Notes on Least-Squares and SLAM
DRAFT

Giorgio Grisetti

November 1, 2015

Contents

Part I

Least Squares and SLAM

Chapter 1

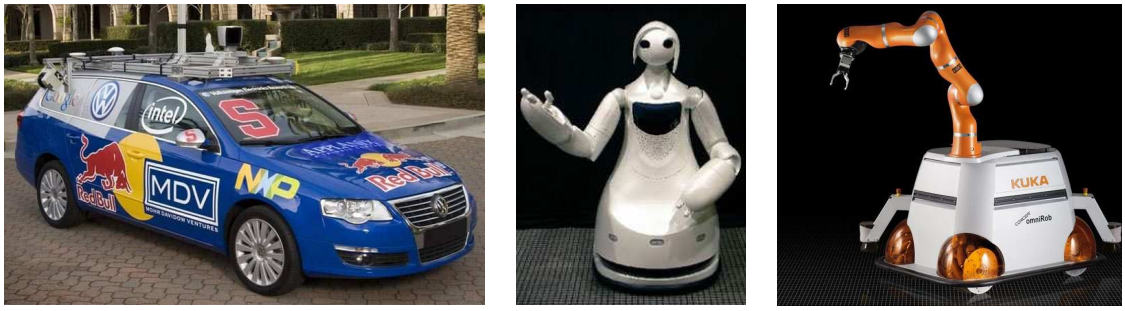
Simultaneous Localization and Mapping

To efficiently solve many tasks envisioned to be carried out by mobile robots including transportation, search and rescue, or automated vacuum cleaning robots need a map of the environment. The availability of an accurate map allows for the design of systems that can operate in complex environments only based on their on-board sensors and without relying on external reference system like, e.g., GPS. The acquisition of maps of indoor environments, where typically no GPS is available, has been a major research focus in the robotics community over the last decades. Learning maps under pose uncertainty is often referred to as the simultaneous localization and mapping (SLAM) problem. In the literature, a large variety of solutions to this problem is available. These approaches can be classified either as filtering or smoothing. Filtering approaches model the problem as an on-line state estimation where the state of the system consists in the *current* robot position and the map. The estimate is augmented and refined by incorporating the new measurements as they become available. Popular techniques like Kalman and information filters [24, 2], particle filters [20, 9, 8], or information filters [5, 25] fall into this category. To highlight their incremental nature, the filtering approaches are usually referred to as on-line SLAM methods. Conversely, smoothing approaches estimate the full trajectory of the robot from the full set of measurements [19, 3, 21]. These approaches address the so-called full SLAM problem, and they typically rely on least-square error minimization techniques.

Figure 1.1 shows three examples of real robotic systems that use SLAM technology: an autonomous car, a tour-guide robot, and an industrial mobile manipulation robot. Image (a) shows the autonomous car Junior as well as a model of a parking garage that has been mapped with that car. Thanks to the acquired model, the car is able to park itself autonomously at user selected locations in the garage. Image (b) shows the TPR-Robina robot developed by Toyota which is also used in the context of guided tours in museums. This robot uses SLAM technology to update its map whenever the environment has been changed. Robot manufacturers such as KUKA, recently presented mobile manipulators as shown in Image (c). Here, SLAM technology is needed to operate such devices in flexible way in changing industrial environments. Figure 1.2 illustrates 2D and 3D maps that can be estimated by the SLAM algorithm discussed in this paper.

An intuitive way to address the SLAM problem is via its so-called graph-based formulation. Solving a graph-based SLAM problem involves to construct a graph whose nodes represent robot poses or landmarks and in which an edge between two nodes encodes a sensor measurement that constrains the connected poses. Obviously, such constraints can be contradictory since observations are always affected by noise. Once such a graph is constructed, the crucial problem is to find a configuration of the nodes that is maximally consistent with the measurements. This involves solving a large error minimization problem.

The graph-based formulation of the SLAM problem has been proposed by Lu and Milios in 1997 [19]. However, it took several years to make this formulation popular due to the comparably high complexity of solving the error minimization problem using standard techniques. Recent insights into the structure of the SLAM problem and advancements in the fields of sparse linear algebra resulted in efficient approaches to the optimization problem at hand. Consequently, graph-based SLAM methods have undergone a



(a) (b) (c)

Figure 1.1: Applications of SLAM technology. (a) An autonomous instrumented car developed at Stanford. This car can acquire maps by utilizing only its on-board sensors. These maps can be subsequently used for autonomous navigation. (b) The museum guide robot TPR-Robina developed by Toyota (picture courtesy of Toyota Motor Company). This robot acquires a new map every time the museum is reconfigured. (c) The KUKA Concept robot "Omnirob", a mobile manipulator designed autonomously navigate and operate in the environment with the sole use of its on-board sensors (picture courtesy of KUKA Roboter GmbH).

renaissance and currently belong to the state-of-the-art techniques with respect to speed and accuracy. The aim of this tutorial is to introduce the SLAM problem in its probabilistic form and to guide the reader to the synthesis of an effective and state-of-the-art graph-based SLAM method. To understand this tutorial a good knowledge of linear algebra, multivariate minimization, and probability theory are required.

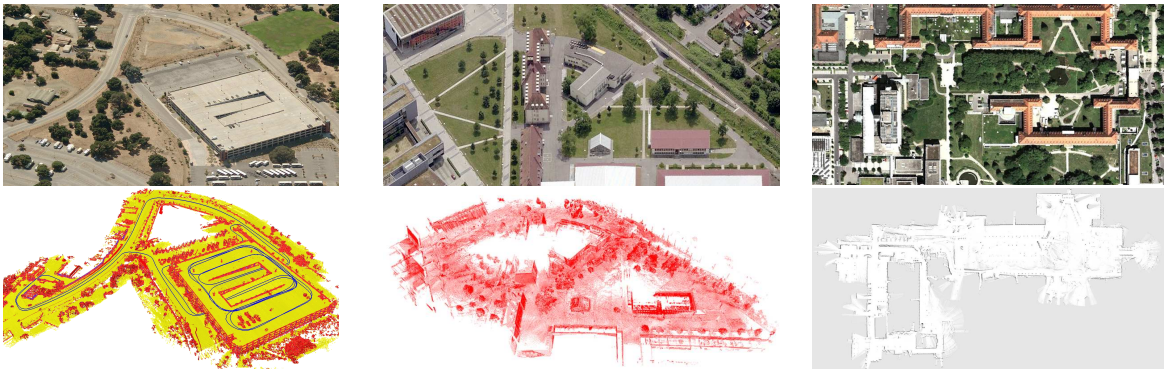
1.1 Probabilistic Formulation of SLAM

Solving the SLAM problem consists of estimating the robot trajectory and the map of the environment as the robot moves in it. Due to the inherent noise in the sensor measurements, a SLAM problem is usually described by means of probabilistic tools. The robot is assumed to move in an unknown environment, along a trajectory described by the sequence of random variables $\mathbf{x}_{1:T} = \{\mathbf{x}_1, \ldots, \mathbf{x}_T\}$. While moving, it acquires a sequence of odometry measurements $\mathbf{u}_{1:T} = \{\mathbf{u}_1, \ldots, \mathbf{u}_T\}$ and perceptions of the environment $\mathbf{z}_{1:T} = \{\mathbf{z}_1, \ldots, \mathbf{z}_T\}$. Solving the full SLAM problem consists of estimating the posterior probability of the robot's trajectory $\mathbf{x}_{1:T}$ and the map $\mathbf{m}$ of the environment given all the measurements plus an initial position $\mathbf{x}_0$:

$$p(\mathbf{x}_{1:T}, \mathbf{m} \mid \mathbf{z}_{1:T}, \mathbf{u}_{1:T}, \mathbf{x}_0). \tag{1.1}$$

The initial position $\mathbf{x}_0$ defines the position of the map and can be chosen arbitrarily. For convenience of notation, in the remainder of this document we will omit $\mathbf{x}_0$. The poses $\mathbf{x}_{1:T}$ and the odometry $\mathbf{u}_{1:T}$ are usually represented as 2D or 3D transformations in $SE(2)$ or in $SE(3)$, while the map can be represented in different ways. Maps can be parametrized as a set of spatially located landmarks, by dense representations like occupancy grids, surface maps, or by raw sensor measurements. The choice of a particular map representation depends on the sensors used, on the characteristics of the environment, and on the estimation algorithm. Landmark maps [24, 20] are often preferred in environments where locally distinguishable features can be identified and especially when cameras are used. In contrast, dense representations [26, 9, 8] are usually used in conjunction with range sensors. Independently of the type of the representation, the map is defined by the measurements and the locations where these measurements have been acquired [14, 15]. Figure 1.2 illustrates three typical dense map representations for 3D and 2D: multilevel surface maps, point clouds and occupancy grids. Figure 1.3 shows a typical 2D landmark based map.

Estimating the posterior given in (1.1) involves operating in high dimensional state spaces. This would not be tractable if the SLAM problem would not have a well defined structure. This structure arises from certain and commonly done assumptions, namely the static world assumption and the Markov assumption. A convenient way to describe this structure is via the dynamic Bayesian network (DBN) depicted in Figure 1.4. A Bayesian network is a graphical model that describes a stochastic process as



(a) (b) (c)

Figure 1.2: (a) A 3D map of the Stanford parking garage acquired with an instrumented car (bottom), and the corresponding satellite view (top). This map has been subsequently used to realize an autonomous parking behavior. (b) Point cloud map acquired at the university of Freiburg (courtesy of Kai. M. Wurm) and relative satellite image. (c) Occupancy grid map acquired at the hospital of Freiburg. Top: a bird's eye view of the area, bottom: the occupancy grid representation. The gray areas represent unobserved regions, the white part represents traversable space while the black points indicate occupied regions.

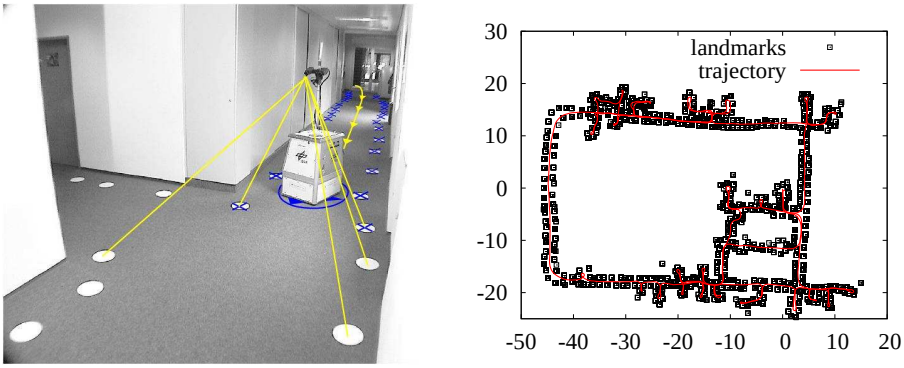


Figure 1.3: Landmark based maps acquired at the German Aerospace Center. In this setup the landmarks consist in white circles painted on the ground that are detected by the robot through vision, as shown in the left image. The right image illustrates the trajectory of the robot and the estimated positions of the landmarks. These images are courtesy of Udo Frese and Christoph Hertzberg.

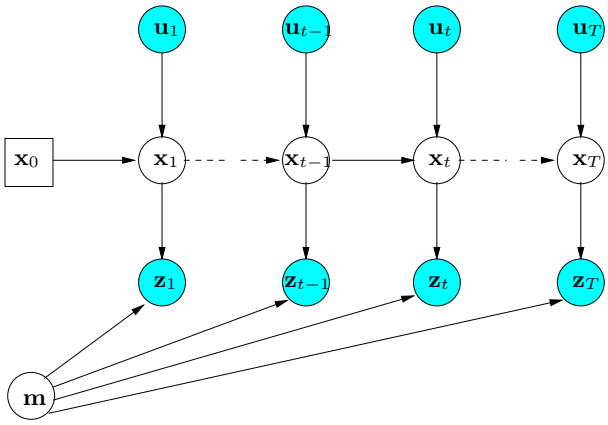


Figure 1.4: Dynamic Bayesian Network of the SLAM process.

a directed graph. The graph has one node for each random variable in the process, and a directed edge (or arrow) between two nodes models a conditional dependence between them.

In Figure 1.4, one can distinguish blue/gray nodes indicating the observed variables (here $\mathbf{z}_{1:T}$ and $\mathbf{u}_{1:T}$) and white nodes which are the hidden variables. The hidden variables $\mathbf{x}_{1:T}$ and $\mathbf{m}$ model the robot's trajectory and the map of the environment. The connectivity of the DBN follows a recurrent pattern characterized by the state transition model and by the observation model. The transition model $p(\mathbf{x}_t \mid \mathbf{x}_{t-1}, \mathbf{u}_t)$ is represented by the two edges leading to $\mathbf{x}_t$ and represents the probability that the robot at time t is in $\mathbf{x}_t$ given that at time $t-1$ it was in $\mathbf{x}_t$ and it acquired an odometry measurement $\mathbf{u}_t$.

The observation model $p(\mathbf{z}_t \mid \mathbf{x}_t, \mathbf{m}_t)$ models the probability of performing the observation $\mathbf{z}_t$ given that the robot is at location $\mathbf{x}_t$ in the map. It is represented by the arrows entering in $\mathbf{z}_t$. The exteroceptive observation $\mathbf{z}_t$ depends only on the current location $\mathbf{x}_t$ of the robot and on the (static) map $\mathbf{m}$. Expressing SLAM as a DBN highlights its temporal structure, and therefore this formalism is well suited to describe filtering processes that can be used to tackle the SLAM problem.

An alternative representation to the DBN is via the so-called "graph-based" or "network-based" formulation of the SLAM problem, that highlights the underlying spatial structure. In graph-based SLAM, the poses of the robot are modeled by nodes in a graph and labeled with their position in the environment [19, 15]. Spatial constraints between poses that result from observations z_t or from odometry measurements $\mathbf{u}_t$ are encoded in the edges between the nodes. More in detail, a graph-based SLAM algorithm constructs a graph out of the raw sensor measurements. Each node in the graph represents a robot position and a measurement acquired at that position. An edge between two nodes represents a spatial constraint relating the two robot poses. A constraint consists in a probability distribution over the relative transformations between the two poses. These transformations are either odometry measurements between sequential robot positions or are determined by aligning the observations acquired at the two robot locations. Once the graph is constructed one seeks to find the configuration of the robot poses that best satisfies the constraints. Thus, in graph-based SLAM the problem is decoupled in two tasks: constructing the graph from the raw measurements (graph construction), determining the most likely configuration of the poses given the edges of the graph (graph optimization). The graph construction is usually called front-end and it is heavily sensor dependent, while the second part is called back-end and relies on an abstract representation of the data which is sensor agnostic. Figure 1.5 depicts an uncorrected pose-graph and the corresponding corrected one.

1.2 Document Structure

The purpose of this document is to give you an understanding of SLAM. However you may have realized that SLAM is a problem that can be solved in many ways. I will discuss in this block of lectures one possible solution to these problems, which is now regarded as the most promising. To this end we will make extensive use of least squares optimization, and of out knowledge about the Gaussian distributions.

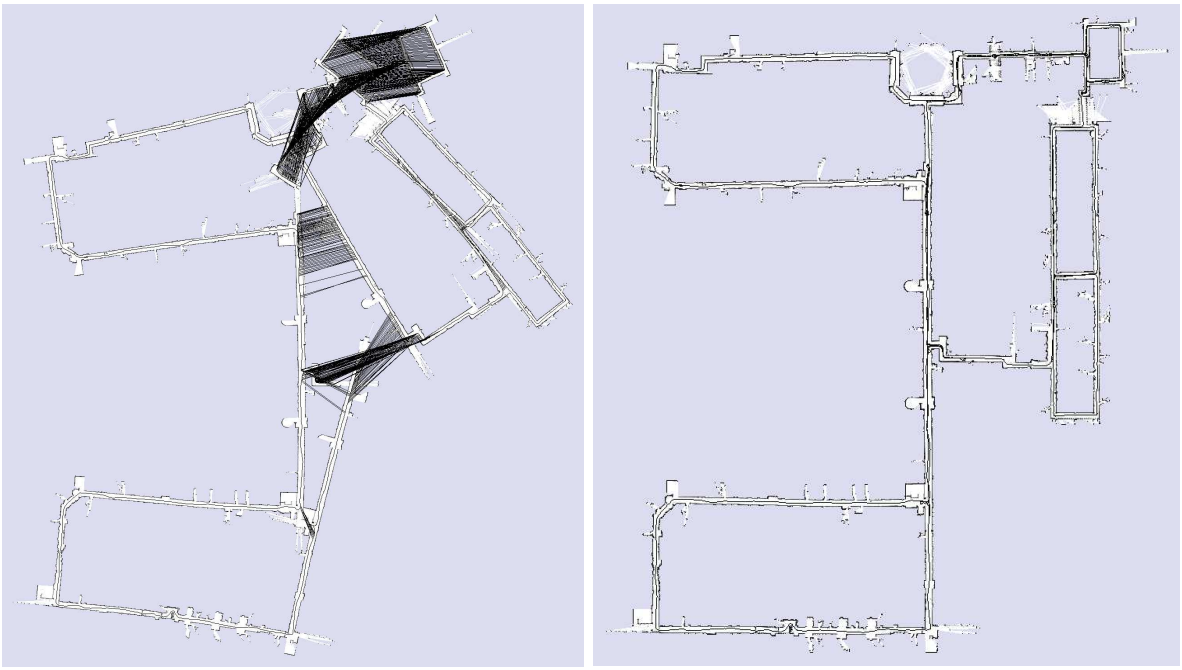


Figure 1.5: Pose-graph corresponding to a data-set recorded at MIT Killian Court (courtesy of Mike Bosse and John Leonard) (left) and after (right) optimization. The maps are obtained by rendering the laser scans according to the robot positions in the graph.

In the remainder of this document you will learn a methodology for approaching problems with least squares. Naive implementations of least squares, however are problematic in circumstances where the state space is not a vector space. This for instance happens when our robot rotates, and we want to estimate its orientation. In these cases, however if the state space that is locally smoother (more formally it is a smooth manifold), we can still deal with it. You will learn these wonders in Chapter 2.

When you become robust on dealing with least squares problems involving a small state space, and you try to use the very same approach on large problems you will be disappointed by its slowness. But you will be enlightened by looking at the structure of the problem and you will discover that you can save a lot of computation. You will reach this wisdom after reading Chapter 3.

After you master least squares, you will be telling yourself: "Yes, i know how to solve a least squares problem, but how do i make one out of the raw data of the robot?". This requires solving the so called Data Association problem, that means distinguishing which variable in your state your system is currently observing. Imagine you have a million needles in a pot, each one with its position. Your ca observe the pot, and thus determine the position of each needle. But how to associate the portion of image capturing a needle to the right variable in the state? In SLAM things don't look that bad. But still you will have to figure out which needle is which. I will disclose you these secrets in Chapter 4.

Chapter 2

State Estimation of a Stationary Non-Linear System

2.1 Problem Definition

Let $\mathcal{W}$ be a stationaty system, whose state is parameterized by a set of state variables $\mathbf{x} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ these variables can span over arbitrary spaces. We observe indirectly the state of the system by a set of sensors and from which we obtain measurements $\mathbf{z} = \{\mathbf{z}_1, \ldots, \mathbf{z}_K\}$. Here $\mathbf{z}_k$ is the k^{th} measurement. This situation is illustrated in Fig. 2.1. The measurements are affected by noise, so in fact $\mathbf{z}_k$ are random variables. We assume that the state models all the knowledge required to predict the distribution of a measurement. Being the measurements affected by noise, it is impossible to estimate *the state* of the system, given the measurements. Instead, we can compute a distribution over the potential states of the system given these measurements. More formally, we want to estimate the probability distribution of the state $\mathbf{x}$, given the measurements $\mathbf{z}$:

$$p(\mathbf{x} \mid \mathbf{z}) = p(\mathbf{x}_1, \ldots, \mathbf{x}_N \mid \mathbf{z}_1, \ldots, \mathbf{z}_K) \tag{2.1}$$

$$= p(\mathbf{x}_{1:N} | \mathbf{z}_{1:K}). \tag{2.2}$$

In Eq. 2.2 $\mathbf{x}_{1:N}$ just indicates all state variables with indices from 1 to N and similarly $\mathbf{z}_{1:K}$ indicates all measurements between 1 and K.

Unfortunately, there are in general no closed forms to compute Eq 2.2, for many reasons. The most important are listed below:

- the measurements $\mathbf{z}_k$ typically allow to observe only a subset of the state variables. Additionally, the mapping between the measurements and the states is in general non-linear. Thus, the probablity distribution can be multi modal and have complex shapes

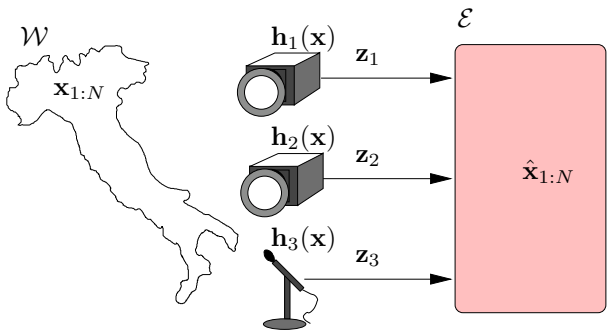


Figure 2.1: Illustration of a state estimation on a stationary system. $\mathcal{W}$ is the phenomena being observed. The state of this phenomena is indirectly measured through a set of sensors each of them measuring a quantity $\mathbf{z}_k$ which depends on the system's state.

- we can have an arbitrary number of measurements. Sometimes not enough to fully characterize the distribution over states. Some other time these measurements could not be explained by any particular state due to noise, thus we have an overdetermined problem
- some of the measurements might be wrong, for instance because we cannot correctly identyfy which state variable generates the a specific observation.

However, we notice that there is another distribution that is in general easy to obtain. This is the conditional distribution of a measurement given the state: $p(\mathbf{z}_k|\mathbf{x})$ and is commonly known as *sensor model* or *observation model*. This is a predictive distribution, that tells what is the probability that we do a certain measurement $\mathbf{z}_k$, *assuming* the state $\mathbf{x}$ to be known. Another hint comes from the fact that the measurements are independent, *given* the state. Put in other words this means that if I know the stationary state, taking a measurement $\mathbf{z}_a$ does not tell me anything on another measurement $\mathbf{z}_b$ because I know the state, and it captures all the relevant information necessary to predict $\mathbf{z}_a$. We can put this prose in probabilistic form by the following:

$$p(\mathbf{z}_{1:K}|\mathbf{x}_{1:N}) = \prod_{k=1}^{K} p(\mathbf{z}_k|\mathbf{x}_{1:N}) \tag{2.3}$$

The term $p(\mathbf{z}_{1:K}|\mathbf{x}_{1:N})$ is known as *likelihood* of the measurements given the states. It is a conditional distribution that changes its shape when changing the states.

Our initial goal was however to compute the distribution of the states, given the measurements. We know that the likelihood can be easily computed. Can we use this fact to simplify our life? The answer is yes (otherwise you would probably not be reading this document). With the bayes rule we can manipulate Eq. 2.2 to highlight the contribution of the likelihood.

$$\begin{aligned} p(\mathbf{x}_{1:N}|\mathbf{z}_{1:K}) &= \frac{\overbrace{p(\mathbf{z}_{1:K}|\mathbf{x}_{1:N})}^{\text{likelihood}} \cdot \overbrace{p(\mathbf{x}_{1:N})}^{\text{prior}}}{\underbrace{p(\mathbf{z}_{1:K})}_{\text{normalizer}}} && (2.4)\\ &= \frac{p(\mathbf{z}_{1:K}|\mathbf{x}_{1:N})p_x}{p_z} && (2.5)\\ &= \eta p_x p(\mathbf{z}_{1:K}|\mathbf{x}_{1:N}) && (2.6)\\ &\propto \prod_{k=1}^{K} p(\mathbf{z}_k|\mathbf{x}_{1:N}) && (2.7) \end{aligned}$$

In Eq. 2.4 the prior $p(\mathbf{x}_{1:N})$ models the potential knowledge we have about the states before we do the measurements. If we know nothing, the prior is a uniform distribution whose value is a constant p_x. and the normalizer $p(\mathbf{z}_{1:K})$ does not depend on the states. We know the measurements, and they do not change, so $p(\mathbf{z}_{1:K})$ is just a constant number p_z (see Eq. 2.5). To drop the fraction, let $\eta = 1/p_z$ (Eq. 2.6). Eq. 2.7 is obtained from the previous by killing the constants and finally utilizing the independence assumption.

Example: Point Registration Imagine the following phenomena: We have a robot moving in a 2D world populated by unique 2D points. Unique means that, for instance each point has a unique color, so that if we observe a point from different viewpoints, we can always determine to which point in the world the observation refers. Our robot is equipped with a powerful 2D sensor capable to determine the color and the position of a point relative to the robot. Let $\mathbf{x} = (t_x\ t_y\ \theta)$ be the position of the robot w.r.t, the origin of our world. Let $\mathbf{p}_i = (\mathbf{x}_i\ \mathbf{y}_i\ 1)$ be the position of one of the points w.r.t. the origin, in homogeneous coordinates. Our problem is to estimate the position of the robot $\mathbf{x}$, given a set of measurements of the surrounding points. We assume to know the positions of these points in the global frame. Put in other words we want to estimate $p(\mathbf{x}|\mathbf{z}_{1:N})$, where $\mathbf{z}_i$ refers to a measurement of one of the points.

To this end we just need to construct a sensor model, that is to construct the function describing the measurement density, given the state: $p(\mathbf{z}_i|\mathbf{x})$. If the robot is located at a certain position, we could

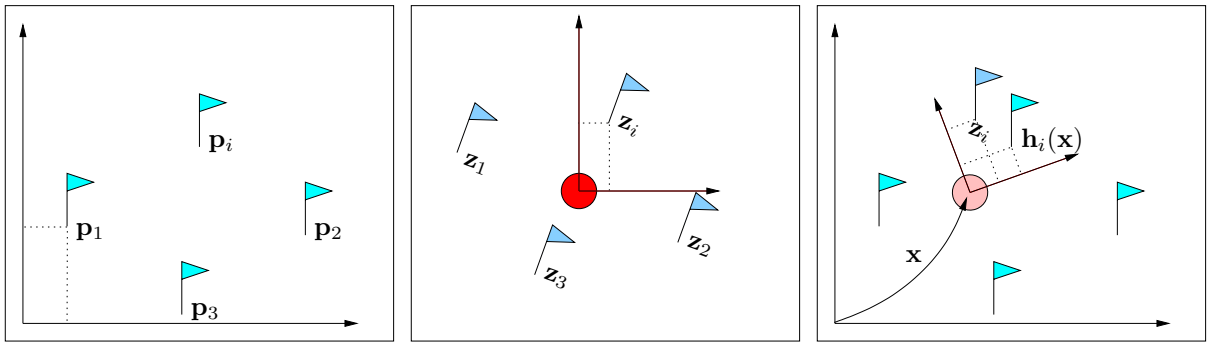


Figure 2.2: Illustration of the scenario of a robot moving in a 2D world and sensing 2D point features. Left: the world is populated by a set of landmarks $\mathbf{p}_i$. Middle: the robot senses a the landmarks in its own reference frame, gathering the observations $\mathbf{z}_i$. Right: If the robot would know its position in the world $\mathbf{x}$, it could predict the position at which he senses the the i^{th} landmark, through the prediction function $\mathbf{h}_i\mathbf{x}$. A good solution is one that pulls the robot pose so that each prediction is close to the actual measurement.

predict the measurement the robot would gather about a certain point. This is the so called observation model

$$\mathbf{h}_i(\mathbf{x}) = \mathbf{T}(\mathbf{x})^{-1}\mathbf{p}_i \tag{2.8}$$

that is the position of the point w.r.t. the robot reference frame. Here i denoted with $\mathbf{T}(x)$ the transformation matrix corresponding to the three dimensional vector x as follows:

$$\mathbf{T}(\mathbf{x}) = \begin{pmatrix} \mathbf{R}_\theta & \mathbf{t} \\ \mathbf{0} & 1 \end{pmatrix} \qquad \mathbf{R}_\theta = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \qquad \mathbf{t} = \begin{pmatrix} t_x \\ t_y \end{pmatrix} \tag{2.9}$$

If the measurements are affected by a zero mean addittive Gaussian noise characterized by an information matrix $\mathbf{\Omega}_i$, we could estimate the distribution of the measurement $\mathbf{z}_i$ of the i^{th} point, given a state $\mathbf{x}$. $p(\mathbf{z}_i|\mathbf{x}) = \mathcal{N}(\mathbf{z}_i, \mathbf{h}_i(\mathbf{x}), \mathbf{\Sigma}_i)$

2.2 Least Squares Estimation: Basics and Probabilistic Interpretation

The conclusion in the previous section is captured by Eq. 2.7. In short, the distribution over the possible states given the measurements is proportional to the likelihood of the measurement given the states. The latter is easy to compute, just by knowing the observation model, which can be easily constructed once we know how the sensor works. If the noise $\mathbf{n}$ affecting the measurements is zero mean normally distributed, the likelihood of the measurement will also be gaussian.

$$p(\mathbf{z}_k|\mathbf{x}) \propto \exp\left(-(\hat{\mathbf{z}}_k - \mathbf{z}_k)^T \mathbf{\Omega}_k (\hat{\mathbf{z}}_k - \mathbf{z}_k)^T\right) \tag{2.10}$$

Here $\mathbf{\Omega}_k = \mathbf{\Sigma}_k^{-1}$ is the information matrix of the conditional measurement and $\hat{\mathbf{z}}_k$ is the prediction of the measurement, given the state $\mathbf{x}$, and represents the mean of the conditional distribution. Also the likelihood of all the measurements $p(\mathbf{z}|\mathbf{x})$ will be Gaussian, being the product of Gaussians, according to Eq. 2.7.

The predicted measurement is obtained from the state by applying the sensor model $\mathbf{h}_k(\cdot)$:

$$\hat{\mathbf{z}}_k = \mathbf{h}_k(\mathbf{x}) \tag{2.11}$$

In case the sensor model is an affine transformation $\mathbf{h}_k(\mathbf{x}) = \mathbf{A}\mathbf{x} + \mathbf{b}$ and the noise in the measurements is Gaussian, $p(\mathbf{x}|\mathbf{z})$ is normally distributed $\mathcal{N}(\mu_\mathbf{x}, \mathbf{\Sigma}_\mathbf{x})$. If you have some problems with this statement, [23] might be a good read. Unlucky us, the sensor model is in general a non-linear function of the states.

However, if it is smooth enough we can approximate it in the neighborhood of a linearization point $\breve{\mathbf{x}}$ by its Taylor expansion

$$\mathbf{h}_k(\breve{\mathbf{x}} + \boldsymbol{\Delta}\mathbf{x}) \simeq \underbrace{\mathbf{h}_k(\breve{\mathbf{x}})}_{\breve{\mathbf{z}}_k} + \underbrace{\left.\frac{\partial \mathbf{h}_k(\mathbf{x})}{\partial \mathbf{x}}\right|_{\mathbf{x}=\breve{\mathbf{x}}}}_{\mathbf{J}_k} \cdot \boldsymbol{\Delta}\mathbf{x}. \tag{2.12}$$

If we choose as linearization point the state $\mathbf{x}^*$ that better explains the measurements

$$\mathbf{x}^* = \underset{\mathbf{x}}{\operatorname{argmax}}\, p(\mathbf{z} \mid \mathbf{x}), \tag{2.13}$$

and we plug in Eq. 2.12 in Eq.2.10 we can have a new distribution over $P(\mathbf{z}|\boldsymbol{\Delta}\mathbf{x})$ *governed by* the increments $\boldsymbol{\Delta}\mathbf{x}$:

$$p(\mathbf{z}_k|\mathbf{x}^* + \boldsymbol{\Delta}\mathbf{x}) \propto \exp\left[-(\mathbf{h}_k(\mathbf{x}^*) + \mathbf{J}_k\boldsymbol{\Delta}\mathbf{x} - \mathbf{z}_k)^T \boldsymbol{\Omega}_k (\mathbf{h}_k(\mathbf{x}^*) + \mathbf{J}_k\boldsymbol{\Delta}\mathbf{x} - \mathbf{z}_k)\right]. \tag{2.14}$$

Since $\mathbf{x}^*$ is the optimal state given the measurements, we can treat it as a constant. Evaluating Eq 2.14 at different $\boldsymbol{\Delta}\mathbf{x}$ tells us how the likelihood fits the data measurement given a perturbation of the optimal state.

As I said before, the measurement models are in general non-linear, and finding the state $\mathbf{x}^*$ that better explains the measurements might require some work. However, once we have found this state, we can construct a local gaussian approximation of our likelihood. Thus, let us split the problem in two parts:

- find the optimal $\mathbf{x}^* = \operatorname{argmax}_{\mathbf{x}} p(\mathbf{z}|\mathbf{x})$
- determine the parameters of the local gaussian approximation of $p(\mathbf{x}|\mathbf{z}) \simeq \mathcal{N}(\mathbf{x}; \mathbf{x}^*, \boldsymbol{\Sigma}_{\mathbf{x}|\mathbf{z}})$. Note that in computing the gaussian approximation around the optimum, I set directly its mean to be the optimum. This makes sense since the mean is the point where the gaussian is higher.

We will discuss how to carry on these steps in the remainder of this section.

2.2.1 Direct Minimization Methods: Gauss, Newton and their friends

We want to find the $\mathbf{x}^*$ that maximizes Eq. 2.10

$$\begin{aligned}
\mathbf{x}^* &= \underset{\mathbf{x}}{\operatorname{argmax}} \prod_{k=1}^{K} p(\mathbf{z}_k|\mathbf{x}) && (2.15)\\
&\quad \text{[gaussian assumption]} \\
&= \underset{\mathbf{x}}{\operatorname{argmax}} \prod_{k=1}^{K} \exp[-(\mathbf{h}_k(\mathbf{x}) - \mathbf{z}_k)^T \boldsymbol{\Omega}_k (\mathbf{h}_k(\mathbf{x}) - \mathbf{z}_k)] && (2.16)\\
&\quad \text{[taking the logarithm]} && (2.17)\\
&= \underset{\mathbf{x}}{\operatorname{argmax}} \sum_{k=1}^{K} [-(\mathbf{h}_k(\mathbf{x}) - \mathbf{z}_k)^T \boldsymbol{\Omega}_k (\mathbf{h}_k(\mathbf{x}) - \mathbf{z}_k)] && (2.18)\\
&\quad \text{[removing the minus]} \\
&= \underset{\mathbf{x}}{\operatorname{argmin}} \sum_{k=1}^{K} (\mathbf{h}_k(\mathbf{x}) - \mathbf{z}_k)^T \boldsymbol{\Omega}_k (\mathbf{h}_k(\mathbf{x}) - \mathbf{z}_k). && (2.19)
\end{aligned}$$

Might be a matter of personal taste, but I like Eq. 2.19 far more than Eq. 2.15. Let us introduce a new function: the error $\mathbf{e}_k(\mathbf{x})$, which is the difference between the prediction and the observation:

$$\mathbf{e}_k(\mathbf{x}) = \mathbf{h}_k(\mathbf{x}) - \mathbf{z}_k. \tag{2.20}$$

Similarly, we define the argument of the minimization as

$$F(\mathbf{x}) = \sum_{k=1}^{K} \underbrace{\mathbf{e}_k(\mathbf{x})^T \boldsymbol{\Omega}_k \mathbf{e}_k(\mathbf{x})}_{e_k(\mathbf{x})} \tag{2.21}$$

Let us assume that we have a reasonable initial guess of the optimum $\breve{\mathbf{x}}$. By exploiting the Taylor approximation in Eq. 2.12, we can rewrite one of the summands in Eq 2.19 as

$$
\begin{aligned}
e_k(\breve{\mathbf{x}}+\boldsymbol{\Delta}\mathbf{x}) &= (\mathbf{h}_k(\breve{\mathbf{x}}+\boldsymbol{\Delta}\mathbf{x})-\mathbf{z}_k)^T\boldsymbol{\Omega}_k(\mathbf{h}_k(\breve{\mathbf{x}}+\boldsymbol{\Delta}\mathbf{x})-\mathbf{z}_k) && (2.22)\\
&\quad \text{[plugging in Eq 2.12]}\\
&\simeq (\mathbf{J}_k\boldsymbol{\Delta}\mathbf{x}+\mathbf{h}_k(\breve{\mathbf{x}})-\mathbf{z}_k)^T\boldsymbol{\Omega}_k(\mathbf{J}_k\boldsymbol{\Delta}\mathbf{x}+\mathbf{h}_k(\breve{\mathbf{x}})-\mathbf{z}_k) && (2.23)\\
&\quad \text{[substituting } \mathbf{h}_k(\breve{\mathbf{x}})-\mathbf{z}_k=\mathbf{e}_k \text{]}\\
&= (\mathbf{J}_k\boldsymbol{\Delta}\mathbf{x}+\mathbf{e}_k)^T\boldsymbol{\Omega}_k(\mathbf{J}_k\boldsymbol{\Delta}\mathbf{x}+\mathbf{e}_k) && (2.24)\\
&\quad \text{[doing the products and grouping the terms]}\\
&= \boldsymbol{\Delta}\mathbf{x}^T\underbrace{\mathbf{J}_k^T\boldsymbol{\Omega}_k\mathbf{J}_k}_{\mathbf{H}_k}\boldsymbol{\Delta}\mathbf{x}+2\underbrace{\mathbf{e}_k^T\boldsymbol{\Omega}_k\mathbf{J}_k}_{\mathbf{b}_k^T}\boldsymbol{\Delta}\mathbf{x}+\mathbf{e}_k^T\boldsymbol{\Omega}_k\mathbf{e}_k && (2.25)\\
&\quad \text{[using the renamed terms]}\\
&= \boldsymbol{\Delta}\mathbf{x}^T\mathbf{H}_k\boldsymbol{\Delta}\mathbf{x}+2\mathbf{b}_k^T\boldsymbol{\Delta}\mathbf{x}+\mathbf{e}_k^T\boldsymbol{\Omega}_k\mathbf{e}_k. && (2.26)
\end{aligned}
$$

Now we are ready to plug Eq. 2.26 in Eq. 2.21

$$
\begin{aligned}
F(\breve{\mathbf{x}}+\boldsymbol{\Delta}\mathbf{x}) &\simeq \sum_{k=1}^{K}\boldsymbol{\Delta}\mathbf{x}^T\mathbf{H}_k\boldsymbol{\Delta}\mathbf{x}-2\mathbf{b}_k^T\boldsymbol{\Delta}\mathbf{x}+\mathbf{e}_k^T\boldsymbol{\Omega}_k\mathbf{e}_k && (2.27)\\
&\quad \text{[pushing in the summations]}\\
&= \boldsymbol{\Delta}\mathbf{x}^T\underbrace{\left[\sum_{k=1}^{K}\mathbf{H}_k\right]}_{\mathbf{H}}\boldsymbol{\Delta}\mathbf{x}+2\underbrace{\left[\sum_{k=1}^{K}\mathbf{b}_k^T\right]}_{\mathbf{b}^T}\boldsymbol{\Delta}\mathbf{x}+\underbrace{\left[\sum_{k=1}^{K}\mathbf{e}_k^T\boldsymbol{\Omega}_k\mathbf{e}_k\right]}_{c} && (2.28)
\end{aligned}
$$

What we really did in Eq. 2.28 is to write an expression of the objective function, under a linear approximation of the sensor model around a neighborhood of the initial estimate $\breve{\mathbf{x}}$. We found out that if we fix the initial estimate, the value of the function in the neighborhood can be approximated by a quadratic form with respect to the increments $\boldsymbol{\Delta}\mathbf{x}$. Thus, we can minimize this quadratic form and get the increment $\boldsymbol{\Delta}\mathbf{x}^*$ that applied to the current guess gives a better solution:

$$\mathbf{x}* = \breve{\mathbf{x}}+\boldsymbol{\Delta}\mathbf{x}^*. \tag{2.29}$$

The question now is: how do we get the $\boldsymbol{\Delta}\mathbf{x}^*$ that minimizes the quadratic form in Eq. 2.28? As we did in the high school: we set the derivative of the function to zero and solve the corresponding equation. The derivative of Eq. 2.28 is

$$\frac{\partial(\boldsymbol{\Delta}\mathbf{x}^T\mathbf{H}\boldsymbol{\Delta}\mathbf{x}-2\mathbf{b}\boldsymbol{\Delta}\mathbf{x}+c)}{\partial\boldsymbol{\Delta}\mathbf{x}} = 2\mathbf{H}\boldsymbol{\Delta}\mathbf{x}-2\mathbf{b}, \tag{2.30}$$

thus we can find the minimum by solving the linear system

$$\mathbf{H}\boldsymbol{\Delta}\mathbf{x}* = \mathbf{b}. \tag{2.31}$$

If the measurement function would be linear, we would find the minimum in just one go. Since this is typically not the case, we have to iterate the procedure until no substantial improvements are made. The whole procedure is the Gauss-Newton (GN) algorithm and is described in detail in Algorithm 1.

Note that the Gauss-Newton algorithm is not guaranteed to converge in general. The convergence depends on a variety of factors, namely:

- how close our initial guess is to the actual minimum,
- how smooth are the error functions and
- how far is our guess from potential singularities.

Algorithm 1 Gauss-Newton minimization algorithm

Require: $\breve{\mathbf{x}}$: initial guess. $\mathcal{C} = \{\langle \mathbf{z}_k(\cdot), \mathbf{\Omega}_k \rangle\}$: measurements
Ensure: $\mathbf{x}^*$: new solution
  //compute the current error
  $F_{\text{new}} \leftarrow \breve{F}$
  // iterate until no substantial improvements are made
  repeat
    $\breve{F} \leftarrow F_{\text{new}}$
    $\mathbf{b} \leftarrow \mathbf{0} \qquad \mathbf{H} \leftarrow \mathbf{0}$
    for all $k = 1 \dots K$ **do**
      // Compute the prediction $\hat{\mathbf{z}}_k$, the error $\mathbf{e}_k$ and the jacobian $\mathbf{J}_k$
      $\hat{\mathbf{z}}_k \leftarrow \mathbf{h}_k(\breve{\mathbf{x}})$
      $\mathbf{e}_k \leftarrow \hat{\mathbf{z}}_k - \mathbf{z}_k$
      $\mathbf{J}_k \leftarrow \left.\frac{\partial \mathbf{h}_k(\mathbf{x})}{\partial \mathbf{x}}\right|_{\mathbf{x}=\breve{\mathbf{x}}}$
      // compute the contribution of this measurement to the linear system
      $\mathbf{H}_k \leftarrow \mathbf{J}_k^T \mathbf{\Omega}_k \mathbf{J}_k$
      $\mathbf{b}_k \leftarrow \mathbf{J}_k^T \mathbf{\Omega}_k \mathbf{e}_k$
      // accumulate the contribution to construct the overall system
      $\mathbf{H} \mathrel{+}= \mathbf{H}_k$
      $\mathbf{b} \mathrel{+}= \mathbf{b}_k$
    end for
    // solve the linear system
    $\mathbf{\Delta x} \leftarrow \text{solve}(\mathbf{H}\,\mathbf{\Delta x} = -\mathbf{b})$
    // update the state
    $\breve{\mathbf{x}} \mathrel{+}= \mathbf{\Delta x}$
    // compute the new error
    $F_{\text{new}} \leftarrow F(\breve{\mathbf{x}})$
  until $\breve{F} - F_{\text{new}} > \epsilon$
  return $\breve{\mathbf{x}}$

Note that the matrix $\mathbf{H}$ should be possibly full rank, if we do not want to have an under-determined solution.

It might also happen that one iteration of Gauss-Newton results in a new solution that is worse than the initial one. To overcome this problem we can use the Levenberg-Marquardt (LM) algorithm, which is a damped version of Gauss-Newton that introduces backup-restore steps to recover from wrong steps. At every iteration, instead of solving the system $\mathbf{H}\mathbf{\Delta x}^* = \mathbf{b}$, LM solves a *damped* version of the system:

$$(\mathbf{H} + \lambda \mathbf{I})\mathbf{\Delta x}^* = \mathbf{b}. \tag{2.32}$$

Intuitively, if $\lambda \to \inf$, $\mathbf{\Delta x}^* \to 0$. So, the higher is the damping factor λ, the smaller are the increments. An abstract implementation of LM is provided in Algorithm 2.

In general, LM algorithm always converges, but the minimum found might be not the global one. In the remainder this document we will further investigate techniques to make the optimization more robust, however for now you have two powerful tools to find the most likely state of a system, given its measurements. In the next section we will revise how to determine the distribution of states, instead of just the most likely state. Let us now focus on a simple example that can be used to ground this theory.

Example: Point Registration Let us back up to the previous example on point registration. As I promised you, once you know the error function (or the likelihood function $p(\mathbf{z}|\mathbf{x})$ you can approach the problem with least squares.

In our previous example on point registration we have found the likelihood function, from which we can easily determine the error function $\mathbf{e}_i(\mathbf{x}) = \mathbf{h}(\mathbf{x}) - \mathbf{z}_i$:

$$\mathbf{e}_i(\mathbf{x}) = \mathbf{R}_\theta^T \begin{pmatrix} x_i - t_x \\ y_i - t_y \end{pmatrix} - \mathbf{z}_i \tag{2.33}$$

Here we dropped the last row of the homogeneous notation, since it does not carry any information. Thus the measurements will be represented by a two dimensional vector.

To implement Algorithm 1, we only need to compute the jacobian $\mathbf{J}_k = \left.\frac{\partial \mathbf{e}_k(\mathbf{x})}{\partial \mathbf{x}}\right|_{\mathbf{x}=\check{\mathbf{x}}}$. Let us rewrite Eq. 2.33 to separate the contributions of $\mathbf{p}_i = (x_i\ y_i)$ and $\mathbf{t} = (\mathbf{t}_x\ \mathbf{t}_y)$, as follows:

$$\mathbf{e}_i(\mathbf{x}) = \mathbf{R}_\theta^T \mathbf{p}_i - \mathbf{R}_\theta^T \mathbf{t} - \mathbf{z}_i. \tag{2.34}$$

In this straightforward case the Jacobian w.r.t. the first two components of the state vector t_x and t_y is just:

$$\frac{\partial \mathbf{e}_i(\mathbf{x})}{\partial \mathbf{t}} = -\mathbf{R}_\theta^T. \tag{2.35}$$

The jacobian w.r.t the angular component θ is

$$\frac{\partial \mathbf{e}_i(\mathbf{x})}{\partial \theta} = \underbrace{\frac{\partial \mathbf{R}_\theta^T}{\partial \theta}}_{\mathbf{R}'_\theta} \underbrace{(\mathbf{p}_i - \mathbf{t})}_{\mathbf{p}'_i}. \tag{2.36}$$

The full Jacobian will then be

$$\frac{\partial \mathbf{e}_i(\mathbf{x})}{\partial \mathbf{x}} = \left(-\mathbf{R}_\theta^T\ \mathbf{R}'_\theta \mathbf{p}'_i\right). \tag{2.37}$$

If you plug these equations in Algorithm 1, it will work.

The reader might notice that we could have substantially simplified our existance if instead fo seeking for the transform $\mathbf{x}$ that maps the position of the robot in the world frame we would have looked for its inverse. This comes from the algebric form of Eq. 2.33. Let then parameterize the state of the robot by the transform $\mathbf{x}' = (t'_x\ t'_y\ \theta')$ such that $\mathbf{T}(\mathbf{x}') = \mathbf{T}^{-1}(\mathbf{x})$. In this case, the error function would have been

$$\mathbf{e}_i(\mathbf{x}') = \mathbf{R}_\theta \mathbf{p}'_i + \mathbf{t} - \mathbf{z}_i. \tag{2.38}$$

If you compare 2.38 with Eq. 2.33 you will notice that the former is much simpler than the latter. Accordingly the the Jacobian of Eq. 2.38 has the following form:

$$\frac{\partial \mathbf{e}_i(\mathbf{x}')}{\partial \mathbf{x}'} = \left(\mathbf{I}_{2\times 2}\ \mathbf{R}'_\theta \mathbf{p}_i\right). \tag{2.39}$$

Example: Bearing only Registration In the previous example we assumed to have a powerful sensor capable of detecting the cartesian coordinates of a point in its reference frame. What if we are poor and we can only afford a much cheaper bearing only sensor? Can we still localize our sensor?

In this simple case, trying is faster than thinking. Let $\mathbf{p}_i$ be world point and $\mathbf{R}_\theta|\mathbf{t}$ be the transform of the origin w.r.t the reference frame of the robot. The point sensed in the robot frame will be

$$\mathbf{p}'_i = \mathbf{g}_i(\mathbf{x}) = \mathbf{R}_\theta \mathbf{p}_i + \mathbf{t} = \begin{pmatrix} x'_i \\ y'_i \end{pmatrix}. \tag{2.40}$$

Here $\mathbf{g}_i(\mathbf{x})$ is a function introduced to highlight the dependency of the projected point from the transform in $\mathbf{x}$. Once we have the point in the robot reference frame we can compute its bearing as:

$$\hat{\mathbf{z}}_i = \mathbf{h}_i(\mathbf{x}) = \operatorname{atan}\left(\frac{y'_i}{x'_i}\right). \tag{2.41}$$

The error function is just the difference between the predicted and sensed bearing

$$\mathbf{e}_i(\mathbf{x}) = \hat{\mathbf{z}}_i - \mathbf{z}_i; \tag{2.42}$$

Ok, the error function is done. To be nice we should fix tghe angulat wraparounds of our estimator, but we will deal with this in the next section. What remains to compute is the Jacobian, which in this case will be 1×3 matrix. We notice that Eq. 2.42 can be rewritten as

$$\mathbf{e}_i(\mathbf{x}) = \operatorname{atan}(\mathbf{p}'_i) - \mathbf{z}_i = \operatorname{atan}(\mathbf{g}_i(\mathbf{x})) - \mathbf{z}_i. \tag{2.43}$$

Here we assumed atan(·) can be fed with a 2D point and returns its bearing; By using the chain rule of derivatives, the Jacobian in the point $\breve{\mathbf{x}}$ becomes:

$$\mathbf{J}_i(\breve{\mathbf{x}}) = \left.\frac{\partial \operatorname{atan}(\mathbf{p})}{\partial \mathbf{p}}\right|_{\mathbf{p}=\mathbf{p}'_i} \cdot \left.\frac{\partial \mathbf{g}_i(\mathbf{x})}{\partial \mathbf{x}}\right|_{\mathbf{x}=\breve{\mathbf{x}}}. \tag{2.44}$$

We already computed the derivative of $\mathbf{g}_i$ w.r.t, $\mathbf{x}$, in Equation 2.39. So we only have to compute $\frac{\partial \operatorname{atan}(\mathbf{p})}{\partial \mathbf{p}}$. Recalling that $d\operatorname{atan}(a)/da = \frac{1}{1+a^2}$ and that $\operatorname{atan}(\mathbf{p}) = \operatorname{atan}(y/x)$ where x and y are the coordinates of $\mathbf{p}$, we can write :

$$\frac{\partial \operatorname{atan}(\mathbf{p})}{\partial \mathbf{p}} = \frac{1}{1+(y/x)^2} \cdot \begin{pmatrix} -\frac{y}{x^2} & \frac{1}{x} \end{pmatrix}. \tag{2.45}$$

Putting it all together in Eq. 2.44 we get

$$\mathbf{J}_i(\breve{\mathbf{x}}) = \frac{1}{x'^2_i + y'^2_i} \cdot (-y'_i \;\; x'_i) \cdot (\mathbf{I}_{2\times 2} \; \mathbf{R}_{\theta'} \mathbf{p}_i). \tag{2.46}$$

With our utmost satisfaction we observe that Eq. 2.46 is dimensionally correct.

2.2.2 Gaussian Conditional: $p(\mathbf{x}|\mathbf{z}) \sim \mathcal{N}(\mathbf{x}^*, \mathbf{H}^{-1})$

In this section we will show that the statement made in its title is true. We will recall some concepts already introduced at the beginning of Section 2.2, namely:

- if the measurements $\mathbf{z}_k$ are affected by a Gaussian noise with information matrices $\mathbf{\Omega}_k$
- under smoothness assumptions, the measurement function $\hat{\mathbf{z}}_k = \mathbf{h}_k(\mathbf{x})$ can be reasonably approximated around the optimum $\mathbf{x}^*$ by its Taylor expansion
- the conditional distribution of the measurements given the increments is normally distributed

$$p(\mathbf{z}_k|\mathbf{\Delta x} + \mathbf{x}^*) \sim \mathcal{N}(\mathbf{J}_k \mathbf{\Delta x} + \mathbf{z}^*_k, \mathbf{\Omega}_k^{-1}). \tag{2.47}$$

  Here $\mathbf{z}^*_k = \mathbf{h}_k(\mathbf{x}^*)$ is the prediction at the optimum.

The conditional over *all* measurements $\mathbf{z}$ is again a multivariate Gaussian

$$p(\mathbf{z}|\mathbf{\Delta x} + \mathbf{x}^*) \sim \mathcal{N}(\mu_{\mathbf{z}}, \mathbf{\Omega}_{\mathbf{z}}^{-1}) \tag{2.48}$$

with

$$\mu_{\mathbf{z}} = \begin{pmatrix} \mathbf{J}_1\mathbf{\Delta x} + \mathbf{z}^*_1 \\ \mathbf{J}_2\mathbf{\Delta x} + \mathbf{z}^*_2 \\ \vdots \\ \mathbf{J}_K\mathbf{\Delta x} + \mathbf{z}^*_K \end{pmatrix} = \underbrace{\begin{pmatrix} \mathbf{J}_1 \\ \mathbf{J}_2 \\ \vdots \\ \mathbf{J}_K \end{pmatrix}}_{\mathbf{J}} \mathbf{\Delta x} + \underbrace{\begin{pmatrix} \mathbf{z}^*_1 \\ \mathbf{z}^*_2 \\ \vdots \\ \mathbf{z}^*_K \end{pmatrix}}_{\mathbf{z}^*} \qquad \mathbf{\Omega}_{\mathbf{z}} = \begin{pmatrix} \mathbf{\Omega}_1 & & & \\ & \mathbf{\Omega}_2 & & \\ & & \ddots & \\ & & & \mathbf{\Omega}_K \end{pmatrix} \tag{2.49}$$

If we have no prior about the states, but we assume they are normally distributed we can describe them with $p(\mathbf{\Delta x}) \sim \mathcal{N}(\mathbf{0}, \mathbf{\Sigma}_{\mathbf{x}})$. To express our total ignorance we set $\mathbf{\Sigma}_{\mathbf{x}}$ to infinity, or more elegantly we set $\mathbf{\Omega}_{\mathbf{x}} = \mathbf{0}$. Recalling Eq. A.14, we have that the joint distribution $p(\mathbf{\Delta x}, \mathbf{z})$ has the following form:

$$p(\mathbf{\Delta x}, \mathbf{z}) \sim \mathcal{N}((0, \mathbf{z}^*)^T, \mathbf{\Omega}^{-1}_{\mathbf{x},\mathbf{z}}) \tag{2.50}$$

Where $\mathbf{z}^* = (\mathbf{z}^*_1, ..., \mathbf{z}^*_K)$ and

$$\mathbf{\Omega}_{\mathbf{x},\mathbf{z}} = \begin{pmatrix} \mathbf{J}^T\mathbf{\Omega}_{\mathbf{z}}\mathbf{J} & -\mathbf{J}^T\mathbf{\Omega}_{\mathbf{z}} \\ -\mathbf{\Omega}_{\mathbf{z}}\mathbf{J} & \mathbf{\Omega}_{\mathbf{z}} \end{pmatrix} \tag{2.51}$$

Let us now consider the upper left block of $\mathbf{\Omega}_{\mathbf{x},\mathbf{z}}$

$$\mathbf{J}^T\mathbf{\Omega}_\mathbf{z}\mathbf{J} = \begin{pmatrix} \mathbf{J}_1^T & \mathbf{J}_2^T & \cdots & \mathbf{J}_K^T \end{pmatrix} \begin{pmatrix} \mathbf{\Omega}_1 & & & \\ & \mathbf{\Omega}_2 & & \\ & & \ddots & \\ & & & \mathbf{\Omega}_K \end{pmatrix} \begin{pmatrix} \mathbf{J}_1 \\ \mathbf{J}_2 \\ \vdots \\ \mathbf{J}_K \end{pmatrix} \tag{2.52}$$

$$= \mathbf{J}_1^T\mathbf{\Omega}_1\mathbf{J}_1 + \mathbf{J}_2^T\mathbf{\Omega}_2\mathbf{J}_2 + \cdots + \underbrace{\mathbf{J}_k^T\mathbf{\Omega}_k\mathbf{J}_k}_{\mathbf{H}_k} + \cdots + \mathbf{J}_K^T\mathbf{\Omega}_K\mathbf{J}_K \tag{2.53}$$

$$= \sum_{k=1}^{K} \mathbf{H}_k \tag{2.54}$$

In substance $\mathbf{H}$ is the upper ledt block of the $\mathbf{\Omega}_{\mathbf{x},\mathbf{z}}$, which is the information matrix of the joint Gaussian distribution of the states and the measurements $p(\mathbf{\Delta x}, \mathbf{z})$.

What we want to do now is to compute the *conditional* distribution $p(\mathbf{\Delta x}|\mathbf{z})$. In particular we are interested in the information matrix $\mathbf{\Omega}_{\mathbf{\Delta x}|\mathbf{z}}$. Thanks to Eq. A.9 we have that conditioning a Gaussian distribution expressed in information form can be easily done by suppressing the rows and the columns of the information matrix corresponding to variables in the conditional. In our case, these blocks are $\mathbf{\Omega}_\mathbf{z}$, $-\mathbf{J}^T\mathbf{\Omega}_\mathbf{z}$ and $-\mathbf{\Omega}_\mathbf{z}\mathbf{J}$. So only $\mathbf{H}$ remains. Intuitively, the mean $\mu_{\mathbf{\Delta x}}$ is $\mathbf{0}$ because we assumed $\mathbf{x}^*$ to be our optimal state.

To conclude we show the initial claim of this section, that is $p(\mathbf{x}|\mathbf{z}) \sim \mathcal{N}(\mathbf{x}^*, \mathbf{H}^{-1})$. To this end, we consider the random variable $\mathbf{x} = \mathbf{x}^* + \mathbf{\Delta x}$. Since $\mathbf{\Delta x} \sim \mathcal{N}(0, \mathbf{\Sigma}_{\mathbf{\Delta x}|\mathbf{z}})$ and $\mathbf{x}^*$ is a constant, $\mathbf{x}$ has the same covariance of $\mathbf{\Delta x}$, and just its mean is shifted of $\mathbf{x}^*$.

2.3 Smooth Manifolds (a.k.a. Non-Euclidean Pain)

In the previous section we put the basis of Least Squares Estimation, and we provided some basic tools for estimating the optimal solution and its uncertainty. However all the derivations we made strongly rely on two aspects, of the input problem, namely: the smoothness of the measurement functions and the fact that the state space spans over an Euclidean domain. All in all, to find a solution with iterative approaches we linearize the problem, compute quadratic approximation of it and compute a set of increments that should decrease the value of the quadratic form. Then, once we have these increments we *sum* them to the previous solution and we pray that they are still good.

Everything works as long as the *sum* is a legal operation in the domain of the state variables considered. In robotics this is often not the case, since angles and rotations are involved. Summing a triplet of Euler angles is not such an elegant thing, and might lead to singular configurations. In fact rotations are not Euclidean, although they admit local Euclidean mappings, they are *smooth manifolds* [17].

To understand the contents of this section, it is a good thing to keep in mind, for instance, the space of the 3D rotations. It is a well known thing that 3D rotations can be parameterized with a rotation matrix $\mathbf{R}$. A rotation matrix, however has 9 numbers to represent 3 angles. If we would use the rotation matrices as parameters in our optimization mechanism we would find non-valid solutions, since the orthogonality constraint is not enforced. Thus, it seems there is no alternative than carrying on the optimization by using a minimal representation, for instance Euler angles. But we just said that Euler angles suffer of singularities. Lucky us, the rotations are a manifold. That is, they admit a local parameterization which is homeomorphic to the vector space $\Re^n$. To explain this sentence, think to a generic rotation $\mathbf{R}(\rho, \theta, \psi)$. We can easily define a mapping between rotation matrices and Euler angles, and viceversa:

$$\mathbf{u} = \text{toVector}(\mathbf{R}) \tag{2.55}$$

$$\mathbf{R} = \text{fromVector}(\mathbf{u}) \tag{2.56}$$

where $\mathbf{u} = (\rho\ \theta\ \psi)^T$ represents the vector containing the 3 Euler angles. Clearly fromVector(toVector($\mathbf{R}$)) = $\mathbf{R}$ and toVector(fromVector($\mathbf{u}$)) = $\mathbf{u}$. If we are around the origin of $\mathbf{u}$, the Euler angles are away from singularities and the euclidean distance measured between two orientations $\mathbf{u}_1$ and $\mathbf{u}_2$ around the origin are closely related to the true distance of their orientations. This situation changes, however when we

are close to a singularity, when a small difference in orientation might lead to an abrupt difference in the local parameters $\mathbf{u}$, with potentially catastrophic consequences on our optimization that strongly relies on the smoothness of the problem.

However, given any arbitrary initial rotation $\mathbf{R}_0$, we can compute a *local* mapping to Euler angles that is away from singularities. In the domain of the rotations, this can be easily done by setting the reference frame so that $\text{toVector}(\mathbf{R}_0) = \mathbf{0}$ is the origin,

$$\mathbf{u} = \mathbf{R} \boxminus \mathbf{R}_0 = \text{toVector}(\mathbf{R}_0^{-1} \cdot \mathbf{R}) \tag{2.57}$$

$$\mathbf{R} = \mathbf{R}_0 \boxplus \mathbf{u} = \mathbf{R}_0 \cdot \text{fromVector}(\mathbf{u}) \tag{2.58}$$

Eq. 2.57 returns the euler angles of the rotation that moves $\mathbf{R}_0$ to $\mathbf{R}$, in the reference frame of $\mathbf{R}_0$. It computes the coordinates in a local map, centered around $\mathbf{R}_0$. If the two rotations have a small difference, also $\mathbf{u}$ will be small, independently on whether $\mathbf{R}_0$ is close to a singularity or not. Similarly, Eq. 2.58 computes a new rotation from $\mathbf{R}_0$, by applying a local perturbation $\mathbf{u}$. Clearly $\mathbf{R}_0 \boxplus (\mathbf{R} \boxminus \mathbf{R}_0) = \mathbf{R}$ and $(\mathbf{R}_0 \boxplus \mathbf{u}) \boxminus \mathbf{R}_0 = \mathbf{u}$. The operators $\boxminus$ and $\boxplus$ are powerful operators that convert a *global* difference in the manifold space in a local perturbation, and viceversa. We have given an example of manifold mapping in the domain of rotation matrices. The question now is how to use this result to improve the optimization.

To this end, let's assume our state space is a smooth manifold. The values in this space are parameterized redundantly by a certain variable $\mathbf{X}$, and minimally by a vector $\mathbf{x}$. A sensor model has the forms $\mathbf{h}_k(\mathbf{X})$ or $\mathbf{h}_k(\mathbf{x})$. If $\breve{\mathbf{X}}$ is a current estimate, we are interested in computing the perturbation of the predicted observation, under a small perturbation of the state variables: $\mathbf{h}_k(\breve{\mathbf{X}} \boxplus \boldsymbol{\Delta}\mathbf{x})$. Consider that $\breve{\mathbf{X}}$ is a constant. Still we can span the whole state space by varying $\delta\mathbf{x}$. We could then compute the Taylor expansion of $\mathbf{h}_k(\breve{\mathbf{X}} \boxplus \boldsymbol{\Delta}\mathbf{x})$, around the point $\boldsymbol{\Delta}\mathbf{x} = \mathbf{0}$, with respect to $\boldsymbol{\Delta}\mathbf{x}$ To this end we can consider the $\breve{\mathbf{X}}$ as a constant

$$\mathbf{h}_k(\breve{\mathbf{X}} \boxplus \boldsymbol{\Delta}\mathbf{x}) \simeq \mathbf{h}_k(\mathbf{X}) + \underbrace{\left.\frac{\partial \mathbf{h}_k(\breve{\mathbf{X}} \boxplus \boldsymbol{\Delta}\mathbf{x})}{\partial \boldsymbol{\Delta}\mathbf{x}}\right|_{\boldsymbol{\Delta}\mathbf{x}=\mathbf{0}}}_{\tilde{\mathbf{J}}_k} \cdot \boldsymbol{\Delta}\mathbf{x} \tag{2.59}$$

$$= \mathbf{h}_k(\breve{\mathbf{X}}) + \tilde{\mathbf{J}}_k \boldsymbol{\Delta}\mathbf{x}. \tag{2.60}$$

This Taylor expansion is very similar to Eq. 2.12, with some notable differences:

- the current linearization point is $\breve{\mathbf{X}}$, and is fixed.
- the expansion gives us the change of the parameters as a function of the variation of the increments, that are applied to the current linearization point through the $\boxplus$ operator.

As a consequence of that, once we find a solution $\boldsymbol{\Delta}\mathbf{x}^*$ for the increments, we need to apply the perturbation to the current estimate by the $\boxplus$ operator:

$$\breve{\mathbf{X}} \leftarrow \breve{\mathbf{X}} \boxplus \boldsymbol{\Delta}\mathbf{x}^*. \tag{2.61}$$

Appropriately dealing with manifold state spaces is essential for direct methods (like GN or LM) to work. Usually, local parametrizations greatly increase the convergence basin of an algorithm.

So said, not only the states can be non-euclidean, but also the measurements. Again, by exploiting the smooth manifolds, we can come up with a general approach that in most of the cases results in smoother error function, thus it generally leads to faster and more robust convergence. In Section 2.2.1 we introduced the error function $\mathbf{e}_k(\mathbf{x}) = \mathbf{h}_k(\mathbf{x}) - \mathbf{z}_k$, that was defined as the Euclidean difference between the prediction and the measurement. Let us consider this alternative error function:

$$\tilde{\mathbf{e}}_k(\mathbf{x}) = \tilde{\mathbf{e}}_k(\hat{\mathbf{z}}_k, \mathbf{z}_k) = \hat{\mathbf{z}}_k \boxminus \mathbf{z}_k = \mathbf{h}_k(\mathbf{x}) \boxminus \mathbf{z}_k \tag{2.62}$$

where we replace the "$-$" with "$\boxminus$". This error function computes the displacement between prediction and observation a minimal space, and selects as center of the "chart" the actual measurement $\mathbf{z}_k$. Since in general prediction and measurement are close in the manifold space of the observations, their displacement computed by using the $\boxminus$ will also be small and regular.

The prediction is distributed according to $p(\mathbf{z}_k|\mathbf{x}) = \mathcal{N}(\hat{\mathbf{z}}_k, \mathbf{\Omega}_k^{-1})$, with $\hat{\mathbf{z}}_k = \mathbf{h}_k(\mathbf{x})$. As usual, we can compute the Taylor expansion of the new error, under a small perturbation of the prediction:

$$\tilde{\mathbf{e}}_k(\hat{\mathbf{z}}_k + \mathbf{\Delta z}_k, \mathbf{z}_k) = (\hat{\mathbf{z}}_k + \mathbf{\Delta z}_k) \boxminus \mathbf{z}_k \tag{2.63}$$

$$\simeq \hat{\mathbf{z}}_k \boxminus \mathbf{z}_k + \underbrace{\frac{\partial \left((\hat{\mathbf{z}}_k + \mathbf{\Delta z}_k) \boxminus \mathbf{z}_k \right)}{\partial \mathbf{\Delta z}_k}}_{\mathbf{J}_{\mathbf{z}_k}} \Bigg|_{\mathbf{\Delta z}_k = \mathbf{0}} \mathbf{\Delta z}_k \tag{2.64}$$

Based on this result, we can compute the gaussian approximation of the conditional distribution of the error as

$$p(\tilde{\mathbf{e}}_k|\mathbf{x}) \sim \mathcal{N}(\hat{\mathbf{z}}_k \boxminus \mathbf{z}_k, \underbrace{\mathbf{J}_{\mathbf{z}_k} \mathbf{\Omega}_k^{-1} \mathbf{J}_{\mathbf{z}_k}^T}_{\mathbf{\Sigma}_{\mathbf{e}_k|\mathbf{x}}}). \tag{2.65}$$

Note that when we project the measurement to a minimal space through $\boxminus$, the conditional covariance of the error $\mathbf{\Sigma}_{\mathbf{e}_k|\mathbf{x}}$ depends on $\hat{\mathbf{z}}_k = \mathbf{h}_k(\mathbf{x})$, thus on $\mathbf{x}$, and needs to be recomputed at every iteration.

The reader might be worried by the complexity of the derivatives one has to compute once exploiting this smooth manifold structure. However, this procedure can be simplified by using the rule on the partial derivatives as:

$$\frac{\partial \mathbf{e}_k(\breve{\mathbf{x}} \boxplus \mathbf{\Delta \tilde{x}})}{\partial \mathbf{\Delta x}} = \underbrace{\frac{\partial \mathbf{e}_k(\mathbf{x})}{\partial \mathbf{x}}}_{\mathbf{J}_k} \Bigg|_{\mathbf{x}=\breve{\mathbf{x}}} \cdot \underbrace{\frac{\partial (\breve{\mathbf{x}} \boxplus \mathbf{\Delta x})}{\partial \mathbf{\Delta x}}}_{\mathbf{M}} \Bigg|_{\mathbf{\Delta x}=\mathbf{0}} \tag{2.66}$$

Accordingly, one can easily derive from the Jacobian *not* defined on a manifold of Eq. 2.12 a Jacobian on a manifold just by multiplying its non-zero blocks by the derivative of the $\boxplus$ operator computed in $\breve{\mathbf{x}}$ Let the Jacobian of $\boxplus$ be denoted by $\mathbf{M}$, as follows:

$$\tilde{\mathbf{J}}_k = \mathbf{J}_k \mathbf{M}. \tag{2.67}$$

With these things in place, we can derive a modified version of the Gauss-Newton algorithm described in Algorithm 1, as shown in Algorithm 3.

Example: Point Registration Again, let us think on how we can modify the point registration example to take into account of manifolds. This is easily done, just by

- characterizing the state space, in this case $SE(2)$, in terms of extended and minimal representations. The extended representation can be the homogeneous transformation matrices. The minimal representation could be the vector $\mathbf{x} = (t_x \; t_y \; \theta)$.
- characterizing the $\boxplus$ and $\boxminus$ operators
  - $\boxplus : SE(2) \times \Re^3 \rightarrow SE(2)$ is defined as follows:

$$\mathbf{X} \boxplus \mathbf{\Delta x} = \mathbf{X} \cdot \text{fromVector}(\mathbf{\Delta x}). \tag{2.68}$$

    Here $\mathbf{X}$ denotes a homogeneous transform and fromVector($\mathbf{\Delta x}$) denotes the transform obtained by converting the vector $\mathbf{x} = (t_x \; t_y \; \theta)$ to a transformation matrix as explained before in this document. An alternative valid form for the $\boxplus$ would be to multiply the increment to the left, as

$$\text{fromVector}(\mathbf{\Delta x}) \cdot \mathbf{X}. \tag{2.69}$$

    Clearly when applying the increments to the state you have to be consistent with the definition of the $\boxplus$ of your choice.

– $\boxminus : SE(2) \times SE(2) \rightarrow \Re^3$ is the operator that computes a minimal vector $\mathbf{\Delta x}$ that applied to the first transform, moves it into the second. It is defined as.

$$\mathbf{X}_2 \boxminus \mathbf{X}_1 = \text{toVector}(\mathbf{X}_1^{-1} \cdot \mathbf{X}_2). \tag{2.70}$$

In sum we compute the "difference" between $\mathbf{X}_2$ and $\mathbf{X}_1$, and convert it to a minimal vector through the toVector(..) function.

- writing the error function, by highlighting the contribution of the perturbation. We refer to the simplified version of the error in Eq. 2.38, for our convenience. In the remainder, I will drop the point index i for sake of notation, and I will denote the point $\mathbf{p} = (p_x \ p_y)$, the transformation matrix $\mathbf{X}$ and the delta increments $\mathbf{\Delta x} = (\Delta t_x \ \Delta t_y \ \Delta\theta)$.

$$\mathbf{e}_i(\mathbf{X} \boxplus \mathbf{\Delta x}) = \text{toHomogeneousMatrix}(\mathbf{\Delta x}) \underbrace{\mathbf{X}\mathbf{p}_i}_{\mathbf{p}'_i} - \mathbf{z}_i. \tag{2.71}$$

note that in the previous equation we used the defnition of the $\boxplus$ in Eq. 2.69. We then can compute the Jacobian as

$$\left.\frac{\partial \mathbf{e}_i(\mathbf{X} \boxplus \mathbf{\Delta x})}{\partial \mathbf{\Delta x}}\right|_{\mathbf{\Delta x}=\mathbf{0}} = \left.\frac{\partial \begin{pmatrix} p'_x \cos\Delta\theta - p'_y \sin\Delta\theta + \Delta t_x \\ p'_x \sin\Delta\theta + p'_y \cos\Delta\theta + \Delta t_y \end{pmatrix}}{\partial \mathbf{\Delta x}}\right|_{\mathbf{\Delta x}=0} \tag{2.72}$$

$$= \begin{pmatrix} \mathbf{I}_{2\times 2} & \begin{matrix} -p'_y \\ p'_x \end{matrix} \end{pmatrix}. \tag{2.73}$$

Once you have the error functiond and the $\boxplus$ operator you can happily construct the least quares problem. In this case you will have to apply the incremental transformation *to the left* of the initial one, according to the choice we have taken for the $\boxplus$.

2.4 Approaching a Least Squares Estimation Problem: Methodology

In the previous sections we presented a bunch of concepts and algorithms to approach a least squares estimation problem. Here we try to wrap these ideas together and to give you a general methodology to construct your solver for a given problem. In fact we have already seen this methodology in the point registration example.

A general estimation problem might be expressed as find $p(\mathbf{x}|\mathbf{z})$, where $\mathbf{x}$ are the state variables and $\mathbf{z} = \mathbf{z}_{1:K}$ are the measurements. To construct a solver you have to follow these sequential steps:

1. **Identify a feasible parameterization for the state variables**: This means that you have to determine a way to represent the possible states of your system. To this end, choose a convenient representation, like transformation matrices for $SE(2)$ or $SE(3)$. This "Extended" parameterization should not be minimal. In case the space is Euclidean, however, the parameterization will be minimal, and it will live in the same space as the perturbation. Let $\mathbf{X}$ this parameterization.

2. **If the state variables do not live in an Euclidean space**

   - **Define a parameterization for the increments**: It is often convenient solving the linearized system in a minimal representation, for the reasons outlined above. Let $\mathbf{\Delta x}$ this minimal parameterization.
   - **Define the $\boxplus$ operator** The $\boxplus$ operator should apply to a state variable the perturbation, expressed in the minimal form.

3. **Define a parameterization for the measurements $\mathbf{z}_k$**: The measurements are "what my sensor observes", and finding a parameterization means defining how a bunch of numbers can be used to describe a particular measurement. Let $\mathbf{Z}$ these parameters.

4. **If the measurements do not live in an Euclidean space**

   - **Define a minimal parameterization for the difference in the measurements**: Analogous to the case of the state variables, if the measurements do not live in an euclidean space, we need to map "differences" between measurements in an euclidean space, in order to cast our linear system. The difference between two identical variables should result in an euclidean vector of $\mathbf{0}$ norm. Let $\boldsymbol{\Delta}\mathbf{z}$ this parameterization.
   - **Define the $\boxminus$ operator** The $\boxminus$ operator should return the euclidean difference between the measurements and the state.

5. **Define the measurement function**: This means "describing what your sensor should measure, given a known configuration of the state variables". Let $\hat{\mathbf{Z}} = \mathbf{h}(\mathbf{X})$ this function.

6. **Define the error function**: With the $\boxminus$ operator in place, you can construct the error function function as

$$\mathbf{e}(\mathbf{X}) = \mathbf{h}(\mathbf{X}) \boxminus \mathbf{Z}, \tag{2.74}$$

   where $\mathbf{Z}$ is the measurement.

7. **computing the Jacobians**: that is

$$\mathbf{J} = \left. \frac{\partial \mathbf{e}(\mathbf{X} \boxplus \boldsymbol{\Delta}\mathbf{x})}{\partial \boldsymbol{\Delta}\mathbf{x}} \right|_{\boldsymbol{\Delta}\mathbf{x}=\mathbf{0}} \tag{2.75}$$

   Note that if you feel bad in computing large derivatives, you can always compute the jacobians numerically.

8. **Put the things together in Algorithm 3**.

Algorithm 2 Levenberg-Marquardt minimization algorithm

Require: $\breve{\mathbf{x}}$: initial guess. $\mathcal{C} = \{\langle \mathbf{e}_k(\cdot), \mathbf{\Omega}_k \rangle\}$: measurements
Ensure: $\mathbf{x}^*$: new solution

```
//compute the current error
F_new ← F̆
//backup the solution
x_bakup ← x̆
λ ← computeInitialLambda(C, x̆)
// iterate until no substantial improvements are made
repeat
  F̆ ← F_new
  b ← 0      H ← 0
  for all k = 1...K do
    // Compute the prediction ẑ_k, the error e_k and the jacobian J_k
    ẑ_k ← h_k(x̆)
    e_k ← ẑ_k − z_k
    J_k ← ∂h_k(x̆ ⊞ Δx)/∂Δx |_{Δx=0}
    // compute the contribution of this measurement to the linear system
    H_k ← J_k^T Ω_k J_k
    b_k ← J_k^T Ω_k e_k
    // accumulate the contribution to construct the overall system
    H += H_k
    b += b_k
  end for
  //Adjust the damping factor and perform backup/restore actions
  t ← 0 // number of iterations the damping factor has been adjusted
  repeat
    // solve the linear system
    Δx ← solve((H + λI)Δx = b)
    // update the state
    x̆ += Δx
    // compute the new error
    F_new ← F(x̆)
    if F_new < F̆ then
      // good step, accept the solution and update the temporaries
      λ ← λ/2
      x_bakup ← x̆
      t ← −1
    else
      // bad step, revert the solution
      λ ← λ * 4
      x̆ = x_bakup
      t += 1
    end if
  until t < t_max ∧ t > 0
until F̆ − F_new > ε ∧ t < t_max
return x̆
```

Algorithm 3 Gauss-Newton minimization algorithm for manifold measurement and state spaces

Require: $\breve{\mathbf{x}}$: initial guess. $\mathcal{C} = \{\langle \mathbf{z}_k(\cdot), \mathbf{\Omega}_k \rangle\}$: measurements
Ensure: $\mathbf{x}^*$: new solution

```
//compute the current error
F_new ← F̆
// iterate until no substantial improvements are made
repeat
  F̆ ← F_new
  b ← 0      H ← 0
  for all k = 1...K do
    // Compute the prediction ẑ_k and the error ẽ_k
    // Note that the measurements z_k are overparameterized
    ẑ_k ← ĥ_k(x)
    ẽ_k ← ẑ_k ⊟ z_k
    and the jacobians J̃_k and J_{z_k}
    // Compute the jacobian of the error function, in the space of the increments
    J_k ← ∂e_k(h_k(x̆ ⊞ Δx), z_k)/∂Δx_k |_{Δx_k = 0}
    // Compute the jacobian of the ominus operator in the measurements
    J_{z_k} ← ∂((ẑ_k + Δz_k) ⊟ z_k)/∂Δz_k |_{Δz_k = 0}
    // Remap the information matrix
    Ω̃_k ← J_{z_k} Ω_k^{-1} J_{z_k}^T
    // compute the contribution of this measurement to the linear system
    H_k ← J_k^T Ω̃_k J_k
    b_k ← J_k^T Ω̃_k e_k
    // accumulate the contribution to construct the overall system
    H += H_k
    b += b_k
  end for
  // solve the linear system
  Δx ← solve(H Δx = −b)
  // update the state
  x̆ ← x̆ ⊞ Δx
  // compute the new error
  F_new ← F(x̆)
until F̆ − F_new > ε
return x̆
```

Chapter 3

Sparse Least Squares

Least Squares Estimation, is a powerful tool that allows us to estimate a set of hidden variables from measurements that depend on these variables. In the previous sections we introduced the general concepts, that remain valid. The dimension of the state vector, for certain classes of problems, might be very large. The least squares solution of the problem involves the solution of a very large linear system, that might become a performance bottleneck.

It is not uncommon to encounter problems where, albeit a large dimension of the state vector, a measurement depends only on a subset of the state variables. In this section we will discover how to take advantage of this fact to enhance the performances of our solver, and approaching problems that otherwise would not be tractable. We start with introducing a simple example that captures one of these problems, and then we will reason on how we can exploit the intuition (hopefully) given to us by our example in a more general way.

3.1 Typical Problems

3.1.1 Example: 2D Multirobot Point Registration

For instance imagine the following problem: we have a robot moving in $SE(2)$, equipped with a point sensor. Let $\mathbf{x}^r_{1:n}$ these robot positions, with $\mathbf{x}^r_i \in SE(2)$. This robot moves in a world populated by a set of *distinguishable* points, and senses them. Let $\mathbf{x}^p_{1:m}$ these point positions, with $\mathbf{x}^p_j \in \Re^2$. For each robot position, our sensor will report a set of measurements corresponding to the $x-y$ positions of the landmarks, in the reference frame of the robot. Let the generic measurement $\mathbf{z}_{i,j}$ a measurement made by the robot at position $\mathbf{x}^r_i$ about the landmark $\mathbf{x}^p_j$, with $\mathbf{z}_{i,j} \in \Re^2$.

Our goal is to estimate:

- the position of all robots $\mathbf{x}^r_{1:n}$,
- the position of all landmarks $\mathbf{x}^p_{1:m}$

given the measurements $\{\mathbf{z}_{i,j}\}$.

To construct this solver we will be following the methodology outlined in Section 2.4

- **Identification of the state variables**
  The state variables are all robot positions and all landmarks.
  - *Robot positions:* The extended parameterization for the i^{th} robot position is a transformation matrix, denoted with $\mathbf{X}^r_i$. To ease the calculations of the jacobians in the remainder of this section, we select to describe the position of the robot, as the transformation that maps the origin w.r.t the local frame of the robot, as we did in the example in Section 3.4.
  - *Landmarks:* It is with immense pleasure that we note the landmarks live in $\Re^2$, so and this is the way we will represent the blocks of the state.

The overall state $\mathbf{x}$ will then be a data structure consisting of n transformation matrices $\mathbf{X}^r_{1:n}$ and m two dimensional vectors $\mathbf{x}^p_{1:m}$, as follows:

$$\mathbf{x} = \{\mathbf{X}^r_{1:n}, \mathbf{x}^p_{i:m}\} \tag{3.1}$$

- **Parameterization of the increments**
  - *Robot positions:* An increment in a robot position can be described just by the 3 vector $\mathbf{\Delta x}^r_i = (t_{x_i}\ t_{y_i}\ \theta_i)$. The boxplus will be the same as in Section 2.3, that is

$$\mathbf{X}^r_i \boxplus \mathbf{\Delta x}^r_i = \text{fromVector}(\mathbf{\Delta x}^r_i) \cdot \mathbf{X}^r_i \tag{3.2}$$

  - *Landmarks:* the landmarks live in an euclidean space, so the increment will just be an euclidean displacement that can be applied to a state variable with the regular vector addition.

Our displacement vector $\mathbf{\Delta_x}$ will be consisting of the chaining of the displacements of all state variables, that is

$$\mathbf{\Delta x} = (\mathbf{\Delta x}^r_1\ \mathbf{\Delta x}^r_2\ \cdots\ \mathbf{\Delta x}^r_n\ \mathbf{\Delta x}^p_1\ \cdots\ \mathbf{\Delta x}^p_m) \tag{3.3}$$

This displacement can be applied to the state by applying the individual perturbation to each state block, according to the $\boxplus$ defined before. More specifically we can construct the global perturbation operator $\mathbf{x}' = \mathbf{x} \boxplus \mathbf{\Delta x}$, where $\mathbf{x}'$ is the global state after applying the perturbation, $\mathbf{x}$ is the initial state and $\mathbf{\Delta x}$ is the perturbation defined in Eq. 3.3 as follows:

$$\begin{aligned}
\mathbf{x} &= \{\mathbf{X}^r_{1:n}, \mathbf{x}^p_{i:m}\} && (3.4)\\
\mathbf{x}' &= \{\mathbf{X}'^r_{1:n}, \mathbf{x}'^p_{i:m}\} && (3.5)\\
\mathbf{X}'^r_i &= \mathbf{X}^r_i \boxplus \mathbf{\Delta x}^r_i && (3.6)\\
\mathbf{x}'^p_j &= \mathbf{x}^p_j + \mathbf{\Delta x}^p_j && (3.7)
\end{aligned}$$

- **Measurement function:** The measurement function predicting the measurement $\mathbf{z}_{i,j}$ will depend on the i^{th} robot pose and on the j^{th} landmark variable, and will map the landmark variable from the global frame to the local frame of the robot, as follows:

$$\mathbf{h}_{i,j}(\mathbf{X}) = \mathbf{X}^r_i \cdot \begin{pmatrix} \mathbf{x}^p_j \\ 1 \end{pmatrix}. \tag{3.8}$$

- **Error function:** The measurement space is Euclidean, thus the error function is just the difference between the prediction and the measurement

$$\mathbf{e}_{i,j}(\mathbf{X}) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \mathbf{X}^r_i \cdot \begin{pmatrix} \mathbf{x}^p_j \\ 1 \end{pmatrix} - \mathbf{z}_{i,j}. \tag{3.9}$$

The matrix $\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$ is just used to eliminate the useless homogeneous component.

- **Jacobians:** To compute the jacobian we first have to write the error function at the perturbations

$$\mathbf{e}_{i,j}(\mathbf{X} \boxplus \mathbf{\Delta x}) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \text{fromVector}(\mathbf{\Delta x}^r_i) \mathbf{X}^r_i \begin{pmatrix} \mathbf{x}^p_j + \mathbf{\Delta x}^p_j \\ 1 \end{pmatrix}. \tag{3.10}$$

We note that the error function depends only on two variables, thus the jacobian will be non zero only in the 2×3 block corresponding to variable $\mathbf{\Delta x}^r_i$ and in the 2×3 block corresponding to $\mathbf{\Delta x}^p_j$, as

$$\frac{\partial \mathbf{e}_{i,j}(\mathbf{X} \boxplus \mathbf{\Delta x})}{\partial \mathbf{\Delta x}} = \left(\mathbf{0}\ \cdots \mathbf{0} \frac{\partial \mathbf{e}_{i,j}(\mathbf{X}^r_i \boxplus \mathbf{\Delta x}^r_i, \mathbf{x}^p_j)}{\partial \mathbf{\Delta x}^r_i} \mathbf{0}\ \cdots \mathbf{0} \frac{\partial \mathbf{e}_{i,j}(\mathbf{X}^r_i, \mathbf{x}^p_j + \mathbf{\Delta x}^p_j)}{\partial \mathbf{\Delta x}^p_j} \mathbf{0}\ \cdots \mathbf{0} \right) \tag{3.11}$$

Analogous to the example of point registration, evaluating the jacobian for $\mathbf{\Delta x} = \mathbf{0}$, we have that

$$\frac{\partial \mathbf{e}_{i,j}(\mathbf{X}_i^r \boxplus \mathbf{\Delta x}_i^r, \mathbf{x}_j^p)}{\partial \mathbf{\Delta x}_i^r} = \begin{pmatrix} \mathbf{I}_{2\times 2} & -p'_y \\ & p'_x \end{pmatrix} \tag{3.12}$$

where $\mathbf{p}' = (-p'_y \; p'_x) = \mathbf{X}_i^r \mathbf{x}_j^p$. Being a linear transformation,

$$\frac{\partial \mathbf{e}_{i,j}(\mathbf{X}_i^r, \mathbf{x}_j^p + \mathbf{\Delta x}_j^p)}{\partial \mathbf{\Delta x}_j^p} = \mathbf{R}(\theta_i). \tag{3.13}$$

3.1.2 Example: Pose SLAM

Here we propose another example: Pose SLAM. Imagine we have a set of robots living in a 2D world, each located at a position $\mathbf{x}_i \in SE(2)$. Each robot is equipped with a fancy sensor capable of reporting the postions of the other robots in its own reference frame. A measurement made by robot i about the location of robot j will be $\mathbf{z}_{i,j} \in SE(2)$. Once again we assume the sensor is capable of determining which other robot it is sensing.

Our goal is to estimate:

- the position of all robots $\mathbf{x}_{1:n}$,

given the measurements $\{\mathbf{z}_{i,j}\}$.

Again, we follow our beloved methodology

- **Identification of the state variables**
  The state variables are all robot positions. The extended parameterization for the i^{th} robot position is a transformation matrix, denoted with $\mathbf{X}_i$.

  The overall state $\mathbf{x}$ will then be a data structure consisiting of n transformation matrices $\mathbf{X}_{1:n}$

  $$\mathbf{x} = \{\mathbf{X}_{1:n}\} \tag{3.14}$$

- **Parameterization of the increments** An increment in a robot position can be described just by the 3 vector $\mathbf{\Delta x}_i = (t_{x_i} \; t_{y_i} \; \theta_i)$. The boxplus will be the same as in Section 3.4, that is

  $$\mathbf{X}_i \boxplus \mathbf{\Delta x}_i = \mathbf{X}_i \cdot \text{fromVector}(\mathbf{\Delta x}_i) \tag{3.15}$$

  Our displacement vector $\mathbf{\Delta}_\mathbf{x}$ will be consisting of the chaining of the displacements of all state variables, that is

  $$\mathbf{\Delta x} = (\mathbf{\Delta x}_1 \; \mathbf{\Delta x}_2 \; \cdots \; \mathbf{\Delta x}_n) \tag{3.16}$$

  This displacement can be applied to the state by applying the individual perturbation to each state block, as in the previous example. Let $\mathbf{x}' = \mathbf{x} \boxplus \mathbf{\Delta x}$, be the global state after applying the perturbation, $\mathbf{x}$ the initial state and $\mathbf{\Delta x}$ is the perturbation defined in Eq. 3.16 as follows:

  $$\begin{aligned} \mathbf{x} &= \{\mathbf{X}_{1:n}\} && (3.17)\\ \mathbf{x}' &= \{\mathbf{X}'_{1:n}\} && (3.18)\\ \mathbf{X}_i &= \mathbf{X}_i \boxplus \mathbf{\Delta x}_i && (3.19) \end{aligned}$$

- **Measurement function:** The measurement function predicting the measurement $\mathbf{z}_{i,j}$ will depend on the i^{th} and the j^{th} robot poses, and will map the position of the robot j in the frame of robot i as follows:

  $$\mathbf{h}_{i,j}(\mathbf{X}) = \mathbf{X}_i^{-1} \cdot \mathbf{X}_j. \tag{3.20}$$

- **Error function:** The measurement space is a manifold, we need to define the $\boxminus$ operator as the operator that maps the "difference" between two reference frames in a minimal perturbation.

  $$\mathbf{X}_a \boxminus \mathbf{X}_b = \text{toVector}(\mathbf{X}_b^{-1} \cdot \mathbf{X}_a) \tag{3.21}$$

  The error will then be

  $$\begin{aligned} \mathbf{e}_{i,j}(\mathbf{X}) &= \mathbf{h}_{i,j}(\mathbf{X}) \boxminus \mathbf{Z}_{i,j} && (3.22)\\ &= \text{toVector}(\mathbf{Z}_{i,j}^{-1} \cdot \mathbf{X}_i^{-1} \cdot \mathbf{X}_j). && (3.23) \end{aligned}$$

- **Jacobians:** Again, to compute the jacobian we have to write the error function at the perturbations

$$\begin{aligned}
\mathbf{e}_{i,j}(\mathbf{X} \boxplus \boldsymbol{\Delta}\mathbf{x}) &= \text{toVector}\left[\mathbf{Z}_{i,j}^{-1} \cdot (\mathbf{X}_i \cdot \text{fromVector}(\boldsymbol{\Delta}\mathbf{x}_i))^{-1} \cdot (\mathbf{X}_j \cdot \text{fromVector}(\boldsymbol{\Delta}\mathbf{x}_j))\right] && (3.24)\\
&= \text{toVector}\left[\mathbf{Z}_{i,j}^{-1}\text{fromVector}(\boldsymbol{\Delta}\mathbf{x}_i)^{-1} \cdot \underbrace{\mathbf{X}_i^{-1} \cdot \mathbf{X}_j}_{\mathbf{X}_{i,j}} \cdot \text{fromVector}(\boldsymbol{\Delta}\mathbf{x}_j)\right] && (3.25)\\
&= \text{toVector}\left[\mathbf{Z}_{i,j}^{-1}\text{fromVector}(\boldsymbol{\Delta}\mathbf{x}_i)^{-1} \cdot \mathbf{X}_{i,j} \cdot \text{fromVector}(\boldsymbol{\Delta}\mathbf{x}_j)\right] && (3.26)
\end{aligned}$$

  As in the previous case, we note that the error function depends only on two variables, thus the jacobian will be non zero only in the 3×3 blocks corresponding to variables $\boldsymbol{\Delta}\mathbf{x}_i$ and $\boldsymbol{\Delta}\mathbf{x}_j$

$$\frac{\partial \mathbf{e}_{i,j}(\mathbf{X} \boxplus \boldsymbol{\Delta}\mathbf{x})}{\partial \boldsymbol{\Delta}\mathbf{x}} = \left(\mathbf{0} \ \cdots \mathbf{0} \frac{\partial \mathbf{e}_{i,j}(\mathbf{X}_i \boxplus \boldsymbol{\Delta}\mathbf{x}_i, \mathbf{X}_j)}{\partial \boldsymbol{\Delta}\mathbf{x}_i} \mathbf{0} \ \cdots \mathbf{0} \frac{\partial \mathbf{e}_{i,j}(\mathbf{X}_i, \mathbf{X}_j \boxplus \boldsymbol{\Delta}\mathbf{x}_j)}{\partial \boldsymbol{\Delta}\mathbf{x}_j} \mathbf{0} \ \cdots \mathbf{0}\right) \quad (3.27)$$

  If the reader (you) feels comfortable and rested I would suggest him to pursue the challenge of computing the jacobians above in analytic form. So said, everything works well even with numeric jacobians.

3.2 A Graphical Representation: Factor Graphs

A least squares minimization problem can be described by the following equation:

$$\begin{aligned}
\mathbf{F}(\mathbf{x}) &= \sum_{k \in \mathcal{C}} \underbrace{\mathbf{e}_k(\mathbf{x}_k, \mathbf{z}_k)^T \boldsymbol{\Omega}_k \mathbf{e}_k(\mathbf{x}_k, \mathbf{z}_k)}_{\mathbf{F}_k} && (3.28)\\
\mathbf{x}^* &= \underset{\mathbf{x}}{\text{argmin}}\, \mathbf{F}(\mathbf{x}). && (3.29)
\end{aligned}$$

Here

- $\mathbf{x} = (\mathbf{x}_1^T, \ \dots \ , \mathbf{x}_n^T)^T$ is a vector of parameters, where each $\mathbf{x}_i$ represents a generic parameter block.
- $\mathbf{x}_k = (\mathbf{x}_{k_1}^T, \ \dots \ , \mathbf{x}_{k_q}^T)^T \subset (\mathbf{x}_1^T, \ \dots \ , \mathbf{x}_n^T)^T$ is the subset of the parameters involved in the k^{th} constraint.
- $\mathbf{z}_k$ and $\boldsymbol{\Omega}_k$ represent respectively the mean and the information matrix of a constraint relating the parameters in $\mathbf{x}_k$.
- $\mathbf{e}_k(\mathbf{x}_k, \mathbf{z}_k)$ is a vector error function that measures how well the parameter blocks in $\mathbf{x}_k$ satisfy the constraint $\mathbf{z}_k$. It is $\mathbf{0}$ when $\mathbf{x}_k$ perfectly matches the constraint. As an example, if one has a measurement function $\hat{\mathbf{z}}_k = \mathbf{h}_k(\mathbf{x}_k)$ that generates a synthetic measurement $\hat{\mathbf{z}}_k$ given an actual configuration of the nodes in $\mathbf{x}_k$. A straightforward error function would then be $\mathbf{e}(\mathbf{x}_k, \mathbf{z}_k) = \mathbf{h}_k(\mathbf{x}_k) - \mathbf{z}_k$.

For simplicity of notation, in the rest of this section we will encode the measurement in the indices of the error function:

$$\mathbf{e}_k(\mathbf{x}_k, \mathbf{z}_k) \ = \ \mathbf{e}_k(\mathbf{x}_k) \ = \ \mathbf{e}_k(\mathbf{x}). \quad (3.30)$$

Note that each error function and each parameter block can span over a different space. A problem in this form can be effectively represented by a directed hyper-graph. A node i of the graph represents the parameter block $\mathbf{x}_i \in \mathbf{x}_k$ and a hyper-edge among the nodes $\mathbf{x}_i \in \mathbf{x}_k$ represents a constraint involving all nodes in $\mathbf{x}_k$. In case the hyper edges have size 2, the hyper-graph becomes an ordinary graph. Figure 3.1 shows an example of mapping between a hyper-graph and an objective function.

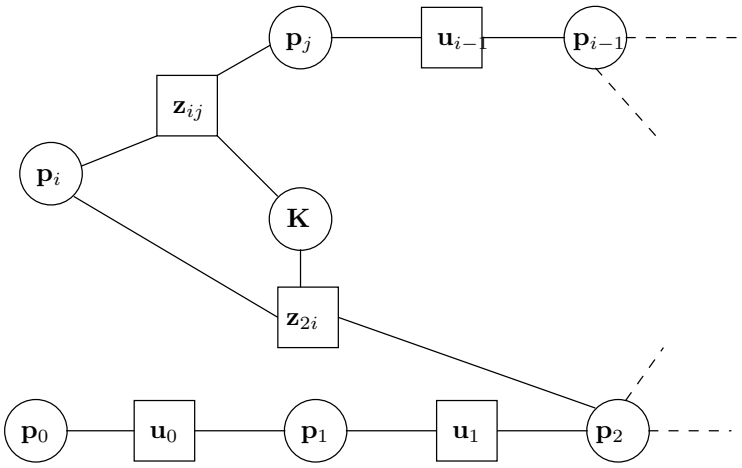


Figure 3.1: This example illustrates how to represent an objective function by a hyper-graph. Here we illustrate a portion of a small SLAM problem [16]. In this example we assume that where the measurement functions are governed by some unknown calibration parameters $\mathbf{K}$. The robot poses are represented by the variables $\mathbf{p}_{1:n}$. These variables are connected by constraints $\mathbf{z}_{ij}$ depicted by the square boxes. The constraints arise, for instance, by matching nearby laser scans *in the laser reference frame*. The relation between a laser match and a robot pose, however, depends on the position of the sensor on the robot, which is modeled by the calibration parameters $\mathbf{K}$. Conversely, subsequent robot poses are connected by binary constraints $\mathbf{u}_k$ arising from odometry measurements. These measurements are made in the frame of the robot mobile base.

3.3 Structure of the Linearized System

According to (2.28), the matrix $\mathbf{H}$ and the vector $\mathbf{b}$ are obtained by summing up a set of matrices and vectors, one for every constraint. If we set $\mathbf{b}_k = \mathbf{J}_k^T \mathbf{\Omega}_k \mathbf{e}_k$ and $\mathbf{H}_k = \mathbf{J}_k^T \mathbf{\Omega}_k \mathbf{J}_k$ we can rewrite $\mathbf{H}$ and $\mathbf{b}$ as

$$\mathbf{b} = \sum_{k \in \mathcal{C}} \mathbf{b}_k \tag{3.31}$$

$$\mathbf{H} = \sum_{k \in \mathcal{C}} \mathbf{H}_k. \tag{3.32}$$

Every constraint will contribute to the system with an addend term. The *structure* of this addend depends on the Jacobian of the error function. Since the error function of a constraint depends only on the values of the nodes $\mathbf{x}_i \in \mathbf{x}_k$, the Jacobian in (2.12) has the following form:

$$\mathbf{J}_k = \left(\mathbf{0} \cdots \mathbf{0}\ \mathbf{J}_{k_1}\ \cdots\ \mathbf{J}_{k_i}\ \cdots \mathbf{0}\ \cdots\ \mathbf{J}_{k_q} \mathbf{0} \cdots \mathbf{0}\right). \tag{3.33}$$

Here $\mathbf{J}_{k_i} = \frac{\partial \mathbf{e}(\mathbf{x}_k)}{\partial \mathbf{x}_{k_i}}$ are the derivatives of the error function with respect to the nodes connected by the k^{th} hyper-edge, with respect to the parameter block $\mathbf{x}_{k_i} \in \mathbf{x}_k$.

From (2.25) we obtain the following structure for the block matrix $\mathbf{H}_{ij}$:

$$\mathbf{H}_k = \begin{pmatrix} \ddots & & & & & & \\ & \mathbf{J}_{k_1}^T \mathbf{\Omega}_k \mathbf{J}_{k_1} & \cdots & \mathbf{J}_{k_1}^T \mathbf{\Omega}_k \mathbf{J}_{k_i} & \cdots & \mathbf{J}_{k_1}^T \mathbf{\Omega}_k \mathbf{J}_{k_q} & \\ & \vdots & & \vdots & & \vdots & \\ & \mathbf{J}_{k_i}^T \mathbf{\Omega}_k \mathbf{J}_{k_1} & \cdots & \mathbf{J}_{k_i}^T \mathbf{\Omega}_k \mathbf{B}_{k_i} & \cdots & \mathbf{J}_{k_i}^T \mathbf{\Omega}_k \mathbf{J}_{k_q} & \\ & \vdots & & \vdots & & \vdots & \\ & \mathbf{J}_{k_q}^T \mathbf{\Omega}_k \mathbf{J}_{k_1} & \cdots & \mathbf{J}_{k_q}^T \mathbf{\Omega}_k \mathbf{B}_{k_i} & \cdots & \mathbf{J}_{k_q}^T \mathbf{\Omega}_k \mathbf{J}_{k_q} & \\ & & & & & & \ddots \end{pmatrix} \tag{3.34}$$

$$\mathbf{b}_k = \begin{pmatrix} \vdots \\ \mathbf{J}_{k_1} \mathbf{\Omega}_k \mathbf{e}_k \\ \vdots \\ \mathbf{J}_{k_i}^T \mathbf{\Omega}_k \mathbf{e}_k \\ \vdots \\ \mathbf{J}_{k_q}^T \mathbf{\Omega}_k \mathbf{e}_k \\ \vdots \end{pmatrix} \tag{3.35}$$

For simplicity of notation we omitted the zero blocks. The reader might notice that the block structure of the matrix $\mathbf{H}$ is the adjacency matrix of the hyper graph. Additionally the Hessian $\mathbf{H}$ is a symmetric matrix, since all the $\mathbf{H}_k$ are symmetric. A single hyper-edge connecting q vertices will introduce q^2 non zero blocks in the Hessian, in correspondence of each pair $\langle \mathbf{x}_{k_i}, \mathbf{x}_{k_j} \rangle$ of nodes connected.

3.4 Sparse Least Squares on Manifold

To deal with parameter blocks that span over a non-Euclidean spaces, it is common to apply the error minimization on a manifold. A manifold is a mathematical space that is not necessarily Euclidean on a global scale, but can be seen as Euclidean on a local scale [18].

For example, in the context of SLAM problem, each parameter block $\mathbf{x}_i$ consists of a translation vector $\mathbf{t}_i$ and a rotational component α_i. The translation $\mathbf{t}_i$ clearly forms a Euclidean space. In contrast to that, the rotational components α_i span over the non-Euclidean 2D or 3D rotation group $SO(2)$ or $SO(3)$. To avoid singularities, these spaces are usually described in an over-parameterized way, e.g., by rotation matrices or quaternions. Directly applying (2.29) to these over-parameterized representations breaks the constraints induced by the over-parameterization. The over-parameterization results in additional degrees of freedom and thus introduces errors in the solution. To overcome this problem, one can use a minimal representation for the rotation (like Euler angles in 3D). This, however, is then subject to singularities.

As discussed in Section 2.3 an alternative idea is to consider the underlying space as a manifold and to define an operator $\boxplus$ that maps a local variation $\mathbf{\Delta x}$ in the Euclidean space to a variation on the manifold, $\mathbf{\Delta x} \mapsto \mathbf{x} \boxplus \mathbf{\Delta x}$. We refer the reader to [11, §1.3] for more mathematical details on manifold in sparse problems. With this operator, a new error function can be defined as

$$\breve{\mathbf{e}}_k(\mathbf{\Delta}\tilde{\mathbf{x}}_k) = \mathbf{e}_k(\breve{\mathbf{x}}_k \boxplus \mathbf{\Delta}\tilde{\mathbf{x}}_k) \tag{3.36}$$

$$= \mathbf{e}_k(\breve{\mathbf{x}} \boxplus \mathbf{\Delta}\tilde{\mathbf{x}}) \simeq \breve{\mathbf{e}}_k + \tilde{\mathbf{J}}_k \mathbf{\Delta}\tilde{\mathbf{x}}, \tag{3.37}$$

where $\breve{\mathbf{x}}$ spans over the original over-parameterized space, for instance quaternions. The term $\mathbf{\Delta}\tilde{\mathbf{x}}$ is a small increment around the original position $\breve{\mathbf{x}}$ and is expressed in a minimal representation. A common choice for $SO(3)$ is to use the vector part of the unit quaternion. In more detail, one can represent the increments $\mathbf{\Delta}\tilde{\mathbf{x}}$ as 6D vectors $\mathbf{\Delta}\tilde{\mathbf{x}}^T = (\mathbf{\Delta}\tilde{\mathbf{t}}^T \; \tilde{\mathbf{q}}^T)$, where $\mathbf{\Delta}\tilde{\mathbf{t}}$ denotes the translation and $\tilde{\mathbf{q}}^T = (\Delta q_x \; \Delta q_y \; \Delta q_z)^T$ is the vector part of the unit quaternion representing the 3D rotation. Conversely, $\breve{\mathbf{x}}^T = (\breve{\mathbf{t}}^T \; \breve{\mathbf{q}}^T)$ uses a quaternion $\breve{\mathbf{q}}$ to encode the rotational part. Thus, the operator $\boxplus$ can be expressed by first converting $\mathbf{\Delta}\tilde{\mathbf{q}}$ to a full quaternion $\mathbf{\Delta q}$ and then applying the transformation $\mathbf{\Delta x}^T = (\mathbf{\Delta t}^T \; \mathbf{\Delta q}^T)$

to $\breve{\mathbf{x}}$. In the equations describing the error minimization, these operations can nicely be encapsulated by the $\boxplus$ operator. The Jacobian $\tilde{\mathbf{J}}_k$ can be expressed by

$$\tilde{\mathbf{J}}_k = \left.\frac{\partial \mathbf{e}_k(\breve{\mathbf{x}} \boxplus \boldsymbol{\Delta}\tilde{\mathbf{x}})}{\partial \boldsymbol{\Delta}\tilde{\mathbf{x}}}\right|_{\boldsymbol{\Delta}\tilde{\mathbf{x}}=\mathbf{0}}. \tag{3.38}$$

Since in the previous equation $\breve{\mathbf{e}}$ depends only on $\boldsymbol{\Delta}\tilde{\mathbf{x}}_{k_i} \in \boldsymbol{\Delta}\tilde{\mathbf{x}}_k$ we can further expand it as follows:

$$\tilde{\mathbf{J}}_k = \left.\frac{\partial \mathbf{e}_k(\breve{\mathbf{x}} \boxplus \boldsymbol{\Delta}\tilde{\mathbf{x}})}{\partial \boldsymbol{\Delta}\tilde{\mathbf{x}}}\right|_{\boldsymbol{\Delta}\tilde{\mathbf{x}}=\mathbf{0}} \tag{3.39}$$

$$= \left(\mathbf{0}\cdots\mathbf{0}\ \tilde{\mathbf{J}}_{k_1}\ \cdots\ \tilde{\mathbf{J}}_{k_i}\ \cdots\mathbf{0}\ \cdots\ \tilde{\mathbf{J}}_{k_q}\mathbf{0}\cdots\mathbf{0}\right). \tag{3.40}$$

With a straightforward extension of notation, we set

$$\tilde{\mathbf{J}}_{k_i} = \left.\frac{\partial \mathbf{e}_k(\breve{\mathbf{x}} \boxplus \boldsymbol{\Delta}\tilde{\mathbf{x}})}{\partial \boldsymbol{\Delta}\tilde{\mathbf{x}}_{k_i}}\right|_{\boldsymbol{\Delta}\tilde{\mathbf{x}}=\mathbf{0}} \tag{3.41}$$

With a straightforward extension of the notation, we can insert (3.37) in (3.31) and (3.32). This leads to the following increments:

$$\tilde{\mathbf{H}}\,\boldsymbol{\Delta}\tilde{\mathbf{x}}^* = -\tilde{\mathbf{b}}. \tag{3.42}$$

Since the increments $\boldsymbol{\Delta}\tilde{\mathbf{x}}^*$ are computed in the local Euclidean surroundings of the initial guess $\breve{\mathbf{x}}$, they need to be re-mapped into the original redundant space by the $\boxplus$ operator. Accordingly, the update rule of (2.29) becomes

$$\mathbf{x}^* = \breve{\mathbf{x}} \boxplus \boldsymbol{\Delta}\tilde{\mathbf{x}}^*. \tag{3.43}$$

meaning that we have to carry on a $\boxplus$ for each variable:

$$\mathbf{x}_k^* = \breve{\mathbf{x}_k} \boxplus \boldsymbol{\Delta}\tilde{\mathbf{x}}_k)^*. \tag{3.44}$$

In summary, formalizing the minimization problem on a manifold consists of first computing a set of increments in a local Euclidean approximation around the initial guess by (3.42), and second accumulating the increments in the global non-Euclidean space by (3.43). Note that the linear system computed on a manifold representation has the same structure like the linear system computed on an Euclidean space. One can easily derive a manifold version of a graph minimization from a non-manifold version, only by defining an $\boxplus$ operator and its Jacobian $\tilde{\mathbf{J}}_{k_i}$ w.r.t. the corresponding parameter block.

3.5 Robust Least Squares

Optionally, the least squares optimization can be robustified. Note, that the error terms in Eq. 3.28 have the following form:

$$\mathbf{F}_k = \mathbf{e}_k^T \Omega_k \mathbf{e}_k = \rho_2\left(\sqrt{\mathbf{e}_k^T \Omega_k \mathbf{e}_k}\right) \quad \text{with} \quad \rho_2(x) := x^2. \tag{3.45}$$

Thus, the error vector $\mathbf{e}_k$ has quadratic influence on $\mathbf{F}$, so that a single potential outlier would have major negative impact. In order to be more outlier robust, the quadratic error function ρ_2 can be replaced by a more robust cost function which weighs large errors less. A common robustifier is the Huber cost function ρ_H can be used

$$\rho_H(x) := \begin{cases} x^2 & \text{if } |x| < b \\ 2b|x| - b^2 & \text{else,} \end{cases} \tag{3.46}$$

which is quadratic for small $|x|$ but linear for large $|x|$. Compared to other, even more robust cost functions, the Huber kernel has to advantage that it is still convex and thus does not introduce new local minima in $\mathbf{F}$ [10, pp.616]. In practice, we do not need to modify Eq. 3.28. An alternative is to use

the following scheme: First the error $\mathbf{e}_k$ compute the error as usual. Then, replace $\mathbf{e}_k$ with a weighted version $w_k\mathbf{e}_k$ such that

$$(w_k\mathbf{e}_k)^T\Omega_k(w_k\mathbf{e}_k) = \rho_H\left(\sqrt{\mathbf{e}_k^T\Omega_k\mathbf{e}_k}\right). \tag{3.47}$$

Then calculate the the weights w_k as:

$$w_k = \frac{\sqrt{\rho_H\left(||\mathbf{e}_k||_\Omega\right)}}{||\mathbf{e}_k||_\Omega} \quad \text{with} \quad ||\mathbf{e}_k||_\Omega := \sqrt{\mathbf{e}_k^T\Omega_k\mathbf{e}_k}. \tag{3.48}$$

Chapter 4

Data Association

So far, we always assumed to know which set of state variables was generates an observation. For example, in the point registration exercise, we assumed to know which point in the map $\mathbf{p}_i$ was responsible of the observation $\mathbf{z}_i$. This could be the case when each landmark has a cue that renders it distinguishable from all others. To this end we can think about each landmark having a unique color.

Unfortunately there are not many cases when we can rely on such a strong assumption, and we have to cope with unknown data associations. In practice we construct our associations incrementally, and we use a large set of aggregated information about the scene to figure to which state variable (landmark/point/pose) generated a specific observation.

If we cannot exploit the structure of the domain, this problem is rather hard. A naive approach would be to consider all possible assignments between observations and state variables, solve the corresponding least squares problem and then pick up the one whose data associations results in a more likely estimate.

4.1 Data Association, Bayesian Information Criterion

Imagine we have a state space characterized by the variables $\mathbf{x}^r_{1:n}$, $\mathbf{x}^l_{1:n}$. To prevent your fantasy from throttling you might think to our multi-point registration of Section 3.1.1 example as a concrete case, where the $\mathbf{x}^r_{1:n}$ are the robot poses and the $\mathbf{x}^l_{1:m}$ are the landmark poses. The robot is equipped with a sensor capable of measuring the relative pose of a landmark with respect to its location.

From a pose, let's say $\mathbf{x}^r_i$, the robot will be able to measure all the landmarks that are in the field of view of the sensor $\mathbf{x}^p_{i_1,\dots,i_k}$. Since all landmarks are the same, all what we get from our sensor are a set of measurements $\mathbf{z}^p_{i,1:i,k}$. We do not know which measurement arises from which landmark. In absence of this information we cannot instantiate a least squares problem and all what we have learned so far is of little use for SLAM. Additionally, a landmark might not have been seen by our system, thus the corresponding state variable might not exist.

Solving the data association, in this context consists in finding for each measurement in $\mathbf{z}^p_{i,j}$ the index of the generating landmark. This can be seen as a function $I(i,j) = r$, that given a robot pose index i, a measurement index j tells us the index of the landmark $\mathbf{x}^p_r$ that produced $\mathbf{z}^p_{i,j}$.

Given an hypothesis of associations $I(\cdot)$, we can formulate a least squares problem and solve it by using the means given in the previous sections. A lot of bad things can then happen. It might be that the initial guess of the solution is rather far from the optimum, thus our problem will not converge. Even in the lucky case where we find convergence for all $I(\cdot)$, we still have to rank them and pick up the best.

This can be trivially done by looking at the value of the cost function at the equilibrium. Once again, we encounter a problem. The best solution, the one with a 0 error is the one that explains each observation with a new landmark. This is clearly not correct, because we have presumably much less landmarks than observations.

We then need to find a compromise between the number of parameters in our state (the number of landmarks), and the likelihood of the measurements. This is accounted by the Bayesian Information Criterion (BIC), that is a measure mixing the likelihood and the number of parameters, and is defined as follows:

$$BIC(I) = -2 \ln \hat{L} + k \ln n. \tag{4.1}$$

Here the meaning of the symbols is the following:

- $\hat{L}$: is the likelihood at the maximum, given the data association. Recalling that in our Gaussian world $\ln(\hat{L}) = F(\mathbf{x}^*)$, eases the computation quite a bit. So the lower is our error, for a given data association, the higher is the BIC.
- k: the number of parameters. It is the dimension of our state space, namely the number of landmarks plus the number of poses.
- n: number of data points. In our problem this is the number of observations about a landmark.

The reader might observe that, while the first term of the BIC decreases with the error in an hypothesis, the second term penalizes the number of parameters.

If we want to approach data association in this way, we are trapped in its combinatorial nature. In practice we do have a guess of the robot pose and of the position of the landmarks we have seen in the previous frames, and this renders the problem tractable in most of the cases. Indeed a single association error is sufficient to lead the system to a catastrophic failure. So a good practice is to get rid of the associations we are unsure about.

4.2 Mahalanobis Distance: the easy case

If we have a decent initial guess we might try to pick up for each observation, the landmark that is maximally consistent with it. In our case this consists of:

- compute the marginal distribution of all landmarks in our state $\mathbf{x}^p_{1:m}$, with respect to the current robot position $\mathbf{x}^r_i$. Let $\mathbf{x}^p_j \sim \mathcal{N}(\mathbf{x}^p_j; \mu^*_j, \mathbf{\Sigma}^*_j)$ be this distribution. This can be done by
  - suppressing the row and the columns of the variable $\mathbf{x}^r_i$ in the $\mathbf{H}^*$ matrix.
  - computing the inverse of $\mathbf{\Sigma}^*$ of $\mathbf{H}^*$, and picking up as covariances of $\mathbf{\Sigma}^*_j$ the blocks along the main diagonal corresponding to the landmarks.
- for each landmark, compute Gaussian approximation of the prediction

$$\hat{\mathbf{z}}_j \sim \mathcal{N}(\mathbf{z}_j; \mathbf{h}_i(\mu_j), \mathbf{J}_i \Sigma^*_j \mathbf{J}_i^T + \mathbf{\Omega}^{-1}_{i,j}). \tag{4.2}$$

  Here $\mathbf{J}_i = \frac{\partial \mathbf{h}_i(\mathbf{x}^r_i, \mathbf{x}^p_j)}{\partial \mathbf{x}^p_j}$.
- for each measurement $\mathbf{z}_{i,k}$ and each landmark $\mathbf{x}^p_j$, we evaluate the log likelihood of the measurement, given the prediction as

$$L_{i,k,j} = ||\mathbf{h}_i(\mu_j) - \mathbf{z}_{i,k}||_{\mathbf{J}_i \Sigma^*_j \mathbf{J}_i^T + \mathbf{\Omega}^{-1}_{i,j}}. \tag{4.3}$$

  The smaller the negative log likelihood the better is the association.

At the end of this procedure we have a $n \times m$ matrix whose elements in position k, j are the likelihoods computed above $L_{i,k,j}$. For a measurement we then select the landmark that has the best likelihood (smaller). Some useful heuristics are also of dropping the measurements whose observations are too ambiguous (two landmarks have a very high likelihood). Measurements that are not well explained generate new landmarks that are then added to the state.

Sometimes one rudely gets the associations without computing the covariance matrices of the predictions $\mathbf{J}_i \Sigma^*_j \mathbf{J}_i^T + \mathbf{\Omega}^{-1}_{i,j}$. This is not a very robust solution as it neglects the noise statistics and might produce even worse associations. Indeed it has the advantage of being a lot simpler to implement.

We remark that the above procedure requires a decent initial guess to work. It is well suited for tracking problems, where we know that the robot motion between subsequent poses is small. Paired with a robust kernel can do a pretty good job.

Certain variants of least-squares + data association, interleave the search for data associations to each optimization step. That is, after each iteration, they recompute the associations and run a new iteration of optimization. The popular ICP algorithm for registering point clouds uses this strategy to progressively refine the data association as the optimization gets closer to the minimum.

4.3 RANSAC

In the previous section we learned how to determine the data association under a good initial guess. Bit what if our estimate is so bad that we have no clue about where we are? Can we do something about it. Well, in this case we can use the RANdom SAmple Consensus (RANSAC) principle.

The input of RANSAC is a set of candidate associations between landmarks and measurements. In the worst case this set contains all possible pairings landmark/measurement. Usually one has some appearance based clue to prune this set. This can be done for instance by extracting featres in correspondences of each measurement. A correspondence between a landmark $\mathbf{x}_j^p$ and a measurement $\mathbf{z}_{i,k}$ is returned only if the descriptors of $\mathbf{z}_{i,k}$ is close to some past descriptors of an observation used to qualify the landmark.

RANSAC in its naivest implementation randomly samples a *minimal* set of correspondences $\mathcal{C}$, that is sufficient to determine a transformation $\mathbf{x}_k$ that allows to align the observations with the map. If we are in 2D, and our observations are points, the minimal set of observations is 2. By knowing 2 point correspondences, we can determine an alignment between two frames. If we are in 3D, we require 3 correspondences. Clearly, the minimal number of features depends on the type of observations and on the dimensionality of the problem.

Once we have this transformation, we can compute the prediction of all landmarks in the map. We then consider all the set of correspondences passed as input to the algorithm. Let j, k, be one of such correspondences between the j^{th} landmark and the k^{th} measurement. Let the prediction $\mathbf{h}_i(\mathbf{x}_j)$, be the predicted measurement of the landmark $\mathbf{x}_j^p$, under the pose $\mathbf{x}_k$ computed exploiting the minimal set. If the distance between $\mathbf{z}_{i,k}$ and $\mathbf{h}_i(\mathbf{x}_j)$ is small, then the minimal set is compatible with the correspondence j, k and we call it an *inlier*. In contrast we call it an outlier. The more inlier we have, the better is the solution.

We repeat the above procedure a certain number of times, and at each round we remember the best solution so far.

If we go on long enough and we have some good correspondences in our initial set, the algorithm will converge at a good solution. Let v be the probability of having a wrong correspondence in the input set, m the minimal number of observations to compute $\mathbf{x}_k$, if we want to succeed with probability p, we need to run

$$\frac{\log(1-p)}{\log(1-(1-v)^m)} \tag{4.4}$$

Appendix A

Gaussian Distribution

The gaussian density is described through its parameters, which are mean μ and covariance matrix $\mathbf{\Sigma}$. A remarkable property of the gaussian is that marginalizing, conditioning and affine transformations of a random variable that is normally distributed are still normally distributed.

The pdf of a normal distribution is the following:

$$p(\mathbf{x}) = \mathcal{N}(\mathbf{x}; \mu, \mathbf{\Sigma}) = \frac{1}{(2\pi)^{n/2}\sqrt{\det \mathbf{\Sigma}}} \exp\left(-\frac{1}{2}(\mathbf{x}-\mu)^T \mathbf{\Sigma}^{-1}(\mathbf{x}-\mu)\right), \tag{A.1}$$

where the notation $\mathcal{N}(\mathbf{x}; \mu, \mathbf{\Sigma})$ makes the parameters explicit, and n is the dimension of the random variable $\mathbf{x}$. Instead of parameterizing the Gaussian through its mean and covariance matrix, one can use the information vector ν and the (fisher) information matrix $\mathbf{\Omega}$, where

$$\nu = \mathbf{\Sigma}^{-1}\mu \tag{A.2}$$

$$\mathbf{\Omega} = \mathbf{\Sigma}^{-1}. \tag{A.3}$$

This leads to the following equivalent expression for Eq. A.1:

$$\mathcal{N}^{-1}(\mathbf{x}; \nu, \mathbf{\Omega}) = \frac{\exp\left(\frac{1}{2}\nu^T \mathbf{\Omega}^{-1}\nu\right)\sqrt{\det \mathbf{\Omega}}}{(2\pi)^{n/2}} \exp\left(-\frac{1}{2}\mathbf{x}^T\mathbf{\Omega}\mathbf{x} + \mathbf{x}^T\nu\right). \tag{A.4}$$

These two parameterizations of the gaussian are called respectively: moments parameterization (Eq. A.1 and canonical parameterization (Eq. A.4).

A.1 Partitioned Gaussian Densities

In the remainder of this paragraph, we will discuss how to perform the basic operations on gaussian distribution, that are: *marginalization, conditioning* and *applying an affine transformation*. Without loss of generality, we will consider partitioned gaussian densities, where the random variable is an n-dimensional vector $\mathbf{x}^T = (\mathbf{x}_a^T\ \mathbf{x}_b^T)$ that consists of two subvectors $\mathbf{x}_a^T$ and $\mathbf{x}_b^T$. We can make the components of $\mathbf{x}$ explicit in the parameterizations of the gaussian, as follows:

$$\mathbf{x} = \begin{pmatrix} \mathbf{x}_a \\ \mathbf{x}_b \end{pmatrix}, \quad \mu = \begin{pmatrix} \mu_a \\ \mu_b \end{pmatrix}, \quad \nu = \begin{pmatrix} \nu_a \\ \nu_b \end{pmatrix}, \quad \mathbf{\Sigma} = \begin{pmatrix} \mathbf{\Sigma}_{aa} & \mathbf{\Sigma}_{ab} \\ \mathbf{\Sigma}_{ba} & \mathbf{\Sigma}_{bb} \end{pmatrix}, \quad \mathbf{\Omega} = \begin{pmatrix} \mathbf{\Omega}_{aa} & \mathbf{\Omega}_{ab} \\ \mathbf{\Omega}_{ba} & \mathbf{\Omega}_{bb} \end{pmatrix} \tag{A.5}$$

Since both covariance matrix $\mathbf{\Sigma}$ and the information matrix $\mathbf{\Omega}$ are symmetric, $\mathbf{\Sigma}_{ba} = \mathbf{\Sigma}_{ab}^T$ and $\mathbf{\Omega}_{ba} = \mathbf{\Omega}_{ab}^T$;

A.2 Marginalization of a Partitioned Gaussian Density

Let $\mathbf{x}^T = (\mathbf{x}_a^T\ \mathbf{x}_b^T)$ be a Gaussian random variable such that $\mathbf{x} \sim \mathcal{N}(\mathbf{x}, \mu, \mathbf{\Sigma})$.
The marginal probability $p(\mathbf{x}_a) = \int_{\mathbf{x}_b} p(\mathbf{x}_a, \mathbf{x}_b) d\mathbf{x}_b$ is Gaussian, with parameters

$$p(\mathbf{x}_a) = \mathcal{N}(\mathbf{x}_a; \mu_a, \mathbf{\Sigma}_a). \tag{A.6}$$

The proof of this statement is in [23].

A.3 Conditioning of a Partitioned Gaussian Density

Let $\mathbf{x}^T = (\mathbf{x}_a^T \ \mathbf{x}_b^T)$ be a Gaussian random variable such that $\mathbf{x} \sim \mathcal{N}^{-1}(\mathbf{x}, \mu, \mathbf{\Sigma})$.
The conditional probability $p(\mathbf{x}_a \mid \mathbf{x}_b) = \frac{p(\mathbf{x}_a, \mathbf{x}_b)}{\int_{\mathbf{x}_a} p(\mathbf{x}_a, \mathbf{x}_b) d\mathbf{x}_a}$ is Gaussian with parameters:

$$p(\mathbf{x}_a) = \mathcal{N}(\mathbf{x}_a; \mu_{a|b}, \mathbf{\Sigma}_{a|b}), \tag{A.7}$$

where

$$\mu_{a|b} = \mu_a + \mathbf{\Sigma}_{ab}\mathbf{\Sigma}_{bb}^{-1}(\mathbf{x}_b - \mu_b) \qquad \text{and} \qquad \mathbf{\Sigma}_{a|b} = \mathbf{\Sigma}_{aa} - \mathbf{\Sigma}_{ab}\mathbf{\Sigma}_{bb}^{-1}\mathbf{\Sigma}_{ba}. \tag{A.8}$$

or alternatively, in information form

$$\nu_{a|b} = \nu_a - \mathbf{\Omega}_{ab}\mathbf{x}_b \qquad \text{and} \qquad \mathbf{\Omega}_{a|b} = \mathbf{\Omega}_{aa}. \tag{A.9}$$

The proof of this statement is in [23]. Note that if the conditioning value is the mean of the b component: $\mathbf{x}_b = \mu_b$, we have that $\mu_{a|b} = \mu_a$ and equivalently $\nu_{a|b} = \nu_a$.

A.4 Affine Transformations

Let $\mathbf{x}_a$ be a random variable normally distributed according to $\mathcal{N}(\mathbf{x}_a; \mu_a, \mathbf{\Sigma}_a)$.
Let $\mathbf{A}$ and $\mathbf{c}$ be respectively a matrix and a vector of compatible dimensions, the random variable

$$\mathbf{x}_b = \mathbf{A}\mathbf{x}_a + \mathbf{c} \tag{A.10}$$

Is normally distributed according to $\mathcal{N}(\mathbf{x}_b; \mu_b, \mathbf{\Sigma}_b)$, and has the following parameters:

$$\mu_b = \mathbf{A}\mu + \mathbf{c} \qquad \mathbf{\Sigma}_b = \mathbf{A}\mathbf{\Sigma}\mathbf{A}^T \tag{A.11}$$

A.5 Chain Rule

Let $\mathbf{x}_a$ be a random variable normally distributed according to $p(\mathbf{x}_a) = \mathcal{N}(\mathbf{x}_a; \mu_a, \mathbf{\Sigma}_a)$.
Let $p(\mathbf{x}_b|\mathbf{x}_a)$ be the distribution of another variable $\mathbf{x}_b$, conditioned to $\mathbf{x}_a$, distributed according to

$$p(\mathbf{x}_b|\mathbf{x}_a) = \mathcal{N}(\mathbf{x}_b; \mathbf{A}\mathbf{x}_a + \mathbf{c}, \mathbf{\Sigma}_{b|a}), \tag{A.12}$$

where $\mathbf{A}$ and $\mathbf{c}$ are respectively a matyrix and a vector of appropriate dimensions. If the above assumptions hold, the joint distribution $p(\mathbf{x}_a, \mathbf{x}_b)$ will be still normally distributed according to

$$p(\mathbf{x}_a, \mathbf{x}_b) = \mathcal{N}\left(\mathbf{x}_{a,b}; \mu_{a,b}, \mathbf{\Sigma}_{a,b}\right) \tag{A.13}$$

Eq. A.13, denotes a Gaussian having dimension $\mathrm{Dim}(\mathbf{x}_a) + \mathrm{Dim}(\mathbf{x}_b)$, where

$$\mathbf{x}_{a,b} = \begin{pmatrix} \mathbf{x}_a \\ \mathbf{x}_b \end{pmatrix} \qquad \mu_{a,b} = \begin{pmatrix} \mu_a \\ \mu_b \end{pmatrix} \qquad \mathbf{\Sigma}_{a,b} = \begin{pmatrix} \mathbf{\Sigma}_a & \mathbf{\Sigma}_a\mathbf{A}^T \\ \mathbf{A}\mathbf{\Sigma}_a & \mathbf{\Sigma}_{b|a} + \mathbf{A}\mathbf{\Sigma}_a\mathbf{A}^T \end{pmatrix} \tag{A.14}$$

The information matrix $\mathbf{\Omega}_{a,b} = \mathbf{\Sigma}_{a,b}^{-1}$ is

$$\mathbf{\Omega}_{a,b} = \begin{pmatrix} \mathbf{A}^T\mathbf{\Omega}_{b|a}\mathbf{A} + \mathbf{\Omega}_a & -\mathbf{A}^T\mathbf{\Omega}_{b|a} \\ -\mathbf{\Omega}_{b|a}\mathbf{A}^T & \mathbf{\Omega}_{b|a} \end{pmatrix} \tag{A.15}$$

Bibliography

[1] S. Arumampalam, S. Maskell, N. Gordon, and T. Clapp. A tutorial on particle fiters for on-line non-linear/non-gaussian bayesian tracking. *IEEE Transactions on Signal Provessing*, XX, 2001.

[2] J.A. Castellanos, J.M.M. Montiel, J. Neira, and J.D. Tardós. The SPmap: A probabilistic framework for simultaneous localization and map building. *IEEE Transactions on Robotics and Automation*, 15(5):948–953, 1999.

[3] F. Dellaert and M. Kaess. Square Root SAM: Simultaneous localization and mapping via square root information smoothing. *Int. Journal of Robotics Research*, 25(12):1181–1204, Dec 2006.

[4] A. Doucet, N. De Freitas, and N. Gordon, editors. *Sequential Monte Carlo Methods in Practice*. Springer Verlag, 2001.

[5] R. Eustice, H. Singh, and J.J. Leonard. Exactly sparse delayed-state filters. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, pages 2428–2435, Barcelona, Spain, 2005.

[6] R. Eustice, M. Walter, and J.J. Leonard. Sparse extended information filters: Insights into sparsification. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 641–648, Edmonton, Cananda, 2005.

[7] N. Gordon, D. Salmond, and C Ewing. A novel approach to nonlinear non gaussian bayesian estimation. In *In IEEE Proceedings, part F*, pages 107–113, 1993.

[8] G. Grisetti, C. Stachniss, and W. Burgard. Improved techniques for grid mapping with rao-blackwellized particle filters. *IEEE Trans. on Robotics*, 23(1):34–46, 2007.

[9] D. Hähnel, W. Burgard, D. Fox, and S. Thrun. An efficient FastSLAM algorithm for generating maps of large-scale cyclic environments from raw laser range measurements. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 206–211, Las Vegas, NV, USA, 2003.

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

[11] C. Hertzberg. A framework for sparse, non-linear least squares problems on manifolds. Master's thesis, Univ. Bremen, 2008.

[12] S Julier, J. Uhlmann, and H. Durrant-White. A new approach for filtering nonlinear systems. In *Proceedings of the American Control Conference*, pages 1628–1632, 1995.

[13] R.E. Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engeneering*, 1960.

[14] K. Konolige. A gradient method for realtime robot control. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2000.

[15] K. Konolige, J. Bowman, J. D. Chen, P. Mihelich, M. Calonder, V. Lepetit, and P. Fua. View-based maps. *International Journal of Robotics Research (IJRR)*, 29(10), 2010.

[16] K. Konolige, G. Grisetti, R. Kümmerle, W. Burgard, B. Limketkai, and R. Vincent. Sparse pose adjustment for 2d mapping. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2010.

[17] J.M. Lee. *Introduction to Smooth Manifolds*. Graduate Texts in Mathematics. Springer, 2002.

[18] J.M. Lee. *Introduction to Smooth Manifolds*, volume 218 of *Graduate Texts in Mathematics*. Springer Verlag, 2003.

[19] F. Lu and E. Milios. Globally consistent range scan alignment for environment mapping. *Autonomous Robots*, 4:333–349, 1997.

[20] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. FastSLAM: A factored solution to simultaneous localization and mapping. In *Proc. of the National Conference on Artificial Intelligence (AAAI)*, pages 593–598, Edmonton, Canada, 2002.

[21] E. Olson, J. Leonard, and S. Teller. Fast iterative optimization of pose graphs with poor initial estimates. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, pages 2262–2269, 2006.

[22] M. Pitt and N. Shephard. Filtering via simulation: Auxiliary particle filters. *Journal of the American Statistical Association*, 94(446):590–599, 1999.

[23] T. Schoen and F. Lindsten. Manipulating the multivariate gaussian density. Technical report, Linkoeping University, 2011. http://www.rt.isy.liu.se/ schon/Publications/SchonL2011.pdf.

[24] R. Smith, M. Self, and P. Cheeseman. Estimating uncertain spatial realtionships in robotics. In I. Cox and G. Wilfong, editors, *Autonomous Robot Vehicles*, pages 167–193. Springer Verlag, 1990.

[25] S. Thrun, Y. Liu, D. Koller, A.Y. Ng, Z. Ghahramani, and H. Durrant-Whyte. Simultaneous localization and mapping with sparse extended information filters. *Int. Journal of Robotics Research*, 23(7/8):693–716, 2004.

[26] R. Triebel, P. Pfaff, and W. Burgard. Multi-level surface maps for outdoor terrain mapping and loop closing. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2006.

[27] R. van der Merwe, N. de Freitas, A. Doucet, and E. Wan. The unscented particle filter. Technical Report CUED/F-INFENG/TR380, Cambridge University Engineering Department, August 2000.

[28] G. Welch and G. Bishop. An introduction to the kalman filter. Technical report, University of North Carolina at Chapel Hill, Chapel Hill, NC 27599-3175, 2001.