SAPIENZA
UNIVERSITÀ DI ROMA

# Extended Measurements in Pose Graph Optimization

Master thesis

Irvin Aloise

aloise.1392066@studenti.uniroma1.it

RoCoCo - Cognitive robot teams laboratory
Department of Computer, Control and Management Engineering Antonio Ruberti
Sapienza University of Rome

**Supervisor:**
Prof. Dr. Giorgio Grisetti
**Cosupervisor:**
Eng. Dominik Schlegel

October 8, 2017

# Abstract

The goal of this Master's Thesis is the design and development of a graph-optimization system tailored for SLAM pipelines. The work focuses on simplicity, accuracy and speed, reaching excellent results in each of the targeted fields.

In particular, this back-end aims attention at 3D pose-graphs and 3D Bundle Adjustment problems, producing outcomes comparable to current state-of-the-art systems like `g2o` [1].

At its core there is a novel error function for $SE(3)$ objects based on the concept of matrices' chordal distance. This approach reduces the problem's non-linearity bringing several benefits to the computation. The fastness of the system is due to a well-designed implementation that performs zero memory copy during the iterative part and that exploits SIMD instructions of modern CPUs and a smart use of the CPU cache.

Finally, the system has been tested on both synthetic and real-world datasets and in both scenarios it succeeded in its purposes, being able to produce results better or comparable to state-of-the-art systems in only 5 thousands lines of code.

# Nomenclature

## Notation

| | |
|---|---|
| ${}^{A}\mathbf{R}_{B}(\alpha, \beta, \gamma)$ | Rotation from frame A to frame B expressed in A |
| ${}^{A}\mathbf{t}_{B}(t_x, t_y, t_z)$ | Translation from frame A to frame B expressed in A |
| ${}^{A}T_{B}(R, \mathbf{t})$ | Transformation from frame A to B expressed in A |
| ${}^{A}\tau_{B}(R, \mathbf{t})$ | Transformation from frame A to B expressed in A (vectorized) |
| $\lfloor \mathbf{t} \rfloor_{\times}$ | Skew-symmetric matrix of a vector $\mathbf{t} \in \mathbb{R}^{3\times3}$ |

## Acronyms and Abbreviations

| | |
|---|---|
| SLAM | Simultaneous localization and mapping |
| SfM | Structure from Motion |
| BA | Bundle Adjustment |
| SCLAM | Simultaneous Calibration Localization and Mapping |
| MAP | Maximum A Posteriori |
| LS | Least Squares |
| PSD | Positive Semi-Definite |
| PDF | Probability Distribution Function |
| Landmark | Salient point in the world (2D or 3D) |
| Feature | Specific structural part of interest in an image (2D) |
| Tracking | Procedure of spatial and temporal re-identification of landmarks |
| Loop closure | Interconnection between landmarks in space |
| GPS | Global Positioning System (generally referring to whole unit) |
| LiDAR | Light Detection and Ranging o Laser Imaging Detection and Ranging |
| Sonar | SOund Navigation And Ranging |
| FLOP | Floating Point OPeration |
| FLOPS | Floating Point Operations Per Second |
| OpenCV | Open source Computer Vision (library), `www.opencv.org` |
| Git | Git revision control, `www.git-scm.com` |

# Contents

# List of Figures

# List of Tables

Chapter 1

# Introduction

Mobile robots, in order to accomplish tasks in real world more easily and in an efficient way, need to have a map of the environment and to localize themselves into the map. Furthermore, in some environment it is not possible to rely on external reference systems - e.g. GPS - and, thus, they can count only on on-board sensors. *Simultaneous Localization and Mapping* (SLAM) addresses the problem of **learning the map under pose uncertainty**.

There are many scenarios in which SLAM is fundamental for the accomplishment of a task, not only in pure Robotics. SLAM, in fact, is a common problem in different domains of application. For example, in Robotics it is fundamental for indoor navigation of mobile robots - e.g. an autonomous vacuum cleaner or a service robot in a museum - or to navigate through extreme environments - e.g. underwater rescues or space exploration. Additionally, new technologies that involve different kind of agents - i.e. not robots - are now using SLAM. One of the most trending is *Augmented Reality* (AR). Always more powerful mobile devices - like smartphones or tablets - are now able to exploit SLAM to deliver stunning virtual experiences. Without any doubts, this technology is going to gain always more popularity and to impact on the research in this topic.

(A)

(B)

Figure 1.1: **Application Examples.** The image in 1.1A represents the well-known Roomba Autonomous vacuum cleaner, which is able to recognize where it is in the room, if it has already cleaned a place or if it is going toward a dangerous path - e.g. stairs; image 1.1B depicts an AR mobile game developed using the Apple ARKit.

As the reader might notice, SLAM is a popular problem and the research community is focusing on it since many years. Several solutions have been proposed through the years and now current state-of-the-art SLAM systems are able to deliver impressive results in real-world scenarios. The most used formulation for the SLAM problem is

the so-called *graph-based SLAM*. In this approach, two sub-systems cooperate with each other to retrieve the best robot trajectory and world configuration given the on-board sensors' measurements. Therefore, the full slam system is composed by:

1. *Front-end*: it exploits sensor data to build an hyper-graph whose nodes are either robot poses or the position of salient points in the world;
2. *Back-end*: it is in charge of performing non-linear optimization of the graph to retrieve the most likely configuration that suits the measurements.

In this work, we propose a back-end system built from scratch that is able to perform fast and accurate graph optimization for 3D environments. The work is focused on simplicity and minimalism also in its implementation, in order to be comprehensible by non-expert people that want to understand how the system works. Despite its minimalistic fashion, system's results are comparable - or even better in some scenarios - to the ones of other state-of-the-art systems, thanks to the use of some novel theoretic ideas and to a well-designed implementation. In particular, this work shows the effectiveness of a **new error function** for $SE(3)$ objects (Section 5.3) and an implementation with **zero memory copy** during the optimization process (Section 6.3).

The remaining of this document is organized as follows:

- Chapter 2: overview of the problem and of methodologies employed through the years, with a particular focus on noteworthy systems;
- Chapter 3: problem statement and fundamental theoretic concepts related to the non-linear optimization problem;
- Chapter 4: sketch of the most common SLAM problem formulations;
- Chapter 5: deeper examination of 3D formulations and further analysis of the proposed approach;
- Chapter 6: details about code design and implementation choices;
- Chapter 7: focus on two full SLAM systems that uses the proposed system as on-line back-end;
- Chapter 8: final considerations and possible future investigations.

Chapter 2

# Related Work

Simultaneous Localization and Mapping represents a well known complex mathematical problem, based on non-linear optimization. It has been studied by the scientific community since the 80s [3] [4]; during this early stage, its statistical formulation has been investigated, proposing interesting results that will constitute, basically, the baseline for all the future SLAM systems.

After some years, in the 90s, the community came-up with early solutions for the SLAM problem. The first systems able to produce appreciable results in terms of speed and accuracy were based on *Extended Kalman Filters* (EKF) [5] [6]. EKFs allow to deal with problem's non-linearity through effective approximations and to represent multivariate distributions with a small number of parameters. This success encouraged the research community to perform deeper investigations in *filtering* approaches [7]. *Particle filters* started to gain popularity, in particular *Rao-Blackwellized Particle Filters* [8]: the work of Montemerlo *et al.* [9] was the first SLAM system able to deal with thousand of landmarks with a good accuracy.

However, *filtering* approaches revealed not to be the best answer to the SLAM problem due to the computational complexity of the solution, especially when dimensions start to grow. Moreover, system's accuracy is affected by the problem's non-linearities, leading to sub-optimal solutions. For these reasons, *Maximum A Posteriori* (MAP) approaches started to be taken in consideration and the community took a step back to the work of Lu *et al.* [10]. Filtering-based approaches align local pose frames incrementally and, thus, different parts of the model are updated independently, generating inconsistencies in the final model. MAP optimization takes into consideration all the local frames and the relations between them at once, leading to a more consistent model and better accuracy. Lu *et al.* embedded all the pose relations into a network with nodes and edges, allowing efficient optimization. However, when they published their work, the computers' computational power was not sufficient to deliver good performance and, thus, this solution was put aside.

Nevertheless, this work represents the precursor to one the most intuitive SLAM formulation, called *graph-based SLAM*, that exploits the computing power of recent robots to deliver impressive performances. In this paradigm, the robot builds an *hyper-graph* whose nodes represent either robot poses or salient points in the world - called *landmarks* - while the hyper-edges encode sensors' measurements between subsets of nodes.

Graph-based SLAM systems have two main components: *front-end* and *back-end*.

The former uses data acquired by robot's sensors to populate the hyper-graph, abstracting raw data into a model that is amenable for optimization. The front-end has to determine the most likely constraint that involves a subset of nodes given an incoming measurement, solving the so-called *data association* problem - *short-term* and *long-term*. Short-term data association has to match corresponding features among consecutive sensor measurements - e.g. stating that visual features detected in multiple consecutive frames represent the same 3D world point. Long-term data association, instead, expresses a more complex problem: it has to associate new measurements to already encountered world points, generating the so-called *loop-closures* - e.g. when a robot passes multiple times across the same place, it has to recognize that it is re-observing the same points in order to generate a map that is consistent with the environment. As for the sensors used, state-of-the-art systems usually acquire data from cameras (RGB or RDB-D) or 3D-LiDARs. The former, in particular, it is gaining much attention from the research community since they are - generally - cheap and can be mounted basically on every electronic device in single or stereo configuration.

This work, however, focuses on system's back-end, assuming that the given front-end provides consistent estimates. The back-end takes as input the graph and computes the most-likely map given all the constraints. Systems based on this formulation represent the gold standard for map optimization, thus, in the next sections it is proposed a brief overview of the most successful implementations.

## Dense Approaches

The work of Lu *et al.* [10] was the first of its kind: map estimation is obtained through global optimization of the error function deriving from constraints between different poses. They employed a combination of *relation-based* and *location-based* representations, where the former were fixed while the latter were treated as free variables. Those pose-relations were used to construct a network whose nodes were robot poses taken from its trajectory while the constraints between nodes were the pose-relations. Finally, the optimization problem exploits the network to obtain an objective function that will be minimized: the total energy will decrease as the difference between estimated relative pose that involves two nodes and the measured value tends to zero.

It is good to notice that, in this formulation, the computational power needed for the optimization grows *cubically* with the number of variables involved, thus, it is $O(N^3)$ where $N$ represents the number of poses. Gutmann and Konolige addressed this problem in their work [11] proposing a method to incrementally build the network and that determines topologically correct relations between poses.

Those approaches opened the path to a series of study in this direction, that will lead to current state-of-the-art optimizers.

## Olson's Gradient Descent

Evident limitations of the previously seen approaches are that their solution highly depends from the initial estimate of the state. The initial guess is derived from dead-reckoning and, thus, if this is not consistent the system will converge to a local minimum, giving a sub-optimal solution. Olson *et al.* [12] addressed this issue, proposing a non-linear optimization algorithm that quickly converges to a good approximation of the global minimum.

They achieved such results combining two new aspects: the first one is the use of a variant of *Stochastic Gradient Descent* algorithm, which is robust against local minima and has a fast convergence rate; the second one is an *alternative state-space representation* that has good stability and computational properties. This last feature, in particular, allows to update many poses with a relatively small computational cost in a single iteration. Moreover, the memory consumption has been lowered together with the run time - respectively $O(N + M)$ and $O(\log(N))$, where $N$ represents the number of poses and $M$ the number of constraints.

## Smoothing and Mapping

Smoothing and Mapping, shortened as *SAM*, follows the path of global trajectory optimization described in the previous Section. Dellaert *et al.* proposed with *square root SAM* ($\sqrt{SAM}$) [13] a system able to deal with *full SLAM problems*: those require the estimation of the entire set of sensor poses along with the parameters of all the features in the environment. This problem is also known in Photogrammetry as *bundle adjustment* and as *structure from motion* in Computer Vision.

$\sqrt{SAM}$ performs fast optimization exploiting the problem's intrinsic sparsity. Knowing that the measurement Jacobian matrix $A$ is sparse, it is possible to solve the relative linear system in a faster way through a good *variable reordering* together with *QR* or *Cholesky* factorization. For those reasons, $\sqrt{SAM}$ can optimize larger graph without losses in terms of performances or accuracy.

Further improvements were introduced with *iSAM* - *incremental* SAM - developed by Kaess *et al.* [14]. The foundations were the same as $\sqrt{SAM}$, but in this case the system operates incrementally, without the need of fully refactoring the whole QR-decomposition, but updating it every time a new measurement is available. With this solution, it was possible to address real-time problems since the optimization process is faster than before.

The next iteration of this branch of solutions, is represented by *iSAM2* [15]. In this case, a new data-structure is proposed, the *Bayes tree*, to map better the square root information matrix of the problem. Employing Bayes trees, the algorithm is able to further improve the performances, exploiting incremental variable re-ordering and fluid relinearization, eliminating the need for periodic batch steps.

## TORO

Grisetti *et al.* proposed in their work TORO [16] - Tree-based netwORk Optimizer - an extension of Olson's algorithm to efficiently manage 2D and 3D graph-based optimization problems.

This frameworks has several features that allow it to achieve good performances in terms of speed and accuracy. The first one is a revisited version of the standard *Stochastic Gradient Descent* used to perform the optimization process, together with a technique to efficiently distribute the *rotational error* over a sequence of 3D poses [17]. In fact, due to the non-commutativity of rotational angles in 3D, major problems may arise when applying approaches that are designed for a 2D world. As a result, TORO converges by orders of magnitude faster with respect to previous approaches.

Moreover, it employs a *tree parametrization of the nodes* [18] that significantly improves the performances and allows to deal with arbitrary network topologies. This consented the authors to bound algorithm complexity to the size of the mapped area and not to the trajectory's length, yielding accurate maps of the environment in a small amount of time.

## G2O

The work of Kümmerle *et al.* [1] is an open-source C++ frameworks for optimizing *non-linear least squares problems* that can be represented as a graph. Its generality together with a cross-platform implementation made `g2o` one of the most successful graph optimization tool, which is employed in many state-of-the-art full SLAM systems.

This work focuses on efficiency, which is achieved at various levels: it exploits graph sparsity and takes advantage of the graph's special structure to perform fast optimization; it uses advanced methods - Cholesky decomposition through CHOLMOD library - to solve sparse linear system; finally, it utilizes modern processor's features to perform fast math operation optimizing cache usage - e.g. SIMD instructions. Moreover, this frameworks offers the possibility to choose between different algorithms - i.e. Gauss-Newton, Levenberg-Marquardt - and linear solvers - direct and iterative.

Our approach still delivers comparable performances with respect to `g2o` while being lightweight and more simple to include in a full SLAM pipeline. In fact, `g2o` is a big framework, with over 40 thousands lines of code and, thus, its inclusion may add weight to the system.

## GT-SAM

GT-SAM is a C++ library developed by Dellaert *et al.* [19] at the Georgia Institute of Technology, which provides solution to a wide range of SLAM and Structure From

Motion (SfM) problems, based on factor-graph optimization. It provides both C++ and MATLAB implementations that allow to easily develop and visualize problem solutions.

This work - like it has been previously seen in `g2o` - also focuses on efficient optimization and takes advantage of the graph sparsity to deliver fast and accurate performances. However, this framework is even more big and complex - over 300 thousands lines of code - making it very difficult to unravel and understand what is under the hood.

## HOG-Man

Grisetti *et al.* in their work [2] proposed an optimization system designed for accurate, fast and memory efficient on-line operations: *HOG-Man* - which stands for Hierarchical Optimization on Manifolds.

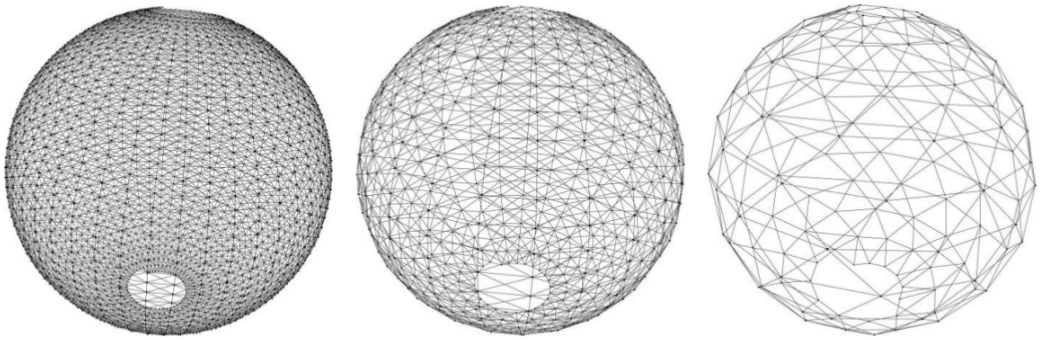


Figure 2.1: **HOG-Man.** The image - courtesy of [2] - sketches the idea behind this approach: the systems creates multiple "views" of the graph's structure, each with a different level of abstraction. Proceeding from left to right it is shown the original structure - i.e. the bottom of the hierarchy - a mid level representation and the final structure - i.e. the top of the hierarchy.

At its core there is a *hierarchical* approach to graph optimization: during on-line mapping, it optimizes only the coarse structure of the environment and not the whole map. The simplified problem that is solved, however, contains all the relevant information to let the front-end solve properly the data association problem.

It is good to notice, that there are *different level of abstractions*: the bottom is the original input, while higher levels are always more compact. When the top levels are modified, only portions of the underlying ones are updated, namely the ones subject to consistent modifications. This method limits the computational power needed for on-line operations while preserving global consistency, outperforming several previous approaches.

## Tectonic-SAM

Ni *et al.* propose a way to reduce the computational effort due to the linearization update, based on sub-map partitioning: *Tectonic SAM* [20] - shortened as *T-SAM*.

The original problem is addressed through a *divide and conquer* approach which produces local sub-maps. Those smaller maps are individually optimized and then the local linearization can be cached and reused when sub-maps are combined into a global map, speeding up the linearization process. Sub-maps have a base node that capture their global position and the authors showed that, under *mild assumptions*, this approach leads to the exact solution.

The next iteration - called T-SAM2 [21] - proposes an algorithm that partitions the SLAM factor-graph into a sub-map tree, performing the optimization from leaves to root. In T-SAM, partitioning was done using edge separators; moreover, T-SAM was not able to maintain hierarchical maps, leading to poor scalability with respect to larger datasets. All those problems were addressed in T-SAM2, where partitioning is done employing the *nested dissection algorithm* that, together with a novel *multi-level* approach, provides a more efficient and robust exact solution.

## Condensed Measurements

The solution of least-squares problems that can be represented as factor-graph - like in SLAM and SfM - is contingent to both *initial estimate* and *sensor models' non-linearities*. Grisetti *et al.* proposed a way to enlarge the convergence basin based on the *divide and conquer* approach that exploits the factor-graph's structure [22].

The core of this formulation is to divide the graph into *small locally connected sub-graph*, each of which represents a sub-problem that can be robustly solved - like it has been suggested in previous systems like HOG-Man [2] and T-SAM [21]. In order to consistently combine the local sub-graphs, the authors build a simple factor-graph from the sub-graphs, constraining the relative positions of the variables in the solution. For this reason, the sub-graphs are called *condensed measurements*. The resulting problem is more convex with respect to the original one and, thus, there are more chances of finding the correct minimum that can be used as initial guess for standard minimization algorithms - e.g. Levenberg-Marquardt.

This formulation allows to recover from bad initial estimations, where most of the other approaches fail - both batch and direct ones - but, unfortunately, it does not deliver real-time performances.

Chapter 3

# Basics

The goal of this chapter is to introduce the reader to the mathematical fundamentals underlying the system developed. Obviously, it will be a brief overview, therefore, references to literature are provided if the reader would like to go more in detail with the proposed concepts.

## Least Square SLAM

In this section it proposed an insight of least-squares state estimation of non-linear stationary systems [23].

Suppose to have a stationary system $\mathcal{W}$ whose state is parametrized by a set of $N$ non-observable **state** variables $\mathbf{x} = \{\mathbf{x}_1, ..., \mathbf{x}_N\}$. Suppose that it is possible to indirectly observe the system state with different generic sensors, those will generate a set of $K$ **measurements** represented by $\mathbf{z} = \{\mathbf{z}_1, ..., \mathbf{z}_K\}$, where $\mathbf{z}_k$ is intended to be the $k^{th}$ measurement. Since the measurements are affected by noise, those are assumed to be **random variables**. Moreover, the state embeds all the knowledge needed to predict the measurements' distribution.

Since measurements are affected by noise, it is impossible to compute the state given the measurements. What is possible to evaluate, instead, is the states' distribution known the measurements, which can be formalized as following *conditional probability*:

$$\begin{aligned} p(\mathbf{x}|\mathbf{z}) &= p(\mathbf{x}_1, ..., \mathbf{x}_N|\mathbf{z}_1, ..., \mathbf{z}_K) = \\ &= p(\mathbf{x}_{1:N}|\mathbf{z}_{1:K}) \end{aligned} \tag{3.1}$$

The probability distribution 3.1 is complex to retrieve in close form, for several reasons:

- The mapping between measurements and states can be highly non-linear, producing a multi-modal probability distribution with a complex shape.
- Each measurement $\mathbf{z}_k$ in general observes only a subset of the state parameters. Moreover, the number of measurements may not be sufficient to fully characterize the state distribution.

- Measurements can be wrong - generating outliers - or it is impossible to map any of the state variable to a specified measurement.

However, what is possible to compute more easily is an estimate of the probability 3.1. To do so, we analyze the conditional distribution $p(\mathbf{z}_k|\mathbf{x})$: this is a predictive distribution called *sensor model* or *observation model*, which formalizes the probability of having a certain measurement *assuming to know system's state*. Extending this to all the measurements, you will get the following distribution:

$$p(\mathbf{z}|\mathbf{x}) = p(\mathbf{z}_{1:K}|\mathbf{x}_{1:N}) \tag{3.2}$$

As it has been stated before, the state fully describes the measurements, rendering the single distributions $p(\mathbf{z}_k|\mathbf{x})$ independent from each other. Exploiting this feature, it is possible to rewrite the 3.2 as follows:

$$p(\mathbf{z}_{1:K}|\mathbf{x}_{1:N}) = \prod_{k=1}^{K} p(\mathbf{z}_k|\mathbf{x}_{1:N}) \tag{3.3}$$

The equation 3.3 describes the measurements' *likelihood* given the state. Recalling the Bayes rule [24] and applying it to 3.1 you will obtain the following relation:

$$\begin{aligned} p(\mathbf{x}_{1:N}|\mathbf{z}_{1:K}) &= \frac{\overbrace{p(\mathbf{z}_{1:K}|\mathbf{x}_{1:N})}^{likelihood}\overbrace{p(\mathbf{x}_{1:N})}^{prior}}{\underbrace{p(\mathbf{z}_{1:K})}_{normalizer}} = \\ &= \frac{\prod_{k=1}^{K} p(\mathbf{z}_k|\mathbf{x}_{1:N})\, p_x}{p_z} = \\ &= \eta_z p_x \prod_{k=1}^{K} p(\mathbf{z}_k|\mathbf{x}_{1:N}) \end{aligned}$$

In this relation, $p(\mathbf{x}_{1:N})$ represents our prior knowledge about the state distribution and, thus, supposing to know nothing about it, it is represented by a uniform distribution whose value is a constant $p_x$. $p(\mathbf{z}_{1:K})$ instead is just a normalizer for the overall probability function and does not depend from the states, therefore it is assumed to be a constant $p_z$. This leads to the following relation:

$$p(\mathbf{x}_{1:N}|\mathbf{z}_{1:K}) \propto \prod_{k=1}^{K} p(\mathbf{z}_k|\mathbf{x}_{1:N}) \tag{3.4}$$

Equation 3.4 represents the core of the entire least-square formulation. This will be exploited in the next subsections to approximate the distribution of interest, minimizing a defined cost function.

### Direct Minimization

Starting from the relation 3.4 it is possible to initialize a minimization problem. Assuming that the measurement are affected by *Additive White Gaussian Noise*, the observation model probability $p\left(\mathbf{z}_k|\mathbf{x}_{1:N}\right)$ will be described by a Gaussian distribution $\mathcal{N}(\mu, \Omega^{-1})$, leading to the equation

$$p\left(\mathbf{z}_k|\mathbf{x}_{1:N}\right) \propto \exp\left(-(\hat{\mathbf{z}}_k - \mathbf{z}_k)\Omega_k(\hat{\mathbf{z}}_k - \mathbf{z}_k)\right) \tag{3.5}$$

where $\hat{\mathbf{z}}_k$ is the **prediction** of the measurement given the state, while $\Omega_k = \Sigma_k^{-1}$ represents conditional measurement's information matrix. The predicted measurement $\hat{\mathbf{z}}_k$ is a function of the state; in particular it is obtained applying the **sensor model** $h_k(\cdot)$ to the state, in formulæ:

$$\hat{\mathbf{z}}_k = h_k(\mathbf{x}) \tag{3.6}$$

In SLAM - and other similar problems like SfM - the sensor model is a highly non-linear function, making the problem more complex and heavy from a computational point of view. Nevertheless, generally the sensor model is smooth enough to be approximated with its *first-order Taylor expansion* in the neighbor of a linearization point $\breve{\mathbf{x}}$, leading to:

$$h_k(\breve{\mathbf{x}} + \Delta\mathbf{x}) \approx h_k(\breve{\mathbf{x}}) + \left.\frac{\partial h_k(\mathbf{x})}{\partial \mathbf{x}}\right|_{\mathbf{x}=\breve{\mathbf{x}}} \Delta\mathbf{x} = h_k(\breve{\mathbf{x}}) + \mathbf{J}_k\Delta\mathbf{x} \tag{3.7}$$

where $\mathbf{J}_k = \left.\frac{\partial h_k(\mathbf{x})}{\partial \mathbf{x}}\right|_{\mathbf{x}=\breve{\mathbf{x}}}$ is the *Jacobian* evaluated in $\mathbf{x} = \breve{\mathbf{x}}$.

The next step consists in finding a linearization point $\mathbf{x}^\star$ that *maximizes the observation model*, leading to the following relations:

$$\begin{aligned}
\mathbf{x}^\star &= \underset{\mathbf{x}}{\operatorname{argmax}}\, p(\mathbf{z}|\mathbf{x}) = \\
&= \underset{\mathbf{x}}{\operatorname{argmax}} \prod_{k=1}^{K} p(\mathbf{z}_k|\mathbf{x}) = \\
&= \underset{\mathbf{x}}{\operatorname{argmax}} \prod_{k=1}^{K} \exp\left(-(h_k(\mathbf{x}) - \mathbf{z}_k)^T\Omega_k(h_k(\mathbf{x}) - \mathbf{z}_k)\right) = \\
&= \underset{\mathbf{x}}{\operatorname{argmin}} \sum_{k=1}^{K} \left((h_k(\mathbf{x}) - \mathbf{z}_k)^T\Omega_k(h_k(\mathbf{x}) - \mathbf{z}_k)\right)
\end{aligned}$$

The relation $\mathbf{e}_k(\mathbf{x}) = h_k(\mathbf{x}) - \mathbf{z}_k$ represents the *error function*, and, thus, the optimal linearization point will be given by the minimization of the following *cost function*:

$$F(\mathbf{x}) = \sum_{k=1}^{K} \underbrace{\mathbf{e}_k^T(\mathbf{x})\Omega_k\mathbf{e}_k(\mathbf{x})}_{\mathbf{e}_k(\mathbf{x})} \tag{3.8}$$

In order to find the optimum linearization point, the system must start from a reasonable initial guess $\breve{\mathbf{x}}$ - to avoid local minima - and apply an increment $\Delta\mathbf{x}$ directed toward $\mathbf{x}^\star$. Applying the perturbation $\Delta\mathbf{x}$ in the error function and approximating again through the first-order Taylor expansion we will obtain

$$\begin{aligned}
\mathbf{e}_k(\breve{\mathbf{x}} + \Delta\mathbf{x}) &= \big(h_k(\breve{\mathbf{x}} + \Delta\mathbf{x}) - \mathbf{z}_k\big)^T\Omega_k\big(h_k(\breve{\mathbf{x}} + \Delta\mathbf{x}) - \mathbf{z}_k\big) = \\
&\approx (\mathbf{J}_k\Delta\mathbf{x} + h_k(\breve{\mathbf{x}}) - \mathbf{z}_k)^T\Omega_k(\mathbf{J}_k\Delta\mathbf{x} + h_k(\breve{\mathbf{x}}) - \mathbf{z}_k) = \\
&= (\mathbf{J}_k\Delta\mathbf{x} + \mathbf{e}_k(\breve{\mathbf{x}}))^T\Omega_k(\mathbf{J}_k\Delta\mathbf{x} + \mathbf{e}_k(\breve{\mathbf{x}}))
\end{aligned} \tag{3.9}$$

Further expanding the quantities in the equation 3.9, it is possible to obtain the following relation:

$$\begin{aligned}
\mathbf{e}_k(\breve{\mathbf{x}} + \Delta\mathbf{x}) &\approx \Delta\mathbf{x}^T\overbrace{\mathbf{J}_k^T\Omega_k\mathbf{J}_k}^{H_k}\Delta\mathbf{x} + 2\,\overbrace{\mathbf{J}_k^T\Omega_k\mathbf{e}_k(\breve{\mathbf{x}})}^{\mathbf{b}_k}\Delta\mathbf{x} + \overbrace{\mathbf{e}_k^T(\breve{\mathbf{x}})\Omega_k\mathbf{e}_k(\breve{\mathbf{x}})}^{\mathbf{c}_k} = \\
&= \Delta\mathbf{x}^T H_k\Delta\mathbf{x} + 2\,\mathbf{b}_k\Delta\mathbf{x} + \mathbf{c}_k
\end{aligned} \tag{3.10}$$

Extending the perturbation to the total cost function expressed in equation 3.8 and plugging what stated in 3.10 we will obtain:

$$\begin{aligned}
F(\breve{\mathbf{x}} + \Delta\mathbf{x}) &\approx \sum_{k=1}^{K}\left[\Delta\mathbf{x}^T H_k\Delta\mathbf{x} + 2\,\mathbf{b}_k\Delta\mathbf{x} + \mathbf{c}_k\right] = \\
&= \Delta\mathbf{x}^T\underbrace{\left[\sum_{k=1}^{K} H_k\right]}_{\mathbf{H}}\Delta\mathbf{x} + 2\underbrace{\left[\sum_{k=1}^{K}\mathbf{b}_k\right]}_{\mathbf{b}}\Delta\mathbf{x} + \underbrace{\sum_{k=1}^{K}\mathbf{c}_k}_{\mathbf{c}} = \\
&= \Delta\mathbf{x}^T\mathbf{H}\Delta\mathbf{x} + 2\,\mathbf{b}\Delta\mathbf{x} + \mathbf{c}
\end{aligned} \tag{3.11}$$

It is good to notice that equation 3.11 represents a *quadratic form* in $\Delta\mathbf{x}$. Thus, finding the minimum of this formula will give us the optimal perturbation $\Delta\mathbf{x}$ such that

$$\mathbf{x}^\star = \breve{\mathbf{x}} + \Delta\mathbf{x}$$

In order to find the minimum of equation 3.11 we derive it in $\Delta\mathbf{x}$, we equal the derivative to zero and finally we solve the resulting equation for $\Delta\mathbf{x}$; in formulæ:

$$\frac{\partial \left(\Delta \mathbf{x}^T \mathbf{H} \Delta \mathbf{x} + 2\,\mathbf{b} \Delta \mathbf{x} + \mathbf{c}\right)}{\partial \Delta \mathbf{x}} = 2\,\mathbf{H} \Delta \mathbf{x} + 2\,\mathbf{b} = 0 \tag{3.12}$$

Therefore, in order to find the solution of equation 3.12 and finally get to the optimal perturbation, we must solve the following *linear system*:

$$\mathbf{H}\,\Delta \mathbf{x}^\star = -\mathbf{b} \tag{3.13}$$

**Algorithm 1** Standard Gauss-Newton minimization algorithm

**Require:** Initial guess $\breve{\mathbf{x}}$; a set of measurements $\mathcal{C} = \{\langle \mathbf{z}_k, \Omega_k \rangle\}$
**Ensure:** Optimal solution $\mathbf{x}^\star$

$F_{new} \leftarrow \breve{F}$ ▷ compute the current error
**while** $\breve{F} - F_{new} > \epsilon$ **do**
  $\breve{F} \leftarrow F_{new}$
  $\mathbf{b} \leftarrow 0$
  $\mathbf{H} \leftarrow 0$
  **for all** $k \in \{1, ..., K\}$ **do**
    $\hat{\mathbf{z}}_k \leftarrow h_k(\breve{\mathbf{x}})$ ▷ compute prediction
    $\mathbf{e}_k \leftarrow \hat{\mathbf{z}}_k - \mathbf{z}_k$ ▷ compute error
    $\mathbf{J}_k \leftarrow \left.\frac{\partial h_k(\mathbf{x})}{\partial \mathbf{x}}\right|_{\mathbf{x}=\breve{\mathbf{x}}}$ ▷ compute Jacobian
    $\mathbf{H}_k \leftarrow \mathbf{J}_k^T \Omega_k \mathbf{J}_k$ ▷ contribution of $\mathbf{z}_k$ in $\mathbf{H}$
    $\mathbf{b}_k \leftarrow \mathbf{J}_k^T \Omega_k \mathbf{e}_k$ ▷ contribution of $\mathbf{z}_k$ in $\mathbf{b}$
    $\mathbf{H}$ += $\mathbf{H}_k$ ▷ Accumulate contributions in $\mathbf{H}$
    $\mathbf{b}$ += $\mathbf{b}_k$ ▷ Accumulate contributions in $\mathbf{b}$
  **end for**
  $\Delta \mathbf{x} \leftarrow solve(\mathbf{H}\Delta \mathbf{x} = -\mathbf{b})$ ▷ Solve linear system
  $\breve{\mathbf{x}}$ += $\Delta \mathbf{x}$ ▷ Apply increment
  $F_{new} \leftarrow F(\breve{\mathbf{x}})$ ▷ Update the error
**end while**
**return** $\breve{\mathbf{x}}$

It is good to notice that, if the *sensor model* $h_k(\cdot)$ is a linear function of the state, it is possible to find the minimum in just one iteration. However, since it is almost never the case in our cases of study, a solution must be found iteratively, until convergence is reached. To do so, it is possible to use the **Gauss-Newton** algorithm - described in Algorithm 1. However, *Gauss-Newton* is not guaranteed to converge in general. The convergence is subject to several factors, like the smoothness of the error function used or the initial guess - i.e. if it is close to potential singularity or far from the optimal solution. *Levenberg-Marquardt* iterative algorithm is variation of *Gauss-Newton* that enforces the convergence - shown in detail in Algorithm 2. It solves a *damped* version of the linear system 3.13, described by the following formula:

$$(\mathbf{H} + \lambda \mathbf{I})\, \Delta\mathbf{x} = -\mathbf{b} \tag{3.14}$$

where $\lambda$ is a scalar *damping* factor. The algorithm does not diverge, but it may converge to a *local minimum* and, thus, retrieving a sub-optimal solution.

It is good to notice that the system developed in this work uses the *Gauss-Newton* algorithm, since the error function has been properly manipulated to be more linear with respect to other approaches. Obviously, more details can be found in the next Chapters.

## Manifolds

In the previous Section it has been shown how to compute the optimal solution to the problem through Least-Squares estimation. However, two strong assumptions have been made:

1. The measurement function is smooth enough to be approximated by its *first-order Taylor expansion* without loss of generality
2. The state space spans over an *Euclidean domain*, and, thus $\mathbf{x} \in \mathbb{R}^n$

While the first hypothesis is true in general, the second one is almost never verified in SLAM problems. For example, if the state of the system is the robot 3D-pose, it involves to deal *Euler angles* or rotations in general; therefore the state belongs to $SE(3)$ - namely the *Special Euclidean group* of dimension 3. In this topological spaces, operations like *sum* or the *product* are not defined as in $\mathbb{R}^n$, thus they are illegal - i.e. summing 2 triads of Euler angles will lead to an inconsistent result or to singular configurations.

However, in SLAM - as in SfM or BA - the state generally belongs to *topological spaces* that *locally resemble* the Euclidean space, called **manifolds**. This means that each point of an $n-$dimensional manifold has a neighborhood that is homeomorphic to $\mathbb{R}^n$. Examples of manifolds are spheres or toruses, which are *locally* flat - the reader can think to the Earth that, for its inhabitants, seems flat.

Stepping back to our problem, it is possible for us to use this property to perform Least-Square estimation also with state spaces that are manifold. Assuming that our state belongs to $SO(3)$ - 3D rotations - if we represent it through a rotation matrix $\mathbf{R}(\phi, \theta, \psi)$ it is not possible to simply sum two quantities since it is not enforced the matrix's orthogonality. A minimal representation, instead, $\mathbf{x} = (\phi, \theta, \psi)$ will lead to singularities. However, if we are in a neighborhood of the origin, Euler angles are away from singularities. Thus we can define an operator *box-plus* that locally sums two quantities, exploiting manifold's property. The same must be done for other mathematical operators. In the case of rotation, we can define the following operators:

$$\mathbf{R} = \mathbf{R}_0 \boxplus \mathbf{u} = toMatrix(\mathbf{u})\, \mathbf{R}_0 \tag{3.15}$$

$$\mathbf{u} = \mathbf{R} \boxminus \mathbf{R}_0 = toEuler(\mathbf{R}_0^T\, \mathbf{R}) \tag{3.16}$$

**Algorithm 2** Standard Levenberg-Marquardt minimization algorithm

**Require:** Initial guess $\breve{\mathbf{x}}$; a set of measurements $\mathcal{C} = \{\langle \mathbf{z}_k, \Omega_k \rangle\}$
**Ensure:** Optimal solution $\mathbf{x}^\star$

$F_{new} \leftarrow \breve{F}$ ▷ compute the current error
$\mathbf{x}_{backup} \leftarrow \breve{\mathbf{x}}$ ▷ backup the solution
$\lambda \leftarrow \text{computeInitialLambda}(\mathcal{C}, \breve{\mathbf{x}})$
**while** $\breve{F} - F_{new} > \epsilon \wedge t < t_{max}$ **do**
  $\breve{F} \leftarrow F_{new}$
  $\mathbf{b} \leftarrow 0$
  $\mathbf{H} \leftarrow 0$
  **for all** $k \in \{1, ..., K\}$ **do**
    $\hat{\mathbf{z}}_k \leftarrow h_k(\breve{\mathbf{x}})$ ▷ compute prediction
    $\mathbf{e}_k \leftarrow \hat{\mathbf{z}}_k - \mathbf{z}_k$ ▷ compute error
    $\mathbf{J}_k \leftarrow \left.\frac{\partial h_k(\mathbf{x})}{\partial \mathbf{x}}\right|_{\mathbf{x}=\breve{\mathbf{x}}}$ ▷ compute Jacobian
    $\mathbf{H}_k \leftarrow \mathbf{J}_k^T \Omega_k \mathbf{J}_k$ ▷ contribution of $\mathbf{z}_k$ in $\mathbf{H}$
    $\mathbf{b}_k \leftarrow \mathbf{J}_k^T \Omega_k \mathbf{e}_k$ ▷ contribution of $\mathbf{z}_k$ in $\mathbf{b}$
    $\mathbf{H} \mathrel{+}= \mathbf{H}_k$ ▷ accumulate contributions in $\mathbf{H}$
    $\mathbf{b} \mathrel{+}= \mathbf{b}_k$ ▷ accumulate contributions in $\mathbf{b}$
  **end for**
  $t \leftarrow 0$ ▷ n. of iterations $\lambda$ has been adjusted
  **while** $t < t_{max} \wedge t > 0$ **do**
    $\Delta\mathbf{x} \leftarrow solve\,((\mathbf{H} + \lambda I)\Delta\mathbf{x} = \mathbf{b})$ ▷ solve damped linear system
    $\breve{\mathbf{x}} \mathrel{+}= \Delta\mathbf{x}$ ▷ apply increment
    $F_{new} \leftarrow F(\breve{\mathbf{x}})$ ▷ update the error
    **if** $F_{new} < \breve{F}$ **then**
      $\lambda \leftarrow \lambda/2$ ▷ good step, accept the solution
      $\mathbf{x}_{backup} \leftarrow \breve{\mathbf{x}}$
      $t \leftarrow t - 1$
    **else**
      $\lambda \leftarrow \lambda * 4$ ▷ bad step, revert the solution
      $\breve{\mathbf{x}} \leftarrow \mathbf{x}_{backup}$
      $t \leftarrow t + 1$
    **end if**
  **end while**
**end while**
**return** $\breve{\mathbf{x}}$

The operators $\boxminus$ and $\boxplus$ convert a global difference in the manifold into a local perturbation and vice versa.

Now we have to feed this new formalizations into the previously seen Least-Squares

algorithm. From now on, $\mathbf{X}$ indicates the over-parametrized representation of the state, while $\mathbf{x}$ the minimal one - i.e. a vector. Recalling the equation 3.7, to linearize the problem it is necessary to apply a perturbation $\Delta\mathbf{x}$ to the current estimate of the state $\breve{\mathbf{X}}$. However, to do so, it must be employed the new operator *sum*. Moreover, $\breve{\mathbf{X}}$ can be treated as a constant since is the current estimate of the system; still, it is possible to span the state space by varying $\Delta\mathbf{x}$ (minimal representation). Therefore, the first order Taylor expansion of $h_k(\breve{\mathbf{X}} \boxplus \Delta\mathbf{x})$ is computed with respect to $\Delta\mathbf{x}$, in the linearization point $\Delta\mathbf{x} = 0$. This leads to the following new formulation:

$$h_k(\breve{\mathbf{X}} \boxplus \Delta\mathbf{x}) \approx h_k(\breve{\mathbf{X}}) + \underbrace{\frac{\partial h_k(\mathbf{X} \boxplus \Delta\mathbf{x})}{\partial \Delta\mathbf{x}}\bigg|_{\Delta\mathbf{x}=0}}_{\tilde{\mathbf{J}}_k} \Delta\mathbf{x} = h_k(\breve{\mathbf{X}}) + \tilde{\mathbf{J}}_k \Delta\mathbf{x} \tag{3.17}$$

It is good to notice that the formulation 3.17 is topologically similar to the equation 3.7 with obvious differences due to the manifold state space.

As a consequence of what we have seen so far, in order to compute the optimal $\breve{\mathbf{X}}$, the operator $\boxplus$ must be used to apply the optimal increment, leading to the following relation:

$$\mathbf{X}^\star = \breve{\mathbf{X}} \boxplus \Delta\mathbf{x}^\star \tag{3.18}$$

Until now, it has been assumed that the measurements lie on $\mathbb{R}^n$. However, in our case of study, those may lie on a manifold too. This means that we have to define some new mathematical operators and minimal/redundant representations also for the measurements, like it has been done for the states. In particular, we know that the generic error function is $\mathbf{e}_k(x) = h_k(\mathbf{x}) - \mathbf{z}_k$; now it is necessary to introduce the new operator *difference*, that leads to the following error formulation:

$$\tilde{\mathbf{e}}_k(\mathbf{X}) = \tilde{\mathbf{e}}_k(\hat{\mathbf{z}}_k, \mathbf{z}_k) = h_k(\mathbf{X}) \boxminus \mathbf{z}_k \tag{3.19}$$

Equation 3.19 computes the error between the predicted measurement and the actual one in the minimal space, centering the computation in $\mathbf{z}_k$. In general the error function will be smooth and regular even using the operator $\boxplus$, since the displacement between $h_k(\mathbf{X})$ and $\mathbf{z}_k$ is generally small.

Applying a small perturbation $\Delta\mathbf{x}$ to the *prediction*, it is possible to compute the first order Taylor expansion of the error, which gives the following result:

$$\begin{aligned}\tilde{\mathbf{e}}_k(\hat{\mathbf{z}}_k + \Delta\mathbf{z}_k, \mathbf{z}_k) &= (\hat{\mathbf{z}}_k + \Delta\mathbf{z}_k) \boxminus \mathbf{z}_k \approx \\ &\approx \hat{\mathbf{z}}_k \boxminus \mathbf{z}_k + \underbrace{\frac{\partial\left((\hat{\mathbf{z}}_k + \Delta\mathbf{z}_k) \boxminus \mathbf{z}_k\right)}{\partial \Delta\mathbf{z}_k}\bigg|_{\Delta\mathbf{z}_k=0}}_{\mathbf{J}_{\mathbf{z}_k}} \Delta\mathbf{z}_k\end{aligned} \tag{3.20}$$

It is good to notice that the approximation of the error conditional distribution is described by a Gaussian with mean $\hat{\mathbf{z}}_k \boxminus \mathbf{z}_k$ and covariance $\Sigma_{\mathbf{e}_k|\mathbf{x}} = \mathbf{J}_{\mathbf{z}_k} \Omega_k^{-1} \mathbf{J}_{\mathbf{z}_k}^T$. The reader must notice that projecting the measurement onto a minimal space using the operator $\boxminus$ makes the covariance of the conditional error distribution a function of the state - since it depends by $\hat{\mathbf{z}}_k$. Therefore, $\Sigma_{\mathbf{e}_k|\mathbf{x}}$ must be computed at every iteration. However, in our work we manged to keep the error function Euclidean, avoiding the computation of $\Sigma_{\mathbf{e}_k|\mathbf{x}}$ even if the measurements lie on $SE(3)$ - e.g. in the case of pose measurements like the ones generated by the odometry.

In order to simplify the computation $\tilde{\mathbf{J}}_k$ it is possible to exploit the chain rule for partial derivatives, which leads to following relation

$$\frac{\partial\, \mathbf{e}_k\left(\breve{\mathbf{x}} \boxplus \Delta\mathbf{x}\right)}{\partial \Delta\mathbf{x}} = \underbrace{\left.\frac{\partial \mathbf{e}_k(\mathbf{x})}{\partial \mathbf{x}}\right|_{\mathbf{x}=\breve{\mathbf{x}}}}_{\mathbf{J}_k} + \underbrace{\left.\frac{\partial(\breve{\mathbf{x}} \boxplus \Delta\mathbf{x})}{\partial \Delta\mathbf{x}}\right|_{\Delta\mathbf{x}=0}}_{\mathbf{M}} \tag{3.21}$$

where $\mathbf{J}_k$ is the simple Jacobian computed in the Euclidean case, while $\mathbf{M}$ represents the derivate of $\boxplus$ operator evaluated in $\breve{\mathbf{x}}$. Given this, the total Jacobian $\tilde{\mathbf{J}}_k$ computed on a manifold is described by the relation

$$\tilde{\mathbf{J}}_k = \mathbf{J}_k \mathbf{M} \tag{3.22}$$

Finally, it is possible to plug all this new elements in the modified version of the *Gauss-Newton* algorithm - or the *Levenberg-Marquardt* one - to perform the optimization process properly with non-euclidean states and measurements, explained in detail in Algorithm 3.

## Sparse Least Squares

In the previous Sections it has been introduced several powerful tools to estimate a set of hidden variables given a set of measurements that are related to those variables. However, the state vector - especially in SLAM applications - may easily become big and, thus, it creates a big bottleneck that kills the performances. Still, it is possible to overcome this dimensional problem exploiting the nature of the measurements. In fact, a single measurement $\mathbf{z}_k$ depends only by a *subset* $\mathbf{x}_k = \mathbf{x}_{r:s} = \{\mathbf{x}_r, ..., \mathbf{x}_s\} \subseteq \mathbf{x}_{1:N}$ of the whole state vector and this will lead to special structure of the *Hessian* matrix $\mathbf{H}$.

Going deeper in this analysis, it is good to notice that each measurement contributes to the *Hessian* $\mathbf{H}$ and the *right-hand-side vector* $\mathbf{b}$ with just one addend - namely $\mathbf{H}_k$ and $\mathbf{b}_k$. The structure of those quantities depends by the Jacobian of the error function and this last one depends only by the state variables involved by the measurement.

Recalling equations 3.7 and 3.22 and on the basis of what just stated, *Jacobian*'s structure will be:

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

**Require:** Initial guess $\breve{\mathbf{x}}$; a set of measurements $\mathcal{C} = \{\langle \mathbf{z}_k, \Omega_k \rangle\}$
**Ensure:** Optimal solution $\mathbf{x}^\star$

$F_{new} \leftarrow \breve{F}$ ▷ compute the current error
**while** $\breve{F} - F_{new} > \epsilon$ **do**
  $\breve{F} \leftarrow F_{new}$
  $\mathbf{b} \leftarrow 0$
  $\mathbf{H} \leftarrow 0$
  **for all** $k \in \{1, ..., K\}$ **do**
    $\hat{\mathbf{z}}_k \leftarrow h_k(\breve{\mathbf{x}})$ ▷ compute prediction
    $\mathbf{e}_k \leftarrow \hat{\mathbf{z}}_k \boxminus \mathbf{z}_k$ ▷ compute error with over-parametrized $\mathbf{z}_k$
    $\tilde{\mathbf{J}}_k \leftarrow \left.\frac{\partial \tilde{\mathbf{e}}_k(h_k(\mathbf{X} \boxplus \Delta\mathbf{x}), \mathbf{z}_k)}{\partial \Delta\mathbf{x}_k}\right|_{\Delta\mathbf{x}_k = 0}$ ▷ compute Jacobian of the error function
    $\mathbf{J}_{\mathbf{z}_k} \leftarrow \left.\frac{\partial((\hat{\mathbf{z}}_k + \Delta\mathbf{z}_k) \boxminus \mathbf{z}_k)}{\partial \Delta\mathbf{z}_k}\right|_{\Delta\mathbf{z}_k = 0}$ ▷ compute Jacobian of the $\boxminus$ w.r.t. $\Delta\mathbf{z}_k$
    $\tilde{\Omega}_k \leftarrow \left(\mathbf{J}_{\mathbf{z}_k} \Omega_k \mathbf{J}_{\mathbf{z}_k}^T\right)^{-1}$ ▷ Remap the information matrix
    $\mathbf{H}_k \leftarrow \mathbf{J}_k^T \tilde{\Omega}_k \mathbf{J}_k$ ▷ contribution of $\mathbf{z}_k$ in $\mathbf{H}$
    $\mathbf{b}_k \leftarrow \mathbf{J}_k^T \tilde{\Omega}_k \mathbf{e}_k$ ▷ contribution of $\mathbf{z}_k$ in $\mathbf{b}$
    $\mathbf{H} \mathrel{+}= \mathbf{H}_k$ ▷ Accumulate contributions in $\mathbf{H}$
    $\mathbf{b} \mathrel{+}= \mathbf{b}_k$ ▷ Accumulate contributions in $\mathbf{b}$
  **end for**
  $\Delta\mathbf{x} \leftarrow solve(\mathbf{H}\Delta\mathbf{x} = -\mathbf{b})$ ▷ Solve linear system
  $\breve{\mathbf{x}} \leftarrow \breve{\mathbf{x}} \boxplus \Delta\mathbf{x}$ ▷ Apply increment
  $F_{new} \leftarrow F(\breve{\mathbf{x}})$ ▷ Update the error
**end while**
**return** $\breve{\mathbf{x}}$

$$\mathbf{J}_k = \underbrace{\begin{bmatrix}\mathbf{0} \cdots \mathbf{0}\, \mathbf{J}_{k_1} \mathbf{0} \cdots \mathbf{0}\, \mathbf{J}_{k_h} \mathbf{0} \cdots \mathbf{0}\, \mathbf{J}_{k_q} \mathbf{0} \cdots \mathbf{0}\end{bmatrix}}_{Nblocks} \tag{3.23}$$

where each non-zero component $\mathbf{J}_{k_h} = \frac{\partial \mathbf{e}_k(\mathbf{x}_k)}{\partial \mathbf{x}_{k_h}}$ represents the partial derivative of the error function deriving from $\mathbf{z}_k$, computed with respect to the parameter block $\mathbf{x}_{k_h} \in \mathbf{x}_k$. According to equation 3.10, the contribution to $\mathbf{H}$ and $\mathbf{b}$ will have the following anatomy:

$$\mathbf{H}_k = \begin{pmatrix} \ddots & & & & & & \\ & \mathbf{J}_{k_1}^T \Omega_k \mathbf{J}_{k_1} & \cdots & \mathbf{J}_{k_1}^T \Omega_k \mathbf{J}_{k_h} & \cdots & \mathbf{J}_{k_1}^T \Omega_k \mathbf{J}_{k_q} & \\ & \vdots & & \vdots & & \vdots & \\ & \mathbf{J}_{k_h}^T \Omega_k \mathbf{J}_{k_1} & \cdots & \mathbf{J}_{k_h}^T \Omega_k \mathbf{J}_{k_h} & \cdots & \mathbf{J}_{k_h}^T \Omega_k \mathbf{J}_{k_q} & \\ & \vdots & & \vdots & & \vdots & \\ & \mathbf{J}_{k_q}^T \Omega_k \mathbf{J}_{k_1} & \cdots & \mathbf{J}_{k_q}^T \Omega_k \mathbf{J}_{k_h} & \cdots & \mathbf{J}_{k_q}^T \Omega_k \mathbf{J}_{k_q} & \\ & & & & & & \ddots \end{pmatrix} \tag{3.24}$$

$$\mathbf{b}_k = \begin{bmatrix} \vdots \\ \mathbf{J}_{k_1} \Omega_k \mathbf{e}_k \\ \vdots \\ \mathbf{J}_{k_h} \Omega_k \mathbf{e}_k \\ \vdots \\ \mathbf{J}_{k_q} \Omega_k \mathbf{e}_k \\ \vdots \end{bmatrix} \tag{3.25}$$

The internal structure of $\mathbf{H}$ and $\mathbf{b}$ just shown, reveals some important properties of those objects:

- $\mathbf{b}$ is a **dense vector** composed by $N$ non-zero blocks.
- The Hessian $\mathbf{H}$ is a **sparse symmetric matrix**.
- Every new measurement $q$ introduces $q^2$ non-zero blocks in the Hessian.

Hessian's sparsity is fundamental to perform efficient and fast optimization, especially to solve the linear system 3.13. The literature proposes many approaches to efficiently solve sparse linear systems, either with *direct* [25] or *iterative* [26] methods. One of the most employed direct method uses the *Cholesky* factorization of the matrix - namely the *LU* decomposition. In this case, the Hessian is decomposed into two triangular matrices $\mathbf{H} = \mathbf{L}\mathbf{U}$, where $\mathbf{U} = \mathbf{L}^T$ and $\mathbf{L}$ is a *lower-triangular matrix*. The solution to the original system is found solving consecutively the following derived systems:

$$\begin{cases} \mathbf{L}\,\mathbf{y} & = \mathbf{b} \\ \mathbf{U}\,\Delta\mathbf{x} & = \mathbf{y} \end{cases} \tag{3.26}$$

Since $L$ and $U$ are triangular matrices, the solutions of equations 3.26 can be easily computed through *Forward* and *Backward Substitution* respectively. It is good to notice that the *Cholesky* decomposition of a sparse matrix will have a greater number of non-zero entries than the source, due to the *fill-in* deriving from the decomposition itself. However, it is possible to reduce the *fill-in* with several techniques, like *variable reordering*. This

is achieved pre and post-multiplying the source matrix - $\mathbf{H}$ - for a suitable *permutation matrix* $\mathbf{P}$. The *Cholesky* factorization will be applied to the permuted matrix $\mathbf{PHP}^T = \hat{\mathbf{L}}\hat{\mathbf{U}}$, preserving the sparsity and, thus, leading to faster solution of the linear system.

The literature proposes a lot of theory on how to retrieve the good ordering to reduce the fill-in, minimize the *FLOPs* or to exploit parallel computation. Some of the most effective orderings to reduce the fill-in are computed by *Minimum Degree* [27, 28], *super-nodal* [29] and *nested dissection based* algorithms [30]. It is good to notice that this is an *NP-hard* problem and several libraries have been developed to compute orderings in a fast and efficient way [31].

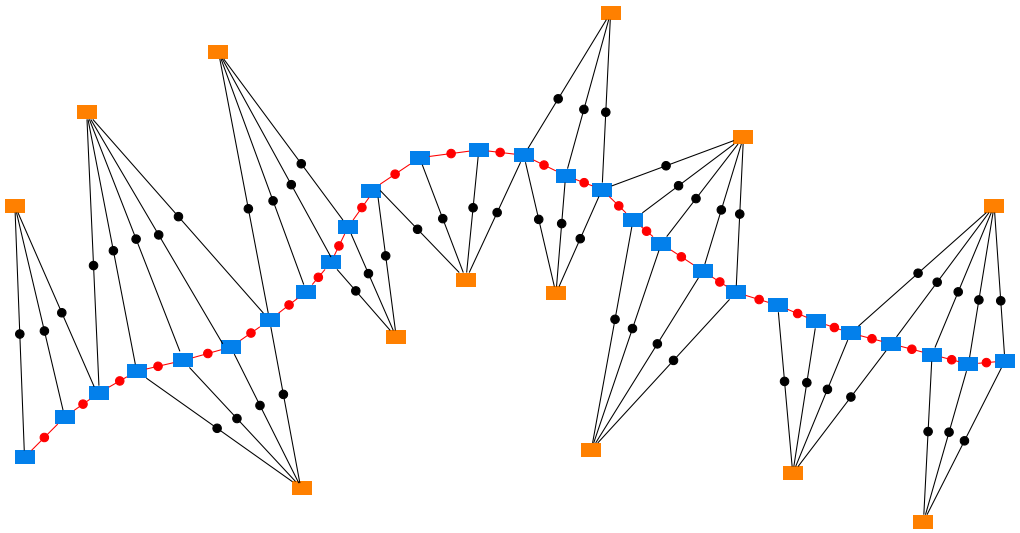


Figure 3.1: **Generic Factor Graph.** The figure depicts the structure of factor graph. The nodes are illustrated with colored squares and they can represent either a *pose* - in blue - or a *salient world point* - in orange. Measurements coming from the sensors are the constraints that connect the nodes, illustrated with circles - red for pose constraints and black for point ones.

## Factor Graphs

Until now, it has been showed the methodology to solve a least-squares minimization problem whose cost function is given by 3.8. In the previous Section, it has been underlined the *sparsity* of the problem. In particular it has been stated that, given the state $\mathbf{x} = (\mathbf{x}_1, ..., \mathbf{x}_N)$, the measurement $\mathbf{z}_k$ represents a **constraint** relating only a subset of

the whole state vector, namely $\mathbf{x}_k = \left(\mathbf{x}_{k_1}, ..., \mathbf{x}_{k_q}\right) \subseteq \mathbf{x} = (\mathbf{x}_1, ..., \mathbf{x}_N)$. In this sense, the error $\mathbf{e}_k(\hat{\mathbf{z}}_k, \mathbf{z}_k)$ describes how well the parameter blocks in $\mathbf{x}_k$ satisfy the constraint $\mathbf{z}_k$ and, in fact, it is $\mathbf{0}$ when $\mathbf{x}_k$ perfectly matches the constraint.

A problem that has this formulation, can be easily represented with a *directed hyper-graph*, where

- Each parameter block $\mathbf{x}_i \in \mathbf{x}_k$ represents a node $i$ in the hyper-graph
- Each constraint $\mathbf{z}_k$ represents an hyper-edge that links all the nodes $\mathbf{x}_i \in \mathbf{x}_k$.

Obviously, when the hyper-edges have size 2, the hyper-graph becomes an ordinary graph. Figure 3.1 shows the concept underlying this problem formulation.

Graph-based formulations are very common for the SLAM problem, since they help the optimization process and, in fact, most of the current state-of-the-art systems are based on this formulation. In fact, exploiting the *topology* of the graph it is possible to achieve better performances, both in terms of speed and accuracy of the solution.

Now that the underlying theory has been exploited, it is possible to better explain the typical formulations of the problem addressed in SLAM. Therefore, the next Chapter will propose an overview of the most common ones, in particular for 3D environments.

Chapter 4

# Typical Problems

In this Chapter the reader will became familiar with typical formulations of SLAM problems, with a particular focus on 3D environments. Obviously, there are several other instances of the problem that will be not shown since they are not strictly related to this work.

## Pose Graphs

Pose Graphs represents the backbone of SLAM. In this problem, the state vector $\mathbf{x} = (\mathbf{x}_1, ..., \mathbf{x}_N)$ is composed by a 2D or 3D isometry, thus, each node of the graph $\mathbf{x}_i$ belongs to $SE(2)$ or $SE(3)$ - i.e. the *Special Euclidean* group of dimension 2 or 3.

The measurements are also 2D or 3D isometries that lie in $SE(2)$ or $SE(3)$. Therefore, an edge $\mathbf{z}_{ij}$ that connects $\mathbf{x}_i$ with $\mathbf{x}_j$ represents the pose of node $j$ expressed in the reference frame of node $i$ - e.g. $\mathbf{z}_{ij} = {}^iT_j$.

This problem is very common in SLAM: suppose that we want to estimate the best trajectory of a 2D robot, given only the *odometry* measurements. The odometry retrieves robot's motion from state $\mathbf{x}_i$ to $\mathbf{x}_j$ and encodes it into the transformation matrix ${}^iT_j$. Moreover, supposing that the robot is also able to retrieve *loop-closures*, this would mean that there is an edge between the nodes $i$ and $k$. The related transformation is encoded into the quantity $\mathbf{z}_{ik} = {}^iT_k$.

Clearly, both states and measurements are non-euclidean. The extended parametrization of those quantities - as it has been stated before - is given by an isometry, while a possible *minimal representation* can be a 3 vector $(t_x\ t_y\ \theta)^T$. The next step is to define the operators *box-plus* and *box-minus*. Therefore, we introduce the operators $t2v$ and $v2t$ that allow to map an isometry into a 3 vector and vice versa. Given those operators, we will have the following relations:

$$\mathbf{X} \boxplus \Delta\mathbf{x} = \mathbf{X} \cdot \mathrm{v2t}(\Delta\mathbf{x}) \tag{4.1}$$

$$\mathbf{X}_a \boxminus \mathbf{X}_b = \mathrm{t2v}\left(\mathbf{X}_b^{-1}\,\mathbf{X}_a\right) \tag{4.2}$$

Since the measurement $\mathbf{z}_{ij}$ expresses pose $\mathbf{X}_j$ with respect to the reference system of $\mathbf{X}_i$, the predicted measurement $\mathbf{z}_{ij}$ can be computed as

$$\hat{\mathbf{Z}}_{ij} = h_{ij}(\mathbf{X}) = \mathbf{X}_i^{-1}\,\mathbf{X}_j \tag{4.3}$$

Given the relations 4.2 and 4.3, the error is computed as follows:

$$\begin{aligned}\mathbf{e}_{ij}\left(\mathbf{X}\right) &= h_{ij}\left(\mathbf{X}\right) \boxminus \mathbf{Z}_{ij} = \\ &= \mathrm{t2v}\left(\mathbf{Z}_{ij}^{-1}\,\mathbf{X}_i^{-1}\,\mathbf{X}_j\right)\end{aligned} \tag{4.4}$$

Finally, Jacobians must be computed and, to do so, we apply a perturbation to the error function 4.4, that leads to the following relation:

$$\begin{aligned}\mathbf{e}_{ij}\left(\mathbf{X} \boxplus \Delta\mathbf{x}\right) &= \mathrm{t2v}\left(\mathbf{Z}_{ij}^{-1}\ \left(\mathbf{X}_i \cdot \mathrm{v2t}(\Delta\mathbf{x}_i)\right)^{-1}\ \left(\mathbf{X}_j \cdot \mathrm{v2t}(\Delta\mathbf{x}_j)\right)\right) = \\ &= \mathrm{t2v}\left(\mathbf{Z}_{ij}^{-1}\,\mathrm{v2t}(\Delta\mathbf{x}_i)^{-1}\ \underbrace{\mathbf{X}_i^{-1}\mathbf{X}_j}_{\hat{\mathbf{Z}}_{ij}}\ \mathrm{v2t}(\Delta\mathbf{x}_j)\right) = \\ &= \mathrm{t2v}\left(\mathbf{Z}_{ij}^{-1}\,\mathrm{v2t}(\Delta\mathbf{x}_i)^{-1}\,\hat{\mathbf{Z}}_{ij}\,\mathrm{v2t}(\Delta\mathbf{x}_j)\right)\end{aligned} \tag{4.5}$$

The remaining part is just the computation of the following partial derivatives:

$$\mathbf{J}_i = \left.\frac{\partial\,\mathrm{t2v}\left(\mathbf{Z}_{ij}^{-1}\,\mathrm{v2t}(\Delta\mathbf{x}_i)^{-1}\,\hat{\mathbf{Z}}_{ij}\,\mathrm{v2t}(\Delta\mathbf{x}_j)\right)}{\partial\Delta\mathbf{x}_i}\right|_{\Delta\mathbf{x}_i=0,\Delta\mathbf{x}_j=0} \tag{4.6}$$

$$\mathbf{J}_j = \left.\frac{\partial\,\mathrm{t2v}\left(\mathbf{Z}_{ij}^{-1}\,\mathrm{v2t}(\Delta\mathbf{x}_i)^{-1}\,\hat{\mathbf{Z}}_{ij}\,\mathrm{v2t}(\Delta\mathbf{x}_j)\right)}{\partial\Delta\mathbf{x}_j}\right|_{\Delta\mathbf{x}_i=0,\Delta\mathbf{x}_j=0} \tag{4.7}$$

The final Jacobian will be non-zero only in the blocks relative to variables $\mathbf{X}_i$ and $\mathbf{X}_j$, in formulæ:

$$\mathbf{J} = [\mathbf{0}\ \cdots\ \mathbf{0}\ \mathbf{J}_i\ \mathbf{0}\ \cdots\ \mathbf{0}\ \mathbf{J}_j\ \mathbf{0}\ \cdots\ \mathbf{0}] \tag{4.8}$$

The reader might notice that the equation 4.5 is highly non-linear - as also the operators t2v and v2t themselves - and this will lead to a less accurate approximation obtained from the first-order Taylor expansion of the error 4.4 and to the computation of complex derivatives given by 4.6 and 4.7. The situation becomes even worse in a 3D environment, which will be better analyzed in the next Chapters.

## Pose-Landmarks Graphs

In this Section we will propose another common formulation of the problem. Landmarks represent the position of salient world points and here are employed to optimize the trajectory of the robot, assuming that the position of those points is known in the environment.

Taking in consideration again the 2D problem for simplicity, the true position of those points in the world will be given by $\mathbf{p} = (p_x\, p_y)^T$. The measurements $\mathbf{z}_k$ are the position of those point in the robot reference frame - namely $\mathbf{z}_k = (z_x\, z_y)^T$. It is good to notice that while the state still belongs to $SE(2)$, the measurements now are Euclidean. Both the minimal and extended parametrization of the state can be taken from the previous formulation - so they will be a 3 vector $\mathbf{x} = (t_x t_y \theta)$ and a 2D isometry $\mathbf{X} = {}^{W}T_R$ respectively. Also the operators $\boxplus$ and $\boxminus$ remain unchanged. The 2D isometry $\mathbf{X}_i$ is composed by a rotational part $\mathbf{R}_\theta$ and a translational one $\mathbf{t}$, in formulæ:

$$\mathbf{X} = \begin{bmatrix} \mathbf{R}_\theta & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix} \qquad \mathbf{R}_\theta = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \qquad \mathbf{t} = \begin{pmatrix} t_x \\ t_y \end{pmatrix} \tag{4.9}$$

Given those considerations, it is possible to define the new *prediction* of the measurement:

$$\begin{aligned} \hat{\mathbf{z}}_{ij} = h_{ij}(\mathbf{X}) &= \mathbf{X}_i^{-1}\, \mathbf{p}_j \\ &= \mathbf{R}_\theta^T\, \mathbf{p}_j - \mathbf{R}_\theta^T \mathbf{t} \end{aligned} \tag{4.10}$$

Since the measurements are Euclidean - i.e. they lie in $\mathbb{R}^2$ - the error is just the standard subtraction between prediction and measurement, namely $\mathbf{e}_{ij}(\mathbf{X}) = \hat{\mathbf{z}}_{ij} - \mathbf{z}_j$. Applying a state perturbation to the error we obtain:

$$\mathbf{e}_{ij}(\mathbf{X} \boxplus \Delta\mathbf{x}) = (\mathbf{X}_i\, \mathrm{v2t}(\Delta\mathbf{x}_i))^{-1}\, \mathbf{p}_j - \mathbf{z}_j \tag{4.11}$$

To facilitate the computation of the derivatives, it is possible to define the *box-plus* operator as the pre-multiplication of $\mathrm{v2t}(\Delta\mathbf{x})$ to the state isometry, namely $\mathbf{X} \boxplus \Delta\mathbf{x} = \mathrm{v2t}(\Delta\mathbf{x})\, \mathbf{X}$. Moreover, instead of considering the state as $\mathbf{X} = {}^{W}T_R$ - i.e. the transformation from world to robot expressed in the world reference frame - it is more advisable to use its inverse. Therefore, the state now is $\mathbf{X} = {}^{R}T_W = {}^{W}T_R^{-1}$ - i.e. the transformation *from robot to world* expressed in the robot reference frame. Introducing this notation together with the new *box-plus* operator, equations 4.10 and 4.11 become:

$$\hat{\mathbf{z}}_{ij} = h_{ij}(\mathbf{X}) = \mathbf{X}_i\, \mathbf{p}_j = \mathbf{R}_\theta \mathbf{p}_j + \mathbf{t} \tag{4.12}$$

$$\mathbf{e}_{ij}(\mathbf{X} \boxplus \Delta\mathbf{x}) = \mathrm{v2t}(\Delta\mathbf{x}) \underbrace{\mathbf{X}_i\, \mathbf{p}_j}_{\tilde{\mathbf{p}}_j} - \mathbf{z}_j \tag{4.13}$$

The Jacobian is now more straightforward and it is computed as follows:

$$\begin{aligned}\mathbf{J}_i &= \left.\frac{\partial \mathbf{e}_{ij}(\mathbf{X} \boxplus \Delta\mathbf{x})}{\partial \Delta\mathbf{x}}\right|_{\Delta\mathbf{x}=0} = \\ &= \left.\frac{\partial \begin{pmatrix} \tilde{p}_x \cos(\Delta\theta) - \tilde{p}_y \sin(\Delta\theta) + \Delta t_x \\ \tilde{p}_x \sin(\Delta\theta) + \tilde{p}_y \cos(\Delta\theta) + \Delta t_y \end{pmatrix}}{\partial \Delta\mathbf{x}}\right|_{\Delta\mathbf{x}=0} = \\ &= \begin{bmatrix} I_{2\times 2} & \begin{matrix} -\tilde{p}_y \\ \tilde{p}_x \end{matrix} \end{bmatrix} \end{aligned} \tag{4.14}$$

Now that we defined every aspect of the problem, it is possible to embed everything in the already seen Least-Squares algorithm to find the optimal state.

## Bundle Adjustment

This problem instantiation is more complex with respect what has been proposed in the previous Sections. In Section 4.1 the constraints were only of type *pose* - i.e. they lie on $SE(n)$ - while in Section 4.2 they were only of type *point* - i.e. they lie on $\mathbb{R}^n$.

Bundle Adjustment (BA) aims to estimate *both* the robot trajectory *and* the world position of the landmarks, given constraints of type *pose* and *point*. Considering again the 2D case for simplicity, the state will be:

$$\mathbf{X} = \left( \overbrace{\mathbf{X}_1^R, ..., \mathbf{X}_N^R}^{N \text{ poses}} \mid \overbrace{\mathbf{x}_{N+1}^L, ..., \mathbf{x}_{N+M}^L}^{M \text{ landmarks}} \right) \tag{4.15}$$

where $\mathbf{X}_i^R = {}^W T_R \in SE(2)$ represents the $i$-th robot pose while $\mathbf{x}_j^L = \mathbf{p} = (p_x p_y)^T \in \mathbb{R}^2$ represents the world position of the $j$-th landmark. The set of increments $\Delta\mathbf{x}$ will have the structure seen in 4.15, namely:

$$\Delta\mathbf{x} = \left( \overbrace{\Delta\mathbf{x}_1^R, ..., \Delta\mathbf{x}_N^R}^{N \text{ poses}} \mid \overbrace{\Delta\mathbf{x}_{N+1}^L, ..., \Delta\mathbf{x}_{N+M}^L}^{M \text{ landmarks}} \right) \tag{4.16}$$

where $\Delta\mathbf{x}_i^R = (\Delta t_{x_i}\, \Delta t_{y_i}\, \Delta\theta_i)^T$ and $\Delta\mathbf{x}_j^L = (\Delta p_{x_j}\, \Delta p_{y_j})^T$.

In order to properly apply the increment, it is necessary to define a suitable operator *box-plus*. In this case, for the *point* increments it is possible to use the Euclidean sum, while for the *pose* ones we will employ the already seen operator $\boxplus$, leading to $\mathbf{X} \boxplus \Delta\mathbf{x}^R = \text{v2t}(\Delta\mathbf{x}^R)\, \mathbf{X}$.

Clearly, depending on the types of constraint that we linearize, it leads to different contribution in the *Hessian* matrix $\mathbf{H}$, so they will be analyzed separately.

### Pose-Pose Constraints

Here, for a 2D environment, it is possible to reuse what it has been shown in Section 4.1. Therefore, the *predicted measurement* $\hat{\mathbf{Z}}_{ij}$, the *error* $\mathbf{e}_{ij}$ and the *perturbed error* are computed as follows:

$$\begin{aligned}\hat{\mathbf{Z}}_{ij} &= h_{ij}(\mathbf{X}) = \mathbf{X}_i^{-1}\,\mathbf{X}_j \\ \mathbf{e}_{ij}(\mathbf{X}) &= \text{t2v}\left(\mathbf{Z}_{ij}^{-1}\,\mathbf{X}_i^{-1}\,\mathbf{X}_j\right) \\ \mathbf{e}_{ij}(\mathbf{X}\boxplus\Delta\mathbf{x}^R) &= \text{t2v}\left(\mathbf{Z}_{ij}^{-1}\,\text{v2t}(\Delta\mathbf{x}_i^R)^{-1}\,\hat{\mathbf{Z}}_{ij}\,\text{v2t}(\Delta\mathbf{x}_j^R)\right)\end{aligned}$$

The complete Jacobian will be structured as in equation 4.8 and its components can be computed employing the *chain-rule* for partial derivatives.

### Pose-Point Constraints

It is possible to refer to what it has been shown in Section 4.2 for this part. The reader might notice that in this formulation the *robot pose* is expressed in the world reference frame - i.e. $\mathbf{X} = {}^{W}T_R$ - and, thus, we must stick to this notation also for *pose-point constraints*. Therefore, supposing that the measurement $\mathbf{z}_{ij}$ relates the $i$-th pose with the $j$-th point - indicated with $\mathbf{X}_i$ and $\mathbf{x}_j = \mathbf{p}_j$ respectively - the *predicted measurement* $\hat{\mathbf{z}}_{ij}$, the *error* $\mathbf{e}_{ij}$ and the *perturbed error* are computed as follows:

$$\begin{aligned}\hat{\mathbf{z}}_{ij} &= h_{ij}(\mathbf{X}) = \mathbf{X}_i^{-1}\,\mathbf{p}_j = \mathbf{R}_i^T\,\mathbf{p}_j - \mathbf{R}_i^T\mathbf{t}_i \\ \mathbf{e}_{ij}(\mathbf{X}) &= \hat{\mathbf{z}}_{ij} - \mathbf{z}_j \\ \mathbf{e}_{ij}(\mathbf{X}_i\boxplus\Delta\mathbf{x}_i^R, \mathbf{x}_j + \Delta\mathbf{x}_j^L) &= \left(\text{v2t}(\Delta\mathbf{x}_i^R)\mathbf{X}_i\right)^{-1}\left(\mathbf{p}_j + \Delta\mathbf{x}_j^L\right) - \mathbf{z}_j\end{aligned}$$

In this formulation the state parameters are $\mathbf{X}_i$ *and* $\mathbf{p}_j$, the final Jacobian $\mathbf{J}$ will have the following structure:

$$\mathbf{J} = [\mathbf{0}\ \cdots\ \mathbf{0}\ \mathbf{J}_R\ \mathbf{0}\ \cdots\ \mathbf{0}\ \mathbf{J}_L\ \mathbf{0}\ \cdot\ \mathbf{0}]$$

where

$$\mathbf{J}_R = \left.\frac{\partial\left[\left(\text{v2t}(\Delta\mathbf{x}_i)\mathbf{X}_i\right)^{-1}\left(\mathbf{p}_j + \Delta\mathbf{x}_j\right) - \mathbf{z}_j\right]}{\partial\Delta\mathbf{x}_i^R}\right|_{\Delta\mathbf{x}_i^R=0,\Delta\mathbf{x}_j^L=0} \tag{4.17}$$

$$\mathbf{J}_L = \left.\frac{\partial\left[\left(\text{v2t}(\Delta\mathbf{x}_i)\mathbf{X}_i\right)^{-1}\left(\mathbf{p}_j + \Delta\mathbf{x}_j\right) - \mathbf{z}_j\right]}{\partial\Delta\mathbf{x}_j^L}\right|_{\Delta\mathbf{x}_i^R=0,\Delta\mathbf{x}_j^L=0} \tag{4.18}$$

The reader might notice how this formulation leads to more complex derivatives with respect to the one reported in equation 4.14.

Once that all the needed objects are defined, it is possible to embed them in the iterative algorithm to find the optimal state configuration.

## Simultaneous Calibration Localization and Mapping

The formulations seen in the previous Sections, they all rely on the fact that all the specific *inner robot parameters are known* - e.g. the position of the sensors mounted on the robot with respect to the robot reference frame. Performing SLAM with a wrong estimation of those parameters reduces the accuracy of the system, leading to non-consistent estimations even after the optimization process.

In general to acquire those information, it is possible to proceed in different ways:

- Get those parameters from the robot's specifics - if they are present;
- Measure those quantity *manually* on the robot;
- Perform ad-hoc calibration *before* employing the robot in a mission.

All those procedures share the same drawbacks: they are not able to estimate non-stationary parameters and, if the robot is subject to hardware changes, the calibration procedure must be repeated.

A possible solution to this problem is to *embed those parameters in the state* and to estimate them together with the robot trajectory and the map. This solution has been proposed by *Kümmerle et al.* in their work [32] and it leads to several benefit to the original SLAM problem. Therefore, in this Section it is proposed a brief overview of this approach.

Given a generic robot in a 2D environment, we have to introduce the following quantities:

- $\mathbf{l} = (l_x \, l_y \, \theta_l)$ the pose of a generic sensor expressed with respect to the robot reference frame;
- $\mathbf{u}_i$ and $\Omega_i^u$ that represent respectively the *motion command* and its relative information matrix;
- $\mathbf{k}$ the parameters of the robot's *forward kinematic*.

The forward kinematic function $K(\mathbf{u}, \mathbf{k})$ converts the wheels' velocities deriving from the motion commands into the actual robot displacement from node $i$ to $i+1$. For example, taking in consideration a differential drive kinematic scheme, the motion commands

might be $\mathbf{u} = (v_r\, v_l)^T$ - respectively the left and right wheel velocity. Those lead to the following relation to compute the relative motion obtained during a time step $\Delta t$:

$$K(\mathbf{u}, \mathbf{k}) = \begin{pmatrix} \mathbf{R}(\Delta t\, \omega) & \mathbf{0} \\ \mathbf{0} & 1 \end{pmatrix} \begin{pmatrix} -ICC \\ 0 \end{pmatrix} + \begin{pmatrix} ICC \\ \Delta t\, \omega \end{pmatrix} \tag{4.19}$$

where

$$ICC = \begin{pmatrix} 0 \\ \frac{b}{2} \frac{r_l v_l + r_r v_r}{r_l v_l - r_r v_r} \end{pmatrix}$$
$$\omega = \frac{r_l v_l - r_r v_r}{b}$$

In this robot configuration $r_l$ and $r_r$ represent respectively the left and right wheel radii, while $b$ is the distance between the two driving wheels. Those quantities represent the forward kinematic parameters $\mathbf{k} = (r_r\, r_k\, b)^T$ that we embed in the estimation process.

Given those objects, the optimization process has to retrieve the optimal configuration $[\mathbf{x}^\star\, \mathbf{l}^\star\, \mathbf{k}^\star]$ that minimizes the following negative log-likelihood function $F(\mathbf{x}, \mathbf{l}, \mathbf{k})$:

$$F(\mathbf{x}, \mathbf{l}, \mathbf{k}) = \sum_{i,j} \mathbf{e}_{ij}^l(\mathbf{x})^T \Omega_{ij}^l\, \mathbf{e}_{ij}^l(\mathbf{x}) + \sum_i \mathbf{e}_i^u(\mathbf{x})^T \tilde{\Omega}_i^l\, \mathbf{e}_i^u(\mathbf{x}) \tag{4.20}$$

where $\mathbf{e}_{ij}^l(\mathbf{x})$ describes how well the parameter blocks $\mathbf{x}_i$, $\mathbf{x}_j$ and $\mathbf{l}$ satisfy the constraint $\mathbf{z}_{ij}$, while $\mathbf{e}_i^u(\mathbf{x})$ measures the likelihood of the parameter blocks $\mathbf{x}_i$, $\mathbf{x}_j$ and $\mathbf{k}$ with respect to the constraint $\mathbf{u}_i$. The two error functions $\mathbf{e}_{ij}^l(\mathbf{x})$ and $\mathbf{e}_i^u(\mathbf{x})$ are described analytically by the following relations:

$$\mathbf{e}_{ij}^l(\mathbf{x}) = ((\mathbf{x}_j \boxplus \mathbf{l}) \boxminus (\mathbf{x}_i \boxplus \mathbf{l})) \boxminus \mathbf{z}_{ij}$$
$$\mathbf{e}_{ij}^l(\mathbf{x}) = (\mathbf{x}_{i+1} \boxminus \mathbf{x}_i) \boxminus K(\mathbf{u}_i, \mathbf{k})$$

where the operators $\boxplus$ and $\boxminus$ are the same of the ones described in the previous Sections. It is good to notice that, in order to obtain consistent results, the information matrix $\Omega_i^u$ must be projected through the forward kinematic function $K(\mathbf{u}, \mathbf{k})$, for example using the *Unscented Transform* [33].

Given the main notions about the most common formulations of the problem, in the next Chapter it will be analyzed more in detail the solution for 3D factor graphs, explaining also the main contributions brought by this work.

Chapter 5

# Solving Factor Graphs with SE3 Variables

In this project it has been mainly addressed the optimization of **3D factor graphs**. In particular it has been developed from scratch a back-end system to efficiently solve *pose-graph optimization* and *bundle adjustment*.

The reader might feel already comfortable with the formulation of those problems, thus, in this Section will be shown more in detail the approaches used in order to achieve **real-time performances** and how $SE(3)$ constraints have been manipulated in order to **reduce problem's non-linearity**.

## Exploit Sparsity

As it as been already mentioned in Chapter 3, standard optimization algorithms like *Gauss-Newton* or *Levenberg-Marquardt* reduce the non-linear problem to the solution of a *linear system*, namely:

$$\mathbf{H}\Delta\mathbf{x} = -\mathbf{b}$$

Here, $\Delta\mathbf{x}$ and $\mathbf{b}$ are dense vectors, while the *Hessian*'s approximation $\mathbf{H}$ is a sparse matrix. The literature proposes a lot of methods to solve efficiently sparse linear systems. Those can be categorized in two main groups:

1. **Iterative methods** [26] that compute a solution iteratively.
2. **Direct methods** [25] that solve the system in just one step.

The former ones like *Conjugate Gradient* or *Generalized Minimal Residual Method* - shortened as GMRES - exploit fast matrix-vector product to deliver good performances even if the solution is found iteratively. Those often use also *preconditioning* that consist in the application of a transformation - called the preconditioner - that turns the system into a form more suitable for numerical solving methods - e.g. *Preconditioned Conjugate Gradient* (PCG). The latter group, instead, exploits matrix decompositions like *Cholesky* or the $QR$-decomposition to efficiently retrieve a solution in one step. Crucial for those

kind of methods is the *fill-in* - i.e. the increase of non-zero elements in the decomposition with respect to the source matrix.

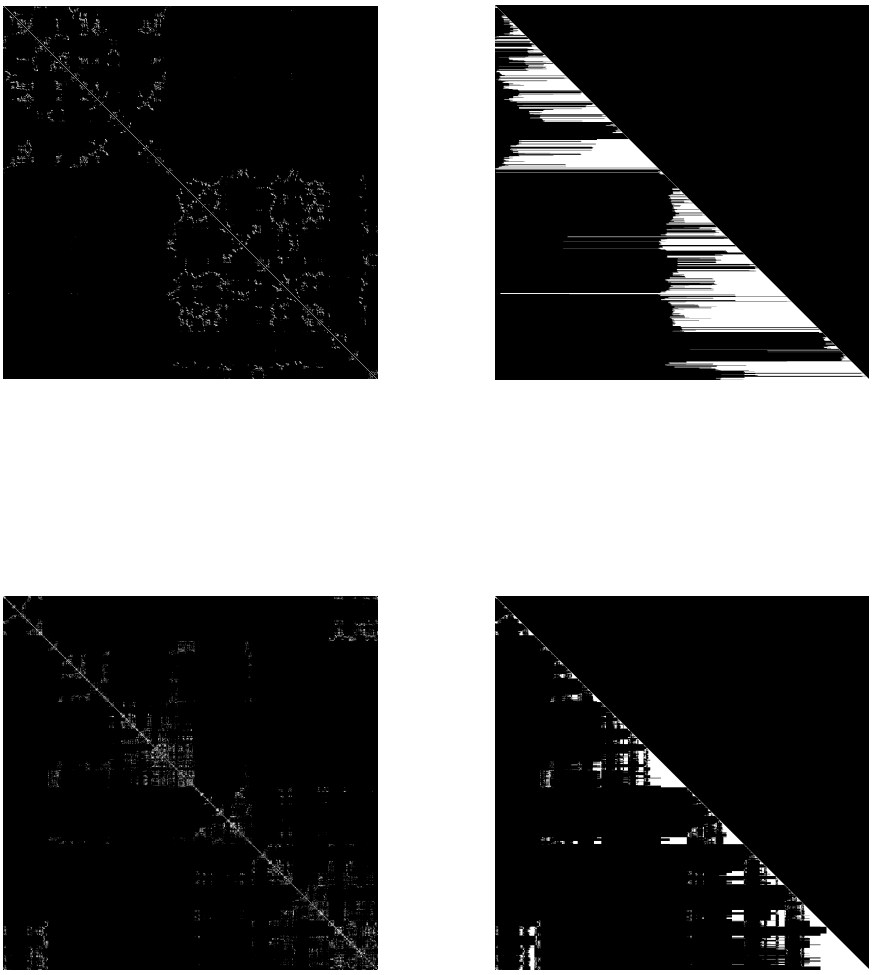


Figure 5.1: **Cholesky Fill-In.** The figure highlights the fill-in due to the factorization of a $(2000 \times 2000)$ symmetric PSD matrix: non-zero blocks are depicted in white, while null-blocks in black. The first row illustrates the patterns of matrices $\mathbf{H}$ and its decomposition $\mathbf{L}$ - respectively on the left and on the right. In the bottom row, the same matrices after the permutation of $\mathbf{H}$ using the AMD ordering. It is clear the ordering contribution in reducing the fill-in of the factorization, minimizing the memory required to store $\mathbf{L}$ and the number of block-matrices operations.

As already mentioned in Section 3.3, variable reordering techniques, that consists in applying a suitable permutation to the source matrix, are able to reduce dramatically

the fill-in, allowing a faster decomposition. Several algorithm are proposed in literature to compute the proper permutation - e.g. AMD, COLAMD. For the Cholesky $LU$ decomposition it is also possible to evaluate the pattern of the $L$ matrix before computing the actual decomposition, through the *Symbolic Cholesky Decomposition*. The reader might appreciate the variable reordering contribution in Figure 5.1.

In the remaining of the Section, it will be given a more detailed description on how to solve a sparse linear system using the Cholesky decomposition of the matrix $\mathbf{H}$.

### Storage Methods for Sparse Matrices

Another core aspect of sparse matrices is that there are several techniques to *store* them more efficiently, reducing the amount of memory needed. The most general ones are *Compressed Row Storage* (CRS) or *Compressed Column Storage* (CCS), since they do not make any assumption on the structure of the matrix but the do not store unnecessary elements - i.e. the zeros. Those method store the matrix using only 3 vectors: one vector for floating point numbers that represent the non-zero entries and two for the column and row indexes respectively. As an example, given a matrix $A$

$$A = \begin{pmatrix} 10 & 0 & 0 & 0 & -2 & 0 \\ 3 & 9 & 0 & 0 & 0 & 3 \\ 0 & 7 & 8 & 7 & 0 & 0 \\ 3 & 0 & 8 & 7 & 5 & 0 \\ 0 & 8 & 0 & 9 & 9 & 13 \\ 0 & 4 & 0 & 0 & 2 & -1 \end{pmatrix}$$

in CRS format is represented by the following vectors - using zero-based indexing:

$$\begin{aligned} val &= \begin{bmatrix} 10 & -2 & 3 & 9 & 3 & 7 & \cdots & 2 & -1 \end{bmatrix} \\ col &= \begin{bmatrix} 0 & 4 & 0 & 1 & 5 & 1 & \cdots & 4 & 5 \end{bmatrix} \\ row &= \begin{bmatrix} 0 & 2 & 5 & 8 & 12 & 16 & 19 \end{bmatrix} \end{aligned}$$

The vector $row$ contains the indexes of the elements that correspond to the first non-zero entry of each row in the sparse matrix A.

*List of Lists* (LIL) represents another effective and easy-to-implement storage method. In this case, each row-vector is represented as a list of pairs that denotes the column index and the element's value.

Recalling equation 3.24, since in this problem it has been considered only *binary constraints*, the linearization of each edge generates 4 block-contributions to the Hessian, namely:

$$\mathbf{H}_{ii} = \mathbf{J}_i^T \Omega_k \mathbf{J}_i \qquad \mathbf{H}_{jj} = \mathbf{J}_j^T \Omega_k \mathbf{J}_j$$
$$\mathbf{H}_{ij} = \mathbf{J}_i^T \Omega_k \mathbf{J}_j \qquad \mathbf{H}_{ji} = \mathbf{J}_j^T \Omega_k \mathbf{J}_i$$

Therefore, the Hessian can be intended to be a *sparse block-matrix* where each entry is a matrix itself of a given size. In our work, it has been chosen to employ the LIL storage method with block-entries, in order to speed-up the numerical computations - better explained in the next Chapter.

### Cholesky Decomposition

In linear algebra, the Cholesky decomposition (or factorization) is the decomposition of a *Hermitian*, positive semi-definite (PSD) matrix into the product of a *lower triangular matrix* and its *conjugate transpose*, namely:

$$\mathbf{A} = \mathbf{L}\,\mathbf{L}^\star \tag{5.1}$$

In this particular problem formulation, the matrices involved are composed by real numbers, therefore the conjugate transpose of $\mathbf{L}$ becomes its transposed $\mathbf{L}^T = \mathbf{U}$. In this sense, it represents the square-root operator for symmetric PSD matrices.

A more stable variant of the classical Cholesky decomposition is the **LDL** decomposition. In this case, the original matrix is decomposed into the following product:

$$\mathbf{A} = \mathbf{L}\mathbf{D}\mathbf{L}^\star \tag{5.2}$$

where $\mathbf{L}$ is a lower *unit* triangular matrix - i.e. all the entries on the main diagonal are 1 - and $\mathbf{D}$ a diagonal matrix. This variant requires the same space and computational effort with respect to the original one but avoids the square-roots extraction. In this way, even matrices that do not have a Cholesky decomposition can be factorized with the *LDL* one. However, in this work, since the matrices take into account are symmetric and PSD by construction, it has been chosen the original Cholesky decomposition.

In general the computational complexity for the factorization of a $(n \times n)$ matrix is $O(n^3)$, requiring about $n^3/3$ FLOPs. There are several algorithm available to compute the factorization, however, one of the most common is the *Cholesky-Banachiewicz*. In this algorithm the computation starts from the top-left corner of the matrix $\mathbf{L}$ and proceeds the computation row-by-row as follows:

$$\mathbf{A} = \mathbf{L}\mathbf{L}^T = \begin{pmatrix} L_{00} & 0 & 0 \\ L_{10} & L_{11} & 0 \\ L_{20} & L_{12} & L_{22} \end{pmatrix} \begin{pmatrix} L_{00} & L_{10} & L_{20} \\ 0 & L_{11} & L_{21} \\ 0 & 0 & L_{22} \end{pmatrix}$$

where

$$\begin{cases} L_{jj} = \sqrt{A_{jj} - \sum_{k=1}^{j-1} L_{jk}^2} \\ L_{ij} = \frac{1}{L_{jj}} \left[ A_{ij} - \sum_{k=1}^{j-1} L_{ik} L_{jk} \right] \qquad \text{for} \quad i > j \end{cases} \tag{5.3}$$

The *Cholesky-Crout*'s algorithm, instead, is a column-wise version of the previous one. It is good to notice that both algorithms allow to perform the computation also *in-place*. Moreover, both algorithms can be employed in sparse block-matrices, leading to a block-Cholesky factorization. The blocks are also computed as in Equation 5.3 but, in the block case, the *square root* operator is applied to the matrix-block and it consists in the *Cholesky decomposition of the block* itself.

As it has been already mentioned, the main use of the Cholesky decomposition is in the solution of linear systems. Given a symmetric PSD real matrix $\mathbf{A}$, the solution of the linear system $\mathbf{A}\,\mathbf{x} = \mathbf{b}$ is computed through the following steps:

1. **Cholesky factorization** of the source matrix $\mathbf{A} = \mathbf{L}\mathbf{L}^T$
2. **Forward substitution** to solve the linear system $\mathbf{L}\,\mathbf{y} = \mathbf{b}$
3. **Backward substitution** to solve the linear system $\mathbf{L}^T\,\mathbf{x} = \mathbf{y}$

Clearly, since in the problem in analysis $\mathbf{H}$ and its factorization $\mathbf{L}$ are sparse *block-matrices*, also $\mathbf{b}$ and $\mathbf{y}$ are dense *block-vector* - with the number of blocks $N$ equal to the number of vertexes in the graph.

## Manifold Representation

As it has been already stated at the begin of the Chapter, this work focuses on 3D formulations of pose-graph and bundle adjustment. In this Section it will be better analyzed the representation of all the objects required for the LS estimation in both the formulations.

### 3D Pose-Graph

Pose-graph optimization in 3D represents the backbone of SLAM, allowing to estimate the robot trajectory *in space* through MAP estimation. In this formulation, the state $\mathbf{x}$ includes the 3D orientation of the nodes which represent the main reason why pose-SLAM is a complex problem. Rotations in space can be over-parametrized through a 3D rotation matrix $\mathbf{R} \in SO(3)$. Therefore, the over-parametrized state can be represented by a 3D isometry $\mathbf{X} = {}^{W}T_R \in SE(3)$ which represents the robot pose in the world reference frame - i.e. a $(4 \times 4)$ homogeneous transformation matrix.

A possible *minimal representation* of the state can be a 6 vector $\mathbf{x} = (t_x\, t_y\, t_z\, \alpha\, \beta\, \gamma)^T$, where the triplet $\mathbf{r} = (\alpha\, \beta\, \gamma)^T$ represents the Euler Angles that compose the rotational

part of the isometry, while $\mathbf{t} = (t_x\, t_y\, t_z)^T$ is the translational one. Summarizing, in formulæ:

$$\mathbf{X} = \begin{pmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^T & 1 \end{pmatrix} \qquad \mathbf{x} = \begin{bmatrix} t_x & t_y & t_z & \alpha & \beta & \gamma \end{bmatrix}^T$$

where

$$\mathbf{R} = \mathbf{R}_x(\alpha)\, \mathbf{R}_y(\beta)\, \mathbf{R}_z(\gamma) \tag{5.4}$$

The measurements are also of type *pose*, thus, it is possible to use the same notation that describes the state. Therefore $\mathbf{Z}_{ij}$ represents the over-parametrized measurement of node $j$ with respect to node $i$ - i.e. a 3D isometry ${}^iT_j$.

The next required step concerns the definitions of suitable operators *box-plus* and *box-minus*. We introduce again the operators *v2t* and *t2v* that allow to map the over-parametrized representation into the minimal one and vice versa. Those two operators allow to define the following relations:

$$\mathbf{X} \boxplus \Delta\mathbf{x} = \text{v2t}(\Delta\mathbf{x})\, \mathbf{X} \tag{5.5}$$

$$\mathbf{X}_a \boxminus \mathbf{X}_b = \text{t2v}\left(\mathbf{X}_b^{-1}\mathbf{X}_a\right) \tag{5.6}$$

For $SE(3)$ object, the *v2t* function computes the rotational part of $\mathbf{X}$ through Equation 5.4 and then composes the isometry adding the translational part $\mathbf{t} = (t_x\, t_y\, t_z)^T$. In Equation 5.4 the factors $\mathbf{R}_x$, $\mathbf{R}_y$ and $\mathbf{R}_z$ represent the 3D rotation respectively around the $x$, $y$ and $z$ axis; they are defined as follows:

$$\mathbf{R}_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix} \tag{5.7}$$

$$\mathbf{R}_y(\beta) = \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix} \tag{5.8}$$

$$\mathbf{R}_z(\gamma) = \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.9}$$

Instead, to retrieve the Euler given the rotation matrix $\mathbf{R}$ - that is done in the *t2v* operator - it is necessary to equate each element in $\mathbf{R}$ with its corresponding element in the matrix product $\mathbf{R}_x(\alpha)\, \mathbf{R}_y(\beta)\, \mathbf{R}_z(\gamma)$, in formulæ:

$$\mathbf{R} = \begin{bmatrix} R_{00} & R_{01} & R_{02} \\ R_{10} & R_{11} & R_{12} \\ R_{20} & R_{21} & R_{22} \end{bmatrix} = \mathbf{R}_x(\alpha)\,\mathbf{R}_y(\beta)\,\mathbf{R}_z(\gamma) = \tag{5.10}$$
$$= \begin{bmatrix} \cos\beta\,\cos\gamma & -\cos\beta\,\sin\gamma & \sin\beta \\ \cos\alpha\,\sin\gamma + \sin\alpha\,\cos\gamma\,\sin\beta & \cos\alpha\,\cos\gamma - \sin\alpha\,\sin\beta\,\sin\gamma & -\cos\beta\,\sin\alpha \\ \sin\alpha\,\sin\gamma - \cos\alpha\,\cos\gamma\,\sin\beta & \sin\alpha\,\cos\gamma + \cos\alpha\,\sin\beta\,\sin\gamma & \cos\beta\,\cos\alpha \end{bmatrix}$$

The reader might notice the non-linearities introduced by the functions t2v and v2t. Those imply the computation of complex non-linear derivatives to retrieve the Jacobian in the linearization phase. However, this problem - and the solution proposed in this work - will be better analyzed in the next Section.

### 3D Bundle Adjustment

In this formulation, as already explained in Section 4.3, the system has to estimate both the robot trajectory and the position of salient world points - i.e. the 3D landmarks. Therefore, the system's *state* and *increment* are described as follows:

$$\mathbf{X} = \left( \overbrace{\mathbf{X}_1^R, ..., \mathbf{X}_N^R}^{Nposes} \mid \overbrace{\mathbf{x}_{N+1}^L, ..., \mathbf{x}_{N+M}^L}^{Mlandmarks} \right)$$
$$\Delta\mathbf{x} = \left( \overbrace{\Delta\mathbf{x}_1^R, ..., \Delta\mathbf{x}_N^R}^{Nposes} \mid \overbrace{\Delta\mathbf{x}_{N+1}^L, ..., \Delta\mathbf{x}_{N+M}^L}^{Mlandmarks} \right)$$

The formalization for $SE(3)$ nodes remains unchanged from the previous Sub-Section, therefore, it is necessary to characterize only the nodes that describe the landmarks.

Landmarks' nodes lie on $\mathbb{R}^3$, so it is not necessary to define anything else - i.e. no *box-plus/box-minus* operator needed. As a consequence of this, a measurement $\mathbf{z}_{ij} \in \mathbb{R}^3$ is a simple 3 vector that describes the position of point $j$ in the $i$-th pose reference frame.

However, it is necessary to consider also the fact that the sensor's reference frame and the robot's one might not coincide, but they are related through the transformation $\mathbf{S} = {}^R T_S \in SE(3)$. Thus, given the state, the *predicted measurement* is:

$$\tilde{\mathbf{z}}_{ij} = h_{ij}(\mathbf{X}) = \underbrace{\mathbf{S}^{-1}\,\mathbf{X}_i^{-1}}_{\mathbf{K}}\,\mathbf{p}_j = \mathbf{R}_K\mathbf{p}_j + \mathbf{t}_K \tag{5.11}$$

In light of this, without loss of generality, the error between the predicted and the actual measurement is computed as follows:

$$\mathbf{e}_{ij} = \tilde{\mathbf{z}}_{ij} - \mathbf{z}_{ij} = \mathbf{S}^{-1}\mathbf{X}_i^{-1}\mathbf{p}_j - \mathbf{z}_{ij} \tag{5.12}$$

Given Equation 5.12, the perturbed error will be:

$$\mathbf{e}_{ij}(\mathbf{X}_i \boxplus \Delta\mathbf{x}_i^R, \mathbf{x}_j + \Delta\mathbf{x}_j^L) = \mathbf{S}^{-1}\left[\left(\text{v2t}(\Delta\mathbf{x}_i^R)\mathbf{X}_i\right)^{-1}\left(\mathbf{p}_j + \Delta\mathbf{x}_j^L\right)\right] - \mathbf{z}_{ij} \tag{5.13}$$

Finally, it is necessary to compute the Jacobian $\mathbf{J}$ deriving from the constraint $\mathbf{z}_{ij}$, that is structured as follows:

$$\mathbf{J} = [\mathbf{0} \cdots \mathbf{0}\ \mathbf{J}_R\ \mathbf{0} \cdots \mathbf{0}\ \mathbf{J}_L\ \mathbf{0} \cdots \mathbf{0}]$$

where

$$\begin{aligned}
\mathbf{J}_R &= \frac{\partial\, \mathbf{S}^{-1}\left[\left(\text{v2t}(\Delta\mathbf{x}_i^R)\mathbf{X}_i\right)^{-1}\left(\mathbf{p}_j + \Delta\mathbf{x}_j^L\right) - \mathbf{z}_j\right]}{\partial \Delta\mathbf{x}_i^R}\bigg|_{\substack{\Delta\mathbf{x}_i^R = 0 \\ \Delta\mathbf{x}_j^L = 0}} = \\
&= \frac{\partial\, \left[\mathbf{R}_S^T\left[\text{v2t}(\Delta\mathbf{x}_i^R)\mathbf{X}_i\right]^{-1}\mathbf{p}_j - \mathbf{R}_S^T\mathbf{t}_S\right]}{\partial \Delta\mathbf{x}_i^R}\bigg|_{\substack{\Delta\mathbf{x}_i^R = 0 \\ \Delta\mathbf{x}_j^L = 0}} = \\
&= \frac{\partial\, \mathbf{R}_S^T\mathbf{X}_i^{-1}\left[\text{v2t}(\Delta\mathbf{x}_i^R)\right]^{-1}\mathbf{p}_j}{\partial \Delta\mathbf{x}_i^R}\bigg|_{\substack{\Delta\mathbf{x}_i^R = 0 \\ \Delta\mathbf{x}_j^L = 0}} = \\
&= \frac{\partial\, \left[\mathbf{R}_S^T\mathbf{R}_R^T\left[\text{v2t}(\Delta\mathbf{x}_i^R)\right]^{-1}\mathbf{p}_j - \mathbf{R}_R^T\mathbf{t}_R\right]}{\partial \Delta\mathbf{x}_i^R}\bigg|_{\substack{\Delta\mathbf{x}_i^R = 0 \\ \Delta\mathbf{x}_j^L = 0}} = \\
&= \frac{\partial\, \mathbf{R}_S^T\mathbf{R}_R^T\left(\left[\text{v2t}(\Delta\mathbf{x}_i^R)\right]^{-1}\mathbf{p}_j\right)}{\partial \Delta\mathbf{x}_i^R}\bigg|_{\substack{\Delta\mathbf{x}_i^R = 0 \\ \Delta\mathbf{x}_j^L = 0}} = \mathbf{R}_S^T\mathbf{R}_R^T\frac{\partial\left[\mathbf{R}_{\Delta\mathbf{x}_i^R}\mathbf{p}_j - \mathbf{R}_{\Delta\mathbf{x}_i^R}\mathbf{t}_{\Delta\mathbf{x}_i^R}\right]}{\partial \Delta\mathbf{x}_i^R}\bigg|_{\substack{\Delta\mathbf{x}_i^R = 0 \\ \Delta\mathbf{x}_j^L = 0}}
\end{aligned}$$

Exploiting the fact that the derivative expressed in the previous equation is evaluated in $\Delta\mathbf{x}_i^R = 0$, it leads to the following relation:

$$\mathbf{J}_R = \mathbf{R}_S^T\mathbf{R}_R^T\left[-I_{3\times 3} \quad | \quad -\lfloor\mathbf{p}_j\rfloor_\times\right] \tag{5.14}$$

The other component of $\mathbf{J}$ - namely $\mathbf{J}_L$ - instead can be computed as follows:

$$\mathbf{J}_L = \frac{\partial \mathbf{S}^{-1}\left[\left(\text{v2t}(\Delta\mathbf{x}_i^R)\mathbf{X}_i\right)^{-1}\left(\mathbf{p}_j + \Delta\mathbf{x}_j^L\right) - \mathbf{z}_j\right]}{\partial \Delta\mathbf{x}_j^L}\Bigg|_{\substack{\Delta\mathbf{x}_i^R = 0 \\ \Delta\mathbf{x}_j^L = 0}} =$$

$$= \frac{\partial\left[\mathbf{R}_S^T\,\mathbf{X}_i^{-1}\left(\mathbf{p}_j + \Delta\mathbf{x}_j^L\right) - \mathbf{R}_S^T\,\mathbf{t}_S\right]}{\partial \Delta\mathbf{x}_j^L}\Bigg|_{\substack{\Delta\mathbf{x}_i^R = 0 \\ \Delta\mathbf{x}_j^L = 0}} =$$

$$= \frac{\partial\left[\left(\mathbf{R}_S^T\,\mathbf{X}_i^{-1}\mathbf{p}_j\right) + \left(\mathbf{R}_S^T\,\mathbf{X}_i^{-1}\Delta\mathbf{x}_j^L\right)\right]}{\partial \Delta\mathbf{x}_j^L}\Bigg|_{\substack{\Delta\mathbf{x}_i^R = 0 \\ \Delta\mathbf{x}_j^L = 0}} = \mathbf{R}_S^T\,\frac{\partial\left[\mathbf{R}_R^T\,\Delta\mathbf{x}_j^L - \mathbf{R}_R^T\mathbf{t}_R\right]}{\partial \Delta\mathbf{x}_j^L}\Bigg|_{\substack{\Delta\mathbf{x}_i^R = 0 \\ \Delta\mathbf{x}_j^L = 0}}$$

This result - expanding the derivatives and exploiting that the linearization point is $\Delta\mathbf{x}_j^L = 0$ - leads to the following relation:

$$\mathbf{J}_L = \mathbf{R}_S^T\,\mathbf{R}_R^T \tag{5.15}$$

The reader might notice that $\mathbf{J}_R$ is a $(3 \times 6)$ matrix - since the minimal representation of $SE(3)$ states has 6 components - while $\mathbf{J}_L$ is a $(3 \times 3)$ matrix - because $\mathbb{R}^3$ states are vectors with only 3 components.

In the next Section it is proposed a deeper analysis on the error representation and the linearization of 3D pose constraints, highlighting the non-linearity of the computation and the proposed approach to overcome this issue.

## Dealing with SE3 Objects

3D poses are complex objects to manage, due to their rotational part that introduces many an highly non-linear contribution in the linearization process. In this section it is proposed an approach that aims to reduce those non-linearities while delivering performances comparable to other state-of-the-art systems.

### Chordal Distance Based Error Function

Recalling 5.2.1, we have defined the functions v2t and t2v that allow to map the minimal representation of the state into the redundant one and vice-versa. Through them, we defined the operators *box-plus* and *box-minus*, described respectively in Equations 5.5 and 5.6. Sticking to this notion, the *predicted measurement* $\hat{\mathbf{Z}}_{ij}$ of pose $j$ from pose $i$ is computed as

$$\hat{\mathbf{Z}}_{ij} = h_{ij}(\mathbf{X}) = \mathbf{X}_i^{-1}\mathbf{X}_j \tag{5.16}$$

that leads to the following error function:

$$\mathbf{e}_{ij}(\mathbf{X}) = \hat{\mathbf{Z}}_{ij} \boxminus \mathbf{Z}_{ij} = \text{t2v}\left(\mathbf{Z}_{ij}^{-1}\,\mathbf{X}_i^{-1}\,\mathbf{X}_j\right) \tag{5.17}$$

The error $\mathbf{e}_{ij}$ is a 6 vector that expresses the mismatch of each component of state's minimal representation. Proceeding with the error perturbation, the result will be:

$$\mathbf{e}_{ij}(\mathbf{X}_i \boxplus \Delta\mathbf{x}_i, \mathbf{X}_j \boxplus \Delta\mathbf{x}_j) = \text{t2v}\left(\mathbf{Z}_{ij}^{-1}\,(\text{v2t}(\Delta\mathbf{x}_i)\,\mathbf{X}_i)^{-1}\,(\text{v2t}(\Delta\mathbf{x}_j)\,\mathbf{X}_j)\right) \tag{5.18}$$

The full Jacobian $\mathbf{J} = [\mathbf{0}\ \cdots\ \mathbf{0}\ \mathbf{J}_i\ \mathbf{0}\ \cdots\ \mathbf{0}\ \mathbf{J}_j\ \mathbf{0}\ \cdots\ \mathbf{0}]$ will be quite complex to compute due to the derivative of the t2v and v2t functions, requiring also many FLOPs and, thus, slowing the optimization process. In this formulation $\mathbf{J}_i$ and $\mathbf{J}_j$ are $(6\times6)$ matrices.

It is possible to approach this problem using a different error formulation that leads to easy-to-compute derivatives. In order to do so, we define the $L_{p,q}$ norm of a $(m \times n)$ matrix $A$ as follows:

$$\|A\|_{p,q} = \left(\sum_{j=1}^{n}\left(\sum_{i=1}^{m}|a_{ij}|^p\right)^{q/p}\right)^{1/q}$$

For $p = q = 2$ the becomes

$$\|A\|_F = \left(\sum_{j=1}^{n}\left(\sum_{i=1}^{m}|a_{ij}|^2\right)^{2/2}\right)^{1/2} = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|a_{ij}|^2} \tag{5.19}$$

that represents the *Frobenius norm* of a matrix, an *entrywise* norm, which is also *invariant under rotation* constraint. Based on this concept, it is possible to define the *chordal distance* between two rotation matrix $\mathbf{R}_A$ and $\mathbf{R}_B$ as follows:

$$d_{\text{chord}}(\mathbf{R}_A, \mathbf{R}_B) = \|\mathbf{R}_A - \mathbf{R}_B\|_F \tag{5.20}$$

It is good to notice that the difference operator employed in Equation 5.20 is the standard Euclidean *minus*, executed element-wise.

We define also a new function, that given a 3D isometry returns a 12 vector made with its components, namely:

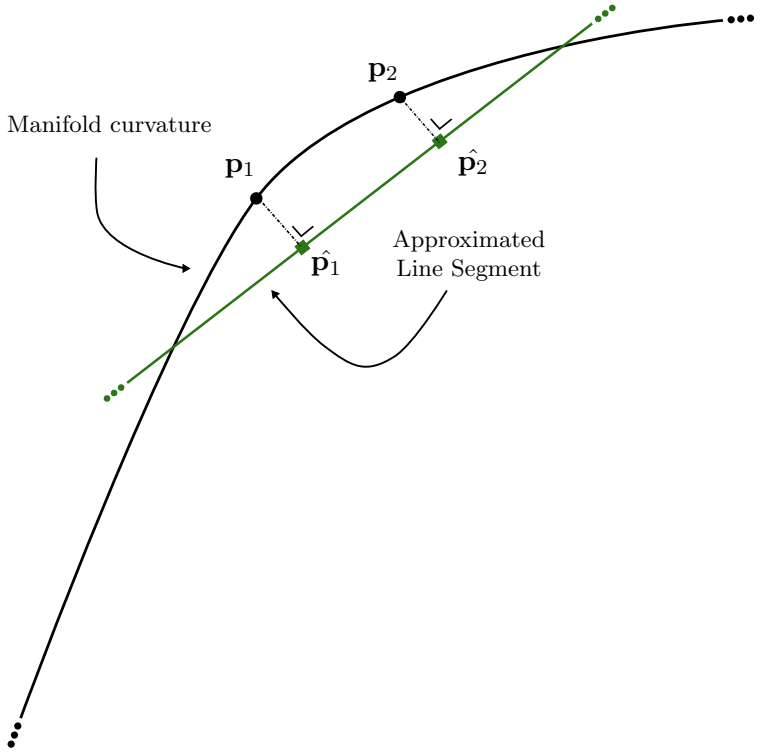



Figure 5.2: **Chordal Distance.** This figure shows the underlying concept of the new error function: the distance between $\mathbf{p}_1$ and $\mathbf{p}_2$ can be approximated with the Euclidean distance computed between the projection of those points onto the relative chord - namely between $\hat{\mathbf{p}}_1$ and $\hat{\mathbf{p}}_2$.

$$\begin{aligned} \mathbf{X} &= \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix} \\ \mathbf{R} &= \begin{pmatrix} \mathbf{r}_1 | \mathbf{r}_2 | \mathbf{r}_3 \end{pmatrix} \\ \text{flatten}(\mathbf{X}) &= \begin{pmatrix} \mathbf{r}_1 \\ \mathbf{r}_2 \\ \mathbf{r}_3 \\ \mathbf{t} \end{pmatrix} \end{aligned} \tag{5.21}$$

where $\mathbf{r}_k$ represents the $k$-th column of $\mathbf{R}$.

Finally, we introduce the following relations to express operators *box-plus* and *box-minus*:

$$\mathbf{X} \boxplus \Delta\mathbf{x} = \text{v2t}(\Delta\mathbf{x})\, \mathbf{X} \tag{5.22}$$

$$\mathbf{X}_a \boxminus \mathbf{X}_b = \text{flatten}\left(\mathbf{X}_a\right) - \text{flatten}\left(\mathbf{X}_b\right) \tag{5.23}$$

Given those mathematical concepts, it is possible to define the error between two $SE(3)$ objects through a relaxed version of the chordal distance, that leads to the following relations:

$$\hat{\mathbf{Z}}_{ij} = h_{ij}(\mathbf{X}) = \text{flatten}\left(\mathbf{X}_i^{-1}\,\mathbf{X}_j\right) \tag{5.24}$$

$$\mathbf{e}_{ij}(\mathbf{X}) = \hat{\mathbf{Z}}_{ij} - \mathbf{Z}_{ij} = \text{flatten}\left(\mathbf{X}_i^{-1}\,\mathbf{X}_j\right) - \text{flatten}\left(\mathbf{Z}_{ij}\right) \tag{5.25}$$

It is good to notice that $\mathbf{e}_{ij}$ now is a 12 vector, therefore, the Jacobian's components $\mathbf{J}_i$ and $\mathbf{J}_j$ will be $(12 \times 6)$ matrices. Speaking about this, applying the state perturbation to the error will lead to the following relation:

$$\mathbf{e}_{ij}(\mathbf{X}_i \boxplus \Delta\mathbf{x}_i, \mathbf{X}_j \boxplus \Delta\mathbf{x}_j) = \text{flatten}\left(\left(\text{v2t}(\Delta\mathbf{x}_i)\mathbf{X}_i\right)^{-1}\,\left(\text{v2t}(\Delta\mathbf{x}_j)\mathbf{X}_j\right)\right) - \text{flatten}\left(\mathbf{Z}_{ij}\right) \tag{5.26}$$

It is already possible to notice the reduced complexity of the derivatives required to linearize the constraint. In fact, from Equation 5.26 it is possible to retrieve the following relations - stating that $\mathbf{R}_{\Delta\mathbf{x}_i} = \mathbf{R}^x_{\Delta\mathbf{x}_i}\,\mathbf{R}^y_{\Delta\mathbf{x}_i}\,\mathbf{R}^z_{\Delta\mathbf{x}_i}$:

$$\mathbf{J}_j = \frac{\partial\,\mathbf{e}_{ij}(\mathbf{X}_i \boxplus \Delta\mathbf{x}_i, \mathbf{X}_j \boxplus \Delta\mathbf{x}_j)}{\partial\Delta\mathbf{x}_j}\bigg|_{\substack{\Delta\mathbf{x}_i = 0 \\ \Delta\mathbf{x}_j = 0}} =$$

$$= \frac{\partial\left[\text{flatten}\left(\overbrace{\begin{bmatrix}\mathbf{R}_i^T & -\mathbf{R}_i^T\mathbf{t}_i \\ \mathbf{0} & 1\end{bmatrix}\begin{bmatrix}\left(\mathbf{R}^x_{\Delta\mathbf{x}_i}\,\mathbf{R}^y_{\Delta\mathbf{x}_i}\,\mathbf{R}^z_{\Delta\mathbf{x}_i}\right)^T & -\mathbf{R}^T_{\Delta\mathbf{x}_i}\mathbf{t}_i \\ \mathbf{0} & 1\end{bmatrix}\begin{bmatrix}\mathbf{R}_j & \mathbf{t}_j \\ \mathbf{0} & 1\end{bmatrix}}^{\mathbf{G}}\right)\right]}{\partial\Delta\mathbf{x}_j}\bigg|_{\substack{\Delta\mathbf{x}_i = 0 \\ \Delta\mathbf{x}_j = 0}}$$

Expanding the derivatives, it is possible to define the following - intuitively retrieved - objects:

- $\mathbf{R}'_{x0}$, $\mathbf{R}'_{y0}$ and $\mathbf{R}'_{z0}$ that represent derivatives with respect to $\Delta\alpha$, $\Delta\beta$ and $\Delta\gamma$ of the base rotation $\mathbf{R}_k(\cdot)$, evaluated in 0 and with $k = \{x, y, z\}$;
- $\hat{\mathbf{R}}'_{x0}$, $\hat{\mathbf{R}}'_{y0}$, and $\hat{\mathbf{R}}'_{z0}$ that are the derivatives with respect to $\Delta\alpha$, $\Delta\beta$ and $\Delta\gamma$ of the rotational part of matrix $\mathbf{G}$, computed as $\hat{\mathbf{R}}'_{k0} = \mathbf{R}_i^T\,\mathbf{R}'_{k0}\,\mathbf{R}_j$ with $k = \{x, y, z\}$;
- $\hat{\mathbf{r}}'_{k0}$ which describes the 9 vector obtained stacking the columns of $\hat{\mathbf{R}}'_{k0}$ - with $k = \{x, y, z\}$.

In the light of what we just stated, it is possible to retrieve this final formulation of $\mathbf{J}_j$:

$$\mathbf{J}_j = \begin{pmatrix} \mathbf{0}_{(9\times3)} & \left[\hat{\mathbf{r}}'_{x0} \mid \hat{\mathbf{r}}'_{y0} \mid \hat{\mathbf{r}}'_{z0}\right]_{(9\times3)} \\ \mathbf{R}_i^T & -\mathbf{R}_i^T \lfloor \mathbf{t}_j \rfloor_\times \end{pmatrix} \tag{5.27}$$

Intuitively, the other component of the Jacobian will be simply

$$\mathbf{J}_i = -\mathbf{J}_j \tag{5.28}$$

The reader might noticed how the new error function reduced the computational cost of the derivatives, due to the elimination of the highly non-linear function t2v. Moreover, the use of standard Euclidean *minus* operator to express differences between transforms, leads to a better approximation by the first-order Taylor expansion of the perturbed error. This better approximation enlarges the converge basin of the optimization process and it improves the avoidance of local minima.

### Benefits in the Re-linearization

The reader might remember that, since also the measurements live on a non-Euclidean space, it is necessary to project the measurement information matrices through the *box-minus* operator. Therefore, using the standard *box-minus* described in Equation 5.6 and given $\hat{\mathbf{Z}}_{ij}$ associated to $\mathbf{Z}_{ij} = \mathbf{Z}_k$, it is required to compute the following objects **at each iteration** of the LS optimization:

$$\mathbf{J}_{\mathbf{Z}_k} = \left.\frac{\partial\left(\hat{\mathbf{Z}}_{ij} \boxminus \mathbf{Z}\right)}{\partial \mathbf{Z}}\right|_{\mathbf{Z}=\mathbf{Z}_k} \tag{5.29}$$

$$\tilde{\Omega}_k = \left(\mathbf{J}_{\mathbf{Z}_k}\, \Omega_k\, \mathbf{J}_{\mathbf{Z}_k}^T\right)^{-1} \tag{5.30}$$

The error function based on the chordal distance that has been proposed in the previous Sub-section, brings benefits also in this sense. In fact, since the operator employed to compute the error is the standard Euclidean *minus*, it is not required to recompute the information matrix $\Omega_k$ at each iteration.

It is good to notice that the information matrix associated with the measurement $\mathbf{Z}_k$ is a $(6 \times 6)$ matrix $\Omega_k$ , since the state's minimal representation has dimension 6 in this formalization. However, it is necessary to consider the contribute of function flatten $(\cdot)$ in this process: the new error space has dimension 12, thus it is necessary to project the information matrix into the new higher dimensional space. To this end, it will be proposed now some new mathematical concepts useful in order to estimate the result of applying a non-linear transformation to a probability distribution function.

Given a Gaussian distribution $p(x) = \mathcal{N}(x; \mu, \Sigma)$ with mean $\mu$ and covariance $\Sigma$, it is possible to represent it through a set of weighted points called *Sigma Points*. Each Sigma Point $\chi^i$ is described by a set of parameters:

- a **position** $x^i$
- a **weight for the mean** $w_m^i \in \mathbb{R}^+$
- a **weight for the covariance** $w_c^i \in \mathbb{R}^+$

Clearly, the conversion from the original parameters $(\mu, \Sigma)$ to the Sigma Points $\chi^i = (x^i, w_m^i, w_c^i)$ must be invertible. In fact, it is possible to reconstruct the Gaussian parameters from the Sigma Points as follows:

$$\mu = \sum_i w_m^i \, x^i \tag{5.31}$$

$$\Sigma = \sum_i w_c^i \, (x^i - \mu)(x^i - \mu)^T \tag{5.32}$$

It is good to notice that, if the original Gaussian distribution has dimension $n$, a suitable number of Sigma Point required to approximate it without loosing information will be $N = 2n + 1$. In order to control how far the Sigma Points are sampled from the mean $\mu$ though, it is possible to tune the scalar parameters $\kappa \in \mathbb{R}^+$ and $\alpha \in (0, 1]$. Therefore, the position of each Sigma Point is computed as follows:

$$x^{(0)} = \mu \tag{5.33}$$

$$x^{(i)} = \begin{cases} \mu + [L]_i & \text{for } i \in [1 \dots n] \\ \mu - [L]_{n-i} & \text{for } i \in [n+1 \dots 2n] \end{cases} \tag{5.34}$$

where $L = \sqrt{(n + \lambda)\Sigma}$ and $\lambda = \alpha^2(n + \kappa) - n$. Given the scalar parameter $\beta = 2$ - tuned for Gaussian PDFs - the weights are retrieved as follows:

$$w_m^{(0)} = \frac{\lambda}{\lambda + n} \tag{5.35}$$

$$w_c^{(0)} = w_m^{(0)} + (1 - \alpha^2 + \beta) \tag{5.36}$$

$$w_c^{(i)} = w_m^{(i)} = \frac{1}{2(n + \lambda)} \tag{5.37}$$

This transformation between actual Gaussian parameters and the Sigma Points is called **Unscented Transform** [33]. The core feature of this mathematical function is that it can be used to apply a transformation to a PDF in a straightforward way. Sticking to Gaussian PDFs, given $X_a \sim \mathcal{N}(x_a; \mu_a, \Sigma_a)$ and its Unscented Transform $x_a \sim \mathcal{UT}(x_a; x_a^{(i)}, w_m^{(i)}, w_c^{(i)})$, computing the Gaussian $X_b = g(X_a)$ can require a lot of effort. However, a good approximation of $X_b$ can be computed applying the function $g(\cdot)$ to the Unscented Transformation of $X_a$. This translates in the application of the $g(\cdot)$ function to each Sigma Point of $x_a$, namely

$$\chi_b^{(i)} = g\left(\chi_a^{(i)}\right) \qquad i = 1 \dots N \tag{5.38}$$

Going back to our problem, it is possible to transform the information matrix of each measurement $\Omega_k$ through the Unscented Transform. The required steps are the following:

1. Compute the $N = 2n + 1$ Sigma Points $\chi_{original}^{(i)} = (x_{original}^{(i)}, w_m^{(i)}, w_c^{(i)})$ from $\mathcal{N}(\mu_k, \Sigma_k)$, where $n$ is the minimal representation's dimension (in this case $n = 6$), $\mu_k = \text{t2v}(\mathbf{Z}_k)$ and $\Sigma_k = \Omega_k^{-1}$.
2. Compute the new position of $\chi_{transformed}^{(i)}$ as $x_{transformed}^{(i)} = \text{flatten}\left(\text{v2t}(x_{original}^{(i)})\right)$.
3. Reconstruct the new Gaussian $\mathcal{N}(\bar{\mu}_k, \bar{\Sigma}_k)$ from the Sigma Points $\chi_{transformed}^{(i)}$, leaving the parameters $w_m^{(i)}$ and $w_c^{(i)}$ unchanged.

The adapted information matrix $\bar{\Omega}_k = \bar{\Sigma}_k^{-1}$ is a $(12 \times 12)$ matrix. However, since we are mapping a 6-dimensional space in a 12-dimensional space with only $N = 2n+1 = 13$ Sigma Points, it is possible to retrieve covariance matrix subject to (multiple) rank-loss. Therefore, it is necessary to add a non-zero scalar $\epsilon$ to the main diagonal of $\bar{\Sigma}_k$, in order to avoid numerical issues during its inversion.

Thanks to the Unscented Transform it is possible to have a fast and quite accurate approximation of the adapted information matrix, leading to consistent results from the optimization process. Moreover, since the minus operator employed in this formulation is the standard Euclidean one, the computation is performed just one time for each measurement $\mathbf{Z}_k$, reducing the computational effort of each optimization step.

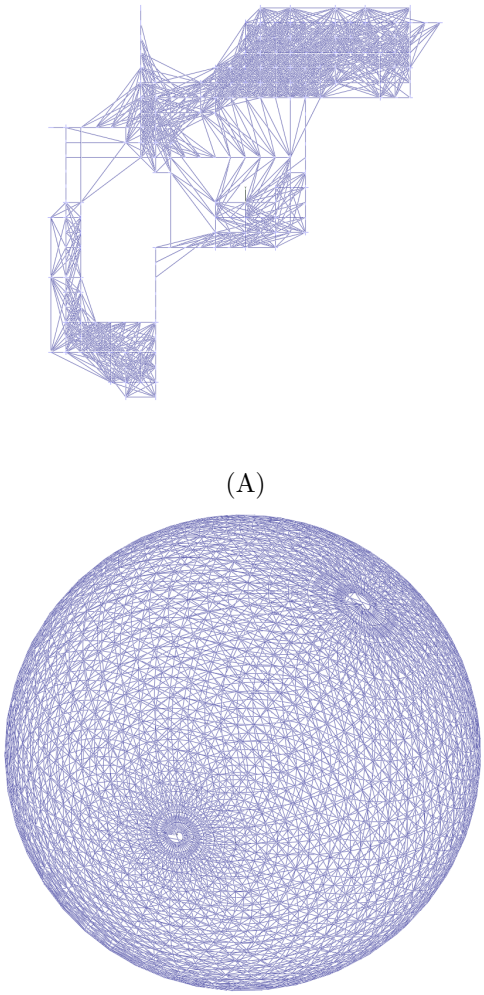
## Convergence Results

In the previous Sections, it has been proposed to the reader a complete overview of the theory underlying our optimization system. This Section, instead, proposes some tangible results brought by the novel approach described in this work.

To test the convergence of the system, we took two reference pose-graphs and we add to them different kind of noises in order to create multiple initial guess for the optimization algorithm. The graphs used are:

1. **Synthetic world:** that has 501 vertices and 4277 measurements (easier);
2. **Synthetic sphere:** 2500 vertexes and 9799 edges (hard).

Figure 5.3 shows the two graphs solved. Here we propose the comparison between our system and the state-of-the-art optimizer `g2o` [1].



(A)

(B)

Figure 5.3: **Solved Graphs** Top image: the synthetic world graph solved. Bottom: synthetic sphere solved.

The first test done consisted in applying the noise only on the *translational part* of the nodes; the employed graph is the synthetic world - Figure 5.3A. Obviously, both the system preformed well recreating the original graph, as shown in Figure 5.4.

However, our system claims to be more effective to manipulate rotations. Consequently, the second test focused on this: the initial guess, in fact, was created from the *synthetic world* graph adding white noise - i.e. sampled from $\mathcal{N}(0, 1)$ - *only* to the rotation $\mathbf{R}$ of each node.

With this initial guess, `g2o` suffers from the non-linearities introduced by the v2t and t2v functions as the reader might notice in Figure 5.5A. To confirm the qualitative result obtained, we decided to compare `g2o`'s `chi2` with the one obtained with our approach,

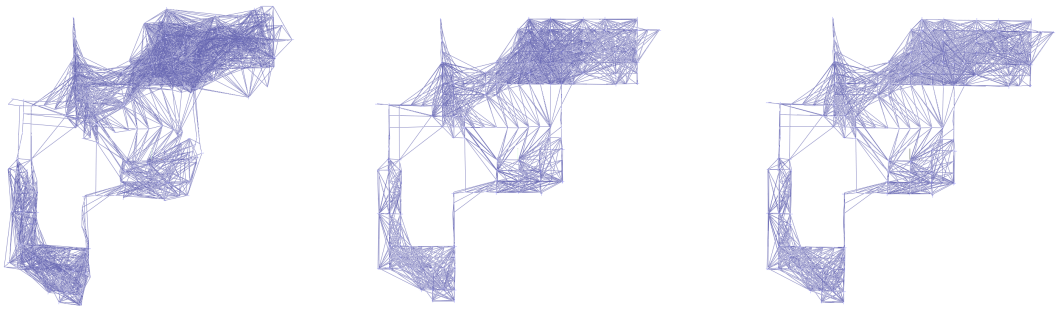
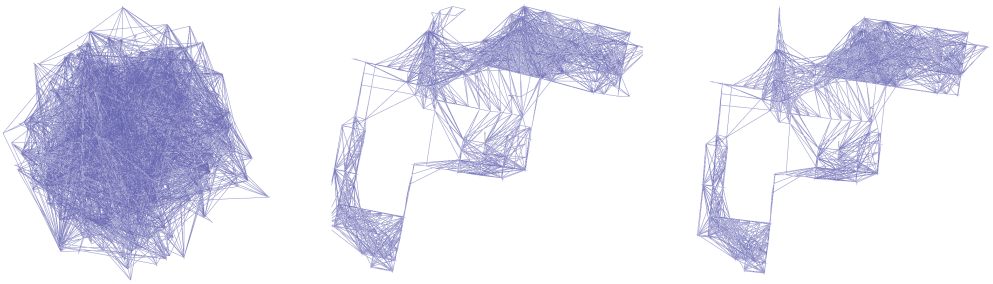
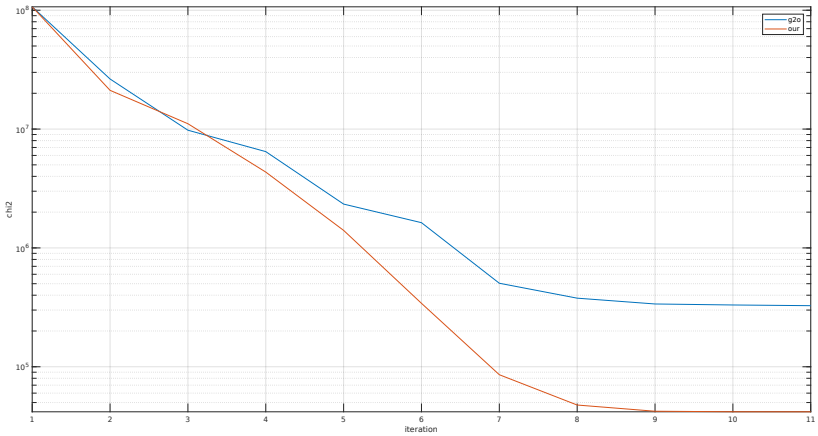


Figure 5.4: **Translational Noise.** From left to right: *initial guess*, `g2o` solution, *our* solution. Both the approaches generate consistent results.



(A)



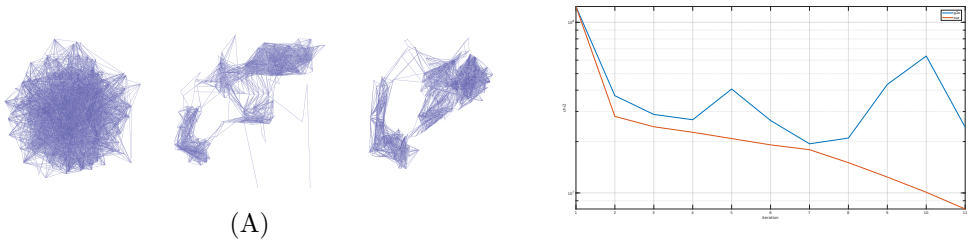

(B)

Figure 5.5: **Rotational Noise.** Figure (A) from left to right: *initial guess*, `g2o` solution, *our* solution. Figure (B) shows the `chi2` of both approaches at each iteration: it is clear the convergence gap between `g2o` - in blue - and our approach - in orange. The $y$ axis' scale is logarithmic. The iterations performed are 10, therefore, the point $x = 1$ indicates the `chi2` of the initial guess.

iteration by iteration. In order to obtain comparable objects - and to not bias the comparison - it has been saved the graph obtained with our system at each iteration and

evaluated its *initial* `chi2` in `g2o`. The result of this comparison is shown in figure 5.5B.

In order to further confirm the quality of the chosen approach, is has been generated another initial guess for the same graph, but the noise figure was sampled from $\mathcal{N}(0, 1000)$. In this case both the systems struggle to find a solution in only 10 iterations, but the trend is better with our error function - see Figure 5.6.



(A)

(B)

Figure 5.6: **Advanced Rotational Noise.** Figure (A) from left to right: *initial guess*, `g2o` solution, *our* solution. Figure (B) shows the `chi2` of both approaches at each iteration. `g2o`'s trend might indicate that the system got stuck in a local minimum.

The next step consists in applying both noise figures to the *sphere* graph, together with a translational perturbation (therefore a 6 DoF noise vector), generating a very harsh initial guess.

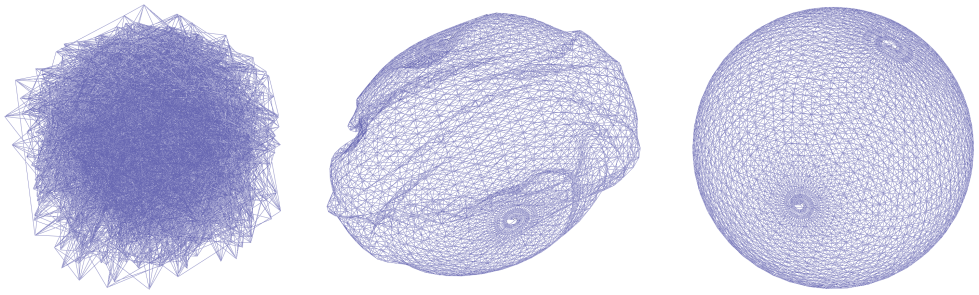
Given the complexity of the initial guess, we decided to perform more iterations with respect the previous graph - i.e. 100. Figure 5.7 shows the qualitative and quantitative results of both `g2o` and our approach, after applying 6 DoF Gaussian noise - sampled from $\mathcal{N}(0, 1)$. The reader might appreciate the fact that `g2o` converges to a *local minimum* far from the optimum, instead, our approach is able to **reach the optimum in less than 40 iterations**.

Finally, in the last test it has been increased the rotational component of the noise, which, in this extreme case, is sampled from $\mathcal{N}(0, 1)$. Here, without a kernel both the systems struggles to optimize the graph.

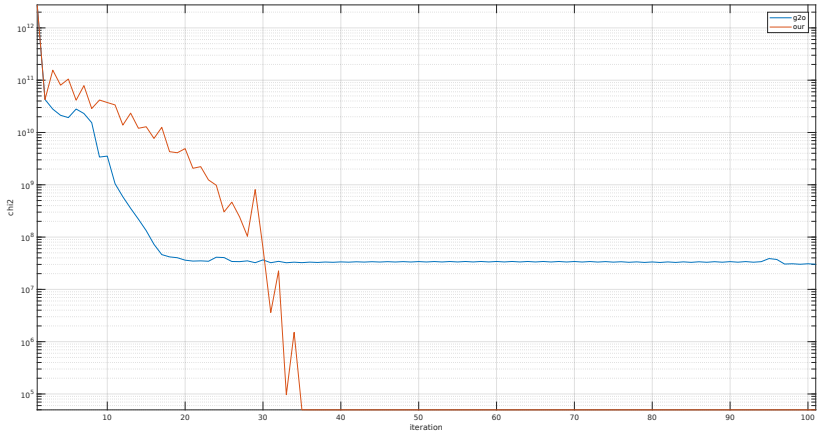
Figure 5.8 shows the outcomes of both the systems: clearly neither `g2o` nor our system reached the optimum, however the solution retrieved with the proposed method is evidently more consistent.

In Subsection 5.3.2, it has been highlighted that converting the information matrices from a 6-dimensional space to a 12-dimensional one through the *Unscented Transform* may generate multiple rank loss, thus, the matrix $\bar{\Sigma}_k = \bar{\Omega}_k^{-1}$ must be conditioned somehow to avoid singularities. Given this, it has been tested the effects of different conditioning methods, however the most promising ones where basically 3:

1. **Soft Conditioning.** It is computed the *Singular Value Decomposition* of matrix $\bar{\Sigma}_k = UDV^\star$, then it is added a *non-zero* value $\epsilon$ only to the degenerated eigenvalues and finally it is computed the matrix $\bar{\Sigma}_k^{conditioned} = U\tilde{D}V^\star$.

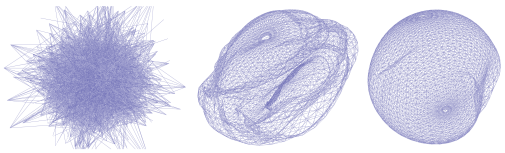


(A)

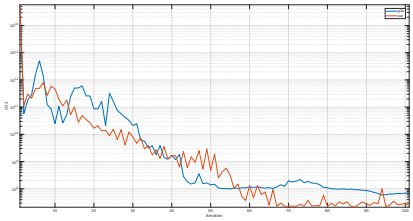



(B)

Figure 5.7: **Sphere 6 DoF Noise.** Figure (A) from left to right: *initial guess*, `g2o` solution, *our* solution. Figure (B) shows the `chi2` of both approaches at each iteration. From the plot it is clear that `g2o` converges to a local minimum distant from the optimum. Here our approach instead is able to converge to the proper solution in less than 40 iterations.



(A)



(B)

Figure 5.8: **Advanced Rotational Noise.** Figure (A) from left to right: *initial guess*, `g2o` solution, *our* solution. Figure (B) shows the `chi2` of both approaches at each iteration. Even if both systems fail in reaching the optimum, our system is able to generate a fair solution that can be further refined to reach the proper convergence.

2. **Mid Conditioning.** This is the method actually used to obtain the results seen

until now, and consists in applying a non-zero value $\epsilon$ to all the elements on the diagonal of $\bar{\Sigma}_k$. Reasonable values are $\epsilon \in [10^{-2}, 10^{-4}]$.

3. **Hard Conditioning.** In this case, all the values outside the main diagonal are simply ignored - i.e. set to zero.

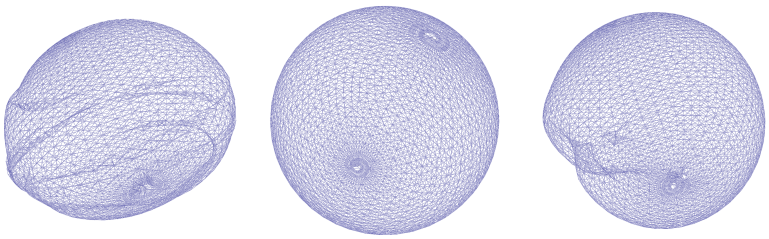


Figure 5.9: **Conditioning Methods.** The outcomes of the three conditioning methods used to compute the measurements' information matrix. Given the initial guess relative to Figure 5.7A, from left to right are illustrated: soft, mid and hard conditioning.

This comparison opens a new path for further investigations, in order to retrieve the best conditioning method to compute the measurements' information matrices.

As this Section showed, the novel features introduced so far lead to an accurate and more robust optimizer, that delivers results **comparable or better** to other state-of-the-art systems. In the next Chapter, the reader will have an insight on the actual implementation of the system, in order to better understand how all those mathematical concepts have been applied in practice and to show the speed performances reached by our system.

Chapter 6

# Software Implementation of the Optimizer

This Chapter will better analyze the actual implementation of a 3D Optimizer. In particular, it has been developed a self-consistent C++ library that provides all the tools needed to create and optimize 3D graphs that contain *pose* or *point* objects. The main components of the system are basically two:

1. The `Optimizer` itself that runs the Gauss-Newton algorithm to retrieve the best state configuration given the constraints.
2. The `Graph`, that contains the actual nodes and edges generated by a suitable front-end or read from file.

While the `Graph` is just a *container* for nodes and edges, the optimizer has to perform several computations in order to retrieve the linear system $\mathbf{H}\Delta\mathbf{x} = -\mathbf{b}$ and then solve it. Therefore, underlying the optimizer a linear solver is required to efficiently solve the aforementioned system.

Once that the front-end populates the graph (or it is loaded from a file), it is fed as input of the optimizer; the general work-flow of a graph optimizer is the following:

1. **Linearization**: for each edge it is computed the error with respect to the current system state and then the relative Jacobian. The output of this phase is the contribution of the edge to the Hessian matrix and the right-hand-side vector.
2. **Permutation**: once that the Hessian $\mathbf{H}$ is built, it is necessary to compute a proper permutation to reduce Cholesky factorization's fill-in and apply it to Hessian and the right-hand-side vector.
3. **Linear Solver**: now it is possible to compute the actual Cholesky factorization $\mathbf{L}$ and retrieve $\Delta\mathbf{x}$ via *Forward/Backward Substitution.*
4. **Update**: finally, once that $\Delta\mathbf{x}$ is computed, it is possible to apply it to the current system state - i.e. to all the graph's `Vertex`.

It is good to notice that our system is almost completely self-contained: the only external libraries employed are *Eigen* [34] - to efficiently manage small matrices - and *SuiteSparse* [35] - in order to compute the right permutation of the Hessian. Furthermore, the library has been developed with simplicity and keeping a minimalistic approach, in order to be comprehensible also by researchers that are not expert in this field. Just for comparison, the entire library has less than 6 thousand lines of code; other state-of-the-art systems are deployed in huge libraries with thousands of lines of code - e.g. `g2o` [1] is over 40 thousands lines of code, `Ceres` [36] is over 90 thousands lines of code and `gtsam` [19] over 300 thousands lines of code.

## Graph

The `Graph` is basically constituted by a collection of edges and nodes. The nodes can be either of type *pose* or *point* - represented respectively by the objects `VertexSE3` and `VertexR3`. Each vertex is represented by

- Its **estimate**, which can be a 3D isometry or a 3D position - namely a `Pose3D` or a `Point3D`;
- A unique **index** that is used to recognize the vertex;
- A boolean variable that indicates whether the node is **fixed** or not. A fixed Vertex must not be involved in the optimization process. It is good to notice that at least one fixed vertex must exists in the graph, otherwise the optimization problem is undefined.

Analogously, the edge are of type *pose-pose* or *pose-point* - represented respectively by the objects `EdgeSE3` and `EdgeSE3_R3`. The edge have several fields, namely:

- The actual **measurement** that can be either a `Pose3D` or a `Point3D`;
- The **data association** that consists in a pair of `Vertex` which represents the nodes involved in the edge;
- The **information matrix** related to the measurement - namely a `Matrix6` or a `Matrix3`.

Once that the front-end populates the graph, it is fed into the `Optimizer` to perform MAP estimation of the best state configuration given the constraints.

## The Optimizer

The `Optimizer` represents systems' heart: it takes as input the graph, retrieves the linear system $\mathbf{H}\Delta\mathbf{x} = -\mathbf{b}$, solves for $\Delta\mathbf{x}$ and finally updates the graph. In the following Subsections the core elements of the `Optimizer` will be better investigated.

### Linearization and Hessian Composition

One of `Optimizer`'s tasks is to compute the contribution that each edge brings to both the Hessian matrix $\mathbf{H}$ and the right-hand-side vector $\mathbf{b}$. Those contribution are retrieved during the *linearization* of the graph.

Each edge of the graph is analyzed, whether it is a `EdgeSE3` or a `EdgeSE3_R3`. For each measurement the block matrices $\mathbf{H}_{ii}$, $\mathbf{H}_{ij}$, $\mathbf{H}_{ji}$ and $\mathbf{H}_{jj}$ are computed as

$$\begin{aligned} \mathbf{H}_{ii} &= \mathbf{J}_i^T \Omega_k \mathbf{J}_i \quad \mathbf{H}_{jj} = \mathbf{J}_j^T \Omega_k \mathbf{J}_j \\ \mathbf{H}_{ij} &= \mathbf{J}_i^T \Omega_k \mathbf{J}_j \quad \mathbf{H}_{ji} = \mathbf{J}_j^T \Omega_k \mathbf{J}_i \end{aligned} \tag{6.1}$$

while the right-hand-side contributions are computed as

$$\mathbf{b}_i = \mathbf{J}_i^T \Omega_k \mathbf{e}_k \qquad \mathbf{b}_j = \mathbf{J}_j^T \Omega_k \mathbf{e}_k \tag{6.2}$$

The subscript indexes $\langle i, j \rangle$ are the node unique indexes in the current edge's data-association. The tuple $\langle i, j \rangle$ indicates the position of each block in the full Hessian $\mathbf{H}$ (and in the full right-hand-side vector $\mathbf{b}$).

The Jacobians $\mathbf{J}_i$ and $\mathbf{J}_j$ are using Equations 5.28 and 5.27 for pose constraints, while for point ones Equations 5.14 and 5.15. Clearly, as reported in Section 5.3, the information matrix of each `EdgeSE3` is adapted through the Unscented Transform before starting the optimization process - and *not* at each iteration.

### Sparse Linear Solver

The linear solver employed is based on Cholesky decomposition of the Hessian - i.e. Subsection 5.1.2. Clearly, the first thing needed in order to efficiently solve a sparse linear system is a proper matrix data-structure.

In this work, sparse matrices are represented through the `SparseBlockMatrix` object, which uses as storage method the *List of Lists* (LIL) approach. It is possible to select between *fixed-size* blocks - i.e. in pose graph optimization - and *dynamic* blocks - i.e. in bundle adjustment. In the former case it is possible to take advantage of CPU cache while doing matrix operations and, thus, boosting the computation; using dynamic blocks matrix operations are slower but it adds flexibility to the system. `SparseBlockMatrix` main feature is the fact that object's memory - used to actually *store* the blocks - can be isolated from the objects itself. Therefore, the actual matrix is just a *view* of those blocks, intended as a LIL of pointers to those memory blocks. This formulation allows to manipulate the `SparseBlockMatrix` **without touching the memory**, just rearranging matrix's *view*. The object `DenseBlockVector` is designed on the basis of this and it used to store dense vectors like the right-hand-side vector $\mathbf{b}$ and the update vector $\Delta\mathbf{x}$.

Before computing the Cholesky factorization, Hessian's non-zero pattern is analyzed through the SuiteSparse library in order to retrieve a suitable ordering that reduces

Cholesky's fill-in. The user can choose between different algorithms - i.e. AMD, COLAMD, CHOLMOD - or let the system retrieve the best one.

Once the permutation is found, `SparseBlockMatrix`'s view of matrix $\mathbf{H}$ is rearranged based on the permutation, together with the `DenseBlockVector` $\mathbf{b}$. Now it is possible to compute the Cholesky factorization and retrieve the matrices $\mathbf{L}$ and $\mathbf{L}^T$. Finally, through Forward-Backward Substitution, the update vector $\Delta\mathbf{x}$ is found.

It is good to notice that before solving the linear system, it is necessary to remove fixed vertexes' contribution from the system. Supposing that node $f$ is fixed, then it is necessary to remove the $f$-th row and column from the Hessian, together with the $f$-th block of the $\mathbf{b}$. Therefore the system we solve has dimension $n - F$, where $F$ represents the number of fixed vertexes.

### Graph Update

Once that the `DenseBlockVector` $\Delta\mathbf{x}$ is computed, it is necessary to apply to each node $k$ the relative block $\Delta\mathbf{x}_k$ of the update vector.

The update is applied node by node through the *box-plus* operator described in Equations 5.22 for $SE(3)$ nodes and the Euclidean minus for $\mathbb{R}^3$ ones. Clearly, fixed nodes will remain unchanged.

## Bottlenecks

The system has been designed to deliver real-time performances, therefore, an ad-hoc implementation is required to achieve such result. The system described in the previous Section has two main time-consuming operations, both performed at each iteration:

1. Allocation and management of the memory for the Hessian's blocks;
2. The actual computation of the Hessian's blocks described in Equation 6.1.

However, the proposed system tackles both the issue, reducing the time required for each iteration to values comparable to state-of-the-art systems.

### Memory management

Allocate the memory blocks for the Hessian $\mathbf{H}$ is a time-expensive operation. Furthermore, it is required to allocate memory also for its decomposition $\mathbf{L}$ and $\mathbf{L}^T$ and for the dense vector $\mathbf{b}$, $\Delta\mathbf{x}$ and $\mathbf{y}$. Moreover, copying memory - for example to create a permutation of a matrix - is another time and resource consuming operation.

The memory copy has been addressed through the separation of matrix view and memory in the `SparseBlockMatrix` object, as mentioned before. In this way it is possible to create multiple views of a matrix that share the same physical memory, avoiding

memory copy and, thus, reducing the memory consumption of the system. The same feature is found in the `DenseBlockVector` object. The physical memory is managed by a separated object called `MemoryManager`. It is in charge of allocating, delete and copy memory for matrix and vectors.

However, even if we are able to avoid memory copy, it is necessary to allocate the matrices and vectors at each iteration. Once again, this process is avoided through a clever solution: once the graph is built, it is possible to exploit its structure to evaluate the non-zero pattern of the Hessian $\mathbf{H}$. The reader might notice that the non-zero pattern will remain unchanged for the entire optimization process, while the *numeric values* of the block will change at each iteration. Moreover, all the `DenseBlockVector`s will have a fixed dimension equal to the number of active vertexes $n - F$. Therefore, it is possible to allocate all the memory required to store Hessian's and vectors' block **only once** before starting the optimization process.

Once that we have the Hessian's non-zero pattern, it is possible to compute both the permutation and the *Cholesky symbolic decomposition*, that retrieves the non-zero pattern of matrix $\mathbf{L}$ (and thus also $\mathbf{L}^T$). In this way, during the optimization steps there is **no memory allocation or copy**, speeding up the entire system.

### Hessian blocks computation

Using the *Eigen* library to perform matrix operations, it is possible to appreciate a bottleneck in the computation of the product $\mathbf{H}_{ij} = \mathbf{J}_i^T \Omega_k \mathbf{J}_j$ - and all the other blocks $\mathbf{H}_{ii}$, $\mathbf{H}_{jj}$ and $\mathbf{H}_{ji}$. The slowing down is particularly emphasized for $(6 \times 6)$ *static* or *dynamic* blocks.

However, for *pose-pose* constraints, given the product $\mathbf{A} = \mathbf{J}_i^T \Omega_k \mathbf{J}_i$, the following relations hold:

$$\begin{aligned} \mathbf{H}_{ii} &= \mathbf{A} & \mathbf{H}_{jj} &= \mathbf{A} \\ \mathbf{H}_{ij} &= -\mathbf{A} & \mathbf{H}_{ji} &= -\mathbf{A} \end{aligned}$$

In this way the computation is performed just one time, thus there is a 75% boost in terms of speed. Furthermore, since $\mathbf{A}$ is a symmetric matrix, it is possible to compute just the upper triangular part and then replicate it in the lower triangular, reducing even more the computational time.

## Performance Results

The library developed has been tested on different synthetic datasets both in *pose-graph* and *Bundle Adjustment* scenarios. All the test have been performed on a Lenovo ThinkPad W540 laptop equipped with:

- CPU: Intel(R) Core(TM) i7-4700MQ CPU @ 2.40GHz

- RAM: $2 \times 4$GB $+ 2 \times 8$GB SODIMM DDR3 - total 24GB
- Disk: SanDisk(R) Ultra II(TM) 240GB Solid State Drive
- OS: Ubuntu 16.04 LTS

Since the works focuses on *pose-graph* optimization, the results will be mostly related to this kind of problem. In this case the blocks composing the Hessian have fixed size - i.e. $(6 \times 6)$ - and, therefore, it is possible to employ a `SparseBlockMatrix` with static fix-sized blocks.

The experiments proposed are obtained using three different datasets that have an increasing number of edges:

1. The open-loop graph created by ProSLAM_stud from the *Kitti_00* run (further information in Chapter 7), with 752 nodes and 903 edges (easy);
2. A synthetic sphere with 2500 vertexes and 9799 edges (medium);
3. A synthetic dataset wit 2001 poses and 24422 edges (hard).

Figure 6.1 shows the step-time's evolution of both systems during the optimization of the three datasets. Furthermore, in Table 6.1 are reported the total optimization time employed by both systems to execute 10 steps.

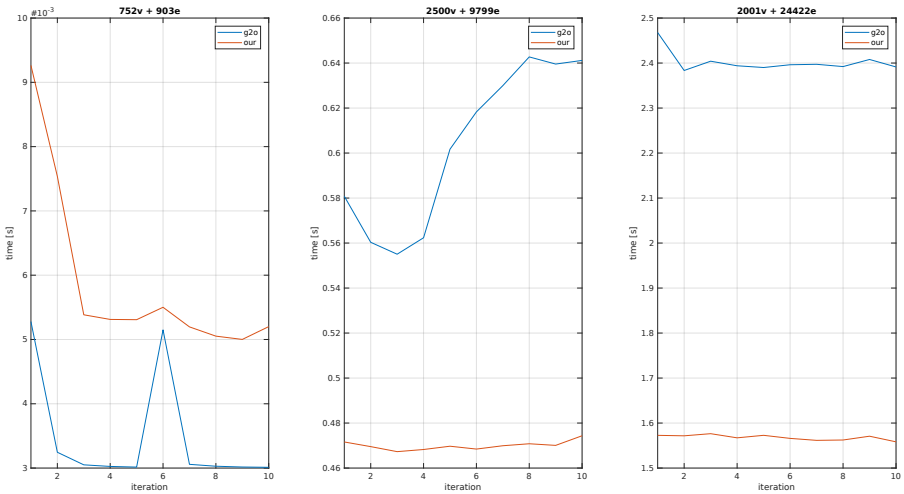



Figure 6.1: **Pose Graph Optimization Step Time Comparison.** Comparison between `g2o` - in blue - and *our system* - in orange - of the time required to execute an optimization step. Despite the minimalistic implementation, our approach is significantly faster than `g2o`; the gap increases with the number of edges, indicating a slightly better scalability of the proposed system with respect to `g2o`.

The reader might notice how well our systems behaves, delivering performances that are better or at least comparable with the one obtained through a state-of-the-art complex system like `g2o`. Furthermore, the results show that an increased number of edges corresponds to a bigger gap between our approach and `g2o`, indicating that our implementation is very efficient and slightly more scalable.

| **Total Optimization Time** | | | |
|---|---|---|---|
| System | *easy* $[s]$ | *medium* $[s]$ | *hard* $[s]$ |
| `g2o` | 0.0349 | 6.0319 | 24.0242 |
| our | 0.0588 | 4.7000 | 15.6795 |

Table 6.1: **Pose Graph Optimization Total Time Comparison.** In this table are reported the total optimization time required by the two systems to complete 10 iterations. The reader might notice that in graphs with more edges, our approach performs better than `g2o`.

For Bundle Adjustment graphs, our systems struggles to obtain the same results seen in pose-graph optimization. This is mainly due to use of a `SparseBlockMatrix` with dynamic blocks, that makes our implementation slower than its competitor `g2o`, as it is reported in Figure 6.2 and Table 6.2.

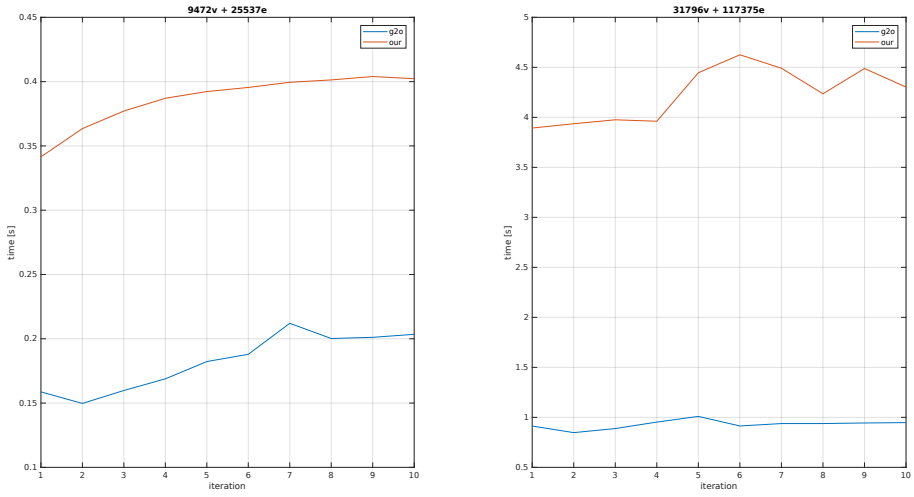



Figure 6.2: **BA Optimization Step Time Comparison.** Comparison between `g2o` - in blue - and *our system* - in orange - of the time required to execute an optimization step. In this case `g2o` leads always the comparison.

The results are obtained running the two systems on two datasets that have an increasing number of nodes and edges. In particular:

1. A synthetic dataset with 9472 vertexes (1001 `SE3` + 8471 `R3`) and 25537 edges (6013 `SE3` + 19524 `SE3_R3`) - called *small*;

2. A bigger synthetic dataset that has 31796 vertexes (2001 `SE3` + 29795 `R3`) and 117375 edges (25327 `SE3` + 92048 `SE3_R3`) - called *big*.

| **Total Optimization Time** | | |
|---|---|---|
| System | *small* $[s]$ | *big* $[s]$ |
| `g2o` | 1.8242 | 9.2920 |
| our | 3.8641 | 42.3554 |

Table 6.2: **BA Optimization Total Time Comparison.** In this table are reported the total optimization time required by the two systems to complete 10 iterations. The reader might notice an inverted trend with respect to pure pose-graph optimization, with our system struggling when the number of edges increases.

One possible solution to this bad trend can be the use of static blocks also for BA problems, solving the linear system $\mathbf{H}\Delta\mathbf{x} = -\mathbf{b}$ using the *Schur Complement*.

Chapter 7

# Use Cases

As already mentioned, the system developed aims to deliver real-time performances also when used in real-world applications. Furthermore, the system provides some easy-to-embed APIs in order to be easily integrated in a full SLAM pipeline. Therefore, in this Chapter we provide two front-end systems developed in the Ro.Co.Co.Lab. at Sapienza University that actually use this work as their back-end. In both cases, our system is used to perform 3D *pose-graph* optimization.

## ProSLAM

The work of Schlegel *et al.* [37] presents a stereo-visual system capable of mapping dynamic large-scale environments called ProSLAM. The system is designed with simplicity and modularity in mind and, thus, it is easy to implement and to understand also from people who are not Computer Vision or SLAM experts.

This works is also almost entirely self-contained and employs only a minimal set of external libraries - among which there is the library described in this work.

ProSLAM, despite its simplicity, is able to provide results comparable to other more complex state-of-the-art front-ends. Its goal is to generate a 3D map from the processing of a sequence of stereo-images. The map is intended as a collection of *landmarks* - salient 3D points in the world characterized in its appearances by a unique descriptor - together with the camera trajectory.

Landmarks acquired in a nearby region define a *local map*; each local map constitutes a node of the graph - i.e. a $SE(3)$ transformation matrix. Edges between local maps encode the spatial constraints correlating local maps close in space. Those constraints are generated by two kind of events:

1. **Tracking** of the camera motion between temporal subsequent maps;
2. **Alignment** of local maps acquired at distant times as a consequence of *relocalization* events - i.e. *loop-closures*.

Re-localization is more complex to address with respect to tracking. In fact, to achieve this goal it is necessary to compare descriptors of all the local-maps. This is an expensive operation, and it is efficiently performed by the *Hamming Binary Search Tree*

(HBST) [38] library. The system periodically triggers graph optimization and this helps also the re-localization process.

It is good to notice that ProSLAM runs on *single thread*, delivering performances comparable with other more complex and multi-thread systems - e.g. ORB-SLAM2 [39].

## ProSLAM_stud

Colosi *et al.* propose in their work ProSLAM_stud [41] a further iteration on minimalism from original ProSLAM. It is a *Master thesis* work, therefore its approach is more didactic than the original one. However, the system still delivers quite good performances both in terms of speed and accuracy.

This systems embeds a new *tracking* method based on KD-Tree that is able to provide a boost in speed with respect to ProSLAM, at the cost of little loss in accuracy.

ProSLAM_stud keeps the single-treaded implementation and despite the minimal approach, it can push up to more than 80Hz - on average.

## KITTI Dataset

Both ProSLAM and ProSLAM_stud have been tested on two sequences of the KITTI dataset [42]. All the 22 available sequences, have been acquired using a car equipped with several sensors - e.g. stereo-cameras, Velodyne laser scanner and localization system that combines data from GPS, GLONASS and IMU - all calibrated and synchronized.

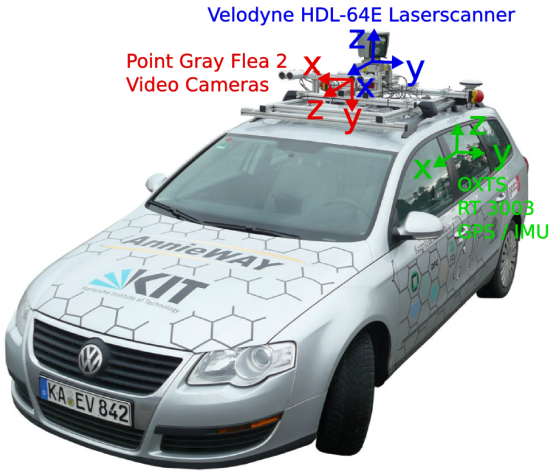



Figure 7.1: **KIT AnnieWAY acquisition method.** The KITTI dataset is acquired using an autonomous driving car, equipped with several sensor: a stereo-rig of high resolution cameras, Velodyne 3D laser scanner and a localization unit based on GPS/GLONASS/IMU.

Clearly, the selected sequences are the ones with most loop-closures, in order to

highlight the benefits of the optimization process. Figure 7.2 proposes a qualitative comparison of the performances using ProSLAM in sequences 00 and 06 respectively.

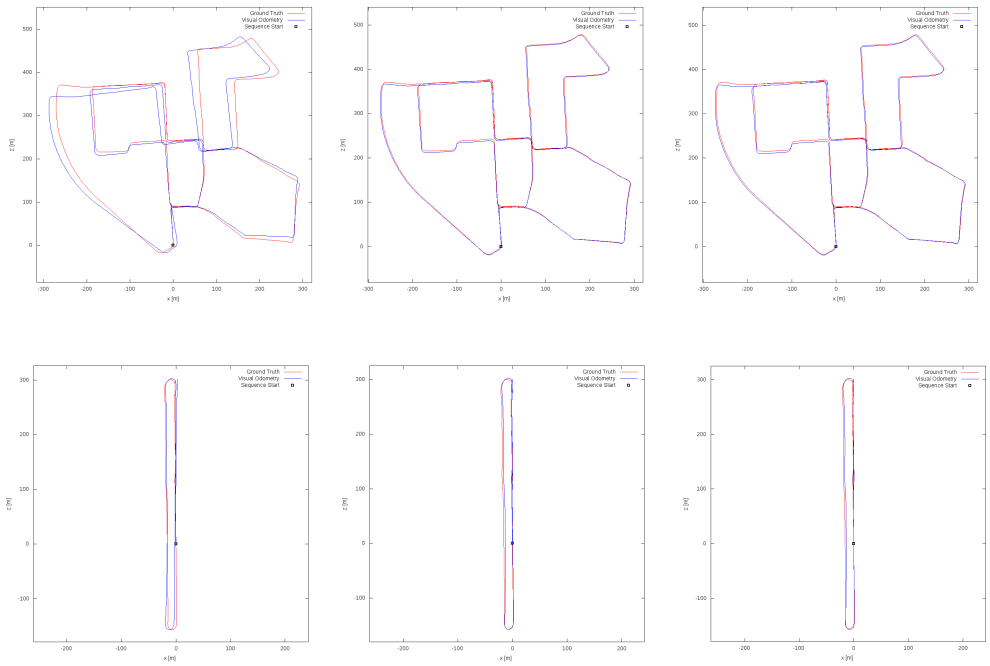


Figure 7.2: **ProSLAM results.** Starting from the top image it is proposed a comparison between the estimated and the real camera trajectory of sequence 00 - respectively in *blue* and in *red*. Proceeding from left to right it is proposed the estimation with *no map optimization*, using `g2o` and using *our approach* as optimizer. In the bottom image it is shown the same comparison but using sequence 06.

The reader might already notice that the estimated trajectory is more consistent using a back-end that performs graph optimization; the figure also show how well our approach performs the optimization compared with another state-of-the-art system - i.e. `g2o` [1].

| **ProSLAM Trajectory Error Evaluation** | | | | |
|---|---|---|---|---|
| Config | Sequence 00 | | Sequence 06 | |
| | Rotation $[\,^{deg}/_{100m}]$ | Translation [%] | Rotation $[\,^{deg}/_{100m}]$ | Translation [%] |
| OL | 0.397155 | 1.117624 | 0.393365 | 0.861104 |
| `g2o` | 0.301858 | 0.731966 | 0.224632 | 0.733721 |
| our | 0.272738 | 0.723581 | 0.219253 | 0.664294 |

Table 7.1: **ProSLAM Trajectory Error.** In this Table are reported the *translation* and *rotation* error of the final trajectory on both sequences. The contribution of a map optimizer is undeniable, however, the reader might appreciate the results obtained with our minimalistic approach, which are not far from the one obtained using the state-of-the-art system `g2o`.

Furthermore, in Table 7.1 are proposed quantitative results about the final trajectory's

error for each map optimizer used: the reader might appreciate the closeness of our approach with the `g2o`.

The same considerations can be made also using ProSLAM_stud system, as depicted in Figure 7.3 and in Table 7.2.

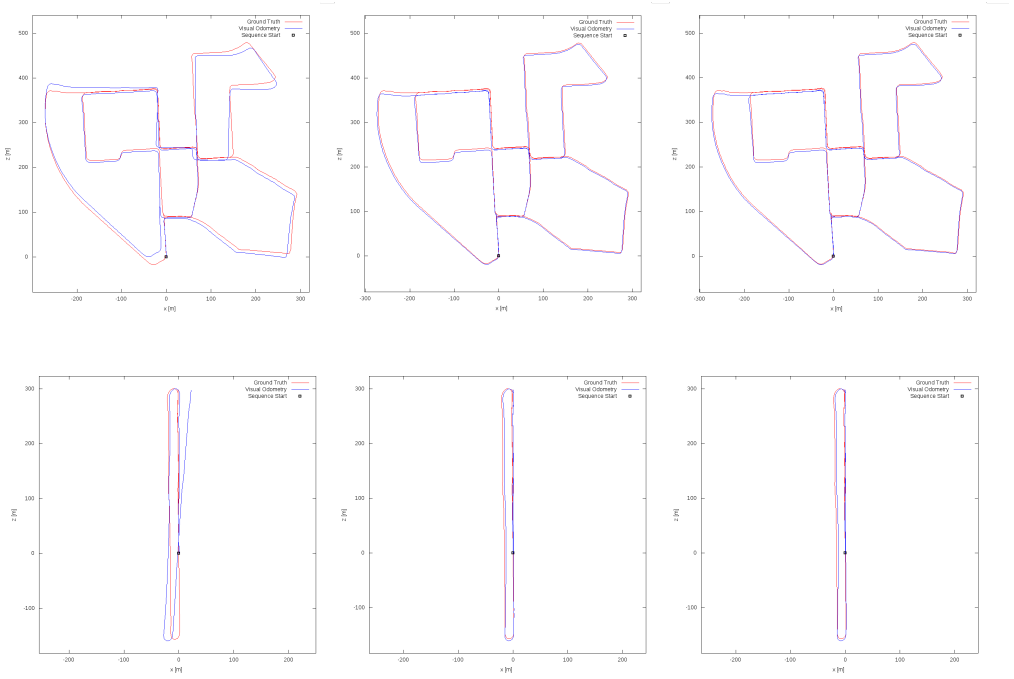


Figure 7.3: **ProSLAM_stud results.** The top image proposes the estimated and real camera trajectory of sequence 00 - respectively in *blue* and in *red*. Again it is proposed the comparison between open-loop estimation, `g2o` and our approach. In the bottom image it is shown the same comparison but using sequence 06.

| **ProSLAM_stud Trajectory Error Evaluation** | | | | |
|---|---|---|---|---|
| Config | Sequence 00 | | Sequence 06 | |
| | Rotation $[\,^{deg}/_{100m}]$ | Translation [%] | Rotation $[\,^{deg}/_{100m}]$ | Translation [%] |
| OL | 0.615349 | 1.285283 | 0.745911 | 1.036123 |
| `g2o` | 0.452409 | 0.941609 | 0.259771 | 0.785845 |
| our | 0.357097 | 0.850793 | 0.272893 | 0.794151 |

Table 7.2: **ProSLAM_stud Trajectory Error.** In this Table are reported the *translation* and *rotation* error of the final trajectory on both sequences. Even in this case, our approach delivers consistent results.

From this quantitative and qualitative analysis it is clear the contribution of a good back-end in the SLAM pipeline. Furthermore, they highlighted the quality reached by our system in real-world scenarios, delivering fast and accurate estimation despite it simplicity and minimalism.

Chapter 8

# Conclusions

In this Master's thesis, it has been reached the goal of creating from scratch a *graph optimizer* able to cope with the principal 3D SLAM problems in real-time - namely *pose-graph optimization* and *Bundle Adjustment*.

The system, as mentioned in Chapter 6, delivers quite good performances despite its minimal and didactic approach, combining simplicity and effectiveness. It is entirely developed in C++ in less than 5500 lines of code and employs only essential external libraries. This makes our work comprehensible also by researchers that approach to this problem for the first time and are not SLAM experts.

The system performs well thanks to a novel approach to manage $SE(3)$ objects - i.e. 3D poses in the space - and an efficient C++ implementation. The former consists in a new **error function** for $SE(3)$ objects that reduces the non-linearities of the problem, reducing also the computing time and facilitating system's convergence. The latter concerns a well designed memory management, letting the system perform the optimization steps with **0 memory copy**. Thanks to this, the system proposed in this work scales well also to big graphs with thousands of poses and points.

## Applications

The system can be employed both for on-line and off-line applications, for example:

- Together with a front-end in a full SLAM pipeline, as mentioned in Chapter 7. In fact, thanks to its fast implementation, it can be embedded on actual mobile robots - no matter if they are on wheels, UAVs or humanoids.
- As a matter of fact, many LS algorithms fail or get stuck in local minima when the initial guess is far from the the optimum. Therefore, our system can bootstrap those algorithm providing a better initial guess in order to further optimize the graph using fine-grain LS algorithms.
- It can be embedded in the map itself, in order to keep always consistent the world representation.

The generality, the ability to adapt to different scenarios and the easy-to-embed provided APIs make our system a good choice in several scenarios. Clearly it is not the perfect

system and, thus, in the next Section it is proposed a set of future works related to our system.

## Future Works

The reader might know that the time available to complete a Master's thesis is limited, and this biased a lot project development. Many compromises have been made and a lot of aspects are not addressed. In this Section it is proposed an insight of what might be the future iterations of this work.

### Expand the Addressed Problems

Until now, our system only deals with 3D pose-graph optimization or 3D bundle-adjustment. All state-of-the-art systems provide APIs to address a many of the typical problems mentioned in Chapter 4, both in 3D and 2D environments.

Therefore, a good starting point might be the extension of the problems addressed by our current system to 2D scenarios. Then, once that core SLAM problems have been successfully addressed, all the other formulations could be added.

### Hierarchical Approach

Hierarchical approaches represents the most interesting evolution of graph-based SLAM formulation. This formulation allows to create multiple graph's *views*, each of which has a different granularity: the bottom level represents the original input, while higher levels capture the structural properties of the environment in a always more compact manner. This approach is similar to what has been proposed in the work of Grisetti *et al.* [2].

This hierarchical formulation of the problem allows to update only the coarse structure of the scene during online mapping - i.e. only the highest level. When a substantial change happens in the top level, the update is propagated through the other lower levels, reducing the computational effort while providing an accurate estimate.

# Bibliography


[1] Kümmerle, R., Grisetti, G., Strasdat, H., Konolige, K., Burgard, W.: g 2 o: A general framework for graph optimization. In: Robotics and Automation (ICRA), 2011 IEEE International Conference on, IEEE (2011) 3607–3613

[2] Grisetti, G., Kümmerle, R., Stachniss, C., Frese, U., Hertzberg, C.: Hierarchical optimization on manifolds for online 2d and 3d mapping. In: Robotics and Automation (ICRA), 2010 IEEE International Conference on, IEEE (2010) 273–278

[3] Durrant-Whyte, H., Bailey, T.: Simultaneous localization and mapping: part i. IEEE robotics & automation magazine **13**(2) (2006) 99–110

[4] Bailey, T., Durrant-Whyte, H.: Simultaneous localization and mapping (slam): Part ii. IEEE Robotics & Automation Magazine **13**(3) (2006) 108–117

[5] Leonard, J., Durrant-Whyte, H., Cox, I.J.: Dynamic map building for autonomous mobile robot. In: Intelligent Robots and Systems' 90.'Towards a New Frontier of Applications', Proceedings. IROS'90. IEEE International Workshop on, IEEE (1990) 89–96

[6] Dissanayake, M.G., Newman, P., Clark, S., Durrant-Whyte, H.F., Csorba, M.: A solution to the simultaneous localization and map building (slam) problem. IEEE Transactions on robotics and automation **17**(3) (2001) 229–241

[7] Aulinas, J., Petillot, Y.R., Salvi, J., Lladó, X.: The slam problem: a survey. CCIA **184**(1) (2008) 363–371

[8] Grisetti, G., Stachniss, C., Burgard, W.: Improving grid-based slam with rao-blackwellized particle filters by adaptive proposals and selective resampling. In: Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on, IEEE (2005) 2432–2437

[9] Montemerlo, M., Thrun, S., Koller, D., Wegbreit, B., et al.: Fastslam: A factored solution to the simultaneous localization and mapping problem. In: Aaai/iaai. (2002) 593–598

[10] Lu, F., Milios, E.: Globally consistent range scan alignment for environment mapping. Autonomous robots **4**(4) (1997) 333–349

[11] Gutmann, J.S., Konolige, K.: Incremental mapping of large cyclic environments. In: Computational Intelligence in Robotics and Automation, 1999. CIRA'99. Proceedings. 1999 IEEE International Symposium on, IEEE (1999) 318–325

[12] Olson, E., Leonard, J., Teller, S.: Fast iterative alignment of pose graphs with poor initial estimates. In: Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on, IEEE (2006) 2262–2269

[13] Dellaert, F., Kaess, M.: Square root sam: Simultaneous localization and mapping via square root information smoothing. The International Journal of Robotics Research **25**(12) (2006) 1181–1203

[14] Kaess, M., Ranganathan, A., Dellaert, F.: isam: Fast incremental smoothing and mapping with efficient data association. In: Robotics and Automation, 2007 IEEE International Conference on, IEEE (2007) 1670–1677

[15] Kaess, M., Johannsson, H., Roberts, R., Ila, V., Leonard, J.J., Dellaert, F.: isam2: Incremental smoothing and mapping using the bayes tree. The International Journal of Robotics Research **31**(2) (2012) 216–235

[16] Grisetti, G., Stachniss, C., Grzonka, S., Burgard, W.: Toro. https://www.openslam.org/toro.html

[17] Grisetti, G., Grzonka, S., Stachniss, C., Pfaff, P., Burgard, W.: Efficient estimation of accurate maximum likelihood maps in 3d. In: Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on, IEEE (2007) 3472–3478

[18] Grisetti, G., Stachniss, C., Grzonka, S., Burgard, W.: A tree parameterization for efficiently computing maximum likelihood maps using gradient descent. In: Robotics: Science and Systems. (2007) 27–30

[19] Dellaert, F.: Factor graphs and gtsam: A hands-on introduction. Technical report, Georgia Institute of Technology (2012)

[20] Ni, K., Steedly, D., Dellaert, F.: Tectonic sam: Exact, out-of-core, submap-based slam. In: Robotics and Automation, 2007 IEEE International Conference on, IEEE (2007) 1678–1685

[21] Ni, K., Dellaert, F.: Multi-level submap based slam using nested dissection. In: Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on, IEEE (2010) 2558–2565

[22] Grisetti, G., Kümmerle, R., Ni, K.: Robust optimization of factor graphs by using condensed measurements. In: Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on, IEEE (2012) 581–588

[23] Charnes, A., Frome, E.L., Yu, P.L.: The equivalence of generalized least squares and maximum likelihood estimates in the exponential family. Journal of the American Statistical Association **71**(353) (1976) 169–171

[24] Joyce, J.: Bayes' theorem. In Zalta, E.N., ed.: The Stanford Encyclopedia of Philosophy. Winter 2016 edn. Metaphysics Research Lab, Stanford University (2016)

[25] Davis, T.A.: Direct methods for sparse linear systems. SIAM (2006)

[26] Saad, Y.: Iterative methods for sparse linear systems. SIAM (2003)

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

[28] Davis, T.A., Gilbert, J.R., Larimore, S.I., Ng, E.G.: A column approximate minimum degree ordering algorithm. ACM Transactions on Mathematical Software (TOMS) **30**(3) (2004) 353–376

[29] Cleveland Ashcraft, C., Grimes, R.G., Lewis, J.G., Peyton, B.W., Simon, H.D., Bjørstad, P.E.: Progress in sparse matrix methods for large linear systems on vector supercomputers. The International Journal of Supercomputing Applications **1**(4) (1987) 10–30

[30] Karypis, G., Kumar, V.: Multilevelk-way partitioning scheme for irregular graphs. Journal of Parallel and Distributed computing **48**(1) (1998) 96–129

[31] Agarwal, P., Olson, E.: Variable reordering strategies for slam. In: Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on, IEEE (2012) 3844–3850

[32] Kümmerle, R., Grisetti, G., Burgard, W.: Simultaneous calibration, localization, and mapping. In: Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on, IEEE (2011) 3716–3721

[33] Julier, S.J.: The scaled unscented transformation. In: American Control Conference, 2002. Proceedings of the 2002. Volume 6., IEEE (2002) 4555–4559

[34] : Eigen template libray for linear algebra. http://eigen.tuxfamily.org/index.php?title=Main_Page

[35] : Suitesparse. http://faculty.cse.tamu.edu/davis/suitesparse.html

[36] Agarwal, S., Mierle, K., Others: Ceres solver. http://ceres-solver.org

[37] Schlegel, D., Colosi, M., Grisetti, G.: Proslam: Graph slam from a programmer's perspective. arXiv preprint arXiv:1709.04377 (2017)

[38] Schlegel, D., Grisetti, G.: Visual localization and loop closing using decision trees and binary features. In: Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on, IEEE (2016) 4616–4623

[39] Mur-Artal, R., Tardós, J.D.: Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. IEEE Transactions on Robotics (2017)

[40] Engel, J., Schöps, T., Cremers, D.: Lsd-slam: Large-scale direct monocular slam. In: European Conference on Computer Vision, Springer (2014) 834–849

[41] Colosi, M., Schlegel, D., Grisetti, G.: Proslam student edition: a minimalistic stereo visual slam system. Master's thesis, Sapienza University of Rome (2017)

[42] Geiger, A., Lenz, P., Urtasun, R.: Are we ready for autonomous driving? the kitti vision benchmark suite. In: Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on, IEEE (2012) 3354–3361