Continuous-time Trajectory Representation for Simultaneous Localization and Mapping

AUTHOR: CHAO QU
ADVISOR: CAMILLO JOSE TAYLOR

COMMITTEE MEMBERS:
KOSTAS DANIILIDIS (CHAIR)
JEAN GALLIER
CAMILLO JOSE TAYLOR

DEPARTMENT OF COMPUTER AND INFORMATION SCIENCE
UNIVERSITY OF PENNSYLVANIA

WRITTEN PRELIMINARY EXAMINATION II
MAY 7, 2021

Abstract

Simultaneous Localization and Mapping (SLAM) [1] is a fundamental problem in robotics research. It is the process by which a mobile robot builds a model of the environment (the *map*) and at the same time *localizes* itself within. From its inception more than three decades ago, SLAM adopted a probabilistic formulation in discrete time, with a set of poses representing the robot trajectory. The discrete-time approach proves to be sufficient for many tasks where measurements arrive at relatively low frequency. However, naively applying it to high-rate sensors would cause the state size to grow beyond the capability of onboard computation.

In this report, we explore *continuous-time trajectory representation* as a remedy to the above problem. We start by reviewing the canonical discrete-time SLAM formulation, with its graphical model and special structure [2]. We then compare three continuous-time variations: 1) a parametric one using a weighted sum of temporal basis functions [3], 2) a non-parametric one that models the trajectory as a Gaussian process [4], and 3) a hybrid one that decouples trajectory estimate and error and only model the error term in continuous-time [5].

Contents

List of Figures

List of Tables

List of Symbols

x	a scalar value
$\mathbf{x}$	a column vector
$\mathbf{M}$	a matrix
$\mathbf{x}^\top, \mathbf{M}^\top$	transpose of a vector or a matrix
$\|\mathbf{x}\|$	norm (length) of a vector $\mathbf{x}$
$\|\mathbf{x}\|_p$	L_p norm of a vector $\mathbf{x}$
$\bar{x}$	linearization point of x
$\check{x}$	prior estimate of x
Δx	small perturbation of x
δx	error state of x
$\hat{x}$	posterior estimate of x
$\|\mathbf{x}\|_\Sigma$	Mahalanobis distance of a vector $\mathbf{x}$ with covariance matrix Σ
$\mathcal{N}(x; \mu, \sigma^2)$	a Gaussian random variable with mean μ and variance σ^2
$\|X\|$	cardinality of a set
$p(x)$	probability density of a continuous random variable x
X	a set containing elements $x_1, \ldots x_n$
x^*	optimal estimation of x

1 Introduction

1.1 A Gentle Introduction to SLAM

1.1.1 What is SLAM?

SLAM stands for **Simultaneous Localization and Mapping**. It consists of the simultaneous estimation of the robot state and the construction of a model of the environment [1]. The robot state is typically described by its pose (position and orientation), or the entire history of poses (trajectory), although other variables such as velocity, sensor biases and calibration parameters can be added. The map is an abstract model that represents the environment where the robot operates in, which can take various form depends on the sensors used and the specific application. Together, they form the full state variable of the SLAM problem.

When the entire trajectory of the robot is desired, we say that we are performing *batch* SLAM. When only the latest pose of the robot is of interest, the algorithm is called *recursive* or *filtering*. If the robot trajectory is given, SLAM reduces to the *mapping* problem. If the map is known in advance, it reduces to the *localization* problem. However, due to the presence of noises in the measurements and uncertainties in the *a priori* knowledge (the known map or trajectory), it is still recommended to refine all variables in a joint optimization for maximum accuracy.

1.1.2 System Architecture

A SLAM system is typically divided into two major components: the *front-end* and *the back-end* [6]. Figure 1.1 depicts such an architecture. The front-end is tasked with abstracting sensor data into various constraints that can be used by the back-end. The back-end then performs inference on the abstracted data from the front-end to produce an estimate of the state. Note the arrow going back from the back-end to the front-end. The output from the back-end can be used to facilitates and robustify part of the front-end. For example, an accurate estimate of the robot's metric pose can not only speed up the search for a loop closure (making it more efficient), but also reduce the risk of

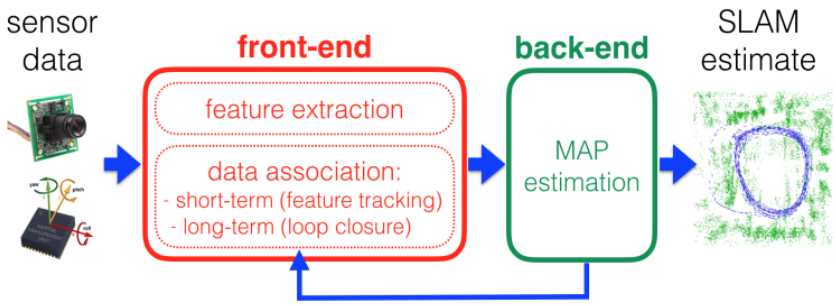


Figure 1.1: Front-end and back-end in a typical SLAM system [6].

(a) CodeSLAM [10]

(b) gradSLAM [13]

Figure 1.2: Results of various deep learning powered SLAM algorithms

finding false positives due to perceptual aliases (making it more robust). This coupling indicates a strong relationship between the two sub-systems.

The front-end of a SLAM system is usually sensor-dependent, whereas the back-end is relatively general, as it operates on the abstraction created by the former. Oftentimes, it is impractical or inconvenient to write directly the sensor data as an analytic function of the state. This is due to the fact that we are not able to design a general yet tractable representation of the environment [6]. For this reason, we rely on the front-end to extract meaningful and relevant features from the raw sensor data. In addition, the front-end is responsible for associating these features with each other and/or to the map, which produces constraints for the back-end.

Lately, deep learning techniques have also been applied to SLAM with moderate success. This is due to the strong representation learning power of deep neural networks (DNNs). Most of these systems try to substitute part of the SLAM system with a DNN [10, 11] and keep the rest of the pipeline fixed. Others make attempts to replace the entire system with several networks or differentiable modules that directly output robot poses and map [12, 13]. These are exciting and promising directions [14], but are out of the scope of this report. Some qualitative results are show in Figure 1.2 for the interested readers.

1.1.3 Is SLAM Solved?

As SLAM becomes a mature research field within the robotics community, many have started to considered it a solved problem. After all, the probabilistic formulation of SLAM has been developed and refined for over three decades. However, asking whether SLAM is solved is like asking whether matrix multiplication is solved [15]. The textbook method for multiplying two compatible matrices is clear as day, which requires taking the inner products of corresponding rows and columns. For centuries people believe $O(n^3)$ was the best complexity one could achieve, despite the wishful thinking that $O(n^2)$ should be as fast as physically possible. However, till this day, scientists are still pushing the theoretical limits closer to the this goal, with the latest result achieving $O(n^{2.3728596})$ [16]. In this sense, matrix multiplication is **not** solved.

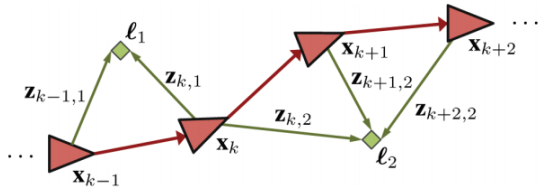
The same argument applies to SLAM as well. One could argue that mapping an indoor environment with a wheeled robot equipped with a laser scanner can be largely considered solved. But current SLAM algorithms are still not robust and scalable enough for large-scale real-world applications. State-of-the-art SLAM systems can easily be induced to fail, or are unable to follow strict performance requirements under challenging environments. The review paper by Cadena *et al.* [6] lists the following areas with many open problems:

1. Robustness: failure recovery (software and hardware), metric relocalization, automatic parameter tunning, *etc*.
2. Scalability: optimal map and trajectory representation, robust distributed mapping, resource-constrained platforms, *etc*.
3. Metric Map Models: high-level, optimal and adaptive representations, *etc*.
4. Semantic Map Models: semantic-based reasoning, joint SLAM and semantics inference, consistent semantic-metric fusion *etc*.

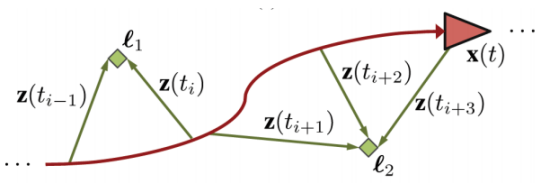
The topic of this report, continous-time trajectory representation, is deeply related to the scalability issue of SLAM, which we will discuss later.

1.2 Motivation

The discrete-time formulation typically introduces a new pose variable at every measurement time instance. This works well in the early phase of the SLAM development, when sensor readings arrive at relatively low frequency. However, this paradigm tend to be pushed to their limits under the following problematic situations [17]:



(a) The conventional discrete-time formulation, which discretizes robot trajectory into a set of poses.



(b) Continuous-time formulation, where the robot trajectory is modelled as a smooth function of time.

Figure 1.3: Comparison of the two SLAM formulations from [4]. The robot poses are depicted by the red lines and triangles, while landmarks by green diamonds. The green arrow indicates a measurements.

1. When a sensor is capturing data at high frequency.
2. When a sensor is scanning continuously while the robot is in motion.
3. When multiple sensor readings arrive asynchronously.
4. When a robot is operating for a prolonged period of time.

Although these are distinct situations, the fundamental problem is the same: to prevent the state size from growing out of control.

Many approximate solutions have been proposed to address this issue. For example, IMU pre-integration was developed to summarize many measurements between two visual keyframes into a single relative measurement (called a IMU pre-integration factor) [18]. For lasers that produce a huge point cloud within each scan, one could use the rough rotation estimate from the IMU or a motion model to undistort the scan and treat it as a snapshot [19]. Edge sparsification and node removal are techniques that directly modify the graph structure to reduce computation complexity [20]

Continuous-time trajectory representations provide a principled and elegant solution to this problem. By parameterizing the robot motion as a function of time, we can drastically reduce the size of the state. In addition, it allows one to query the state of the robot at any given time along the trajectory. A graphical comparison of the two approaches is shown in Figure 1.3.

2 | Discrete-time SLAM

In this chapter, we provide a detailed review to the discrete-time SLAM problem. We limit our scope to batch estimation, where all robot poses within a time window are considered. The recursive version is just a special case of the batch one. The chapter is structured as follows. We start with the probabilistic formulation of SLAM and its graphical representations. We then derive the objective function and find the optimal solution using nonlinear least squares. Finally, we discuss the special structure inherent to the problem, namely its sparsity pattern. There exists a plethora of tutorials on SLAM and our exposition mainly follows ideas from [2, 7, 21].

2.1 Probabilistic Formulation

The goal of SLAM is for the robot to localize itself using the information from its sensors. On top of that, it has limited prior knowledge of the environment, and thus has to construct the map in the process. Due to measurement uncertainty, it is impossible to recover the true state of the robot and the map. Therefore, we aim for a *probabilistic* description of what could be inferred from the measurements.

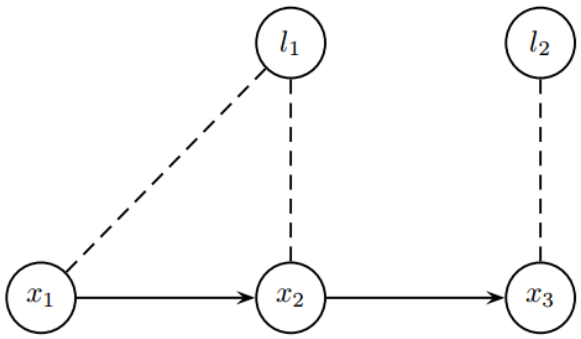


Figure 2.1: A toy SLAM problem with three robot poses and two landmarks. Arrows denote robot motion while dotted lines denote landmark measurements [7].

2.1.1 Graphical Representations

Let's start with a schematic drawing of a toy SLAM problem as shown in Figure 2.1. In SLAM, we want to characterize our knowledge about the unknown variables X, given a set of measurements Z. Here, X is the set of all vertices (circles) and Z is the set of all edges (dotted lines and arrows) in the graph. The specific quantity we are interested in is the conditional density $p(X|Z)$, also called the posterior probability of X. The process of computing this term is called *probabilistic inference*.

To this end, we need to specify a probabilistic model for the variables and this is usually achieved with *probabilistic graphical models (PGMs)* [22]. PGMs provide a mechanism to describe complex probability densities by exploiting the underlying structure of the problem. As such, high-dimensional probability densities can be factorized into a product of individual densities over much smaller domains. [7].

Bayesian Network

One of the popular choices of such graphical models is Bayesian networks [23]. Formally, a Bayesian network or **Bayes net** is a directed graph where the nodes represent variables θ_j. A Bayes net defines a joint probability density $p(\Theta)$ over all variables $\Theta = \{\theta_1, \ldots, \theta_n\}$ as the product of many conditional densities

$$p(\Theta) \coloneqq \prod_j p(\theta_j|\pi_j) \tag{2.1}$$

where π_j is an assignment of values to the *parent* of θ_j. In a Bayes net, the graph structure, specifically the node-parent relationships, dictates the factorization of the joint density. This property can also be leveraged to simulate the Bayes net by topologically sorting the vertices in the graph and sample the parent nodes before the child [24]. This is also why Bayes nets are the language of choice for generative modeling.

The Bayes net of the toy example is shown in Figure 2.2. The joint density $p(X, Z)$ can be written as a product of four different sets of conditional densities

$$\begin{aligned}
p(X, Z) &= p(x_1)p(x_2|x_1)p(x_3|x_2) && \text{(Markov property)} && (2.2)\\
&\times p(l_1)p(l_2) && \text{(Prior)} && (2.3)\\
&\times p(z_1|x_1) && \text{(Pose measurement)} && (2.4)\\
&\times p(z_2|x_1, l_1)p(z_3|x_2, l_1)p(z_4|x_3, l_2) && \text{(Landmark measurements)} && (2.5)
\end{aligned}$$

Note that once we have the joint density $p(X, Z)$, it is possible to compute the posterior via $p(X|Z) = p(X, Z)/p(Z)$.

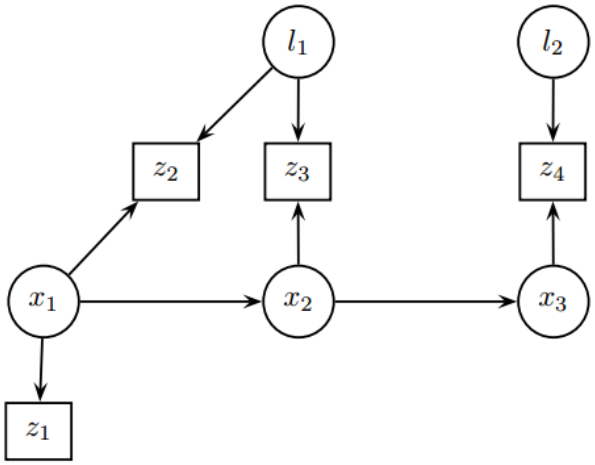


Figure 2.2: Bayesian network model for the toy SLAM example from Figure 2.1. Measurements are denoted by squares. Arrows indicate conditional dependencies between variables [7].

Factor Graphs

Recently, the robotics community has converged to using factor graphs as the *de facto* tool for describing SLAM problems. Like Bayes nets, factor graphs allow one to specify a joint density as a product of factors [25]. Unlike Bayes nets, they can specify any factored (potential) function $\phi(X)$ over a set of variables X, not just probability densities.

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

$$\phi(X) = \prod_i \phi_i(X_i) \tag{2.6}$$

The factor graph model of the toy SLAM problem is shown in Figure 2.3. Converting a Bayes net $P(X, Z)$ to a factor graph (by conditioning on the evidence Z) gives a representation of the posterior $\phi(X) \propto p(X|Z)$ [7]. To see this, we apply Bayes' rule to the joint probability $p(X, Z)$ to get the posterior $P(X|Z)$. Because Z is given as evidence, they are not represented explicitly in factor graphs but simply become fixed parameters in the corresponding factor. Applying this to the toy problem, we

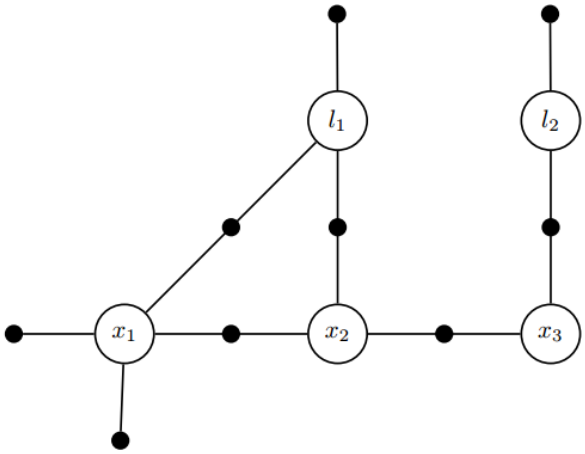


Figure 2.3: Factor graph model for the toy SLAM example from Figure 2.1 [7].

have

$$p(X|Z) = p(Z|X)p(X)/p(Z) \propto l(X;Z)p(X) \tag{2.7}$$
$$= p(x_1)p(x_2|x_1)p(x_3|x_2) \tag{2.8}$$
$$\times\, p(l_1)p(l_2) \tag{2.9}$$
$$\times\, l(x_1;z_1) \tag{2.10}$$
$$\times\, l(x_1,l_1;z_2)l(x_2,l_1;z_3)p(x_3,l_2;z_4) \tag{2.11}$$

where $l(X;Z)$ is the likelihood of the states X given the measurements Z and is proportional to $p(Z|X)$. And each term in the above equation can be represented by a factor from the factor graph, *e.g.* $\phi_7(x_1,l_1) = l(x_1,l_1;z_2) \propto p(z_2|x_1,l_1)$.

2.1.2 Problem Modeling

So far, we've only described the SLAM problem from a high-level perspective, *i.e.* in terms of its graphical models and joint probability density. In this section, we make things a bit more concrete by specifying the form of the densities that are widely used in SLAM problems. The exact form of the densities depends on the application and sensors, but the most common choice is the *multivariate Gaussian distribution*, with probability density

$$\mathcal{N}(\theta;\mu,\Sigma) = \frac{1}{\sqrt{|2\pi\Sigma|}} \exp\left\{-\frac{1}{2}\|\theta-\mu\|_\Sigma^2\right\} \tag{2.12}$$

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

$$\|\theta-\mu\|_\Sigma^2 := (\theta-\mu)^\top \Sigma^{-1}(\theta-\mu) \tag{2.13}$$

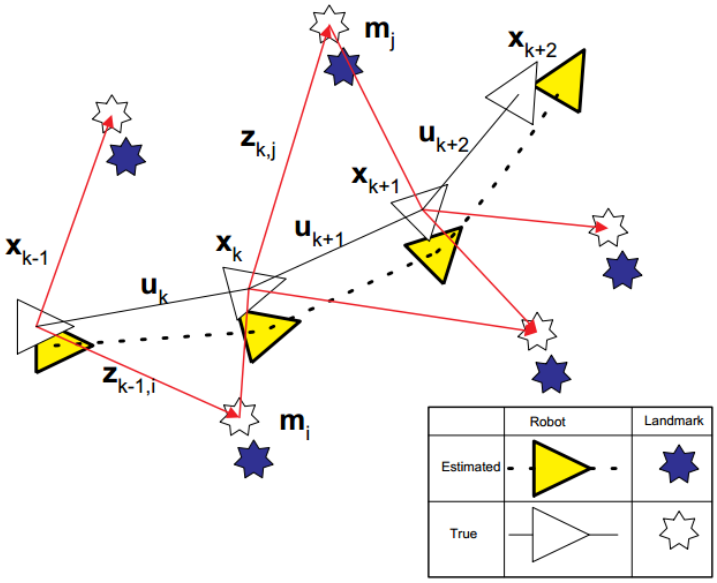


Figure 2.4: The full SLAM problem from [1].

the squared Mahalanobis distance.

Figure 2.4 shows a general drawing of the SLAM problem. Before going into the details, we first defined the following quantities at time k, with $t_k \in [t_0, t_K]$:

- x_k: the state vector describing the pose of the robot at time k.
- u_k: the control input, applied at time $k-1$ to drive the robot to state x_k.
- m_i: the state of landmark i whose true state is assumed to be time invariant
- z_{ki}: a measurement of landmark i from the robot at time k.

Measurement Model

It is common to define a function that relates the measurement to the state. This is called a *measurement model*, which is assumed to be corrupted by zero-mean Gaussian noise

$$z_i = h(x_i, m) + n_i, \quad n_i \sim \mathcal{N}(0, R_i) \tag{2.14}$$

This results in the following conditional density $p(z|x, m)$ on the measurement z

$$p(z_i|x_i, m) = \mathcal{N}(z_i; h(x_i, m), R_i) = \frac{1}{\sqrt{|2\pi R_i|}} \exp\left\{-\frac{1}{2}\|h(x_i, m) - z_i\|^2_{R_i}\right\} \tag{2.15}$$

Motion Model

Not all probability densities involved are derived from measurements. In SLAM, one often specify a *motion model* which the robot is assumed to obey. This typically arises

from some control input u_k and are also corrupted by zero-mean Gaussian noise

$$x_k = f(x_{k-1}, u_k) + w_k, \quad w_k \sim \mathcal{N}(0, Q_k) \tag{2.16}$$

the corresponding conditional density is

$$p(x_k|x_{k-1}, u_k) = \frac{1}{\sqrt{|2\pi Q_k|}} \exp\left\{-\frac{1}{2}\|f(x_{k-1}, u_k) - x_k\|^2_{Q_k}\right\} \tag{2.17}$$

Prior Information

Finally, it is often useful to summarize what we've already known about the world into some prior knowledge. For example, this could represent our belief about the starting pose of the robot, or some previously mapped landmarks. We use a Gaussian density to represent such information

$$x \sim \mathcal{N}(\check{x}, P) \tag{2.18}$$

where $\check{x}$ denotes a prior estimate on x. The probability density function representing this prior knowledge is simply

$$p(x) = \frac{1}{\sqrt{|2\pi P|}} \exp\left\{-\frac{1}{2}\|\check{x} - x\|^2_P\right\} \tag{2.19}$$

2.2 Finding Optimal Solution

Now that we have the means to model the problem, we can perform inference given information about it. From our previous discussion we can write the joint density $p(\Theta)$ from the Bayes net and the posterior $p(X|Z)$ from the corresponding factor graph. What we want to known is the unknown state variables X. At this point, there are two ways going forward: (1) Bayesian inference, and (2) Maximum-a-Posteriori (MAP) Estimation.

2.2.1 Bayesian Inference

Bayesian inference aims to compute the full posterior probability

$$p(X|Z) = \frac{p(Z|X)p(X)}{p(Z)} \qquad \text{(Bayes' Theorem)} \tag{2.20}$$

$$= \frac{p(Z|X)p(X)}{\int p(X, Z)dX} = \frac{p(Z|X)p(X)}{\int p(Z|X)p(X)dX} \qquad \text{(marginalization)} \tag{2.21}$$

The denominator $p(Z)$ in equation (2.20) takes the form of an integration which is usually intractable to compute given nonlinear motion/measurement models. For this reason, practitioners often turn to *Maximum-a-Posteriori (MAP)* estimation as a solution that finds the state that maximize the posterior probability.

It is important to know that the solution of Bayesian inference and MAP are generally not the same for non-linear problems; Bayesian inference computes the mean of the posterior probability while MAP computes the mode (maximum). They are equal only under the linear and Gaussian (LG) case [26]. This is because for LG problems, the full Bayesian posterior is exactly Gaussian, and the mean and mode are thus the same (for Gaussian densities).

2.2.2 Maximum-A-Posteriori Estimation

For MAP estimation, we are interested in the value X that maximizes the posterior probability density $p(X|Z)$ (rather than the full posterior density)

$$X^* = \arg\max_X p(X|Z) \qquad \text{(Posterior probability)} \tag{2.22}$$

$$= \arg\max_X \frac{p(Z|X)p(X)}{p(Z)} \qquad \text{(Bayes' rule)} \tag{2.23}$$

$$= \arg\max_X p(Z|X)p(X) \qquad \text{(Dropping constant denominator)} \tag{2.24}$$

where X^* denotes the optimal solution to the above problem. Since measurements Z are given, the normalization factor $p(Z)$ is just a constant and can be dropped from the maximization without affecting the result. The prior probability $p(X)$ includes any prior knowledge about X. If no prior knowledge is available, $p(X)$ becomes a constant (uniform distribution) and can also be dropped from the optimization. In this case, MAP estimation reduces to *maximum likelihood (ML)* estimation [6].

To be more specific, let us plug our previously defined symbols of state and measurements (2.14, 2.16, 2.18) back into the posterior probability (2.24)

$$p(X|Z) = p(x_{0:K}, m|u_{1:K}, z_{1:N}) \qquad \text{(Posterior)} \tag{2.25}$$

$$= \frac{p(x_{0:K}, m|u_{1:K})p(z_{1:N}|x_{0:K}, m, u_{1:K})}{p(z_{1:N}|u_{1:K})} \qquad \text{(Bayes' rule)} \tag{2.26}$$

$$= \frac{p(x_{0:K}, m|u_{1:K})\prod_{n=1}^{N} p(z_n|x_n, m)}{p(z_{1:N})} \qquad \text{(Independence)} \tag{2.27}$$

$$= \frac{p(m)p(x_0)\prod_{k=1}^{K} p(x_k|x_{k-1}, u_k)\prod_{n=1}^{N} p(z_n|x_n, m)}{p(z_{1:N})} \qquad \text{(Markov)} \tag{2.28}$$

In the above derivation, we make the following assumptions:

1. the state transition is a Markov process: the next state x_{k+1} depends only on the current state x_k and the applied control input u_k.
2. all measurements $z_{1:N}$ are independent of control inputs $u_{1:K}$.
3. each measurement z_i are conditionally independent given the pose and the map.

This posterior probability is thus factored into a product of many (Gaussian) probability densities. Together with the models specified above, we may start to derive the objective function for our MAP estimator. Note that maximizing the posterior probability is equivalent to minimizing its negative log

$$X^* = \arg\max_X p(X|Z) = \arg\min_X -\ln p(X|Z) \tag{2.29}$$

Let θ be the full state vector that is the concatenation of poses $x_{0:K}$ and map m

$$\theta := \begin{bmatrix} x \\ m \end{bmatrix} = \begin{bmatrix} x_{0:K} \\ m \end{bmatrix} \tag{2.30}$$

Equation (2.29) can be re-written as

$$\theta^* = \arg\min_\theta \{-\ln p(x_{0:K}, m|u_{1:K}, z_{1:N})\} \tag{2.31}$$

We further expand the negative-log posterior to be

$$-\ln p(x_{0:K}, m|u_{1:K}, z_{1:N}) = \eta + L_p + L_u + L_z \tag{2.32}$$

where η is a constant term that consumes the combined normalization terms of all Gaussian densities. For each individual term (prior, motion, and measurement), we can write down their negative log likelihood explicitly as

$$-\ln p(m) \propto L_p = \frac{1}{2}\|m - \check{m}\|_P^2 \qquad \text{(Prior)} \tag{2.33}$$

$$-\ln \prod_{k=1}^{K} p(x_k|x_{k-1}, u_k) \propto L_u = \frac{1}{2}\sum_{k=1}^{K} \|f(x_{k-1}, u_k) - x_k\|_{Q_k}^2 \qquad \text{(Motion)} \tag{2.34}$$

$$-\ln \prod_{i=1}^{N} p(z_k|x_k, m) \propto L_z = \frac{1}{2}\sum_{i=1}^{N} \|h(x_i, m) - z_i\|_{R_i}^2 \qquad \text{(Measurement)} \tag{2.35}$$

Note that although $p(x_0)$ is considered a prior on the initial pose of the robot, it is subsumed by the motion term for notational convenience. It is sometimes useful to also define a lifted (matrix) form of the above equations to reduce clutter. We can concatenate all the individual process models together to write the system dynamic

state space model as

$$x = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_K \end{bmatrix}, \; f(x,u) \coloneqq f(x_{0:}, u_{1:K}) = \begin{bmatrix} \check{x}_0 \\ f(x_0, u_1) \\ \vdots \\ f(x_{K-1}, u_K) \end{bmatrix}, \; Q = \begin{bmatrix} P_0 & & & \\ & Q_1 & & \\ & & \ddots & \\ & & & Q_K \end{bmatrix} \tag{2.36}$$

Likewise, concatenating all the measurements

$$z = \begin{bmatrix} z_1 \\ \vdots \\ z_N \end{bmatrix}, \quad h(\theta) \coloneqq h(x_{0:K}, m) = \begin{bmatrix} h(x_1, m) \\ \vdots \\ h(x_N, m) \end{bmatrix}, \quad R = \begin{bmatrix} R_1 & & \\ & \ddots & \\ & & R_N \end{bmatrix} \tag{2.37}$$

Note that here the the subscript i of the state x_i is not a time instance, but an assignment that fetches the corresponding state x_k given the i-th measurement. The lifted objective functions are thus

$$L_p = \frac{1}{2}\|m - \check{m}\|_P^2 \qquad \text{(Prior)} \tag{2.38}$$

$$L_u = \frac{1}{2}\|f(x,u) - x\|_Q^2 \qquad \text{(Motion)} \tag{2.39}$$

$$L_z = \frac{1}{2}\|h(\theta) - z\|_R^2 \qquad \text{(Measurements)} \tag{2.40}$$

Dropping the constant η, we define $L(\theta) = L_p + L_u + L_m$ and obtain the same MAP estimator as (2.31)

$$\theta^* = \arg\min_\theta L(\theta) = \arg\min_\theta \frac{1}{2}\left\{\|m - \check{m}\|_P^2 + \|f(x,u) - x\|_Q^2 + \|h(\theta) - z\|_R^2\right\} \tag{2.41}$$

2.2.3 Nonlinear Least Squares

Recall that we established in the previous section that finding an MAP solution is equivalent to minimizing a quadratic objective function $L(\theta)$. In this section, we describe how to use nonlinear least squares to find the optimal solution θ^* [21].

To better focus on the topic at hand, let us group all individual terms in $L(\theta)$ together and abuse the notation a little bit (reusing x as the full state vector instead of θ) to define

$$L(x) = \frac{1}{2}\sum_i \|e(x_i, z_i)\|_{\Sigma_i}^2 = \frac{1}{2}\sum_i e(x_i, z_i)^\top \Omega_i e(x_i, z_i) \tag{2.42}$$

- x is the full state vector, which consists of the poses $x_{0:K}$ and the map m.
- z_i is a measurement that depends on the state variable x_i. Note that the subscript

i is not time but an assignment that follows the conditional dependency.

- Ω_i is the information matrix and Σ_i the covariance matrix, $\Omega_i = \Sigma_i^{-1}$.
- $e(x_i, z_i)$ is an error function that computes the difference between the expected measurement $h(x_i)$ and the real one z_i. $h(x_i)$ is a generalization that incorporates prior, motion and measurement model.

For example,

$$e_p(x) = h_p(x) - \check{x} = x - \check{x} \tag{2.43}$$
$$e_u(x) = h_u(x) - 0 = f(x, u) - x \tag{2.44}$$
$$e_z(x) = h_z(x) - z = h(x) - z \tag{2.45}$$

For simplicity, we encode the measurement in the indices of the error function and define

$$e(x_i, z_i) := e_i(x) := e_i = h(x_i) - z_i \tag{2.46}$$

It is generally not easy to find a global solution given a nonlinear objective function. Hence, we turn ourselves towards a local minimizer x^* that satisfies the following condition

$$\forall x : \|x^* - x\| < \epsilon \Rightarrow L(x^*) \leq L(x) \tag{2.47}$$

If the error function is differentiable, we can then employ gradient descent methods to determine a step in the opposite direction of the gradient such that the objective is reduced. To this end, we find the gradient of $L(x)$

$$\frac{\partial L(x_i)}{\partial x_i} = \sum_i J_i^\top \Omega_i e_i, \quad J_i = \frac{\partial e_i(x)}{\partial x} \tag{2.48}$$

where J_i is the *Jacobian* of the error function. For the sake of simplicity, we will work with the lifted form

$$\frac{\partial L(x)}{\partial x} = J^\top \Omega e, \quad J = \frac{\partial e(x)}{\partial x} \tag{2.49}$$

Even though the error functions are nonlinear, if we have a decent initial guess, then gradient descent methods such as *Gauss-Newton (GN)* or *Levenberg-Marquart (LM)* [27] will be able to converge to a local minimum. They do so by repeatedly linearize the nonlinear system at some linearization point and solve a linear approximation which will hopefully bring the solution closer to the minimum.

Linearization

Given an initial estimate $\bar{x}$ for the variable, which is also called a linearization point, we can write the measurement model by its Taylor expansion

$$h(\bar{x}+\Delta x) = h(\bar{x}) + H\Delta x + O(\|\Delta x\|^2), \quad H = \left.\frac{\partial h(x)}{\partial x}\right|_{x=\bar{x}} \tag{2.50}$$

Assuming that the Jacobian is a sufficient approximation of the sensor model or Δx is small, we can neglect the higher order terms and have the following linear approximation

$$h(\bar{x}+\Delta x) \approx h(\bar{x}) + H\Delta x = \bar{h} + H\Delta x \tag{2.51}$$

Inserting this approximation back into the error function (2.46) gives us

$$\begin{aligned} e(x) = e(\bar{x}+\Delta x) &= h(\bar{x}+\Delta x) - z && (2.52)\\ &\approx \bar{h} - z + H\Delta x = \bar{e} + J\Delta x && (2.53) \end{aligned}$$

Note that here $H = J$ given our definition of the error function. We then substitute the linear approximation of the error function into the objective function (2.42)

$$\begin{aligned} L(x) &= e(x)^\top \Omega e(x) && (2.54)\\ &\approx (\bar{e}+J\Delta x)^\top \Omega (\bar{e}+J\Delta x) && (2.55)\\ &= \bar{e}^\top \Omega \bar{e} + 2J^\top \Omega \bar{e}\Delta x + \Delta x J^\top \Omega J \Delta x && (2.56)\\ &= c + 2b^\top \Delta x + \Delta x^\top A \Delta x && (2.57)\\ &:= L'(\Delta x) && (2.58) \end{aligned}$$

where $c = \bar{e}^\top \Omega \bar{e}$, $b = J^\top \Omega \bar{e}$, and $A = J^\top \Omega J$. We also define a new objective function $L'(\Delta x)$, which is quadratic in Δx. $L'(\Delta x)$ is the linear approximation of $L(\bar{x}+\Delta x)$. We thus have a new optimization problem

$$\Delta x^* = \arg\min_{\Delta x} L'(\Delta x) \tag{2.59}$$

This is much easier to solve compared to the original one. Because it is quadratic in Δx, we can find Δx^* by simply setting the first derivative of $L'(\Delta x)$ to 0

$$\frac{\partial L'(\Delta x)}{\Delta x} = 2A\Delta x + 2b = 0 \tag{2.60}$$

This leads to the linear system

$$A\Delta x^* = J^\top \Omega J \Delta x^* = -J^\top \Omega \bar{e} = -b \tag{2.61}$$

where A is often called the *Information matrix* or *Hessian*. Solving the above system yields the optimal increments $\Delta x^* = -A^{-1}b$, which can be used to update the current estimate by

$$\bar{x} \leftarrow \bar{x} + \Delta x^* \tag{2.62}$$

Solution to the Linear System

In general, due to the linear approximation of the error functions and thus the objective function, we will not reach the local minimum within one step (unless the system is linear and Gaussian). The solution Δx^* is only the optimal increment at our current linearization point $\bar{x}$. To overcome the approximation error, a typical solution is to carry out repeated linearization and solve for the best increment to update the current state estimates, starting from an initial guess. This is done until some convergence criteria is met, or reaching a maximum number of iterations.

In practice, we never explicitly compute the inverse of A, since the size of A depends on the size of the state vector and can be prohibitively expensive to invert directly. We will see in the next section that due to the sparsity of the graph structure, the corresponding Jacobian and information matrices are also sparse, and can be efficiently decomposed via (sparse) Cholesky factorization $A = U^\top U$, where U is an upper-triangular matrix. Δx^* can then be computed via forward and backward substitution below.

$$U^\top U \Delta x^* = -b \tag{2.63}$$

$$U^\top y = -b \tag{2.64}$$

$$U \Delta x^* = y \tag{2.65}$$

2.2.4 Exploiting Sparsity

In the previous section, we established that performing MAP inference in nonlinear SLAM problems requires repeatedly solving potentially large but sparse linear systems. In this section, we take a closer look at the underlying linear system $A\Delta x^* = -b$, which is arguably the most crucial part of the SLAM problem. We see that the matrix A and vector b are constructed by summing up a set of matrices and vectors from the prior, motion and measurement model respectively

$$A\Delta x^* = (A_p + A_u + A_z)\Delta x^* = -(b_p + b_u + b_z) = -b \tag{2.66}$$

Each of these terms has its own special structure and can be utilized to efficiently solve the linearized system.

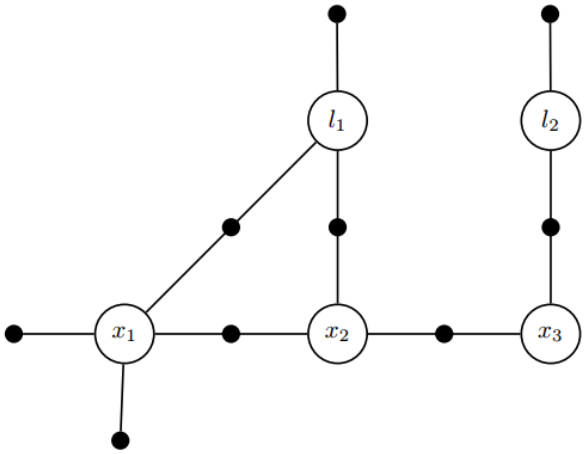


Figure 2.5: Factor graph model for the toy SLAM example from Figure 2.1 [7].

The structure of the linear system is closely related to the sparsity of the corresponding factor graph. To see this, let us rewrite the linear system in the inner loop of the least squares optimization as a summation of individual terms

$$J^\top \Omega J \Delta x^* = -J^\top \Omega \bar{e} \tag{2.67}$$

$$\sum_i J_i^\top \Omega_i J_i \Delta x^* = -\sum_i J_i^\top \Omega_i \bar{e}_i \tag{2.68}$$

$$(J_p^\top \Omega_p J_p + J_u^\top \Omega_u J_u + J_z^\top \Omega_z J_z)\Delta x^* = -(J_p^\top \Omega_p \bar{e}_p + J_u^\top \Omega_u \bar{e} + K_z^\top \Omega_z \bar{e}_z) \tag{2.69}$$

Each term above is derived from a factor in the original, nonlinear SLAM problem and linearized at the current linearization point $\bar{x}$. We will derive and visualize the Jacobian and Hessian matrices for each group, using the same toy problem as an example (shown again in Figure 2.5).

Sparse Jacobian Matrix

For the prior term we have the following quantities

$$e_p(x) = x - \check{x} \qquad \text{(Error function)} \tag{2.70}$$

$$e_p(\bar{x} + \Delta x) = \bar{x} + \Delta x - \check{x} = \bar{e}_p + \Delta x \qquad \text{(Linearization)} \tag{2.71}$$

$$J_p = \frac{\partial e_p(x)}{\partial \Delta x} = I \qquad \text{(Jacobian)} \tag{2.72}$$

$$A_p = J_p^\top \Omega_p J_p = P^{-1} \qquad \text{(Hessian)} \tag{2.73}$$

$$b_p = J_p^\top \Omega_p \bar{e}_p = P^{-1}(\bar{x} - \check{x}) \qquad \text{(rhs)} \tag{2.74}$$

The Jacobian matrix for the prior term in the toy problem is

$$J_p = \begin{array}{c} \begin{matrix} x_1 & x_2 & x_3 & l_1 & l_2 \end{matrix} \\ \left[\begin{array}{ccc|cc} I & & & & \\ & & & I & \\ & & & & I \end{array}\right] \end{array} \begin{matrix} x_1 \\ l_1 \\ l_2 \end{matrix} \tag{2.75}$$

For motion model we have the following quantities

$$e_u(x) = f(x,u) - x = g(x) \qquad \text{(Error function)} \tag{2.76}$$

$$e_u(\bar{x} + \Delta x) \approx f(\bar{x},u) + F\Delta x - \bar{x} - \Delta x = \bar{e}_u + (F - I)\Delta x \qquad \text{(Linearization)} \tag{2.77}$$

$$J_u = \frac{\partial e_u(x)}{\partial \Delta x} = F - I = G, \quad F = \left.\frac{\partial f(x,u)}{\partial x}\right|_{x=\bar{x}} \qquad \text{(Jacobian)} \tag{2.78}$$

$$A_u = J_u^\top \Omega_u J_u = G^\top Q^{-1} G \qquad \text{(Hessian)} \tag{2.79}$$

$$b_u = J_u^\top \Omega_u \bar{e}_u = G^\top Q^{-1}(\bar{f} - \bar{x}) = G^\top Q^{-1}\bar{g} \qquad \text{(rhs)} \tag{2.80}$$

The Jacobian matrix for the motion term in the toy problem is

$$J_u = \begin{array}{c} \begin{matrix} x_1 & x_2 & x_3 & l_1 & l_2 \end{matrix} \\ \left[\begin{array}{ccc|cc} F_1 & -I & & & \\ & F_2 & -I & & \end{array}\right] \end{array} \begin{matrix} u_1 \\ u_2 \end{matrix} \tag{2.81}$$

For measurement model we have the following quantities

$$e_z(x) = h(x) - z \qquad \text{(Error function)} \tag{2.82}$$

$$e_z(\bar{x} + \Delta x) \approx h(\bar{x}) + H\Delta x - z = \bar{e}_z + H\Delta x \qquad \text{(Linearization)} \tag{2.83}$$

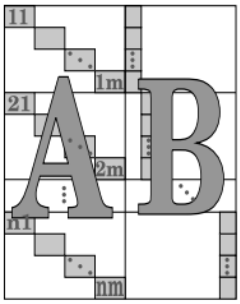
$$J_z = \frac{\partial e_z(x)}{\partial \Delta x} = H, \quad H = \left.\frac{\partial h(x)}{\partial x}\right|_{x=\bar{x}} \qquad \text{(Jacobian)} \tag{2.84}$$

$$A_z = J_z^\top \Omega_z J_z = H^\top R^{-1} H \qquad \text{(Hessian)} \tag{2.85}$$

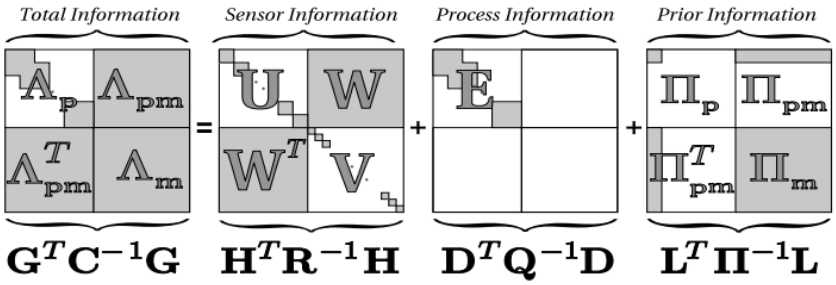
$$b_z = J_z^\top \Omega_z \bar{e}_z = H^\top R^{-1}(\bar{h} - z) \qquad \text{(rhs)} \tag{2.86}$$

The Jacobian matrix for the measurement term in the toy problem is

$$J_z = \begin{array}{c} \begin{matrix} x_1 & x_2 & x_3 & l_1 & l_2 \end{matrix} \\ \left[\begin{array}{ccc|cc} H_1^x & & & & \\ H_{11}^x & & & H_{11}^l & \\ & H_{21}^x & & & H_{21}^l \\ & & H_{32}^x & & H_{32}^l \end{array}\right] \end{array} \begin{matrix} z_1 \\ z_{11} \\ z_{21} \\ z_{32} \end{matrix} \tag{2.87}$$



(a) Structure of the measurement Jacobian $J_z = H$



(b) Structure of the Hessian, as a composition from prior, motion and measurement. Note the difference in notation.

Figure 2.6: Visualization of the Jacobian and Hessian of a general SLAM problem [2].

It is clear from the above derivations that every factor corresponds to a block row and every variable corresponds to a block column in the Jacobian matrix. If we combine all three Jacobians together, there will be a total of 9 non-zero blocks in the full Jacobian matrix J, one for every factor in the factor graph of the toy SLAM problem. Figure 2.6a shows the Jacobian matrix for a general SLAM problem, where the state are partitioned into a pose group (left A) and a map group (right B) [2].

Sparse Information Matrix

We then turn our attention to the information matrix (Hessian) $A = J^\top \Omega J$. Since the Jacobian J is block sparse and the weight Ω matrix is block diagonal, the information matrix A is expected to be sparse as well. The information matrix of the toy problem looks like

$$A = J^\top \Omega J = \begin{array}{c} \begin{array}{ccccc} x_1 & x_2 & x_3 & l_1 & l_2 \end{array} \\ \left[\begin{array}{ccc|cc} A_{11} & A_{12} & & A_{14} & \\ A_{21} & A_{22} & A_{23} & A_{24} & \\ & A_{32} & A_{33} & & A_{35} \\ \hline A_{41} & A_{42} & & A_{44} & \\ & & A_{53} & & A_{55} \end{array}\right] \end{array} \begin{array}{c} x_1 \\ x_2 \\ x_3 \\ l_1 \\ l_2 \end{array} \tag{2.88}$$

The information matrix A is by construction symmetric, and positive definite if a unique solution to the linear system exists. The information matrix of a general SLAM problem is shown in Figure 2.6b (albeit with a slightly different notation).

It is worth pointing out that the block sparsity of the information matrix can be interpreted as the adjacency matrix of an undirected graph where non-zero off-diagonal entries encode conditional dependencies between variables. This graphical model is called a *Markov random field (MRF)* and is shown in Figure 2.7. An MRF is similar to a Bayes net in that it involves only the variables, except that it is undirected and only

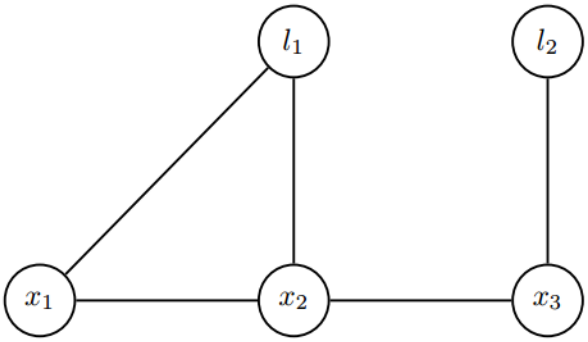


Figure 2.7: The information matrix A can be interpreted as the adjacency matrix of the Markov random filed of the problem [7].

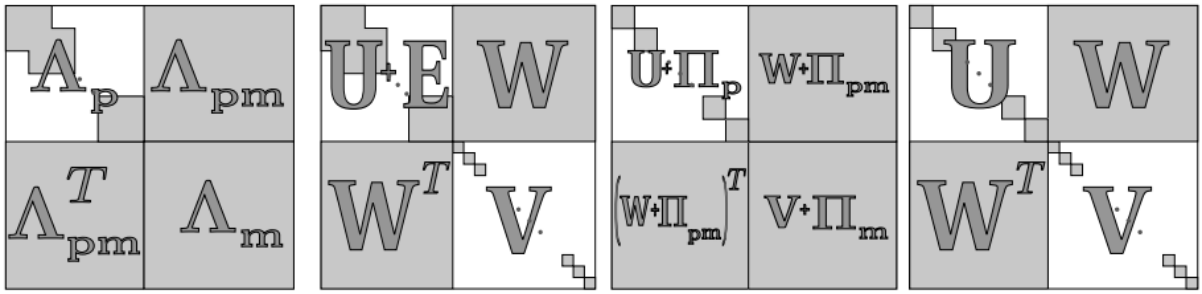


Figure 2.8: Patterns of information matrix of different SLAM variations [2].

indicates some interaction between the variables involved.

We plug all the above terms back in to the linear system in equation (2.66) to obtain

$$(P^{-1} + G^\top Q^{-1} G + H^\top R^{-1} H)\Delta\theta^* \tag{2.89}$$
$$= -\left(P^{-1}(\bar{m} - \check{m}) + G^\top Q^{-1}(\bar{f} - \bar{x}) + H^\top R^{-1}(\bar{h} - z)\right) \tag{2.90}$$

The information matrix shows different sparsity patterns depends on the type of information available. This is visually illustrated in Figure 2.8, where we follow the convention to group variables into pose and map. The leftmost one is the full SLAM problem where prior, motion and measurement all contribute to the problem. The second one to the left is the GraphSLAM [28] problem without prior information. The third one is when we don't have motion information but has prior knowledge. And the last one is the well-known *Bundle Adjustment (BA)* [29] problem where all we have is measurements. Note the slight difference in the top left corner of the Hessian (the pose block) when motion information is and is not available. In discrete-time formulation, due to the Markov assumption, the motion information matrix is block tridiagonal, which is different from the exact block diagonal matrix without motion information.

Reduced System via Schur Complement

We further investigate the type of problems that have sparse motion and map information matrix (without a dense prior information block). This special structure can be taken advantage of to efficiently solve the linear system. To start, let us write the linear system in its block-partitioned form

$$\begin{bmatrix} A_{pp} & A_{pm} \\ A_{pm}^\top & A_{mm} \end{bmatrix} \begin{bmatrix} \Delta x_p^* \\ \Delta x_m^* \end{bmatrix} = \begin{bmatrix} -b_p \\ -b_m \end{bmatrix} \tag{2.91}$$

Note that A_{mm} is exactly block diagonal (no correlation between landmarks), and A_{pp} is block tridiagonal (Markov property). We can use Schur complement to reduce the lower right block A_{mm} onto the upper left block A_{pp}.

$$\begin{bmatrix} A_{pp} - A_{pm}A_{mm}^{-1}A_{pm}^\top & 0 \\ A_{pm}^\top & A_{mm} \end{bmatrix} \begin{bmatrix} \Delta x_p^* \\ \Delta x_m^* \end{bmatrix} = \begin{bmatrix} -b_p + A_{pm}A_{mm}^{-1}b_m \\ -b_m \end{bmatrix} \tag{2.92}$$

Because A_{mm} is block-diagonal, its inverse can be computed by inverting each (relatively small) block individually. Solving the top part of the new system yields the optimal increment of poses Δx_p^*

$$(A_{pp} - A_{pm}A_{mm}^{-1}A_{pm}^\top)\Delta x_p^* = -b_p + A_{pm}A_{mm}^{-1}b_m \tag{2.93}$$

$$\text{or} \quad A'_{pp}\Delta x_p^* = b'_p \tag{2.94}$$

after which Δx_m^* can be obtained via back-substitution with Δx_p^*

$$A_{pm}^\top \Delta x_p^* + A_{mm}\Delta x_m^* = -b_m \tag{2.95}$$

Alternatively, one could also reduce the pose Hessian to the map Hessian. This particular choice depends on the size and sparsity pattern of the system at hand.

Unlike the original information blocks (A_{pp} and A_{mm}) which are exactly sparse, the reduced information matrix A'_{pp} is typically not excatly sparse and has some fill-ins originated from the covisibility of landmarks from different poses. However, for very large systems, A'_{pp} is still relatively sparse due to the fact that not all landmarks are observed by all poses [30].

Reduced System via Marginal Distribution

Another way to look at this is to view the full linear system as a multivariate Gaussian distribution on Δx^*

$$\mathcal{N}(\Delta x^*; \mu, \Sigma) = \mathcal{N}\left(\begin{bmatrix}\Delta x_p^* \\ \Delta x_m^*\end{bmatrix}; \begin{bmatrix}\mu_p \\ \mu_m\end{bmatrix}, \begin{bmatrix}A_{pp} & A_{pm} \\ A_{pm}^\top & A_{mm}\end{bmatrix}^{-1}\right) \tag{2.96}$$

where

$$\begin{bmatrix}\mu_p \\ \mu_m\end{bmatrix} = \begin{bmatrix}A_{pp} & A_{pm} \\ A_{pm}^\top & A_{mm}\end{bmatrix}^{-1} \begin{bmatrix}-b_p \\ -b_m\end{bmatrix} \tag{2.97}$$

Under this viewpoint, the reduced linear system is simply the marginal distribution of a multivariate Gaussian in covariance form [24]

$$\mathcal{N}(\Delta x_p^*; \mu_p, \Sigma_{pp}) \tag{2.98}$$

After manipulating the Gaussian we have

$$\Sigma_{pp} = (A_{pp} - A_{pm} A_{mm}^{-1} A_{pm}^\top)^{-1} \tag{2.99}$$

$$\mu_p = \Sigma_{pp}(-b_p + A_{pm} A_{mm}^{-1} b_m) \tag{2.100}$$

and is the same as the result via Schur complement.

Partitioned Gaussian Distribution

A joint Gaussian distribution $p(x) = p(x_a, x_b)$ can be written in both covariance and information (canonical) form

$$p(x) \sim \mathcal{N}(\mu, \Sigma) = \mathcal{N}(\Lambda^{-1}\eta, \Lambda^{-1}) \tag{2.101}$$

with the following partition

$$x = \begin{bmatrix}x_a \\ x_b\end{bmatrix}, \quad \mu = \begin{bmatrix}\mu_a \\ \mu_b\end{bmatrix}, \quad \eta = \begin{bmatrix}\eta_a \\ \eta_b\end{bmatrix}, \quad \Sigma = \begin{bmatrix}\Sigma_{aa} & \Sigma_{ab} \\ \Sigma_{ba} & \Sigma_{bb}\end{bmatrix}, \quad \Lambda = \begin{bmatrix}\Lambda_{aa} & \Lambda_{ab} \\ \Lambda_{ba} & \Lambda_{bb}\end{bmatrix} \tag{2.102}$$

The marginal distribution is simple in covariance form, which is simply expressed in terms of the partitioned covariance matrix, but more complicated in information

form

$$p(x_a) = \mathcal{N}(\mu_a, \Sigma_{aa}) \tag{2.103}$$
$$\Sigma_{aa} = (\Lambda_{aa} - \Lambda_{ab}\Lambda_{bb}^{-1}\Lambda_{ba})^{-1} \tag{2.104}$$
$$\mu_a = \Sigma_{aa}(\eta_a - \Lambda_{ab}\Lambda_{bb}^{-1}\eta_b) \tag{2.105}$$

On the other hand, the conditional distribution is relatively simple in information form, but complicated in covariance form

$$p(x_a|x_b) = \mathcal{N}(\mu_{a|b}, \Sigma_{a|b}) \tag{2.106}$$
$$\Sigma_{a|b} = \Lambda_{aa}^{-1} = \Sigma_{aa} - \Sigma_{ab}\Sigma_{bb}^{-1}\Sigma_{ba} \tag{2.107}$$
$$\mu_{a|b} = \mu_a - \Lambda_{aa}^{-1}\Lambda_{ab}(x_b - \mu_b) \tag{2.108}$$
$$= \mu_a + \Sigma_{ab}\Sigma_{bb}^{-1}(x_b - \mu_b) \tag{2.109}$$

3 | Continuous-time SLAM

In the previous chapter, we derived the probabilistic formulation for discrete-time SLAM and explored its special structure. We conclude that there are two main factors that determine how efficiently a SLAM problem can be solved: (1) the size of the state vector, and (2) the sparsity of the information matrix. We saw that the discrete-time formulation does maintain the sparsity of the problem (by virtue of the Markov assumption on the robot motion), but naively applying it in some problematic situations (high-rate sensor, long-term navigation, *etc.*) will cause the state vector to grow out of control.

In this chapter, we will look at various continuous-time formulations which were designed to mitigate this issue. We start with a parametric approach using a weighted sum of a set of temporal basis functions [3]. We then show an equivalent, non-parametric formulation using modeled after Gaussian processes [4]. Finally, we observe that the complexity of SLAM (solving the linearized system) only relies on the error state vector, and thus one could decouple the trajectory estimate from its error, and obtain a hybrid representation [5].

3.1 Parametric Representation

In this section, we review **Continuous-time Batch Trajectory Estimation using Temporal Basis Functions** by Furgale *et.al.* [3]. Their main idea is to parameterize the robot motion by time, and then use the weighted sum of a finite set of basis functions (also parameterized by time, thus are named *temporal basis functions*) to approximate it. With this modification, the state vector is no longer a collection of all robot poses at measurement times, but a set of coefficients for the basis functions. Depending on the complexity of and assumption we made on the robot motion (in real world, robot motions are typically smooth), one could use a significantly smaller state vector to represent the same trajectory.

3.1.1 Problem Formulation

We start by derive the probabilistic formulation for the full SLAM problem with a continuous-time state and discrete-time measurements. Our derivation closely follows that of the discrete-time one from Section 2.1.2 in order to present the reader with a direct comparison of the two approaches. Most of the quantities from the discrete-time version remain the same in continuous-time domain, with the only difference that we parameterize the robot trajectory and controls by time t:

- $x(t)$: the state vector describing the pose of the robot at time t.
- $u(t)$: the control input to the robot at time t.

With these modifications, the posterior probability from equation (2.28) becomes

$$
\begin{aligned}
p(X|Z) &= p(x(t), m|u(t), z_{1:N}) && (3.1)\\
&= \frac{p(x(t), m|u(t))p(z_{1:N}|x(t), m, u(t))}{p(z_{1:N}|u(t))} && (3.2)\\
&= \frac{p(x(t), m|u(t))p(z_{1:N}|x(t), m)}{p(z_{1:N})} && (3.3)\\
&= \frac{p(m)p(x(t)|u(t))\prod_{n=1}^{N} p(z_n|x(t_i), m)}{p(z_{1:N})} && (3.4)
\end{aligned}
$$

Note that we do not have an explicit prior on the first pose $p(x(t_0))$, which is subsumed by the motion term $p(x(t)|u(t))$. The measurement model (2.14) with a continuous-time state now becomes

$$
\begin{aligned}
z_i &= h(x(t_i), m) + n_i, \quad n_i \sim \mathcal{N}(0, R_i) && (3.5)\\
p(z_i|x(t_i), m) &\sim \mathcal{N}(h(x(t_i), m), R_i) && (3.6)
\end{aligned}
$$

The motion probability density $p(x(t)|u(t))$ may be specified as a continuous stochastic dynamical system described by the following differential equation

$$
\dot{x}(t) = f(x(t), u(t)) + w(t), \quad w(t) \sim \mathcal{GP}(0, Q\delta(t - t')) \tag{3.7}
$$

where $w(t)$ is a zero-mean white Gaussian process, $\delta(\cdot)$ the Dirac delta function and Q a power-spectral-density matrix. We assume Q to be a given parameter from a system identification process. The probability density of the motion model is

$$
p(x(t)|u(t)) \propto p(x(t_0)) \exp\left\{-\frac{1}{2}\int_{t_0}^{t_K} \|\dot{x}(\tau) - f(x(\tau), u(\tau))\|_Q^2 d\tau\right\} \tag{3.8}
$$

This factor is a prior over the entire robot trajectory that will be modified by the mea-

	Discrete-time	Continuous-time
Posterior	$p(x_{0:K}, m \mid u_{1:K}, z_{1:N})$	$p(x(t), m \mid u(t), z_{1:N})$
Prior	$\theta \sim \mathcal{N}(\check{\theta}, P)$	same
Motion	$x_k \sim \mathcal{N}(f(x_{k-1}, u_k), Q_k)$	$\dot{x}(t) = f(x(t), u(t)) + w(t)$ $w(t) \sim \mathcal{GP}(0, Q\delta(t - t'))$
Measurement	$z_i \sim \mathcal{N}(h(x_i, m), R_i)$	$z_i \sim \mathcal{N}(h(x(t_i), m), R_i)$

Table 3.1: A comparison of derivations of discrete-time and continuous-time SLAM [3]

surements. The matrix Q is a hyperparameter that can be used to adjust the smoothness of the solution. The prior also plays a critical role in solving problems that are under-constrained.

We have now specified a from for all of the probability densities in the posterior except $p(z_i)$, which is constant and inconsequential to our problem. A direct comparison of the discrete-time and continous-time formulation is summarized in Table 3.1.

3.1.2 MAP Estimation

The MAP estimator for our continuous-time formulation is

$$\theta^* = \arg\max_{\theta} p(x(t), m \mid u(t), z_{1:N}) \tag{3.9}$$

$$= \arg\min_{\theta} \{-\ln p(x(t), m \mid u(t), z_{1:N})\} \tag{3.10}$$

where we define

$$\theta(t) := \begin{bmatrix} x(t) \\ m \end{bmatrix} \tag{3.11}$$

to be the combined state vector. Similar to Equation (2.32), this can be break down into the same set of groups of prior, motion and measurement

$$-\ln p(x(t), m \mid u(t), z_{1:N}) = \eta + L_p + L_u + L_z \tag{3.12}$$

with the only notable difference regarding the motion term

$$-\ln p(x(t) \mid u(t)) \propto L_u = \frac{1}{2} \int_{t_0}^{t_K} \|f(x(\tau), u(\tau)) - \dot{x}(\tau)\|_Q^2 d\tau + \frac{1}{2} \|x_0 - \check{x}_0\|_{P_0}^2 \tag{3.13}$$

The rest of the objectives are essentially the same from our discrete-time variant (2.40).

3.1.3 Basis Functions Formulation

Up till now we have been working with a time parameterized trajectory $x(t)$, but haven't specify what form it would take. A natural choice is to approximate $x(t)$ as a weighted sum of a finite set of known temporal basis functions

$$\Phi(t) \coloneqq \begin{bmatrix} \phi_1(t) & \cdots & \phi_M(t) \end{bmatrix}, \quad x(t) \coloneqq \Phi(t)c \tag{3.14}$$

where c is a $M \times 1$ vector of weights or coefficients. Each individual basis function $\phi_j(t)$ has the same dimension as the state of the robot. Under the basis function formulation, our goal is to estimate the weight vector c, instead of a set of poses as in the discrete-time one. In general the length of c is much smaller than x, and significant reduction in computation can be achieved.

Let use also define the following terms, where β is the new state vector of the continuous-time formulation

$$\theta(t) = \begin{bmatrix} x(t) \\ m \end{bmatrix} = \begin{bmatrix} \Phi(t)c \\ m \end{bmatrix} = \Psi(t)\beta, \quad \Psi(t) = \begin{bmatrix} \Phi(t) & \\ & 1 \end{bmatrix}, \quad \beta = \begin{bmatrix} c \\ m \end{bmatrix} \tag{3.15}$$

and under the basis function formulation, the optimal increment we are looking for at each iteration of the nonlinear least squares problem is $\Delta\beta^*$ instead of $\Delta\theta^*$.

This is also called a *parametric model* [24] where our model has a specific functional form (the basis functions $\Phi(t)$) governed by a small number of parameters (the coefficient c) which are to be determined from the data. As with all parametric models, the chosen functional forms might be a poor approximation to the underlying process, thus resulting in poor approximation. This is also called *underfitting*.

Without specifying the exact form of the basis functions, we may now write down the MAP estimator in terms of β

$$\beta^* = \arg\min_{\beta} \frac{1}{2} \left\{ \|m - \check{m}\|^2_{P_m} + \int_{t_0}^{t_K} \|e_u(\tau)\|^2_Q d\tau + \|h(\Psi\beta) - z\|^2_R \right\} \tag{3.16}$$

For the motion term in the objective function we have

$$L_u = \frac{1}{2} \int_{t_0}^{t_K} \|e_u(\tau)\|^2_Q d\tau, \quad e_u(t) = f(x(t), u(t)) - \dot{x}(t) \tag{3.17}$$

Linearizing the motion error function at $\bar{c}$ gives us the following

$$
\begin{aligned}
e_u(t) &= f(x(t), u(t)) - \dot{x}(t) = g(t) && \text{(Error function)} && (3.18)\\
&= f(\Phi(t)(\bar{c} + \Delta c), u(t)) - \dot{\Phi}(t)(\bar{c} + \Delta c) && && (3.19)\\
&= f(\Phi(t)\bar{c}, u(t)) + \left.\frac{\partial f}{\partial x}\right|_{\bar{x}(t)} \Phi(t)\Delta c - \dot{\Phi}(t)\bar{c} - \dot{\Phi}(t)\Delta c && && (3.20)\\
&= \left\{ f(\Phi(t)\bar{c}, u(t)) - \dot{\Phi}(t)\bar{c} \right\} + \left\{ F(t)\Phi(t) - \dot{\Phi}(t) \right\} \Delta c && && (3.21)\\
&= \bar{e}_u(t) + J_u(t)\Delta c && \text{(Linearization)} && (3.22)\\
J_u(t) &= F(t)\Phi(t) - \dot{\Phi}(t) = G(t) && \text{(Jacobian)} && (3.23)
\end{aligned}
$$

However, due to the special form of the motion objective (involves an integration), we cannot directly use the discrete-time equation (2.66). Instead, we directly linearize the motion objective L_u and take the derivative with respect to Δc (we intentionally ignore the map m part of the state as it is not relevant to this term)

$$
\begin{aligned}
L_u &= \frac{1}{2}\int_{t_0}^{t_K} \|e_u(\tau)\|_Q^2 d\tau \approx \frac{1}{2}\int_{t_0}^{t_K} \|\bar{e}_u(\tau) + J_u(\tau)\Delta c\|_Q^2 d\tau && (3.24)\\
\frac{\partial L_u}{\partial \Delta c} &= \left(\int_{t_0}^{t_K} G(\tau)^\top Q^{-1} G(\tau) d\tau \right) \Delta c + \int_{t_0}^{t_K} G(\tau)^\top Q^{-1} \bar{g}(\tau) d\tau && (3.25)
\end{aligned}
$$

Setting the derivative to 0 gives us

$$
\begin{aligned}
A_u &= \int_{t_0}^{t_K} G(\tau)^\top Q^{-1} G(\tau) d\tau && \text{(Hessian)} && (3.26)\\
b_u &= \int_{t_0}^{t_K} G(\tau)^\top Q^{-1} \bar{g}(\tau) d\tau && \text{(Information vector)} && (3.27)
\end{aligned}
$$

For the measurement term in the objective function we have

$$
\begin{aligned}
e_z(\theta) &= h(\theta) - z = h(\Psi\beta) - z && \text{(Error function)} && (3.28)\\
e_z(\bar{\theta} + \Delta\theta) &\approx h(\Psi\bar{\beta}) + H\Psi\Delta\beta - z = \bar{e}_z + H\Psi\Delta\beta && \text{(Linearization)} && (3.29)\\
J_z &= \frac{\partial e_z(\theta)}{\partial \Delta\beta} = H\Psi, \quad H = \left.\frac{\partial h(\theta)}{\partial \theta}\right|_{\theta=\bar{\theta}} && \text{(Jacobian)} && (3.30)\\
A_z &= J_z^\top \Omega_z J_z = \Psi^\top H^\top R^{-1} H\Psi && \text{(Hessian)} && (3.31)\\
b_z &= J_z^\top \Omega_z \bar{e}_z = \Psi^\top H^\top R^{-1}(\bar{h} - z) && \text{(Information vector)} && (3.32)
\end{aligned}
$$

A direct comparison of the information matrices and vectors are shown in Table 3.2.

Once the the building blocks of each individual objective term are derived, optimization can be carried out in the same manner, with the difference that the state

	Discrete-time	Continuous-time
A_u	$G^\top Q^{-1} G$	$\int_{t_0}^{t_K} G(\tau)^\top Q^{-1} G(\tau) d\tau$
b_u	$G^\top Q^{-1}(\bar{f} - \bar{x})$	$\int_{t_0}^{t_K} G(\tau)^\top Q^{-1} \bar{g}(\tau) d\tau$
A_z	$H^\top R^{-1} H$	$\Psi^\top H^\top R^{-1} H \Psi$
b_z	$H^\top R^{-1}(\bar{h} - z)$	$\Psi^\top H^\top R^{-1}(\bar{h} - z)$

Table 3.2: A comparison of linear systems of discrete-time and continuous-time SLAM.

vector now contains the weight for the basis fucntions instead of discrete poses

$$\begin{aligned} &(A_p + A_u + A_z)\Delta\beta^* = -(b_p + b_u + b_z) \\ &\left(P_m^{-1} + \int_{t_0}^{t_K} G(\tau)^\top Q^{-1} G(\tau) d\tau + \Psi^\top H^\top R^{-1} H \Psi\right) \Delta\beta^* \\ =& -\left(P_m^{-1}(\bar{m} - \check{m}) + \int_{t_0}^{t_K} G(\tau)^\top Q^{-1} \bar{g}(\tau) d\tau + \Psi^\top H^\top R^{-1}(\bar{h} - z)\right) \end{aligned} \tag{3.33}$$

We start from an initial guess $\bar{\beta}$, solve the linearized system and update the linearization point until convergence.

3.1.4 Discussion

Representing robot trajectory as a weighted sum of many basis functions has the direct advantage of reducing the size of the state vector. However, if we are not careful, it will destroy the sparsity of the motion information block (top left) in the information matrix. This can be shown in Figure 3.1 where we compare information matrices of several formulation. The leftmost one is the information matrix of a discrete-time formulation. The upper left motion block is sparse (block-tridiagonal) due to the Markov property. The middle one shows the information matrix resulting from a general set of basis functions. We see that even though the size of the motion block is much smaller than that of the discrete-time one, it is completely dense. This is easy to understand because any pose $x(t)$ is a function of all the basis coefficients c.

To restore sparsity, we need a set of basis functions that has the following properties:

1. Local support: the contribution of any single basis function should be local in time.
2. Continuity: we would like the basis functions to be continuous in its second order derivatives.
3. Simple analytical derivatives: this allows the Jacobian matrices to be computed efficiently.

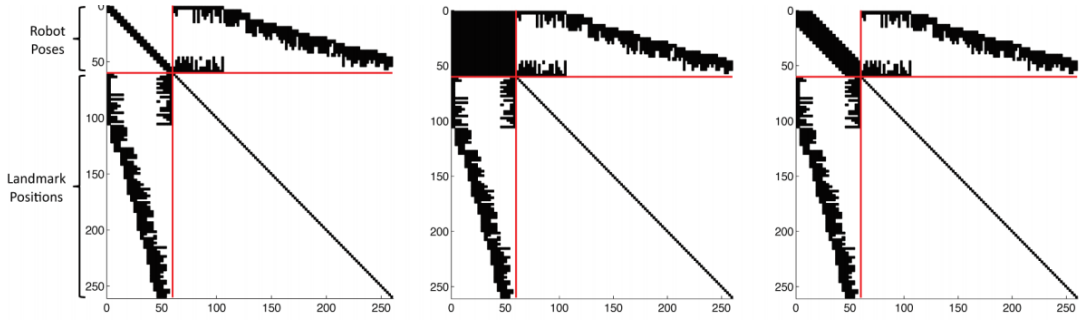


Figure 3.1: Sparsity patterns of the information matrix for a sample two-dimensional SLAM problem involving 20 robot poses and 100 landmarks [4].

B-spline functions become an attractive choice under these criteria. For example, if we employ a cubic B-Spline as our basis functions, then the state vector becomes the control points of the B-spline. Any pose in time is a linear combination of 4 neighbouring control points and the pose information block will become band-diagonal again as shown in the rightmost image in Figure 3.1. One can control the complexity of the trajectory by placing more knots and thus more control points in the B-spline basis. For more details about representing elements of Lie Groups with B-splines and computing their derivatives, please see [31].

3.2 Non-parametric Representation

In this section, we review **Gaussian Process Gauss-Newton for Non-parametric Simultaneous Localization and Mapping** by Tong *et.al.* [4]. Their work introduces Gaussian Process Gauss-Newton (GPGN), which is an algorithm for non-parametric, continuous-time, nonlinear, batch state estimation.

Gaussian process [32] is a commonly used tool in the machine learning community for supervised learning. In supervised learning, the goal of a regression problem is to predict values for an unknown continuous function at new inputs given data. State estimation can be thought of as a special type of regression problem, where we seek to estimate robot states at some instants of time given noisy measurements. However, the subtle difference is that instead of making predictions about future measurements, we want to recover the latent states that generated the observed measurements. In SLAM, the latent states are the robot trajectory and the map.

As we've hinted in the previous section, there are generally two approaches to the regression problem: 1) parametric and 2) non-parametric. In parametric regression, the unknown function is assumed to have a particular form governed by some parameters. For example, as the weighted sum of a set of known basis functions [3]. With this model, the goal is to find the optimal set of parameters that allows the function to

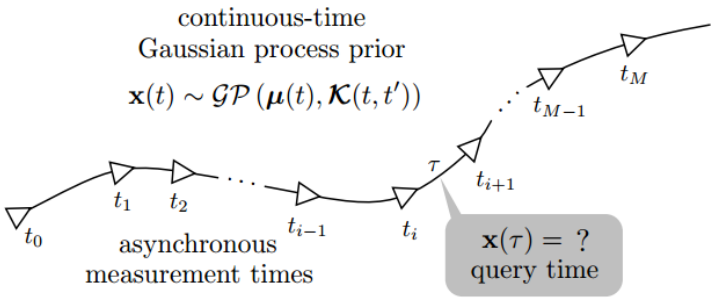


Figure 3.2: The trajectory of the robot can be represented by a continuous-time Gaussian process prior [8]. This allows us to query the trajectory at any time of interest τ. Note that measurements are still at discrete time.

best fit the data. New predictions can be made by evaluating the function at different points. The discrete-time SLAM problem can be viewed as a parametric model, where the robot trajectory is parameterized by the set of poses at instants of time. It can be extended to continuous domain by adopting a continuous state representation such as B-splines [33].

Unlike parametric methods, non-parametric ones make fewer assumptions about the form of the function. Whereas the number of parameters in a parametric model stays the same irrespective of the amount of data, in a non-parametric model, the effective parameters grows with the number of data points.

3.2.1 Problem Formulation

The formulation of the non-parametric model bears a lot of similarities to that of the parametric one. We directly model the motion prior as a Gaussian process shown in Figure 3.2

$$x(t) \sim \mathcal{GP}(\check{x}(t), K(t,t')) \tag{3.34}$$

where $\check{x}(t)$ is the mean function and $K(t,t')$ is the covariance function. This means that any finite number of robot poses will have a joint Gaussian distribution.

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \check{x}_1 \\ \check{x}_2 \end{bmatrix}, \begin{bmatrix} K_{11} & K_{12} \\ K_{21} & K_{22} \end{bmatrix} \right) \tag{3.35}$$

where $x_i := x(t_i)$ and $K_{ij} := K(t_i, t_j)$. Prior and measurement model remain the same

$$p(m) \sim \mathcal{N}(\check{m}, P_m) \tag{3.36}$$

$$p(z_i | x(t_i, m)) \sim \mathcal{N}(h(x(t_i), m), R_i) \tag{3.37}$$

In contrast to GPs in traditional regression settings, we assume that the hyper-parameters of the GP are already known. We combine the time-variant and time-invariant variables (x and m) into a full state vector θ

$$\theta(t) = \begin{bmatrix} x(t) \\ m \end{bmatrix}, \quad P(t,t') = \begin{bmatrix} K(t,t') & 0 \\ 0 & P_m \end{bmatrix} \tag{3.38}$$

which results in the following simplified system equations

$$\theta(t) \sim \mathcal{GP}(\check{\theta}(t), P(t,t')) \tag{3.39}$$

$$z_i = h(\theta(t_i)) + n_i \tag{3.40}$$

Like the parametric version, the posterior probability is essentially the same

$$p(X|Z) = p(x(t), m|u(t), z_{1:N}) \tag{3.41}$$

$$= \frac{p(m)p(x(t)|u(t)) \prod_{n=1}^{N} p(z_n|x(t_i), m)}{p(z_{1:N})} \tag{3.42}$$

The MAP estimator of the above posterior is

$$\theta^* = \arg\min_{\theta} \frac{1}{2} \left\{ \|\theta - \check{\theta}\|_P^2 + \|h(\theta) - z\|_R^2 \right\} \tag{3.43}$$

with the following linear system

$$(P^{-1} + H^\top R^{-1} H)\Delta\theta^* = -(P^{-1}(\bar{\theta} - \check{\theta}) + H^\top R^{-1}(\bar{h} - z)) \tag{3.44}$$

with

$$P = \begin{bmatrix} K & 0 \\ 0 & P_m \end{bmatrix}, \quad K \coloneqq [K(t_i,t_j)]\big|_{1 \le i,j \le N} = \begin{bmatrix} K(t_1,t_1) & \cdots & K(t_1,t_N) \\ \vdots & \ddots & \vdots \\ K(t_N,t_1) & \cdots & K(t_N,t_N) \end{bmatrix} \tag{3.45}$$

3.2.2 Weight-space Derivation

We first show a parametric, weight-space derivation using the GP framework. Recall the basis functions representation for the state is

$$\theta(t) = \Psi(t)\beta \tag{3.46}$$

$$\Psi(t) = \begin{bmatrix} \Phi(t) & 0 \\ 0 & 1 \end{bmatrix}, \quad \beta = \begin{bmatrix} c \\ m \end{bmatrix} \tag{3.47}$$

$$\Phi(t) = \begin{bmatrix} \phi_1(t) & \cdots \phi_M(t) \end{bmatrix} \tag{3.48}$$

These are the same definitions that we had before for the parametric model. However, we assume that the number of basis functions, M, is very large or even infinite. This provides substantial representational power, but with the restriction that we cannot evaluate or store either. With this representation, the state can be written as

$$\theta(t) \sim \mathcal{GP}(\check{\theta}(t), P(t,t')) \tag{3.49}$$

$$\check{\theta}(t) = \Psi(t)\check{\beta}, \quad P(t,t') = \Psi(t)B\Psi(t')^\top \tag{3.50}$$

where β has the following distribution

$$\beta \sim \mathcal{N}(\check{\beta}, B), \quad B = \begin{bmatrix} P_c & 0 \\ 0 & P_m \end{bmatrix} \tag{3.51}$$

These substitutions return us to the familiar Gaussian random variable domain, which is the opposite of applying the kernel trick [24, 32].

The MAP estimator now becomes

$$\theta^* = \arg\min_\theta \frac{1}{2}\left\{ \|\theta - \check{\theta}\|_P^2 + \|h(\theta) - z\|_R^2 \right\} \tag{3.52}$$

$$\Rightarrow \beta^* = \arg\min_\beta \frac{1}{2}\left\{ \|\Psi(\beta - \check{\beta})\|_{\Psi B \Psi^\top}^2 + \|h(\Psi\beta) - z\|_R^2 \right\} \tag{3.53}$$

$$= \arg\min_\beta \frac{1}{2}\left\{ \|\beta - \check{\beta}\|_B^2 + \|h(\Psi\beta) - z\|_R^2 \right\} \tag{3.54}$$

Linearizing the objective function and setting the derivative to 0 we have

$$(B^{-1} + \Psi^\top H^\top R^{-1} H \Psi)\Delta\beta^* = -(B^{-1}(\bar{\beta} - \check{\beta}) + \Psi^\top H^\top R^{-1}(\bar{h} - z)) \tag{3.55}$$

which is exactly the same as equation (3.33) if we use the same motion model

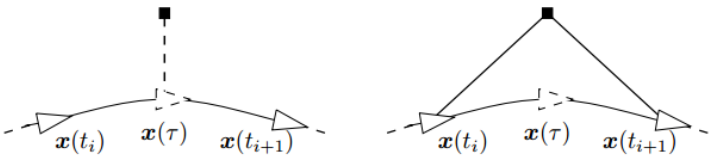


Figure 3.3: Measurement at state $x(\tau)$ (a) does not create an actual factor, the state $x(\tau)$ is instead interpolated by nearby states [9].

3.2.3 Prediction and Interpolation

To understand how to apply the non-parametric method to SLAM, we need to first understand how to make predictions in Gaussian processes. Assume we have a 1-D GP of the following

$$x(t) \sim \mathcal{GP}(m(t), k(t, t')) \tag{3.56}$$

where $m(t)$ is the mean function and $k(t, t')$ the covariance function. In the context of supervised learning, given training data $y = \{y_1, \ldots, y_M\}$, the joint probability distribution of y and new predictions $x = \{x_1, \ldots, x_N\}$ is given by the following density

$$\begin{bmatrix} y \\ x \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \check{y} \\ \check{x} \end{bmatrix}, \begin{bmatrix} K_{yy} & K_{yx} \\ K_{xy} & K_{xx} \end{bmatrix} \right) \tag{3.57}$$

Therefore, we can infer the posterior probability of $p(x|y)$ by leveraging the conditional distribution of multivariate Gaussian in equation (2.109)

$$p(x|y) = \mathcal{N}(\check{x} + K_{xy}K_{yy}^{-1}(y - \check{y}), K_{xx} - K_{xy}K_{yy}^{-1}K_{yx}) \tag{3.58}$$

Note that the mean prediction of x is a linear combination of observations y and is referred to as a linear predictor [32].

Now returning to our SLAM problem. With the assumption that the robot trajectory has a Gaussian process prior, given the current estimation of poses at some instances of time, we can use equation (3.58) to predict poses at other times of interest. For example, Figure 3.3 shows a local view of a factor graph of a SLAM problem. We have a measurement z at time τ from pose $x(\tau)$. Using the GP model, the state $x(\tau)$ can be interpolated by its nearby states $x(t_i)$ and $x(t_{i+1})$ that are actually in the state variable. Thus we can add a binary factor to the factor graph without adding $x(\tau)$ to the state variable.

Let us denote the set of poses that we will actually estimate be $y = y_{1:M}$, which is a subset of all the poses x. We can then predict or interpolate other poses x using the exact same equation

$$x = \check{x} + K_{xy}K_{yy}^{-1}(y - \check{y}) \tag{3.59}$$

We can linearize it and obtain the following relationship of the delta terms

$$\bar{x} + \Delta x = \check{x} + K_{xy} K_{yy}^{-1} (\bar{y} + \Delta y - \check{y}) \tag{3.60}$$

$$\Delta x = K_{xy} K_{yy}^{-1} \Delta y \tag{3.61}$$

We see that given the covariance matrix, we can express any pose as a linear combination of a subset of poses y, which is of much smaller size. Therefore we define a new state vector

$$\gamma = \begin{bmatrix} y \\ m \end{bmatrix} \tag{3.62}$$

where y is a subset of all the poses x that we wish to estimate. We have the following relationship between the original state vector θ and the reduced one γ

$$\theta = \check{\theta} + P_{xy} P_{yy}^{-1} (\gamma - \check{\gamma}) = \check{\theta} + \Pi(\gamma - \check{\gamma}) \tag{3.63}$$

$$\Delta\theta = P_{xy} P_{yy}^{-1} \Delta\gamma = \Pi \Delta\gamma \tag{3.64}$$

Now our MAP estimator becomes

$$\gamma^* = \arg\min_{\gamma} \left\{ \frac{1}{2} \|\theta - \check{\theta}\|_P^2 + \|h(\theta) - z\|_R^2 \right\} \tag{3.65}$$

$$= \arg\min_{\gamma} \left\{ \frac{1}{2} \|\Pi(\gamma - \check{\gamma})\|_P^2 + \|h(\check{\theta} + \Pi(\gamma - \check{\gamma})) - z\|_R^2 \right\} \tag{3.66}$$

Linearizing at $\bar{\gamma}$ and setting the derivative to 0 we have

$$(\Pi^\top P^{-1} \Pi + \Pi^\top H^\top R^{-1} H \Pi) \Delta\gamma^* = -(\Pi^\top P^{-1} (\bar{\theta} - \check{\theta}) + \Pi^\top H^\top R^{-1} (\bar{h} - z)) \tag{3.67}$$

If y is the same as x, then we have $\Pi = P_{xx} P_{xx}^{-1} = I$ and $\theta = \gamma$. We see that the above linear system becomes exactly the discrete-time linear system in equation (3.44)

$$(P^{-1} + H^\top R^{-1} H) \Delta\theta^* = -(P^{-1} (\bar{\theta} - \check{\theta}) + H^\top R^{-1} (\bar{h} - z)) \tag{3.68}$$

Also note that we have

$$\Delta\theta = \Psi \Delta\beta = \Pi \Delta\gamma, \quad P = \Psi B \Psi^\top \tag{3.69}$$

Plugging the above relationship back into equation (3.67) we arrive at excatly the para-

metric continous-time linear system in equation (3.55)

$$
\begin{aligned}
&(\Pi^\top P^{-1}\Pi + \Pi^\top H^\top R^{-1} H\Pi)\Delta\gamma^* && && (3.70)\\
=&-(\Pi^\top P^{-1}(\bar{\theta}-\check{\theta}) + \Pi^\top H^\top R^{-1}(\bar{h}-z)) && \text{(GPGN)} && (3.71)\\
&(\Pi^\top(\Psi B\Psi^\top)^{-1}\Pi + \Pi^\top H^\top R^{-1} H\Pi)\Pi^{-1}\Psi\Delta\beta^* && && (3.72)\\
=&-(\Pi^\top(\Psi B\Psi^\top)^{-1}\Psi(\bar{\beta}-\check{\beta}) + \Pi^\top H^\top R^{-1}(\bar{h}-z)) && && (3.73)\\
&(B^{-1} + \Psi^\top H^\top R^{-1} H\Psi)\Delta\beta^* && && (3.74)\\
=&-(B^{-1}(\bar{\beta}-\check{\beta}) + \Psi^\top H^\top R^{-1}(\bar{h}-z)) && \text{(Basis)} && (3.75)
\end{aligned}
$$

Hence, we see that all three different formulations (discrete, parametric and non-parametric) are connected and share certain similarities.

3.2.4 The GPGN Algorithm

In summary, the GPGN algorithm is performed as follows:

1. Start by extracting a subset of poses at all measurement times (y or γ), which will be in the state vector and will be estimated. The rest of the states will be interpolated by this set.
2. Initialize the state values for the state that will be estimated ($\bar{y}$ or $\bar{\gamma}$).
3. Interpolate the state values for the rest of the measurement times using equation (3.64).
4. Linearize and construct the linear system from equation (3.67).
5. Solve for the optimal perturbation for the estimated poses $\Delta\gamma^*$.
6. Update the linearization point with $\bar{\gamma} \rightarrow \bar{\gamma} + \Delta\gamma^*$.
7. Repeat 3-6 until convergence.

3.2.5 Discussion

The tradeoffs between this non-parametric approach and the basis function approach are related to the time-varying state representation. In the parametric approach, the computational complexity depends on the number of basis functions. While in the non-parametric approach, it is proportional to the number of estimated poses. To control the complexity of the trajectory, one could simply add or remove poses from the estimated set. Due to the usage of the Kernel trick, we see that the basis and the GP formulation are essentially the same. This means that if we use a general kernel function for the GP, we would also end up with the same dense motion information

block as before. However, Anderson *et.al.* have shown that by carefully choosing the GP prior, the resulting inverse kernel matrix is exactly sparse [8].

3.3 Hybrid Representation

The previous two papers showed that one could use either a parametric or a non-parametric formulation to model continuous-time trajectory and the two approaches are mathematically equivalent. However, one critical question is left unanswered: How does one choose the number of basis functions for the parametric model (or the number of poses for the non-parametric one)? This is essentially the model selection problem and are related to the bias-variance trade-off in machine learning. The choice of model complexity really depends on the motion of the robot, yet in real world, robot motions have vastly different characteristics depending on their applications. For example, a driving vehicle will have a much smoother trajectory than a hand-held sensor rig.

The work **Decoupled Representation of the Error and Trajectory Estimates for Efficient Pose Estimation** by Zheng *et.al.* [5] tackles this problem by decoupling the parameterization of the trajectory estimate from the parameterization of the trajectory error. We name it hybrid because the trajectory estimate is represented in discrete-time, as a set of poses, while the trajectory error is represented in continuous-time, using either a parametric or a non-parametric model.

3.3.1 Motivation

As we have seen over and over again in previous sections, the core of the SLAM algorithm is a *linearization-based*, gradient descent method. Notably, the complexity of the algorithm is a function of the dimension of the linearized system, or the *error-state* vector, which is directly related to the complexity of the sparse Cholesky decomposition of the information matrix. In discrete-time approaches, the dimension of the estimator's error state is directly determined by the number of poses in the trajectory. In continuous-time approaches, the dimension of the error state is drastically reduced, but so does the dimension of the trajectory estimate, which may cause underfitting to complex robot motions. In a hybrid approach, these two quantities are decoupled. The trajectory estimate remains discrete to be able to represent highly complex motions, while the trajectory error is modeled as a continous-time process to reduce the computation cost.

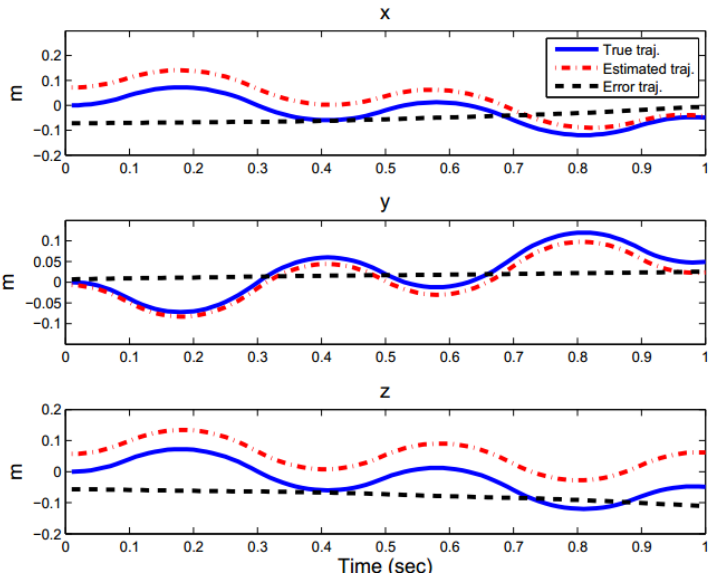


Figure 3.4: Example trajectory and estimation error during a one-second long interval of visual-inertial navigation [5].

Recall the linearized system of a discrete-time SLAM problem

$$(G^\top Q^{-1} G + H^\top R^{-1} H)\Delta\theta^* = -(G^\top Q^{-1}(\bar{f} - \bar{x}) + H^\top R^{-1}(\bar{h} - z)) \tag{3.76}$$

where we ignore the prior term for simplicity. We point out that any linearization-based estimator relies on the computation of (1) measurement residuals, *e.g.*, $\bar{h} - z$, and (2) a linear relationship between the residual and the error state. To compute (1) we need the state estimate, while (2) only needs state error. Therefore, one could use *different representations* for each of these. Specifically, we could represent the state of a robot at a certain time as

$$x(t) = \hat{x}(t) + \delta x(t) \tag{3.77}$$

where $\hat{x}(t)$ is the state estimate and $\delta x(t)$ is the error in this estimate. The state estimate could be represented in discrete-time, where one pose vector is stored for each time instant when a measurement is available. This has the advantage that it makes few assumptions about the robot motion and can thus model any complex trajectory. The state error is represented in continous-time, either using a parametric or non-parametric model, and the resulting information matrix will have the same dimension as the state error vector and can be explicitly controlled by the user. Since the error state are generally much smoother than the trajectory, one could choose a low-dimensional representation for the continous-time error model without worrying about underfitting.

This is empirically demonstrated in Figure 3.4, which shows the trajectory of a visual-inertial odometry within a one-second long window. Notice that while the ac-

tual trajectory is quite complex (blue line), the estimation error (black dotted line) are much smoother and can be effectively described by a low-dimensional model. This is very useful in practice as it allows us to choose the representation power of the continous error model without having prior knowledge of the robot motion. For example, if we wish to directly model the full trajectory using a B-spline, we would need many more knots to faithfully represent a complex motion, compared to that needed for the trajectory error.

3.3.2 Problem Formulation

The trajectory estimate is defined in discrete time by $\hat{x} := \hat{x}_{1:N}$ and the trajectory error δx is defined in continuous time

$$\delta x \sim \mathcal{GP}(\delta\check{x}(t), K(t,t')) \tag{3.78}$$
$$\delta\theta \sim \mathcal{GP}(\delta\check{\theta}(t), P(t,t')) \tag{3.79}$$

Since we've already shown before that the parametric and non-parametric approaches are equivalent, we model the error state as a GP for simplicity. The robot pose at a particular measurement time t_i is the sum of the estimate and the error

$$x_i := x(t_i) = \hat{x}_i + \delta x(t_i) \tag{3.80}$$
$$\theta_i := \theta(t_i) = \hat{\theta}_i + \delta\theta(t_i) \tag{3.81}$$

where $\hat{x}_i$ is the trajectory estimate at time t_i. This estimate can be obtained by integrating an IMU. Following the non-parametric continuous SLAM derivation, we define the subset state vector r and its error vector $\delta\gamma$

$$\gamma = \begin{bmatrix} y \\ m \end{bmatrix}, \quad \delta\gamma = \begin{bmatrix} \delta y \\ 0 \end{bmatrix} \tag{3.82}$$

with the following relationship similar to equation (3.64)

$$\delta\theta = \delta\check{\theta} + P_{xy}P_{yy}^{-1}(\delta\gamma - \delta\check{\gamma}) = \delta\check{\theta} + \Pi(\delta\gamma - \delta\check{\gamma}) \tag{3.83}$$
$$\Delta\delta\theta = P_{xy}P_{yy}^{-1}\Delta\delta\gamma = \Pi\Delta\delta\gamma \tag{3.84}$$

3.3.3 MAP Estimation

The MAP estimator can be derived from the discrete-time one

$$\theta^* = \arg\min_{\theta} \frac{1}{2} \left\{ \|\theta - \check{\theta}\|_P^2 + \|h(\theta) - z\|_R^2 \right\} \tag{3.85}$$

$$\delta\theta^* = \arg\min_{\delta\theta} \frac{1}{2} \left\{ \|\hat{\theta} - \check{\theta} + \delta\theta\|_P^2 + \|h(\hat{\theta} + \delta\theta) - z\|_R^2 \right\} \tag{3.86}$$

$$\delta\gamma^* = \arg\min_{\delta\gamma} \frac{1}{2} \|\hat{\theta} - \check{\theta} + \delta\check{\theta} + \Pi(\delta\gamma - \delta\check{\gamma})\|_P^2 \tag{3.87}$$

$$+ \frac{1}{2} \|h(\hat{\theta} + \delta\check{\theta} + \Pi(\delta\gamma - \delta\check{\gamma})) - z\|_R^2 \tag{3.88}$$

Linearizing at $\delta\bar{\gamma}$ and setting derivative to 0 we have

$$(\Pi^\top P^{-1} \Pi + \Pi^\top H^\top R^{-1} H \Pi) \Delta\delta\gamma^* \tag{3.89}$$

$$= -(\Pi^\top P^{-1} (\hat{\theta} - \check{\theta} + \delta\bar{\theta}) + \Pi^\top H^\top R^{-1} (\bar{h} - z)) \tag{3.90}$$

$$= -(\Pi^\top P^{-1} (\bar{\theta} - \check{\theta}) + \Pi^\top H^\top R^{-1} (\bar{h} - z)) \tag{3.91}$$

The linearized system is again very similar to the GP formulation, with the only difference that the linearization points at which the residual terms are computed are the sum of the trajectory estimate and trajectory error.

3.3.4 Discussion

The hybrid formulation has a clear advantage over both the discrete and continuous time ones. Compared to a discrete-time representation, it has reduced computation complexity. Compared to a continous-time representation, it can model arbitrarily complex trajectories. Also, by modeling trajectory error, which is typically small, we can handle 3D motions by assuming the error lies in the tangent space of the Lie Group, which is just a vector space and is easy to work with. The only requirement for the hybrid approach is to have a good trajectory estimate in the first place, this is not hard to achieve given the prevalence of IMU sensors.

4 | Conclusion

In this report, we examined several continuous-time trajectory formulations that aim to reduce the computation complexity of SLAM problems under certain situations.

We started by revisiting the traditional discrete-time SLAM formulation. In SLAM, what we are interested in is the posterior probability of the states given some measurements. A SLAM problem can be naturally described by a factor graph, which allows us to perform inference on it. However, in practice it is intractable to recover the full posterior probability. Thus the next best thing we can do is MAP inference, which gives us the states that maximize the posterior. After defining suitable models and their corresponding probability densities (usually Gaussian), MAP estimation can be converted to a nonlinear least squares problem. If the factors are differentiable functions (which they usually are), we can quickly find local minima of the objective function via gradient-based methods. This is done by repeatedly linearize the objective function at the current best estimate and solve for an optimal increment from a set of linear equations. Due to the sparse connections in the factor graph, the information matrix of the linearized system is also (block) sparse. We can then leverage this special structure to solve the linear system efficiently via variable re-ordering and sparse matrix decomposition. We conclude that the two major factors that affect the computation complexity of a SLAM problem is the size of the state vector and the sparsity of the information matrix.

Directly applying the discrete-time formulation to situations like high-rate sensors or life-long operations will cause the state size to grow too big for real-time operations. Continuous-time representations provide a solution to this problem by parameterizing the robot trajectory by time. It allows us to query the robot pose at any given time without adding it to the state vector. A straightforward parametric formulation represents the trajectory by the weighted sum of a set of temporal basis fucntions. This replaces all the pose variables in the state vector by the coefficients of the basis functions, which can be much smaller in size. Alternatively, we can also derive a non-parametric representation by modeling the robot trajectory as a Gaussian process. Additional poses at other time instances can be obtained by GP interpolation. We showed that

these two formulations are two different viewpoints of the same underlying idea and are thus mathematically equivalent. Both the parametric and non-parametric formulations suffer from the problem of model selection, where a good prior knowledge of the robot motion is required to pick a suitable model complexity. A hybrid approach mitigates this issue by decouple the trajectory estimate and error, and only model the error trajectory in a continuous-time fashion.

Finally, it is worth pointing out that although adopting a continuous-time representation has the advantage of reducing the state size, a naive treatment would also destroy the sparsity of the problem. Many works have been done to ensure the information matrix stays sparse under this circumstance by either using B-spline basis functions or carefully constructing the covariance matrix.

Bibliography

[1] H. F. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part i," *IEEE Robotics & Automation Magazine*, vol. 13, pp. 99–110, 2006.

[2] G. Sibley, "A sliding window filter for slam," in *Technical Report*, 2006.

[3] P. Furgale, C. Tong, T. Barfoot, and G. Sibley, "Continuous-time batch trajectory estimation using temporal basis functions," *The International Journal of Robotics Research*, vol. 34, pp. 1688 – 1710, 2015.

[4] C. Tong, P. Furgale, and T. Barfoot, "Gaussian process gauss–newton for non-parametric simultaneous localization and mapping," *The International Journal of Robotics Research*, vol. 32, pp. 507 – 525, 2013.

[5] X. Zheng, M. Li, and A. I. Mourikis, "Decoupled representation of the error and trajectory estimates for efficient pose estimation," in *Robotics: Science and Systems*, 2015.

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

[7] F. Dellaert and M. Kaess, "Factor graphs for robot perception," *Found. Trends Robotics*, vol. 6, pp. 1–139, 2017.

[8] S. Anderson, T. Barfoot, C. Tong, and S. Särkkä, "Batch nonlinear continuous-time trajectory estimation as exactly sparse gaussian process regression," *Autonomous Robots*, vol. 39, pp. 221–238, 2015.

[9] J. Dong, M. Mukadam, B. Boots, and F. Dellaert, "Sparse gaussian processes on matrix lie groups: A unified framework for optimizing continuous-time trajectories," *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6497–6504, 2018.

[10] M. Bloesch, J. Czarnowski, R. Clark, S. Leutenegger, and A. Davison, "Codeslam - learning a compact, optimisable representation for dense visual slam," *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2560–2568, 2018.

[11] Y. Li, L. Ma, Z. Zhong, F. Liu, D. Cao, J. Li, and M. Chapman, "Deep learning for lidar point clouds in autonomous driving: A review," *IEEE transactions on neural networks and learning systems*, vol. PP, 2020.

[12] T. Zhou, M. Brown, N. Snavely, and D. Lowe, "Unsupervised learning of depth and ego-motion from video," *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6612–6619, 2017.

[13] K. M. Jatavallabhula, S. Saryazdi, G. Iyer, and L. Paull, "gradslam: Automagically differentiable slam." *arXiv: Robotics*, 2020.

[14] A. Davison, "Futuremapping: The computational structure of spatial ai systems," *ArXiv*, vol. abs/1803.11288, 2018.

[15] K. Hartnett. (2021) Matrix multiplication inches closer to mythic goal. [Online]. Available: https://www.quantamagazine.org/mathematicians-inch-closer-to-matrix-multiplication-goal-20210323/

[16] J. Alman and V. V. Williams, "A refined laser method and faster matrix multiplication," in *SODA*, 2021.

[17] P. T. Furgale, T. D. Barfoot, and G. Sibley, "Continuous-time batch estimation using temporal basis functions," *2012 IEEE International Conference on Robotics and Automation*, pp. 2088–2095, 2012.

[18] C. Forster, L. Carlone, F. Dellaert, and D. Scaramuzza, "On-manifold preintegration for real-time visual–inertial odometry," *IEEE Transactions on Robotics*, vol. 33, pp. 1–21, 2017.

[19] J. Zhang and S. Singh, "Loam: Lidar odometry and mapping in real-time," in *Robotics: Science and Systems*, 2014.

[20] K. Eckenhoff, L. Paull, and G. Huang, "Decoupled, consistent node removal and edge sparsification for graph-based slam," *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3275–3282, 2016.

[21] R. Kümmerle, "State estimation and optimization for mobile robot navigation," Ph.D. dissertation, Albert-Ludwigs-University of Freiburg, Department of Computer Science, 2013.

[22] D. Koller and N. Friedman, "Probabilistic graphical models - principles and techniques." MIT Press, 2009.

[23] D. Heckerman, "A tutorial on learning with bayesian networks," in *Innovations in Bayesian Networks*, 2020.

[24] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.

[25] F. R. Kschischang, B. J. Frey, and H.-A. Loeliger, "Factor graphs and the sum-product algorithm," *IEEE TRANSACTIONS ON INFORMATION THEORY*, vol. 47, pp. 498–519, 1998.

[26] T. D. Barfoot, *State Estimation for Robotics*, 1st ed. USA: Cambridge University Press, 2017.

[27] D. Marquardt, "An algorithm for least-squares estimation of nonlinear parameters," *Journal of The Society for Industrial and Applied Mathematics*, vol. 11, pp. 431–441, 1963.

[28] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. Cambridge, Mass.: MIT Press, 2005.

[29] B. Triggs, P. McLauchlan, R. Hartley, and A. Fitzgibbon, "Bundle adjustment - a modern synthesis," in *Workshop on Vision Algorithms*, 1999.

[30] C. Engels, H. Stewénius, and D. Nistér, "Bundle adjustment rules," 2006.

[31] C. Sommer, V. Usenko, D. Schubert, N. Demmel, and D. Cremers, "Efficient derivative computation for cumulative b-splines on lie groups," *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 11 145–11 153, 2020.

[32] C. Rasmussen and C. K. Williams, "Gaussian processes for machine learning," in *Adaptive computation and machine learning*, 2009.

[33] S. Lovegrove, A. Patron-Perez, and G. Sibley, "Spline fusion: A continuous-time representation for visual-inertial fusion with application to rolling shutter cameras," in *BMVC*, 2013.