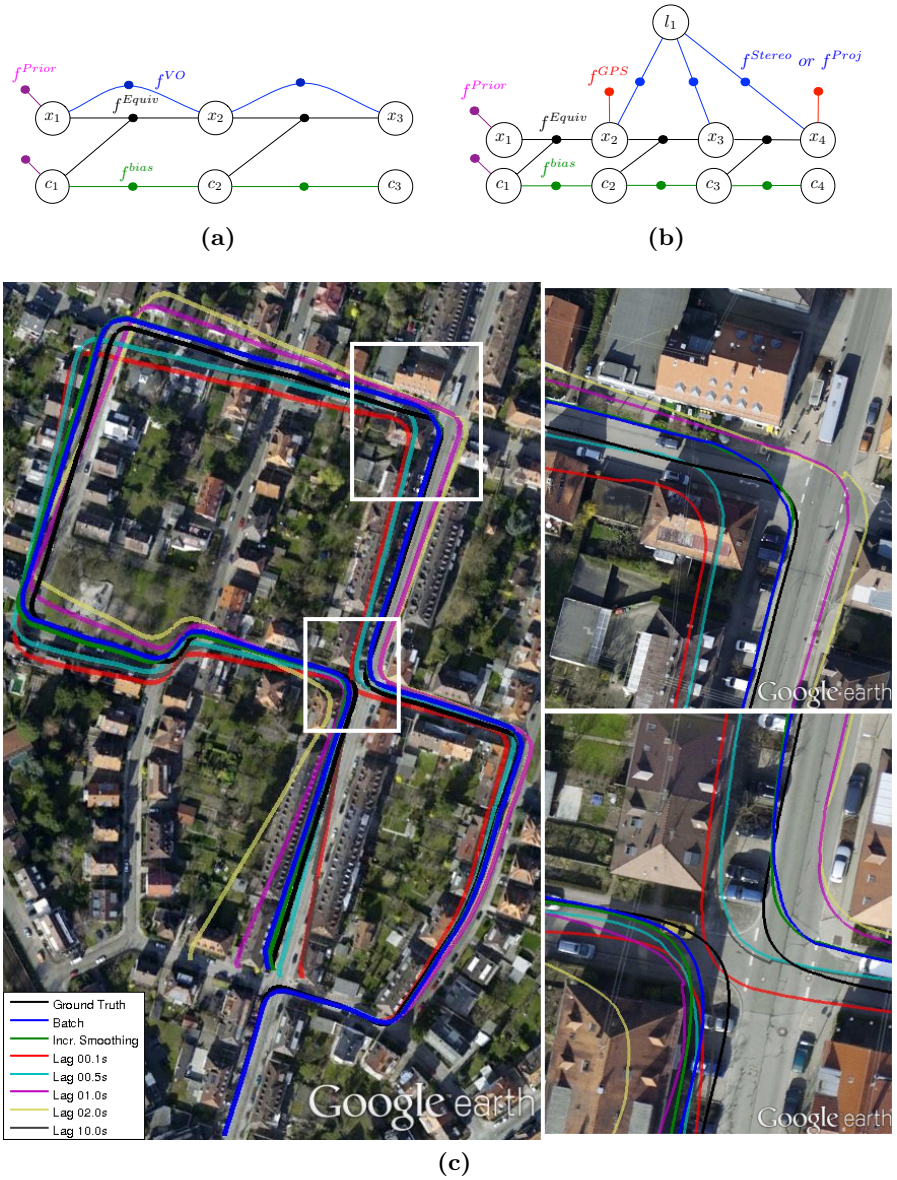


Figure 7.1: Aided inertial navigation [103]. (a) Factor graph for visual odometry integration with inertial bias estimation. (b) GPS integration (red factors) and/or stereo integration (blue). (c) Evaluation on KITTI sequence: Incremental smoothing performs almost as good as full batch optimization. Fixed-lag smoothing solutions are also shown for various lags.

In [102, 103] the plug and play capability of factor graphs is exploited in a navigation context (see Figure 7.1), with incremental smoothing and mapping (iSAM) as the inference engine (see Section 5). To deal with the typically large update rates of IMUs, one can use the idea by Lupton and Sukkarieh [143] to pre-integrate IMU measurements between lower-rate measurements from other sensors such as cameras and LIDARs. The use of IMU pre-integration is an excellent way to balance computational efficiency with fixed-lag smoothing in factor graphs.

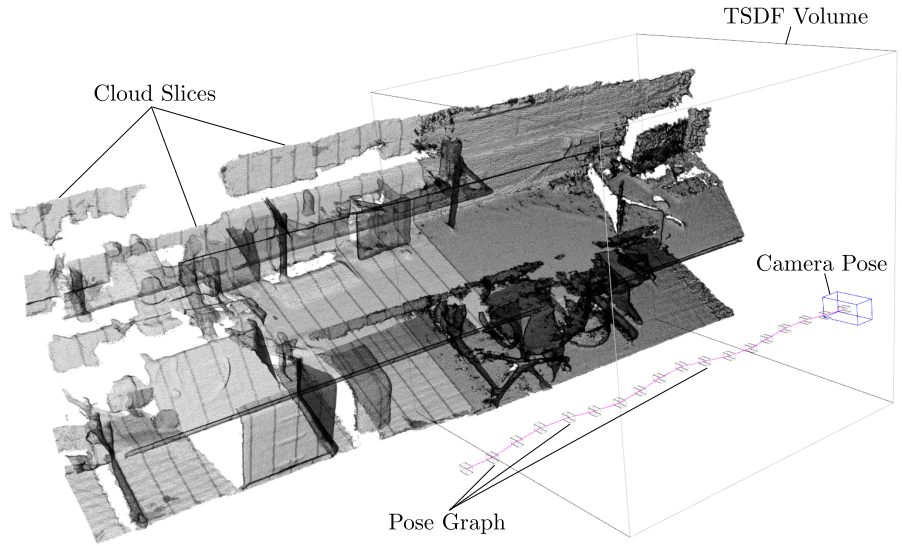
These ideas were also exploited by Forster et al. [67, 68] in the context of **visual-inertial odometry** or VIO. In these papers, a more sophisticated integration scheme is coupled with fixed lag smoothing to yield state-of-the-art performance. Leutenegger et al. [131, 132] also present VIO and SLAM algorithms that are visualized as factor graphs. And Usenko et al. [193] combine a semi-direct odometry algorithm with inertial information, proposing a factor graph approach that essentially follows the approach in [103]. However, instead of incremental smoothing, they apply marginalization to only estimate the most recent state, including inertial biases.

Finally, Mur-Artal and Tardos [149] added inertial sensing to their prior ORB-SLAM system, a visual SLAM approach based on ORB features detected in a video sequence. Here, inertial sensing is incorporated during tracking including bias estimation, while a local bundle adjustment is used when adding new keyframes to the map.

7.2 Dense 3D Mapping

While sparse point clouds are sufficient for keeping a robot localized, interaction with the environment requires a more complete map representation. Common choices for dense 3D map representations include point clouds, surfels (points with orientation), triangle meshes, and planar surfaces.

Whelan et al. [197, 196] present a dense SLAM solution, Kintinuous, that extends the local dense method KinectFusion to large environments, see Figure 7.2. Loop closures are applied to a factor graph



(a)

(b)

Figure 7.2: Real-time dense 3D mapping with Kintinuous [196]: (a) A dense volumetric representation (TSDF) is used locally and shifted along with the camera. The pose graph is drawn in pink with small cuboids as camera poses that have cloud slices associated with them. Loop closure on the pose graph is transferred to the point cloud via mesh deformation. (b) Sequence over two floors of an apartment with over six million vertices that was constructed including online loop closure. Small details such as bathroom fixtures and objects around the environment are clearly reconstructed.

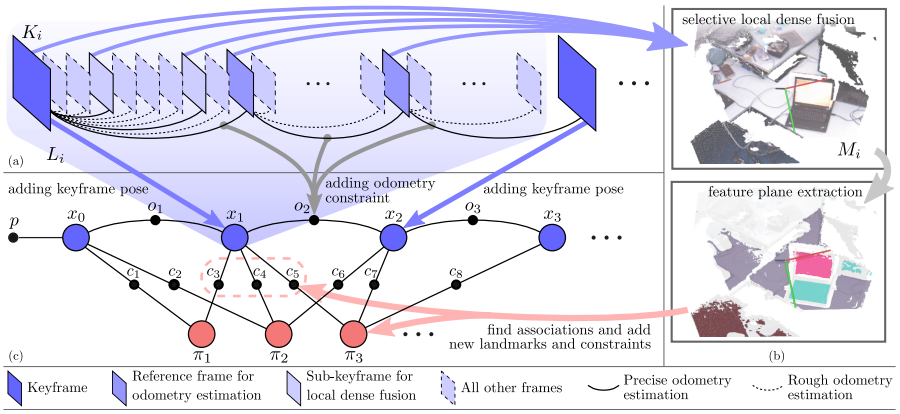
that represents the poses of the trajectory, and the corresponding corrections are transferred to the map, represented as a triangle mesh, by applying a mesh deformation.

Dense methods can also be achieved on the object level. Salas-Moreno et al. [170] present SLAM++, an object-based SLAM system where landmarks correspond to objects such as chairs in the environment, for which a prior dense model can be available. The entire inference problem is posed in terms of factor graph optimization.

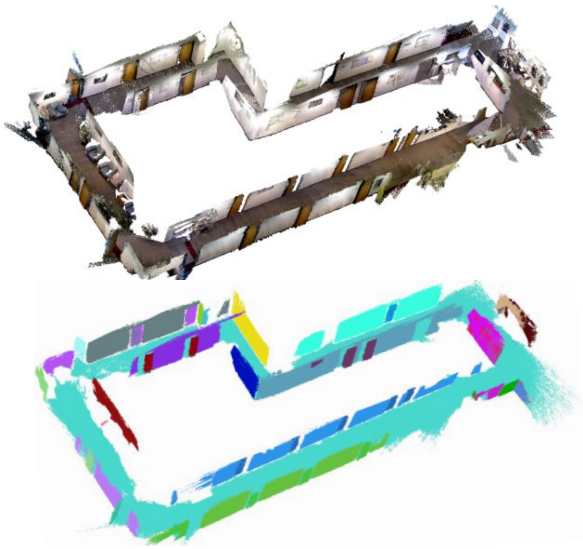
Planar surfaces are predominant in indoor environments, and this information can be exploited in SLAM. Trevor et al. [190] formulate this problem in the context of factor graph optimization. In [106] and [99], we use infinite planes as landmarks in the factor graph and provide a suitable retraction for optimization. The formulation is equivalent to structure-from-motion with infinite planes instead of point features. And, since the number of planar features is small per frame, the optimization can be done in real-time even for large environments. Figure 7.3 provides examples of maps based on infinite planes.

7.3 Field Robotics

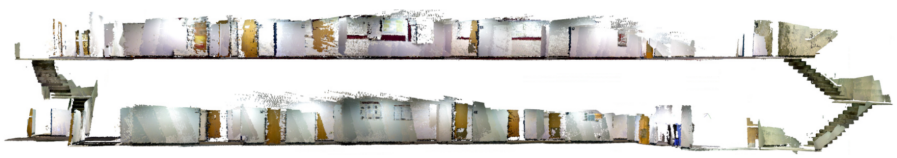
Factor graphs have proven to be useful in underwater robotics. Hover et al. [98] apply them in the inspection of ship hulls and harbor infrastructure with a hovering autonomous underwater vehicle (HAUV), see Figure 7.4. Factor graphs appear in localization and mapping with sonar [185, 194], vision [118], and the creation of sonar mosaics [158]. The sparsification algorithm by Carlevaris-Bianco et al. [23] described in Section 7.5 has been used for both long-term and multi-session operation in underwater scenarios. Beall et al. [7] use factor graphs to describe a large-scale bundle adjustment for underwater 3D reconstruction. Bichucher et al. [13] provide a bathymetric factor graph SLAM algorithm that uses sparse point clouds generated from a Doppler velocity log (DVL). Huang and Kaess [100] recently proposed a novel 3D reconstruction method from multiple 2D sonar images termed acoustic structure from motion (ASFM) that is formulated in terms of factor graphs.



(a)

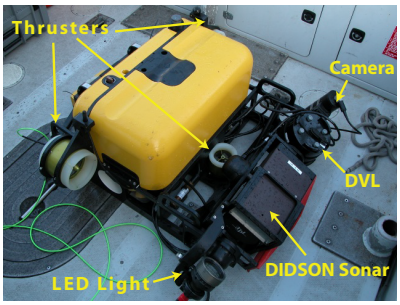


(b)



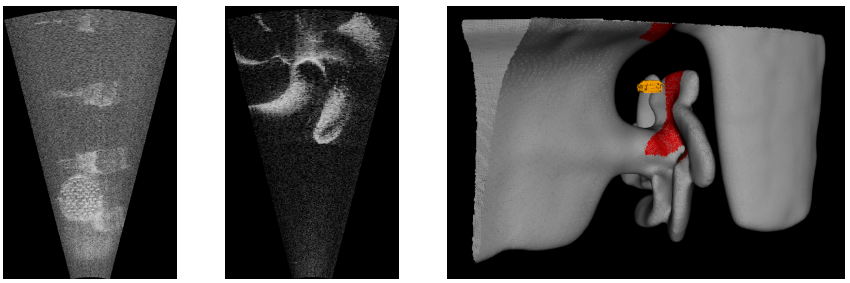
(c)

Figure 7.3: Dense mapping system based on planar surfaces [99]. The factor graph is similar to structure from motion, but with infinite planes instead of point features.

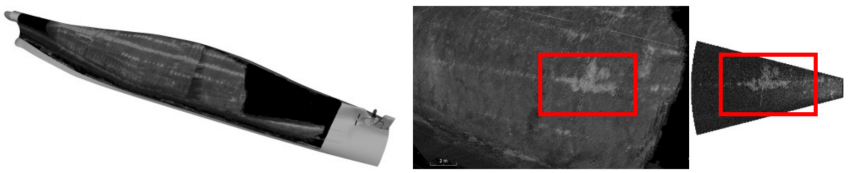


(a) Bluefin HAUV

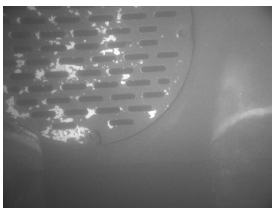
(b) SS Curtiss, San Diego



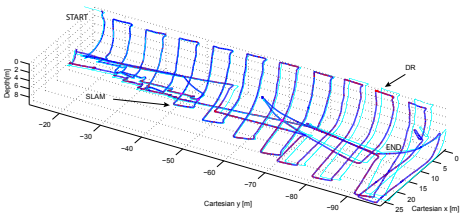
(c) Imaging sonar, profiling sonar, 3D ship model



(d) Imaging sonar mosaic

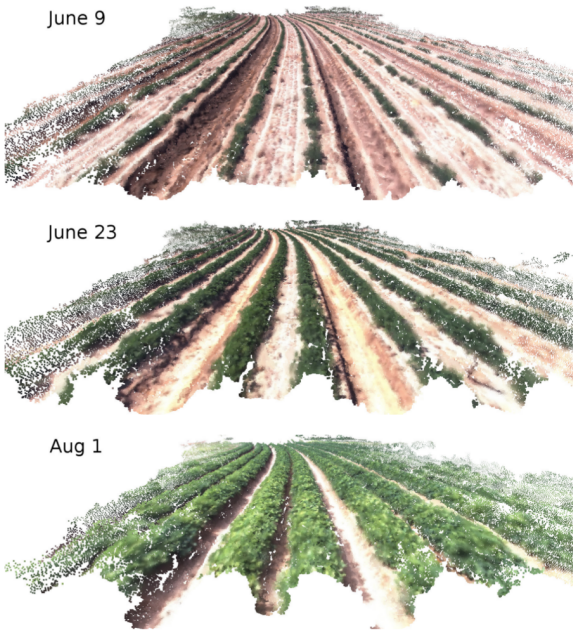


(e) Camera image

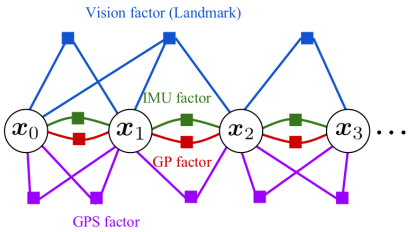


(f) Trajectory from factor graph

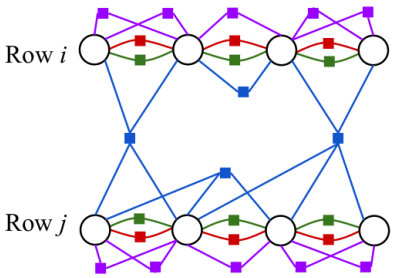
Figure 7.4: Ship hull and harbor infrastructure inspection with an underwater robot [98]. A factor graph formulation is used to fuse navigation and sonar data into 3D models. Camera images can be added when water turbidity allows.



(a)



(b)



(c)

Figure 7.5: Spatio-temporal reconstruction for crop monitoring [53]. (a) Reconstructed 4D model of a peanut field. (b) Factor graph of multi-sensor SLAM. (c) Factor graph connecting two rows with shared landmarks.

In the agriculture domain, Carlone et al. [24], Dong et al. [53] present a spatio-temporal reconstruction for crop monitoring. Figure 7.5 shows a 4D model of crop and the corresponding factor graph formulation for fusing different sensor data. Similarly, Griffith and Pradalier [88] use factor graphs for long-term natural environment monitoring, performing the optimization with the *GTSAM* library.

Factor graphs have also found application in space. Tweddle et al. [192] use factor graphs to model the rigid-body dynamics of a spinning object in space. Their system has been tested with miniature satellites called Spheres onboard the International Space Station.

Zhang et al. [205, 206] present a visual odometry method that is able to use sparse depth information, such as for a separate LIDAR sensor, or from an RGB-D camera with only parts of the scene within range of the sensor. The optimization is implemented using factor graphs and optimized using the iSAM library.

Many applications require a fine grained trajectory resolution: A spinning laser range sensor that also moves takes each measurement at a slightly different location along the trajectory, requiring a pose estimate every fraction of a second. While interpolation between poses in a factor graph is possible, it is natural to investigate continuous time representations that can directly connect measurements and graph optimization. Anderson et al. [4] present a factor-graph based solution using Gaussian process regression. In a follow-up paper [5] they provide a hierarchical wavelet decomposition that selectively provides higher temporal trajectory resolution where needed. An incremental algorithm for sparse GP regression has recently been presented by Yan et al. [202] in the context of continuous trajectory estimation.

7.4 Robust Estimation and Non-Gaussian Inference

It is well known that least-squares optimization is very sensitive to outliers, such as may arise from errors in data association, i.e. wrongly assigning a new observation to an earlier observation. Even with very conservative strategies, it is practically unavoidable that eventually such mistakes are made.

There are various strategies to accommodate outliers in the optimization. One common strategy is the use of robust error functions—error functions that are modified to increase less than quadratically far from the mean. Libraries for general least-squares optimization that use factor graphs including *GTSAM* [47], *iSAM* [111] and *g2o* [127] (the latter uses the terminology "hyper-graph" rather than factor graph) provide the option of using these robust error functions in the factors. Commonly used functions include the Huber and pseudo-Huber cost functions (the Tukey biweight estimator and the Cauchy function are less recommended).

As examples, in the long-term mapping example in Figure 5.1 on page 63 [104], we used pseudo-Huber to deal with the occasional outlier. Rosen et al. [168] introduced an incremental trust-region method that is particularly useful in combination with robust estimators, where standard Gauss-Newton does not perform well, while Levenberg-Marquardt cannot be incrementalized.

Alternatively, uncertain data association decisions can be modeled in the graphical model with a discrete variable. Various strategies have been proposed for solving such graphs, including switchable constraints by Sunderhauf et al. [179, 180], max mixture [156], and "realizing, reversing, recovering" (RRR) [128].

More recently, Segal and Reid [171] showed a method superior to the previous three methods, based on message passing on a junction tree. For the Gaussian case this is equivalent to our Bayes tree. In the presence of discrete variables they provide an iterative algorithm that first estimates the discrete variables, then the continuous ones.

While some of the robust estimation methods above can be regarded as non-Gaussian, they all rely on Gaussian inference and return a Gaussian density as posterior. In contrast, non-Gaussian inference make no such assumption on the posterior and can therefore represent multi-modal posteriors, i.e. ambiguous solutions, where each mode additionally can be non-Gaussian.

Fourie et al. [69] recently presented a novel approach to SLAM that combines nonparametric belief propagation with the Bayes tree formulation, see Figure 7.6. While the approach is necessarily approximate,

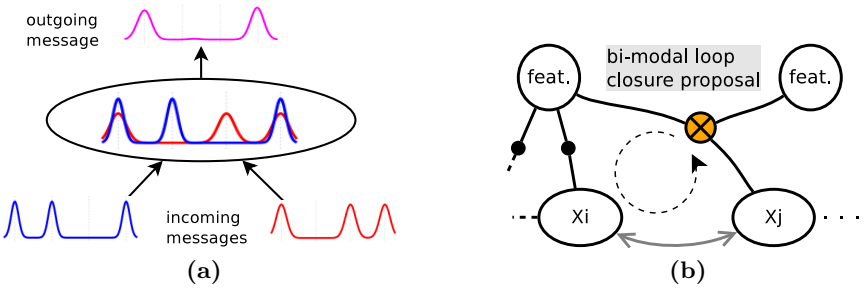


Figure 7.6: Non-Gaussian inference on the Bayes tree using kernel density estimates [69]. (a) Intermediate steps and posterior are multimodal. (b) Data association can be transformed into a multimodal estimation by marginalizing out the discrete variable from the factor graph.

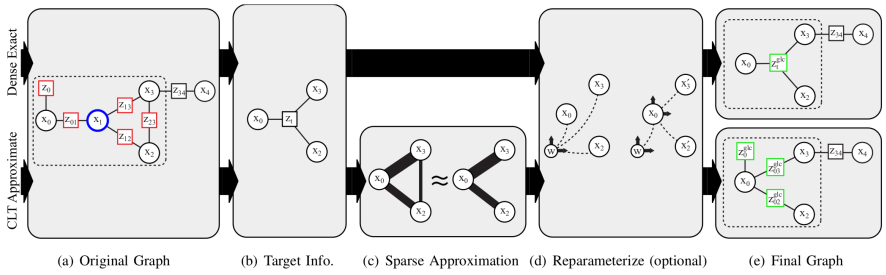
its feasibility has already been demonstrated for a SLAM problem with thousands of variables and high ambiguity with a theoretical number of modes larger than 2^{400}. Here, in contrast to other approaches described above, data association has been converted into a continuous, but multimodal inference problem by integrating out any discrete variables such as those related to uncertainty in data association.

7.5 Long-term Operation and Sparsification

Even with efficient sparse and incremental methods as discussed in this article, managing the growth in computational cost for long-term mapping requires additional strategies.

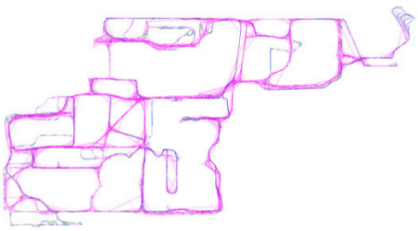
The smoothing and mapping approach to SLAM suffers from unbounded growth over time, not just in the size of the explored environment. Johannsson et al. [104] addressed this problem by introducing the **reduced pose graph**, which reuses previously generated poses in the same area, see Figure 5.1 on page 63. Using this method, growth of the factor graph is kept within bounds during construction.

An alternative and more general approach is to simplify a given graph by **sparsifying** the graph in order to keep computation manageable. Carlevaris-Bianco et al. [23, 21, 22] presented such a method that provides a consistent sparsification. In Figure 7.7, the original factor graph is replaced by a much simpler version that closely approximates

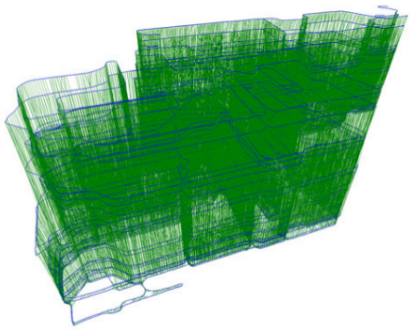


(a) GLC node removal and sparsification algorithm.

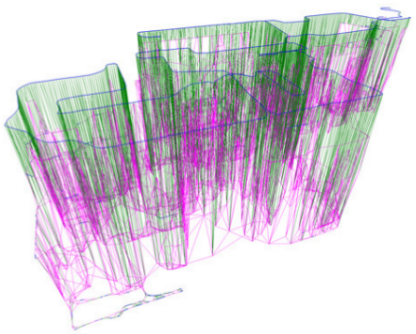
(b) Full graph (top view)



(c) GLC reduced graph (top view)



(d) Full graph (time scaled)



(e) GLC reduced graph (time scaled)

Figure 7.7: Generic linear constraints (GLC) node removal with optional Chow-Liu tree sparsification [23]. (a) Overview of the algorithm. (b)-(e) The example shows extensive outdoor data from 27 mapping sessions over a period of 15 months. The oblique time scaled views visualize the more than ten-fold reduction in both the number of nodes and factors.

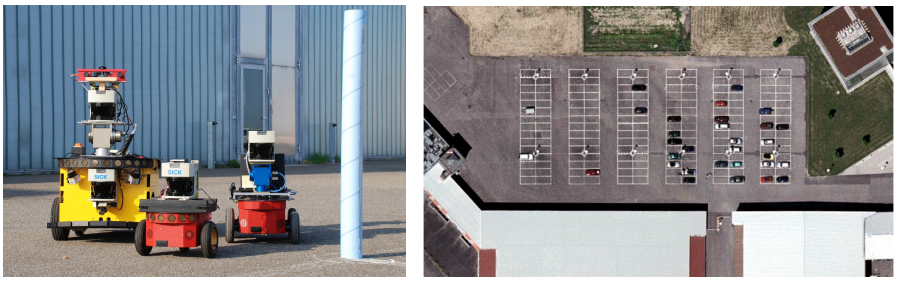


Figure 7.8: Three robots and a synthetic landmark used in the multi-robot mapping experiment from [35] (left), and an aerial view of the parking lot in which the experiment was performed (right).

the original system while maintaining consistency. Sparsification can also be applied in combination with marginalization of variables that are no longer needed, which is another way to keep the remaining graph sparse. Mazuran et al. [146] recently extended this approach by adding a convex optimization to better approximate nonlinear measurement functions throughout the sparsification.

7.6 Large-scale and Distributed SLAM

As discussed in Section 4, large scale mapping can be approached by a divide-and-conquer approach. Ni et al. [154] introduced the tectonic smoothing and mapping (TSAM) algorithm which uses a single-level partitioning of the associated factor graph, and the concept of base nodes that allow a partition to move rigidly when optimizing the separator between partitions. The ordering within each subgraph was computed using the greedy ordering method COLAMD [3, 40]. This was later extended in [152] to a full nested-dissection approach where the graph is *recursively* subdivided. As an example, Figure 4.8 on page 59 shows the recursive partitioning of the well-known Victoria park dataset using this method. Partitioning at each level was done using the METIS package [114]. Similar ideas were applied by the same authors in the area of large-scale 3D reconstruction, in a method called HyperSFM [153], but there the partitioning was done using hyper-graph partitioning, again using METIS.

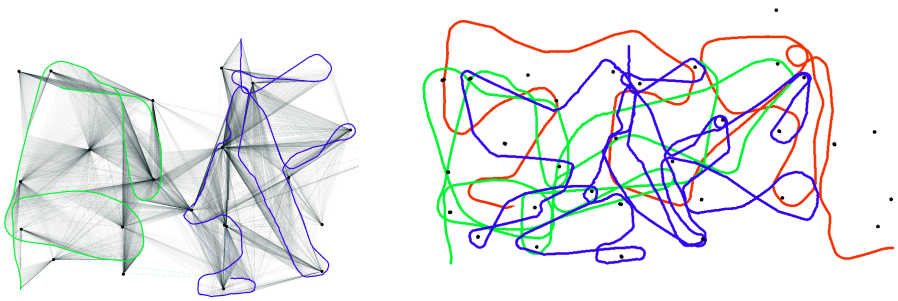


Figure 7.9: Left: data association between 2 (out of 3) trajectories in the middle of an experiment with the three robots from Figure 7.8. Measurement factors are shown as translucent dark lines, trajectories as blue and green paths, and optimized landmarks as black circles. Right: full final results for all three robots, at the end of the experiment, this time omitting factors for clarity.

When multiple robotic platforms need to collaborate on solving a large mapping problem, as illustrated in Figure 7.8, an obvious way of partitioning the corresponding factor graphs is across multiple vehicles. Factors originate from measurements, and hence it makes sense to keep the factors local to the platform they were taken on. This is the approach adopted by DDF-SAM, introduced by Cunningham et al. [34]. Each robot optimizes a local factor graph, and communicates information about shared variables of interest to the other robots in the form of a marginal density (a factor, itself). Coordinate frame transformations are handled using a novel constrained optimization scheme.

Results for the experiment with the robots from Figure 7.8 are shown in Figure 7.9, as reported in [35]. In a subsequent iteration of this framework, DDF-SAM 2 [33], the tedious bookkeeping in DDF-SAM is avoided by introducing **anti-factors,** a new factor type that can be used to subtract out information that would otherwise be double-counted in the communication exchange.

Toohey et al. [189] also present a decentralized cooperative localization algorithm formulated in terms of factor graphs. The algorithm achieves constant message size over time by compressing the graph to only the relevant information needed for the receiver based on inter-vehicle measurements.

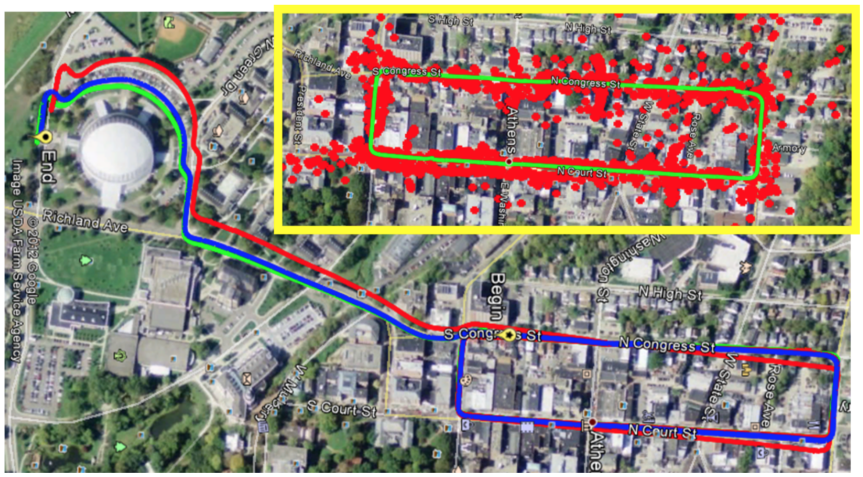


Figure 7.10: Large City Navigation Scenario (3.3 kilometers) from the DARPA ASPN project: Ground truth (blue), solution using only 1-step short-term smoother (red, 3D RMS error: 19.19 meters), solution using the CSM scheme (green, 3D RMS error: 5.85 meters), and the 3D landmarks constructed during the first loop (red points inside the highlighted yellow box).

In a multi-robot setting, estimating the initially unknown transformations between the robot frames is a challenging problem. Kim et al. [119] introduce anchor nodes as a way to establish a common reference frame in the factor graph in a centralized fashion. Indelman et al. [101] provide a distributed algorithm to establish a common reference frame based on EM, which was successfully applied to the real-time distributed mapping of entire buildings using micro air vehicles [54].

In the case of a single camera or robot, cross-registration between multiple sessions can be advantageous. McDonald et al. [147] presented a visual SLAM system for large scale environments that implements this idea. The multi-session capability can also be used to provide additional robustness: If camera tracking is lost, a new session is started and eventually connected back to prior sessions.

Yet another way to distribute computation is among threads on the same robot: a high-priority thread that keeps accurate track of the robot's current state, and a lower-priority thread that refines a more complete map in the background. This was formalized for general

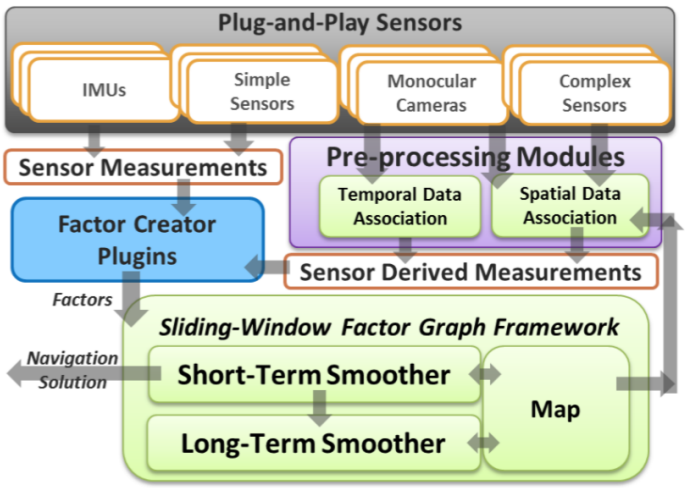


Figure 7.11: Plug-and-play navigation architecture developed in the DARPA ASPN project, as detailed in [29, 30].

factor-graph inference methods in [112, 200] in a technique called concurrent filtering and smoothing, and applied successfully in a number of challenging scenarios in a DARPA funded project (see Figure 7.10). The scenarios included land, sea, and air trials, and used many different sensors that could turn on or off at any given moment. The threaded, concurrent nature of the computation afforded by this method was crucial in attaining real-time performance in all these cases, even when a fixed-lag smoother was substituted for the filter component as done by Chiu et al. [29], termed concurrent smoothing and mapping (CSM).

The latter paper [29] also demonstrated some of the software engineering advantages of working with and thinking about factor graphs. Figure 7.11 shows the plug-and-play navigation architecture developed in the DARPA ASPN project, which was made easy because each type of measurement essentially corresponds to a single factor type. At the bottom of the figure, the CSM scheme that combines a long-term smoother with a short term fixed-lag smoother is visible.

7.7 Summary

It is clear that the above is not meant to be a comprehensive review, and factor graphs have been used in a variety of other settings, from calibration [126] and occupancy grid mapping [51] to connecting language to robotic perception and action [195]. However, from the wide range of applications for which factor graphs have been the method of choice for representation, it becomes clear that they have made a large impact as a unifying representation in robotics. We feel that this is because they really map well to the actual measurements performed, therefore providing a natural representation of the optimization problems arising in robot perception.

Acknowledgements

We would like to thank the co-authors on the many papers that the material above was drawn from or inspired by: Sean Anderson, Tucker Balch, Tim Barfoot, Chris Beall, Byron Boots, Wolfram Burgard, John Burnham, Nicholas Carlevaris-Bianco, Luca Carlone, Han-Pang Chiu, Jason Corso, Alexander Cunningham, Vikas Dhiman, Jing Dong, Ryan Eustice, Maurice Fallon, Stefano Fenu, Christian Forster, Dehann Fourie, Ming Hsiao, Viorela Ila, Vadim Indelman, Hordur Johannsson, Matthew Johnson-Roberson, Zia Khan, Alex Kipp, Peter Krauthausen, Rakesh Kumar, Abhijit Kundu, John Leonard, Ian Mahon, John McDonald, Nathan Michael, Erik Nelson, Kai Ni, Paul Ozog, Manohar Paluri, Glen Rains, Ananth Ranganathan, Richard Roberts, Supun Samarasekera, Davide Scaramuzza, Drew Steedly, Pedro Teixeira, Giancarlo Troni, Eric Westman, Thomas Whelan, Stefan Williams, Stephen Williams, Kai Wurm, Guofeng Zhang, and Xun Zhou. In addition, we are grateful to the following people that provided comments on earlier drafts of the document: Gareth Cross, Saumitro Dasgupta, and Hayk Martirosyan at Skydio; Suresh Kannan at NodeIn; Ivan Jimenez at Georgia Tech; and the students of the Fall 2016 class "Statistical Techniques in Robotics" (16-831) and Spring 2017 class "Robot Localization and Mapping" (16-833) at Carnegie Mellon University. Finally, we gratefully acknowledge support from the various funding agencies who made this work possible over the years, including ARL, DARPA, NSF, ONR, and the U.S. Fullbright program.

Bibliography

[1] Absil, P.-A., Mahony, R., and Sepulchre, R. (2007). *Optimization Algorithms on Matrix Manifolds*. Princeton University Press, Princeton, NJ, USA.

[2] Agarwal, P. and Olson, E. (2012). Variable reordering strategies for SLAM. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.

[3] Amestoy, P., Davis, T., and Duff, I. (1996). An approximate minimum degree ordering algorithm. *SIAM Journal on Matrix Analysis and Applications*, 17(4):886–905.

[4] Anderson, S., Barfoot, T., Tong, C., and Särkkä, S. (2015). Batch nonlinear continuous-time trajectory estimation as exactly sparse Gaussian process regression. *Autonomous Robots*, 39(3):221–238.

[5] Anderson, S., Dellaert, F., and Barfoot, T. (2014). A hierarchical wavelet decomposition for continuous-time SLAM. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.

[6] Bailey, T. and Durrant-Whyte, H. (2006). Simultaneous localisation and mapping (SLAM): Part II state of the art. *Robotics & Automation Magazine*.

[7] Beall, C., Dellaert, F., Mahon, I., and Williams, S. (2011). Bundle adjustment in large-scale 3D reconstructions based on underwater robotic surveys. In *OCEANS, 2011 IEEE-Spain*, pages 1–6. IEEE.

[8] Beeri, C., Fagin, R., Maier, D., Mendelzon, A., Ullman, J., and Yannakakis, M. (1981). Properties of acyclic database schemes. In *ACM Symp. on Theory of Computing (STOC)*, pages 355–362, New York, NY, USA. ACM Press.

[9] Bellman, R. and Dreyfus, S. (1962). *Applied Dynamic Programming*. Princeton University Press.

[10] Bertele, U. and Brioschi, F. (1972a). *Nonserial Dynamic Programming*. Academic Press.

[11] Bertele, U. and Brioschi, F. (1972b). On the theory of the elimination process. *J. Math. Anal. Appl.*, 35(1):48–57.

[12] Bertele, U. and Brioschi, F. (1973). On nonserial dynamic programming. *J. Combinatorial Theory*, 14:137–148.

[13] Bichucher, V., Walls, J., Ozog, P., Skinner, K., and Eustice, R. (2015). Bathymetric factor graph SLAM with sparse point cloud alignment. In *OCEANS, 2015. MTS/IEEE Conference and Exhibition*.

[14] Bierman, G. (1977). *Factorization methods for discrete sequential estimation*, volume 128 of *Mathematics in Science and Engineering*. Academic Press, New York.

[15] Bierman, G. (1978). An application of the square-root information filter to large scale linear interconnected systems. *IEEE Trans. Automat. Contr.*, 23(1):91–93.

[16] Bishop, C. (2006). *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer-Verlag, Secaucus, NJ, USA.

[17] Blair, J. and Peyton, B. (1993). An introduction to chordal graphs and clique trees. In [80], pages 1–27.

[18] Brown, D. C. (1976). The bundle adjustment - progress and prospects. *Int. Archives Photogrammetry*, 21(3).

[19] Cadena, C., Carlone, L., Carrillo, H., Latif, Y., Scaramuzza, D., Neira, J., Reid, I., and Leonard, J. J. (2016). Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Trans. Robotics*, 32(6):1309–1332.

[20] Cannings, C., Thompson, E., and Skolnick, M. (1978). Probability functions on complex pedigrees. *Advances in Applied Probability*, 10:26–61.

[21] Carlevaris-Bianco, N. and Eustice, R. M. (2013a). Generic factor-based node marginalization and edge sparsification for pose-graph SLAM. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 5728–5735.

[22] Carlevaris-Bianco, N. and Eustice, R. M. (2013b). Long-term simultaneous localization and mapping with generic linear constraint node removal. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, Tokyo, Japan.

[23] Carlevaris-Bianco, N., Kaess, M., and Eustice, R. (2014). Generic factor-based node removal: Enabling long-term SLAM. *IEEE Trans. Robotics*, 30(6):1371–1385.

[24] Carlone, L., Dong, J., Fenu, S., Rains, G., and Dellaert, F. (2015). Towards 4D crop analysis in precision agriculture: Estimating plant height and crown radius over time via expectation-maximization. In *ICRA Workshop on Robotics in Agriculture*.

[25] Carré, B. A. (1971). An algebra for network routing problems. *J. Inst. Math. Appl.*, 7:273–294.

[26] Castellanos, J., Montiel, J., Neira, J., and Tardós, J. (1999). The SPmap: A probabilistic framework for simultaneous localization and map building. *IEEE Trans. Robot. Automat.*, 15(5):948–953.

[27] Chatila, R. and Laumond, J.-P. (1985). Position referencing and consistent world modeling for mobile robots. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 138–145.

[28] Chen, Y., Davis, T., Hager, W., and Rajamanickam, S. (2008). Algorithm 887: CHOLMOD, supernodal sparse Cholesky factorization and update/downdate. *ACM Trans. Math. Softw.*, 35(3):22:1–22:14.

[29] Chiu, H., S., Dellaert, F., Samarasekera, S., and Kumar, R. (2013). Robust vision-aided navigation using sliding-window factor graphs. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, Karlsruhe; Germany.

[30] Chiu, H., Zhou, X., Carlone, L., Dellaert, F., Samarasekera, S., and Kumar, R. (2014). Constrained optimal selection for multi-sensor robot navigation using plug-and-play factor graphs. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, Hong Kong.

[31] Cooper, M. and Robson, S. (1996). Theory of close range photogrammetry. In Atkinson, K., editor, *Close range photogrammetry and machine vision*, chapter 1, pages 9–51. Whittles Publishing.

[32] Cowell, R. G., Dawid, A. P., Lauritzen, S. L., and Spiegelhalter, D. J. (1999). *Probabilistic Networks and Expert Systems*. Statistics for Engineering and Information Science. Springer-Verlag.

[33] Cunningham, A., Indelman, V., and Dellaert, F. (2013). DDF-SAM 2.0: Consistent distributed smoothing and mapping. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, Karlsruhe, Germany.

[34] Cunningham, A., Paluri, M., and Dellaert, F. (2010). DDF-SAM: Fully distributed SLAM using constrained factor graphs. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.

[35] Cunningham, A., Wurm, K., Burgard, W., and Dellaert, F. (2012). Fully distributed scalable smoothing and mapping with robust multi-robot data association. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, St. Paul, MN.

[36] Cuthill, E. and McKee, J. (1969). Reducing the bandwidth of sparse symmetric matrices. In *Proc. of the 1969 24th ACM national conference*, pages 157–172, New York, NY, USA. ACM Press.

[37] D'Ambrosio, B. (1994). Symbolic probabilistic inference in large BN2O networks. In *Proc.* 10^{th} *Conf. on Uncertainty in AI (UAI)*, pages 128–135, Seattle, WA.

[38] Darwiche, A. (2009). *Modeling and Reasoning with Bayesian Networks*. Cambridge University Press.

[39] Davis, T. (2011). Algorithm 915: SuiteSparseQR, a multifrontal multithreaded sparse QR factorization package. *ACM Trans. Math. Softw.*, 38(1):8:1–8:22.

[40] Davis, T., Gilbert, J., Larimore, S., and Ng, E. (2004). A column approximate minimum degree ordering algorithm. *ACM Trans. Math. Softw.*, 30(3):353–376.

[41] Davis, T. and Hager, W. (1996). Modifying a sparse Cholesky factorization. *SIAM Journal on Matrix Analysis and Applications*, 20(3):606–627.

[42] Dechter, R. (1996). Bucket elimination: A unifying framework for several probabilistic inference algorithms. In *Proc.* 12^{th} *Conf. on Uncertainty in AI (UAI)*, Portland, OR.

[43] Dechter, R. (1998). Bucket Elimination: A unifying framework for reasoning. In Jordan, M., editor, *Learning in Graphical Models*, pages 75–104. Kluwer Academic Press. Also published by MIT Press, 1999.

[44] Dechter, R. (1999). Bucket Elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113(1-2):41–85.

[45] Dechter, R. and Pearl, J. (1987). Network-based heuristics for constraint-satisfaction problems. *Artificial Intelligence*, 34(1):1–38.

[46] Dellaert, F. (2005). Square Root SAM: Simultaneous location and mapping via square root information smoothing. In *Robotics: Science and Systems (RSS)*.

[47] Dellaert, F. (2012). Factor graphs and GTSAM: A hands-on introduction. Technical Report GT-RIM-CP&R-2012-002, Georgia Institute of Technology.

[48] Dellaert, F. and Kaess, M. (2006). Square Root SAM: Simultaneous localization and mapping via square root information smoothing. *Intl. J. of Robotics Research*, 25(12):1181–1203.

[49] Dellaert, F., Kipp, A., and Krauthausen, P. (2005). A multifrontal QR factorization approach to distributed inference applied to multi-robot localization and mapping. In *Proc. 22^{nd} AAAI National Conference on AI*, pages 1261–1266, Pittsburgh, PA.

[50] Dennis, J. and Schnabel, R. (1983). *Numerical methods for unconstrained optimization and nonlinear equations*. Prentice-Hall.

[51] Dhiman, V., Kundu, A., Dellaert, F., and Corso, J. (2014). Modern map inference methods for accurate and faster occupancy grid mapping on higher order factor graphs. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.

[52] Dissanayake, M., Newman, P., Durrant-Whyte, H., Clark, S., and Csorba, M. (2001). A solution to the simultaneous localization and map building (SLAM) problem. *IEEE Trans. Robot. Automat.*, 17(3):229–241.

[53] Dong, J., Burnham, J., Boots, B., Rains, G., and Dellaert, F. (2017). 4D crop monitoring: Spatio-temporal reconstruction for agriculture. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.

[54] Dong, J., Nelson, E., Indelman, V., Michael, N., and Dellaert, F. (2015). Distributed real-time cooperative localization and mapping using an uncertainty-aware expectation maximization approach. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*.

[55] Doucet, A., de Freitas, N., and Gordon, N., editors (2001). *Sequential Monte Carlo Methods In Practice*. Springer-Verlag, New York.

[56] Duckett, T., Marsland, S., and Shapiro, J. (2002). Fast, on-line learning of globally consistent maps. *Autonomous Robots*, 12(3):287–300.

[57] Duff, I. S. and Reid, J. K. (1983). The multifrontal solution of indefinite sparse symmetric linear systems. *ACM Trans. Math. Softw.*, 9(3):302–325.

[58] Durrant-Whyte, H. (1988). Uncertain geometry in robotics. *IEEE Trans. Robot. Automat.*, 4(1):23–31.

[59] Durrant-Whyte, H. and Bailey, T. (2006). Simultaneous localisation and mapping (SLAM): Part I the essential algorithms. *Robotics & Automation Magazine*.

[60] Estrada, C., Neira, J., and Tardós, J. (2005). Hierarchical SLAM: Real-time accurate mapping of large environments. *IEEE Trans. Robotics*, 21(4):588–596.

[61] Eustice, R., Singh, H., and Leonard, J. (2005). Exactly sparse delayed-state filters. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 2417–2424.

[62] Fagin, R., Mendelzon, A., and Ullman, J. (1982). A simplied universal relation assumption and its properties. *ACM Trans. Database Syst.*, 7(3):343–360.

[63] Farrell, J. (2008). *Aided Navigation: GPS with High Rate Sensors*. McGraw-Hill.

[64] Faugeras, O. (1993). *Three-dimensional computer vision: A geometric viewpoint*. The MIT press, Cambridge, MA.

[65] Fiduccia, C. and Mattheyses, R. (1982). A linear time heuristic for improving network partitions. In *Proc. 19th IEEE Design Automation and Conference*, pages 175–181.

[66] Forney, Jr, G. (2001). Codes on graphs: Normal realizations. *IEEE Trans. Inform. Theory*, 47(2):520–548.

[67] Forster, C., Carlone, L., Dellaert, F., and Scaramuzza, D. (2015). IMU preintegration on manifold for efficient visual-inertial maximum-a-posteriori estimation. In *Robotics: Science and Systems (RSS)*.

[68] Forster, C., Carlone, L., Dellaert, F., and Scaramuzza, D. (2016). On-manifold preintegration for real-time visual-inertial odometry. *IEEE Trans. Robotics*.

[69] Fourie, D., Leonard, J., and Kaess, M. (2016). A nonparametric belief solution to the Bayes tree. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, Daejeon, Korea.

[70] Frese, U. (2005). Treemap: An $O(\log n)$ algorithm for simultaneous localization and mapping. In *Spatial Cognition IV*, pages 455–476. Springer Verlag.

[71] Frese, U., Larsson, P., and Duckett, T. (2005). A multilevel relaxation algorithm for simultaneous localisation and mapping. *IEEE Trans. Robotics*, 21(2):196–207.

[72] Frese, U. and Schröder, L. (2006). Closing a million-landmarks loop. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 5032–5039.

[73] Freuder, E. C. (1982). A sufficient condition for backtrack-free search. *J. ACM*, 29(1):24–32.

[74] Frey, B., Kschischang, F., Loeliger, H.-A., and Wiberg, N. (1997). Factor graphs and algorithms. In *Proc. 35th Allerton Conf. Communications, Control, and Computing*, pages 666–680.

[75] Gauss, C. (1809). *Theoria Motus Corporum Coelestium in Sectionibus Conicis Solem Mabientium [Theory of the Motion of the Heavenly Bodies Moving about the Sun in Conic Sections]*. Perthes and Besser, Hamburg, Germany. English translation available at http://name.umdl.umich.edu/AGG8895.0001.001.

[76] Gauss, C. (1810). Disquisitio de elementis ellipticis Palladis [Disquisition on the elliptical elements of Pallas]. *Göttingische gelehrte Anzeigen*.

[77] Gentleman, W. (1973). Least squares computations by Givens transformations without square roots. *IMA J. of Appl. Math.*, 12:329–336.

[78] George, A. (1973). Nested dissection of a regular finite element mesh. *SIAM Journal on Numerical Analysis*, 10(2):345–363.

[79] George, A., Liu, J., and E., N. (1984). Row-ordering schemes for sparse Givens transformations. I. Bipartite graph model. *Linear Algebra Appl*, 61:55–81.

[80] George, J., Gilbert, J., and Liu, J.-H., editors (1993). *Graph Theory and Sparse Matrix Computations*, volume 56 of *IMA Volumes in Mathematics and its Applications*. Springer-Verlag, New York.

[81] Gilbert, J. and Ng, E. (1993). Predicting structure in nonsymmetric sparse matrix factorizations. In [80].

[82] Gilks, W., Richardson, S., and Spiegelhalter, D., editors (1996). *Markov chain Monte Carlo in practice*. Chapman and Hall.

[83] Gill, P., Golub, G., Murray, W., and Saunders, M. (1974). Methods for modifying matrix factorizations. *Mathematics and Computation*, 28(126):505–535.

[84] Golub, G. and Loan, C. V. (1996). *Matrix Computations*. Johns Hopkins University Press, Baltimore, third edition.

[85] Golub, G. and Plemmons, R. (1980). Large-scale geodetic least-squares adjustment by dissection and orthogonal decomposition. *Linear Algebra and Its Applications*, 34:3–28.

[86] Goodman, N. and Shmueli, O. (1982). Tree queries: a simple class of relational queries. *ACM Trans. Database Syst.*, 7(4):653–677.

[87] Granshaw, S. (1980). Bundle adjustment methods in engineering photogrammetry. *Photogrammetric Record*, 10(56):181–207.

[88] Griffith, S. and Pradalier, C. (2017). Survey registration for long-term natural environment monitoring. *J. of Field Robotics*, 34(1):188–208.

[89] Grisetti, G., Kuemmerle, R., and Ni, K. (2012). Robust optimization of factor graphs by using condensed measurements. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.

[90] Grisetti, G., Kuemmerle, R., Stachniss, C., Frese, U., and Hertzberg, C. (2010). Hierarchical optimization on manifolds for online 2D and 3D mapping. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, Anchorage, Alaska.

[91] Guivant, J. and Nebot, E. (2001). Optimization of the simultaneous localization and map building algorithm for real time implementation. *IEEE Trans. Robot. Automat.*, 17(3):242–257.

[92] Guivant, J., Nebot, E., Nieto, J., and Masson, F. (2004). Navigation and mapping in large unstructured environments. *Intl. J. of Robotics Research*, 23:449–472.

[93] Gupta, A., Karypis, G., and Kumar, V. (1997). Highly scalable parallel algorithms for sparse matrix factorization. *IEEE Trans. Parallel and Distributed Systems*, 8(5):502–520.

[94] Gutmann, J.-S. and Konolige, K. (2000). Incremental mapping of large cyclic environments. In *IEEE Intl. Symp. on Computational Intelligence in Robotics and Automation (CIRA)*, pages 318–325.

[95] Hartley, R. and Zisserman, A. (2000). *Multiple View Geometry in Computer Vision*. Cambridge University Press.

[96] Hartley, R. I. and Zisserman, A. (2004). *Multiple View Geometry in Computer Vision*. Cambridge University Press, second edition.

[97] Heggernes, P. and Matstoms, P. (1996). Finding good column orderings for sparse QR factorization. In *Second SIAM Conference on Sparse Matrices*.

[98] Hover, F., Eustice, R., Kim, A., Englot, B., Johannsson, H., Kaess, M., and Leonard, J. (2012). Advanced perception, navigation and planning for autonomous in-water ship hull inspection. *Intl. J. of Robotics Research*, 31(12):1445–1464.

[99] Hsiao, M., Westman, E., Zhang, G., and Kaess, M. (2017). Keyframe-based dense planar SLAM. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, Singapore.

[100] Huang, T. and Kaess, M. (2015). Towards acoustic structure from motion for imaging sonar. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 758–765, Hamburg, Germany.

[101] Indelman, V., Nelson, E., Dong, J., Michael, N., and Dellaert, F. (2016). Incremental distributed inference from arbitrary poses and unknown data association: Using collaborating robots to establish a common reference. *IEEE Control Systems Magazine (CSM), Special Issue on Distributed Control and Estimation for Robotic Vehicle Networks*, 36(2):41–74.

[102] Indelman, V., Wiliams, S., Kaess, M., and Dellaert, F. (2012). Factor graph based incremental smoothing in inertial navigation systems. In *Intl. Conf. on Information Fusion, FUSION*.

[103] Indelman, V., Wiliams, S., Kaess, M., and Dellaert, F. (2013). Information fusion in navigation systems via factor graph based incremental smoothing. *Robotics and Autonomous Systems*, 61(8):721–738.

[104] Johannsson, H., Kaess, M., Fallon, M., and Leonard, J. (2013). Temporally scalable visual SLAM using a reduced pose graph. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 54–61, Karlsruhe, Germany.

[105] Julier, S. and Uhlmann, J. (2001). A counter example to the theory of simultaneous localization and map building. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, volume 4, pages 4238–4243.

[106] Kaess, M. (2015). Simultaneous localization and mapping with infinite planes. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 4605–4611, Seattle, WA.

[107] Kaess, M., Ila, V., Roberts, R., and Dellaert, F. (2010). The Bayes tree: An algorithmic foundation for probabilistic robot mapping. In *Intl. Workshop on the Algorithmic Foundations of Robotics*.

[108] Kaess, M., Johannsson, H., Roberts, R., Ila, V., Leonard, J., and Dellaert, F. (2011). iSAM2: Incremental smoothing and mapping with fluid relinearization and incremental variable reordering. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, Shanghai, China.

[109] Kaess, M., Johannsson, H., Roberts, R., Ila, V., Leonard, J., and Dellaert, F. (2012a). iSAM2: Incremental smoothing and mapping using the Bayes tree. *Intl. J. of Robotics Research*, 31:217–236.

[110] Kaess, M., Ranganathan, A., and Dellaert, F. (2007). Fast incremental square root information smoothing. In *Intl. Joint Conf. on AI (IJCAI)*, pages 2129–2134, Hyderabad, India.

[111] Kaess, M., Ranganathan, A., and Dellaert, F. (2008). iSAM: Incremental smoothing and mapping. *IEEE Trans. Robotics*, 24(6):1365–1378.

[112] Kaess, M., Wiliams, S., Indelman, V., Roberts, R., Leonard, J., and Dellaert, F. (2012b). Concurrent filtering and smoothing. In *Intl. Conf. on Information Fusion, FUSION*.

[113] Karczmarczuk, J. (1998). Functional differentiation of computer programs. In *Intl. Conf. on Functional Programming (ICFP)*, pages 195–203.

[114] Karypis, G. and Kumar, V. (1998). Multilevel algorithms for multi-constraint graph partitioning. In *Supercomputing '98: Proceedings of the 1998 ACM/IEEE conference on Supercomputing (CDROM)*, pages 1–13, Washington, DC, USA. IEEE Computer Society.

[115] Kelly, III, C. and Barclay, S. (1973). A general Bayesian model for hierarchical inference. *Organizational Behavior and Human Performance*, 10:388–403.

[116] Kernighan, B. and Lin, S. (1970). An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, 49(2):291–307.

[117] Khan, Z., Balch, T., and Dellaert, F. (2006). MCMC data association and sparse factorization updating for real time multitarget tracking with merged and multiple measurements. *IEEE Trans. Pattern Anal. Machine Intell.*, 28(12):1960–1972.

[118] Kim, A. and Eustice, R. M. (2015). Active visual SLAM for robotic area coverage. *Intl. J. of Robotics Research*, 34(4-5):457–475.

[119] Kim, B., Kaess, M., Fletcher, L., Leonard, J., Bachrach, A., Roy, N., and Teller, S. (2010). Multiple relative pose graphs for robust cooperative mapping. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 3185–3192, Anchorage, Alaska.

[120] Knight, J., Davison, A., and Reid, I. (2001). Towards constant time SLAM using postponement. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 405–413.

[121] Koller, D. and Friedman, N. (2009). *Probabilistic Graphical Models: Principles and Techniques*. The MIT Press.

[122] Konolige, K. (2004). Large-scale map-making. In *Proc.* 21^{th} *AAAI National Conference on AI*, San Jose, CA.

[123] Krauthausen, P., Dellaert, F., and Kipp, A. (2006). Exploiting locality by nested dissection for square root smoothing and mapping. In *Robotics: Science and Systems (RSS)*.

[124] Kschischang, F. (2003). Codes defined on graphs. *IEEE Signal Proc. Mag.*, 41:118–125.

[125] Kschischang, F., Frey, B., and Loeliger, H.-A. (2001). Factor graphs and the sum-product algorithm. *IEEE Trans. Inform. Theory*, 47(2).

[126] Kümmerle, R., Grisetti, G., and Burgard, W. (2012). Simultaneous calibration, localization, and mapping. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 3716–3721.

[127] Kümmerle, R., Grisetti, G., Strasdat, H., Konolige, K., and Burgard, W. (2011). g2o: A general framework for graph optimization. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, Shanghai, China.

[128] Latif, Y., Cadena, C., and Neira, J. (2013). Robust loop closing over time for pose graph SLAM. *Intl. J. of Robotics Research*, 32(14):1611–1626.

[129] Leonard, J., Durrant-Whyte, H., and Cox, I. (1992). Dynamic map building for an autonomous mobile robot. *Intl. J. of Robotics Research*, 11(4):286–289.

[130] Leonard, J. and Feder, H. (2001). Decoupled stochastic mapping. *IEEE Journal of Oceanic Engineering*, pages 561–571.

[131] Leutenegger, S., Furgale, P., Rabaud, V., Chli, M., Konolige, K., and Siegwart, R. (2013). Keyframe-based visual-inertial SLAM using nonlinear optimization. In *Robotics: Science and Systems (RSS)*.

[132] Leutenegger, S., Lynen, S., Bosse, M., Siegwart, R., and Furgale, P. (2015). Keyframe-based visual-inertial odometry using nonlinear optimization. *Intl. J. of Robotics Research*, 34(3):314–334.

[133] Levenberg, K. (1944). A method for the solution of certain nonlinear problems in least squares. *Quart. Appl. Math*, 2(2):164–168.

[134] Ling, F. (1991). Givens rotation based least squares lattice related algorithms. *IEEE Trans. Signal Processing*, 39(7):1541–1551.

[135] Lipton, R., Rose, D., and Tarjan, R. (1979). Generalized nested dissection. *SIAM Journal on Applied Mathematics*, 16(2):346–358.

[136] Lipton, R. and Tarjan, R. (1979). A separator theorem for planar graphs. *SIAM Journal on Applied Mathematics*, 36(2):177–189.

[137] Liu, J. (1985). Modification of the minimum-degree algorithm by multiple elimination. *ACM Trans. Math. Softw.*, 11(2):141–153.

[138] Liu, J. W. H. (1989). A graph partitioning algorithm by node separators. *ACM Trans. Math. Softw.*, 15(3):198–219.

[139] Loeliger, H.-A. (2004). An introduction to factor graphs. *IEEE Signal Processing Magazine*, pages 28–41.

[140] Lourakis, M. and Argyros, A. A. (2005). Is Levenberg-Marquardt the most efficient optimization algorithm for implementing bundle adjustment? In *Intl. Conf. on Computer Vision (ICCV)*, volume 2, pages 1526–1531. IEEE.

[141] Lu, F. and Milios, E. (1997a). Globally consistent range scan alignment for environment mapping. *Autonomous Robots*, pages 333–349.

[142] Lu, F. and Milios, E. (1997b). Robot pose estimation in unknown environments by matching 2D range scans. *J. of Intelligent and Robotic Systems*, page 249:275.

[143] Lupton, T. and Sukkarieh, S. (2012). Visual-inertial-aided navigation for high-dynamic motion in built environments without initial conditions. *IEEE Trans. Robotics*, 28(1):61–76.

[144] Marquardt, D. (1963). An algorithm for least-squares estimation of nonlinear parameters. *J. Soc. Indust. Appl. Math.*, 11(2):431–441.

[145] Maybeck, P. (1979). *Stochastic Models, Estimation and Control*, volume 1. Academic Press, New York.

[146] Mazuran, M., Burgard, W., and Tipaldi, G. (2016). Nonlinear factor recovery for long-term SLAM. *Intl. J. of Robotics Research*, 35(1-3):50–72.

[147] McDonald, J., Kaess, M., Cadena, C., Neira, J., and Leonard, J. (2013). Real-time 6-DOF multi-session visual SLAM over large scale environments. *Robotics and Autonomous Systems*, 61(10):1144–1158.

[148] Montanari, U. (1974). Networks of constraints: Fundamental properties and applications to picture processing. *Information Sciences*, 7:95–132.

[149] Mur-Artal, R. and Tardos, J. D. (2016). Visual-inertial monocular SLAM with map reuse. Technical report, Universidad de Zaragoza. arXiv:1610.05949.

[150] Murray, R., Li, Z., and Sastry, S. (1994). *A Mathematical Introduction to Robotic Manipulation*. CRC Press.

[151] Neal, R. (1993). Probabilistic inference using Markov chain Monte Carlo methods. Technical Report CRG-TR-93-1, Dept. of Computer Science, University of Toronto.

[152] Ni, K. and Dellaert, F. (2010). Multi-level submap based SLAM using nested dissection. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.

[153] Ni, K. and Dellaert, F. (2012). HyperSfM. In *3D Imaging, Modeling, Processing, Visualization and Transmission (3DIMPVT), 2012 Second International Conference on*, pages 144–151. IEEE.

[154] Ni, K., Steedly, D., and Dellaert, F. (2007). Tectonic SAM: Exact; out-of-core; submap-based SLAM. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, Rome; Italy.

[155] Nocedal, J. and Wright, S. J. (1999). *Numerical Optimization*. Springer Series in Operations Research. Springer-Verlag.

[156] Olson, E. and Agarwal, P. (2013). Inference on networks of mixtures for robust robot mapping. *Intl. J. of Robotics Research*, 32(7):826–840.

[157] Olson, E., Leonard, J., and Teller, S. (2006). Fast iterative alignment of pose graphs with poor initial estimates. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 2262–2269.

[158] Ozog, P., Troni, G., Kaess, M., Eustice, R., and Johnson-Roberson, M. (2015). Building 3D mosaics from an autonomous underwater vehicle and 2D imaging sonar. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 1137–1143, Seattle, WA.

[159] Parter, S. (1961). The use of linear graphs in Gauss elimination. *SIAM Rev.*, 3(2):119–130.

[160] Paskin, M. (2003). Thin junction tree filters for simultaneous localization and mapping. In *Intl. Joint Conf. on AI (IJCAI)*.

[161] Paz, L. M., Pinies, P., Tardós, J. D., and Neira, J. (2008). Large scale 6DOF SLAM with stereo-in-hand. *IEEE Transactions on Robotics*, 24(5):946–957.

[162] Pearl, J. (1982). Reverend Bayes on inference engines: a distributed hierarchical approach. In *Proc. First AAAI National Conference on AI*, pages 133–136, Carnegie Mellon University, Pittsburgh, PA.

[163] Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann.

[164] Pinies, P., Paz, P., Haner, S., and Heyden, A. (2012). Decomposable bundle adjustment using a junction tree. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 1246–1253. IEEE.

[165] Pothen, A. and Sun, C. (1992). Distributed multifrontal factorization using clique trees. In *Proc. of the Fifth SIAM Conf. on Parallel Processing for Scientific Computing*, pages 34–40. Society for Industrial and Applied Mathematics.

[166] Pothen, A., Simon, H., and Wang, L. (1992). Spectral nested dissection. Technical Report CS-92-01, Penn. State.

[167] Powell, M. (1970). A new algorithm for unconstrained optimization. In Rosen, J., Mangasarian, O., and Ritter, K., editors, *Nonlinear Programming*, pages 31–65. Academic Press.

[168] Rosen, D., Kaess, M., and Leonard, J. (2014). RISE: An incremental trust-region method for robust online sparse least-squares estimation. *IEEE Trans. Robotics*, 30(5):1091–1108.

[169] Rosen, R. (1968). Matrix bandwidth minimization. In *Proceedings of the 1968 23rd ACM national conference*, pages 585–595, New York, NY, USA. ACM Press.

[170] Salas-Moreno, R., Newcombe, R., Strasdat, H., Kelly, P., and Davison, A. (2013). SLAM++: Simultaneous localisation and mapping at the level of objects. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 1352–1359.

[171] Segal, A. and Reid, I. (2014). Hybrid inference optimization for robust pose graph estimation. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 2675–2682.

[172] Seidel, R. (1981). A new method for solving constraint satisfaction problems. In *Intl. Joint Conf. on AI (IJCAI)*, pages 338–342.

[173] Slama, C., editor (1980). *Manual of Photogrammetry*. American Society of Photogrammetry and Remote Sensing, Falls Church, VA.

[174] Smith, R. and Cheeseman, P. (1987). On the representation and estimation of spatial uncertainty. *Intl. J. of Robotics Research*, 5(4):56–68.

[175] Smith, R., Self, M., and Cheeseman, P. (1988). A stochastic map for uncertain spatial relationships. In *Proc. of the Intl. Symp. of Robotics Research (ISRR)*, pages 467–474.

[176] Smith, R., Self, M., and Cheeseman, P. (1990). Estimating uncertain spatial relationships in robotics. In Cox, I. and Wilfong, G., editors, *Autonomous Robot Vehicles*, pages 167–193. Springer-Verlag.

[177] Spivak, M. (1965). *Calculus on manifolds*. Benjamin New York;.

[178] Stewart, G. (2000). The decompositional approach to matrix computation. *Computing in Science & Engineering*, 2(1):50–59. Special issue on the Top 10 Algorithms in Science & Engineering.

[179] Sünderhauf, N. and Protzel, P. (2012a). Switchable constraints for robust pose graph SLAM. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*.

[180] Sünderhauf, N. and Protzel, P. (2012b). Towards a robust back-end for pose graph SLAM. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 1254–1261. IEEE.

[181] Szeliski, R. and Kang, S. (1993). Recovering 3D shape and motion from image streams using non-linear least squares. Technical Report CRL 93/3, DEC Cambridge Research Lab.

[182] Szeliski, R. and Kang, S. (1994). Recovering 3D shape and motion from image streams using nonlinear least squares. *Journal of Visual Communication and Image Representation*, 5(1).

[183] Tanner, R. (1981). A recursive approach to low complexity codes. *IEEE Trans. Inform. Theory*, 27(5):533–547.

[184] Tardós, J., Neira, J., Newman, P., and Leonard, J. (2002). Robust mapping and localization in indoor environments using sonar data. *Intl. J. of Robotics Research*, 21(4):311–330.

[185] Teixeira, P., Kaess, M., Hover, F., and Leonard, J. (2016). Underwater inspection using sonar-based volumetric submaps. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 4288–4295, Daejeon, Korea.

[186] Thrun, S. (2003). Robotic mapping: a survey. In *Exploring artificial intelligence in the new millennium*, pages 1–35. Morgan Kaufmann, Inc.

[187] Thrun, S., Burgard, W., and Fox, D. (2005). *Probabilistic Robotics*. The MIT press, Cambridge, MA.

[188] Thrun, S., Liu, Y., Koller, D., Ng, A., Ghahramani, Z., and Durrant-Whyte, H. (2004). Simultaneous localization and mapping with sparse extended information filters. *Intl. J. of Robotics Research*, 23(7-8):693–716.

[189] Toohey, L., Pizarro, O., and Williams, S. (2014). Multi-vehicle localisation with additive compressed factor graphs. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 4584–4590.

[190] Trevor, A. J. B., Rogers III, J. G., and Christensen, H. I. (2012). Planar surface SLAM with 3D and 2D sensors. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, St. Paul, MN. IEEE.

[191] Triggs, B., McLauchlan, P., Hartley, R., and Fitzgibbon, A. (2000). Bundle adjustment – a modern synthesis. In Triggs, W., Zisserman, A., and Szeliski, R., editors, *Vision Algorithms: Theory and Practice*, volume 1883 of *LNCS*, pages 298–372. Springer Verlag.

[192] Tweddle, B., Saenz-Otero, A., Leonard, J., and Miller, D. (2015). Factor graph modeling of rigid-body dynamics for localization, mapping, and parameter estimation of a spinning object in space. *J. of Field Robotics*, 32(6):897–933.

[193] Usenko, V., Engel, J., Stückler, J., and Cremers, D. (2016). Direct visual-inertial odometry with stereo cameras. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 1885–1892, Stockholm, Sweden.

[194] VanMiddlesworth, M., Kaess, M., Hover, F., and Leonard, J. (2013). Mapping 3D underwater environments with smoothed submaps. In *Field and Service Robotics (FSR)*, pages 17–30, Brisbane, Australia.

[195] Walter, M., Hemachandra, S., Homberg, B., Tellex, S., and Teller, S. (2014). A framework for learning semantic maps from grounded natural language descriptions. *Intl. J. of Robotics Research*, 33(9):1167–1190.

[196] Whelan, T., Kaess, M., Johannsson, H., Fallon, M., Leonard, J., and McDonald, J. (2015). Real-time large scale dense RGB-D SLAM with volumetric fusion. *Intl. J. of Robotics Research*, 34(4-5):598–626.

[197] Whelan, T., Kaess, M., Leonard, J. J., and McDonald, J. (2013). Deformation-based loop closure for large scale dense RGB-D SLAM. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, Tokyo,Japan.

[198] Wiberg, N. (1996). *Codes and Decoding on General Graphs*. PhD thesis, Linkoping University, Sweden.

[199] Wiberg, N., Loeliger, H.-A., and Kötter, R. (1995). Codes and iterative decoding on general graphs. In *Euro. Trans. Telecomm.*, volume 6, pages 513–525.

[200] Williams, S., Indelman, V., Kaess, M., Roberts, R., Leonard, J. J., and Dellaert, F. (2014). Concurrent filtering and smoothing: A parallel architecture for real-time navigation and full smoothing. *Intl. J. of Robotics Research*.

[201] Winkler, G. (1995). *Image analysis, random fields and dynamic Monte Carlo methods*. Springer Verlag.

[202] Yan, X., Indelman, V., and Boots, B. (2017). Incremental sparse GP regression for continuous-time trajectory estimation and mapping. *Robotics and Autonomous Systems*, 87(1):120–132.

[203] Yannakakis, M. (1981). Computing the minimum fill-in is NP-complete. *SIAM J. Algebraic Discrete Methods*, 2.

[204] Yedidia, J., Freeman, W., and Y.Weiss (2000). Generalized belief propagation. In *Advances in Neural Information Processing Systems (NIPS)*, pages 689–695.

[205] Zhang, J., Kaess, M., and Singh, S. (2014). Real-time depth enhanced monocular odometry. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 4973–4980, Chicago, IL.

[206] Zhang, J., Kaess, M., and Singh, S. (2017). A real-time method for depth enhanced monocular odometry. 41(1):31–43.

[207] Zhang, N. and Poole, D. (1994). A simple approach to Bayesian network computations. In *Proc. of the 10th Canadian Conf. on AI*, Banff, Alberta, Canada.

Appendices

A

Multifrontal Cholesky Factorization

We recover sparse **multifrontal Cholesky factorization** if we instead use partial Cholesky factorization when eliminating a single variable. To enable this, when eliminating the variable x_j, the product factor $\psi(x_j, S_j)$ is handled in a slightly different way. In particular, we define the augmented Jacobian matrix $\widehat{A}_j \triangleq [\bar{A}_j | \bar{b}_j]$ associated with the product factor $\psi(x_j, S_j)$, and the corresponding augmented state $\widehat{x} \triangleq [x_j; S_j; 1]$. We then have

$$\left\| \bar{A}_j[x_j; S_j] - \bar{b}_j \right\|_2^2 = \widehat{x}^\top (\widehat{A}_j^\top \widehat{A}_j) \widehat{x}, \tag{A.1}$$

where $\widehat{\Lambda}_j \triangleq \widehat{A}_j^\top \widehat{A}_j$ is the **augmented Hessian matrix** associated with the product factor $\psi(x_j, S_j)$. As an example, eliminating l_2 in the toy example yields the product factor

$$\widehat{\Lambda}_2 = \begin{bmatrix} A_{52}^\top A_{52} + A_{92}^\top A_{92} & A_{92}^\top A_{95} & A_{52}^\top b_5 + A_{92}^\top b_9 \\ - & A_{95}^\top A_{95} & A_{95}^\top b_9 \\ - & - & b_5^\top b_5 + b_9^\top b_9 \end{bmatrix}, \tag{A.2}$$

which one can see to be the sum of two outer products, corresponding to the factors ϕ_5 and ϕ_9.

We partition $\widehat{\Lambda}_j$ into 4 blocks, isolating the blocks associated with the variable x_j, and perform the following partial Cholesky factorization:

$$\widehat{\Lambda}_j = \begin{bmatrix} \widehat{\Lambda}_{11} & \widehat{\Lambda}_{12} \\ \widehat{\Lambda}_{21} & \widehat{\Lambda}_{22} \end{bmatrix} = \begin{bmatrix} R_j^\top & \\ S^\top & L^\top \end{bmatrix} \begin{bmatrix} R_j & S \\ & L \end{bmatrix}. \tag{A.3}$$

The upper triangular matrix R_j, satisfying $R_j^\top R_j = \widehat{\Lambda}_{11}$, will be identical to the one obtained by QR factorization up to possibly sign flips on the diagonal. The remaining blocks S and L can be computed by

$$\begin{aligned} S &= R_j^{-\top} \widehat{\Lambda}_{12} && \text{(A.4)} \\ L^\top L &= S^\top S && \text{(A.5)} \\ &= \widehat{\Lambda}_{22} - \widehat{\Lambda}_{12}^\top \widehat{\Lambda}_{11}^{-1} \widehat{\Lambda}_{12}. && \text{(A.6)} \end{aligned}$$

The latter computation, known as the Schur complement, has a nice information-theoretic interpretation: we *downdate* the information $\widehat{\Lambda}_{22}$ on the separator S_j with the information we "consume" in order to determine the eliminated variable x_j. The more information $\widehat{\Lambda}_{11}$ we had on x_j, the more information remains on the separator S_j.

After the partial Cholesky step, the algorithm proceeds by creating a conditional density from R and S, given by

$$p(x_j|S_j) \propto \exp\left\{-\frac{1}{2}\left\|R_j x_j + T_j S_j - d_j\right\|_2^2\right\} \tag{A.7}$$

with $[T_j|d_j] = S$. This conditional is exactly the same as the one we recover via the QR path. Adding the new factor on the separator S_j corresponding to $L^\top L$ needs some care: we can indeed create a new factor, but with the corresponding error

$$\tau(S_j) = \exp\left\{-\frac{1}{2}\widehat{S_j}^\top (L^\top L)\widehat{S_j}\right\} \tag{A.8}$$

rather than the Jacobian form as used in Equation 3.20 on page 39.

B

Lie Groups and other Manifolds

Many of the unknown variables in robotics live in well-known continuous transformation groups known as **Lie groups**. A rigorous definition will take us too far afield, but roughly speaking a Lie group is simply a manifold with a smooth group operation defined on it. The most important examples are reviewed below.

B.1 2D Rotations

One of the simplest Lie groups is the space of 2D rotations with composition as the group operator, also known as the **Circle Group**. The easiest way to define it is as the subset of all 2×2 invertible matrices that are both orthogonal and have determinant one, i.e., 2×2 rotation matrices. Because of this definition, people often refer to this Lie group the as the **Special Orthogonal Group** in dimension 2, written as $SO(2)$. Here "special" refers to the unit determinant property.

The nonlinear orthogonality and unit determinant constraints define a nonlinear, one-dimensional manifold within the larger 4-dimensional space of 2×2 invertible matrices. In fact, the manifold has the topology of a circle, but it remains a group: matrix multiplica-

tion of two rotation matrices in $SO(2)$ is closed, the identity matrix I_2 is in $SO(2)$, and the inverse element of each rotation R is its transpose $R^\top$, which is also in $SO(2)$. Hence, $SO(2)$ is a subgroup of the **General Linear Group** $GL(2)$ of 2×2 invertible matrices.

What makes this Lie group stand out from all other groups we discuss below is that the group operation is **commutative**: $R_1 R_2 = R_2 R_1$ for all $R_1, R_2 \in SO(2)$. This explains why people often simply represent a planar rotation with an angle $\theta \in \mathbb{R}$, and use scalar addition as a proxy for the group operation. However, while matrix multiplication respects the circle topology, scalar addition does not.

An important representation that *does* respect the wrap-around property is the group of unit-norm complex numbers $\cos\theta + i \sin\theta \in \bar{\mathbb{C}}$ with complex multiplication, which is isomorphic to $SO(2)$.

In summary, these are the three most common representations used for rotations: angles, complex numbers, and 2×2 rotation matrices,

$$\mathbb{R} \to \bar{\mathbb{C}} \leftrightarrow SO(2) \tag{B.1}$$

$$\theta \to \cos\theta + i \sin\theta \leftrightarrow \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}, \tag{B.2}$$

where the first arrow indicates an (undesirable) many to-one mapping.

B.2 2D Rigid Transformations

Equipped with $SO(2)$ we can model the orientation of robots moving in the plane. Just as it was convenient to embed the one-dimensional manifold $SO(2)$ in $GL(2)$, we likewise embed both orientation $R \in SO(2)$ and position $t \in \mathbb{R}^2$ in the space of 3×3 matrices, as follows:

$$T \triangleq \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix}. \tag{B.3}$$

The above defines the **Special Euclidean Group** $SE(2)$. It is a subgroup of the general linear group $GL(3)$, with matrix multiplication as the group operation. The identity element is $I_3 \in GL(3)$, and we have

$$T^{-1} = \begin{bmatrix} R^\top & -R^\top t \\ 0 & 1 \end{bmatrix} \tag{B.4}$$

and

$$T_1 T_2 = \begin{bmatrix} R_1 & t_1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} R_2 & t_2 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} R_1 R_2 & R_1 t_2 + t_1 \\ 0 & 1 \end{bmatrix}. \tag{B.5}$$

Note that composition in $SE(2)$ is *not* commutative.

For planar robots, we can use elements of $SE(2)$ to represent the **2D pose** x of the robot, i.e., $x \in SE(2)$. We can interpret a pose $x_i = T_i \in SE(2)$ as the transformation that would take us from the origin to the coordinate frame associated with the robot's current pose.

Relative poses are also elements of $SE(2)$: suppose $x_i = T_i$ and $x_j = T_j$, then we have

$$x_j = T_j = T_i T_i^{-1} T_j = x_i (T_i^{-1} T_j) = x_i T_j^i \tag{B.6}$$

and hence $T_j^i \triangleq T_i^{-1} T_j$ is the transformation that takes x_i to x_j.

The natural **group action** associated with an element $T_i \in SE(2)$ transforms points $p^i \in \mathbb{R}^2$ in coordinate frame i to points $q^g \in \mathbb{R}^2$ in the global frame by embedding both in $\mathbb{P}^2$ using homogeneous coordinates:

$$\begin{bmatrix} q^g \\ 1 \end{bmatrix} = \begin{bmatrix} R_i & t_i \\ 0 & 1 \end{bmatrix} \begin{bmatrix} p^i \\ 1 \end{bmatrix} = \begin{bmatrix} R_i p^i + t_i \\ 1 \end{bmatrix}. \tag{B.7}$$

We write $q^g = T_i \otimes p^i$, and the change from local to global coordinates is $q^g = R_i p^i + t_i$, i.e., the local point p^i is rotated and then translated.

To model measurements taken from a particular robot pose $x_i = T_i$, a more important question is: if we know the location of a landmark $l_j = q^g \in \mathbb{R}^2$ in the global coordinate frame, what are its coordinates p^i in the robot's frame? Since the inverse of R_i is $R_i^\top$, the inverse transformation follows easily from (B.7) as $p^i = R_i^\top (q^g - t_i)$.

B.3 3D Rotations

The Lie group $SO(3)$ of rotations in 3D (aka spatial rotations) is represented by the set of 3×3 matrices that are orthogonal and have determinant 1. 3D rotations are important in robotics but also in navigation and many other fields, and hence this Lie group is one of the most studied and well-known structures in applied math.

$SO(3)$ is a three-dimensional manifold embedded within a 9-dimensional ambient space, and forms a subgroup within $GL(3)$ in the same way $SO(2)$ is a subgroup of $GL(2)$. However, unlike planar rotations, *spatial rotations do not commute*. In other words,

$$R_1 R_2 \neq R_2 R_1 \tag{B.8}$$

for most $R_1, R_2 \in SO(3)$. Of course, since $SO(2)$ is a subgroup of $SO(3)$ (keep any axis fixed), it is clear that *some* combinations of rotation matrices do commute, just not all.

The subgroup relationship between $SO(2)$ and $SO(3)$ gives rise to the commonly used **axis-angle** representation for spatial rotations. It consists of the pair $(\bar{\omega}, \theta)$, where the axis $\bar{\omega} \in S^2$ is a unit vector on the sphere and $\theta \in \mathbb{R}$ is a rotation angle around this axis. Both can be combined in a single three-vector $\omega = \theta\bar{\omega}$. While convenient for some operations, composition of two rotations is cumbersome and is best achieved by converting back to rotation matrices. In addition, because of the dependence on a scalar angle θ, there is again an undesirable many-to-one mapping from axis-angle to $SO(3)$.

Another, very common way to represent 3D rotations is using **unit quaternions** $q \in \bar{\mathbb{Q}}$, analogous to the role unit complex numbers play for $SO(2)$. Quaternions, like complex numbers, have a real part and an imaginary part, but the imaginary part in quaternions is three-dimensional, with axes i, j, and k. The easiest way to introduce unit-quaternions as a way to represent rotations is by converting from the axis angle representation,

$$(\bar{\omega}, \theta) \rightarrow \cos\frac{\theta}{2} + (\bar{\omega}_x i + \bar{\omega}_y j + \bar{\omega}_z k)\sin\frac{\theta}{2}, \tag{B.9}$$

which highlights that the axis $\bar{\omega}$ is encoded in the imaginary part. Unit quaternions are more compact than 3×3 matrices and, equipped with quaternion multiplication, are *almost* isomorphic to $SO(3)$. Indeed, their only flaw is that there is a two-to-one mapping from $\bar{\mathbb{Q}}$ to $SO(3)$: q and $-q$ represent the same rotation. Despite this minor annoyance, they are a popular representation in robotics.

Finally, the most intuitive but often problematic representation for 3D rotations consists of using **Euler angles**. These are quite useful

from a readability point of view, because rotations around identity can be easily understood as a combination of **roll ϕ, pitch θ, and yaw** ψ—making the three degrees of freedom palatable where rotation matrices and unit quaternions obfuscate. However, far from identity, Euler angles exhibit singularities which complicate optimizing over them when used in those regimes.

In summary, these are the four most common representations used for spatial rotations: axis-angle, unit quaternions, and 3×3 rotation matrices, and Euler angles:

$$S^2 \times \mathbb{R} \leftrightarrow \bar{\mathbb{Q}} \rightrightarrows SO(3) \leftarrow \mathbb{R}^3 \tag{B.10}$$

$$(\bar{\omega}, \theta) \leftrightarrow \cos\frac{\theta}{2} + (\bar{\omega}_x i + \bar{\omega}_y j + \bar{\omega}_z k)\sin\frac{\theta}{2} \rightrightarrows R \leftarrow \phi, \theta, \psi, \tag{B.11}$$

where the double arrow represents the double covering property of unit quaternions, and the last arrow indicates the undesirable many to-one mapping from Euler angles to rotation matrices (even more so now, because of the inherent singularities).

B.4 3D Rigid Transformations

The full 6 DOF pose of a robot operating in free space or on undulating terrain can be represented using rigid 3D transformations. The situation is completely analogous to the 2D case in Section B.2: we embed a rotation matrix $R \in SO(3)$ and a translation vector $t \in \mathbb{R}^3$ in a 4×4 matrix

$$T \triangleq \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \tag{B.12}$$

to define the **Special Euclidean Group** $SE(3)$ of rigid 3D transformations. Again, the group operation is matrix multiplication, and $SE(3)$ is a subgroup of the 4×4 invertible matrices $GL(4)$.

B.5 Directions in 3D

An important nonlinear manifold that is *not* a group is the set of all directions in 3D space. These are useful for reasoning about a robot's

orientation with respect to gravity, such as measured by an accelerometer for instance. Another use case is visual odometry using a monocular camera only, in which case absolute scale is unobservable between two frames, but translation direction is.

A direction in space is conveniently represented by a unit 3-vector, i.e., $p = \begin{bmatrix} x & y & z \end{bmatrix}^\top$ with the nonlinear constraint $x^2 + y^2 + z^2 = 1$. In other words, the manifold of directions in 3D space is the **Sphere in 3D**, typically denoted S^2. It is a *two-dimensional* manifold, as the nonlinear constraint takes away one degree of freedom, and indeed, the sphere is intuitively familiar to us as a two-dimensional surface.