WORCESTER POLYTECHNIC INSTITUTE

Efficient Factor Graph Fusion for Multi-robot Mapping

by

Ramkumar Natarajan

A thesis submitted in partial fulfillment for the degree of Master of Science

in

Robotics Engineering
WORCESTER POLYTECHNIC INSTITUTE

June 2017

I certify that I have read this thesis and that in my opinion
it is fully adequate, in scope and in quality, as a dissertation
for the degree of Master Of Science.

Professor Michael A. Gennert, Thesis Advisor

I certify that I have read this thesis and that in my opinion
it is fully adequate, in scope and in quality, as a dissertation
for the degree of Master Of Science.

Professor William R. Michalson, Thesis Committee Member

I certify that I have read this thesis and that in my opinion
it is fully adequate, in scope and in quality, as a dissertation
for the degree of Master Of Science.

Professor Eugene Eberbach, Thesis Committee Member

"The enchanting charms of this sublime science reveal to only those who have the courage to go deeply into it."

Carl Friedrich Gauss

WORCESTER POLYTECHNIC INSTITUTE

Abstract

Robotics Engineering
Worcester Polytechnic Institute

Master of Science

by Ramkumar Natarajan

This work presents a novel method to efficiently factorize the combination of multiple factor graphs having common variables of estimation. The fast-paced innovation in the algebraic graph theory has enabled new tools of state estimation like factor graphs. Recent factor graph formulation for Simultaneous Localization and Mapping (SLAM) like Incremental Smoothing and Mapping using the Bayes tree (ISAM2) has been very successful and garnered much attention. Variable ordering, a well-known technique in linear algebra is employed for solving the factor graph. Our primary contribution in this work is to reuse the variable ordering of the graphs being combined to find the ordering of the fused graph. In the case of mapping, multiple robots provide a great advantage over single robot by providing a faster map coverage and better estimation quality. This coupled with an inevitable increase in the number of robots around us produce a demand for faster algorithms. For example, a city full of self-driving cars could pool their observation measurements rapidly to plan a traffic free navigation. By reusing the variable ordering of the parent graphs we were able to produce an order-of-magnitude difference in the time required for solving the fused graph. We also provide a formal verification to show that the proposed strategy does not violate any of the relevant standards. A common problem in multi-robot SLAM is relative pose graph initialization to produce a globally consistent map. The other contribution addresses this by minimizing a specially formulated error function as a part of solving the factor graph. The performance is illustrated on a publicly available SuiteSparse dataset and the multi-robot AP Hill dataset.

Acknowledgements

The acknowledgements and the people to thank go here, don't forget to include your project advisor…

Contents

List of Figures

Abbreviations

SLAM	**S**imultaneous **L**ocalization **A**nd **M**apping
ISAM	**I**ncremental **S**moothing **A**nd **M**apping
ISAM2	**I**ncremental **S**moothing **A**nd **M**apping using the Bayes Tree
LIDAR	**LI**ght **D**etection **A**nd **R**anging
IMT	**I**nvertible **M**atrix **T**heorem
HUND	**H**ypergraph **U**nsymmetrical **N**ested **D**issection

Dedicated to my parents. . .

Chapter 1

Introduction

As the boundaries of the definition of robotics are expanding multi-robot systems are increasingly finding its application in various fields. In many applications, including robots in a cluttered environment like a retail store or search and rescue operation and extremely uncertain environment like planetary exploration or surveillance, one of the key challenges is to map the environment and localize the robot simultaneously. This problem is called Simultaneous Localization and Mapping (SLAM) which deals with fusing different sensor measurements to develop a consistent picture of the environment. With the solutions to single robot SLAM getting more matured than ever coupled with the rise of self-driving cars the path forward is to develop solutions for a team of robot explorers which split up at a pathway fork and later meet again to share and merge their maps. Also, with multiple robots, the environment could be mapped more robustly and significantly faster. In particular, we deal with developing a *centralized* optimal map by fusing the measurement estimates from all the robots. Combining the measurements and estimates from multiple robots for centralized mapping is important because it helps to avoid the data redundancy in the overlapping areas and allow the robots to *help each other out* in case of localization loss. Multi-robot SLAM has been extensively studied since the last decade leading to the development of several algorithms [3–5]. The multi-robot scenario also introduces several key issues on top of a single robot case. A large body of the previous work try to address these issues (listed below) in different ways,

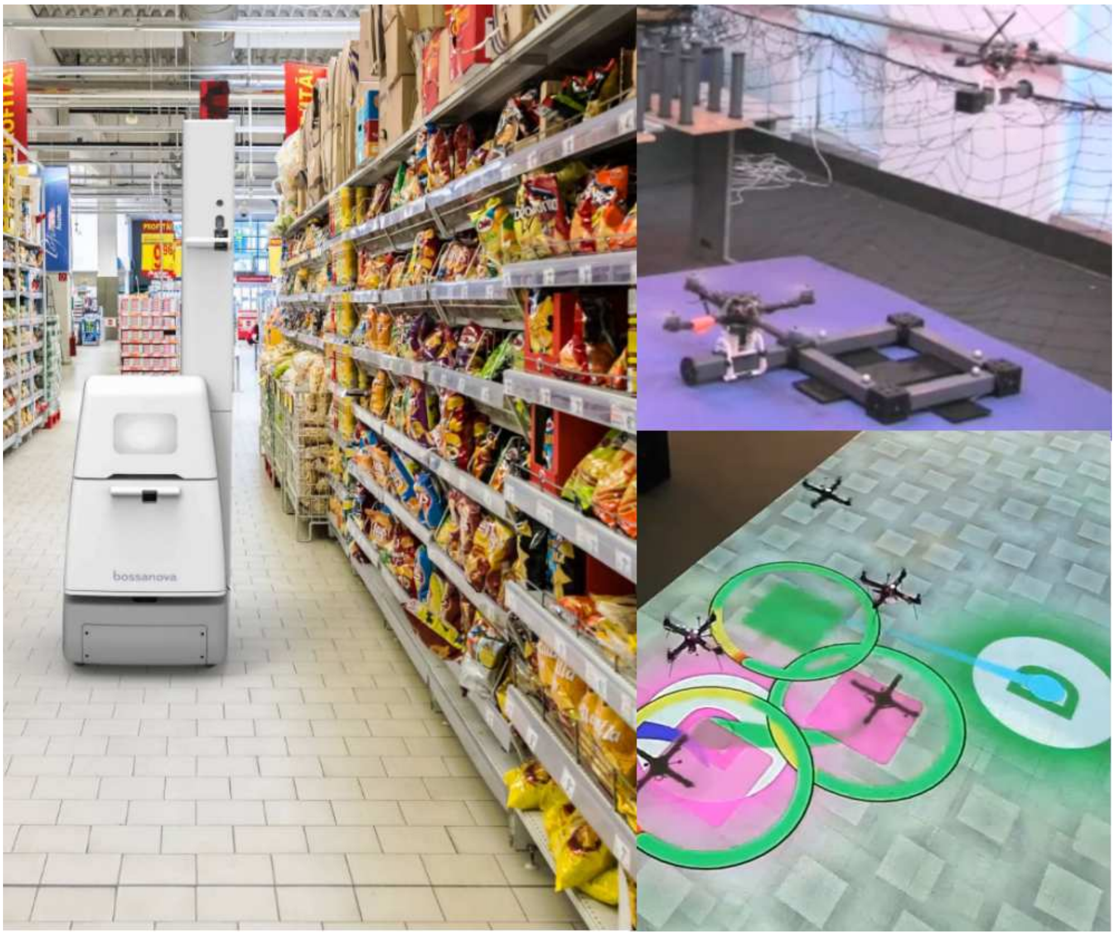


Figure 1.1: Clockwise from left: 1) A mobile robot navigating in a retail store to provide inventory solutions. They are designed to collaborate with other robots at the end of scanning an aisle. 2) Swarm of robots teaming up in a cooperative task of building lego blocks at Grasp Laboratory, University of Pennsylvania. 3) Decentralized control and planning of a multi-robot system at University of New Hampshire.

1. Globally consistent robot pose initialization.

2. Direct and indirect encounters.

3. Multi-robot data association

For multi-robot SLAM, we use *factor graphs* [6] as the underlying framework for state estimation. This work proposes a method to quickly and efficiently fuse the factor graphs of the encountering robots and also address the aforementioned issues simultaneously.

1.1 Thesis

My thesis in this dissertation is the following:

Ordering the variables of the fused factor graph using the ordering of the parent factor graphs provides a superior alternative to the complete reordering approach that is fast, efficient and also numerically stable. Also, by introducing the concept of "global nail" the issue of relative pose graph initialization for different robots is solved.

I split this thesis into three claims that correspond to the Chapters 3, 4 and 6.1 respectively. The goal of my research is efficient factor graph fusion, globally consistent pose initialization for all the robots and rapid multi-channel object detection as explained below:

1. *Fused Graph Ordering:* Variable reordering is a technique used to retain the sparsity of the factor graph during its factorization. This chapter proposes a numerically stable variable ordering strategy for the fused graph by reusing the parent graph ordering that is faster than the naive approach of complete reordering (Chapter 3).

2. *Multi-robot Pose Graph Initialization:* Factor Graph is also referred as Pose Graph in the SLAM context. This chapter introduces a new type of error function used as a factor in the factor graph to estimate the globally consistent trajectory for the robots starting at unknown relative initial positions.

3. *Multi-robot Data Association:* This is another common problem in multi-robot mapping that deals with unknown robot identity during a robot-robot encounter. The experimental real world dataset has colored fiducials attached to the robots and the environment. An improved version of Viola-Jones rapid detection [7] for identifying the fiducials is developed and the computational complexity is studied (Chapter 6.1).

In the reminder of this chapter, I lay down the reasoning leading to my thesis.

1.2 Efficient Factor Graph Fusion

In order to be useful for a multi-robot system, SLAM needs to perform at real-time. Offline or batch solutions to SLAM means the robot has to wait until the calculation for the update is finished. Even in the case of multiple robots with a centralized mapping capability, quicker update times are always useful. With the centralized system calculating the best estimate based on measurements received from all the robots, an update sent back to the individual robots could be used to correct or improve the local estimates.

A real-time algorithm should also be able to update the system incrementally. This means that the centralized system should just require the new set of measurements from the individual robots taken after the last time they communicated to make an update. SLAM by nature itself is a sparse problem with the measurements connected temporally almost always. The centralized system should also be able to calculate an update by just using a local portion of the graph being impacted by the new measurements. For example, the measurements from a robot moving down a particular aisle in a retail store are completely independent of the measurements and observations made in a different and far away aisle. So a centralized system recalculating the estimates of the unaffected portions of the map is not a wise option. This incremental requirement for SLAM problem is well studied, particularly by Kaess and Dellart in [8] and [9]. By using the formulation presented in their work and reusing the variable ordering of the graph being combined we come up with an efficient graph fusion strategy (Chapter 3).

1.3 Multi-robot Pose Graph Initialization

The measurements from multiple robots must be globally consistent to build a unified map of the environment. In order to do this, all the robots should have a prior knowledge about their relative initial positions. It is not necessary or a fair assumption to consider that all the robots of a multi-robot system start at the same position on the map. Any arbitrary value to the initial position will lead to a conflict during a robot-robot encounter in terms of global map alignment. For example, in the case of multiple robots navigating in a large retail store across distinct aisles, lack of knowledge about each robot's initial position gives the freedom to all the independent trajectories to align together in several possible ways. Although there is a closed form solution to resolve the alignment issue with a single robot-robot encounter [5] the devised algorithm should be able to continuously and incrementally integrate the incoming measurements to refine the alignment error.

The encounter could also be *indirect* in which multiple robots visit the same portion of the environment at different time instants. In this case, the variable representing the pose of the landmark (or common portion of visit among different robots) is also involved in alignment estimation. It is therefore essential for the algorithm to incorporate indirect encounters in the alignment of the map. In other words, aligning the map is same as finding the globally consistent initial pose for all the robots. A cost function that tries to minimize the alignment error and also supports multiple encounters between the robots is formulated and optimized in Chapter 4.

1.4 Multi-robot Data Association

Data Association, in general, is an important component of SLAM. It is the process of recognizing previously visited landmarks in the environment to refine the map and the robot's path. There are several ways of extracting the features of interest (landmarks) from the environment, ranging from wireless network-based to computer vision techniques. With multiple robots in place the identity of the fellow robots should also be identified on top of the landmarks in the environment. The data association engine should be able to recognize both the landmarks in the environment and the robot IDs from the extracted features. Simultaneously, these type of sophisticated measurements should not consume a large amount of time as it introduces the problem of synchronization and scheduling.

In this thesis, an object detection based data association is used to demonstrate the results with the experimental dataset. Although this is not the main concentration of the thesis, an improved Viola-Jones rapid object detection [7] is devised to work in the multi-channel image space. As it is able to use multiple image channels beyond color (like depth and intensity), detection could be performed at a much lower resolution saving time per scan. Chapter 6.1 presents a theoretical study of the time complexity of the improved algorithm.

1.5 Organization

The remainder of the dissertation is organized as follows: The background and related work is discussed separately for factor graph ordering and multi-robot map alignment in Chapter 3 and Chapter 4 respectively. The next chapter formally introduces the factor graph and the Bayes tree data structure often used throughout my work. In Chapter 2, a set of key terminologies from graph theory and sparse linear algebra literature are also explained for the sake of better understanding and completeness. The novel algorithm to quickly find the variable ordering of the fused graph is presented in Chapter 3. Experimental results on the standard real world datasets from the sparse linear algebra literature is also presented at the end of Chapter 3. Chapter 4 deals with the problem of pose graph initialization or map alignment and gives a solution by devising an appropriate cost function and "global nail". A detailed numerical section, sensor models & software library used and demonstration of experiments is presented in Chapter 5. The conclusions and the potential future work are also discussed in the same chapter. Finally, Chapter 6.1 presents the improved rapid object detection for multi-channel images.

Chapter 2

Multi-robot Smoothing and Mapping

In this chapter, I will review the SLAM formulation using probabilistic inference as proposed by Dellaert and Kaess [10] but for the scenario of multiple robots. It is referred as the *full* SLAM problem as it contains the entire trajectory and all the landmarks in the state vector. The solution to the SLAM problem using the recent pose graph representations has garnered much attention because of their computational efficiency and robustness. In our work, we will be using the Incremental Smoothing and Mapping using the Bayes tree (ISAM2) [9] as the optimization algorithm for SLAM. Improvements on efficiency of variable reordering for combined graphs, support for cooperative mapping and multi-robot relative pose initialization are demonstrated as an extension to ISAM2. I will also introduce a few key concepts and terminologies of factor graph [6] aiding in the better understanding of the contributions in this thesis. A factor graph is a type of probabilistic graphical model which represents the factorization of a probabilistic distribution function. They are used to model complex estimation problems having wide range of applications in robotics. Formulating the SLAM problem using the factor graph, also known as pose graph in the robotics literature, opens the door for the application of several probabilistic inference algorithms. Although the proposed variable reordering is applicable to any general sparse linear system, our research is multi-robot SLAM and we will use this as an example for formulation throughout the paper. However, to demonstrate its capability outside SLAM, we present some results on the real-world datasets from SuiteSparse [1] in Chapter 3.

In the following section, I will describe the graph structure underlying the multi-robot SLAM which leads to the optimization problem. I then briefly explain the various factorization schemes available upon linearization. I continue with providing an insight on how the various components of the graph are linked with their equivalent matrix representations. The chapter concludes by providing sufficient motivation and getting into the crux of the contribution i.e. efficient variable reordering for fused factor graphs.

2.1 Cooperative SLAM as Probabilistic Inference

In the case of SLAM, the set of constraints obtained from the proprioceptive sensors like odometry measurements, inertial measurement units (IMUs) and exteroceptive sensors like range (LIDAR) and vision measurements form a Markov chain that connects all the variables to be estimated. Consider a single robot navigation task whose pose variables are given by $X = \{\mathbf{x}_i\}_{i=0}^{i=N}$, input commands by $U = \{\mathbf{u}_i\}_{i=1}^{i=N}$, sensor observations by $Z = \{\mathbf{z}_k\}_{k=0}^{i=M}$ and the landmarks as $L = \{\mathbf{l}_j\}_{j=1}^{j=K}$. The belief network showing the interrelationship between these variables is given in Figure 2.1. The estimation of these variables is given by the following probabilistic formulation:

$$P(X, U, Z, L) = p(\mathbf{x}_0) \prod_{i=1}^{N} p(\mathbf{x}_i \mid \mathbf{x}_{i-1}, \mathbf{u}_i) \prod_{k=0}^{M} p(\mathbf{z}_k \mid \mathbf{x}_{i_k}, \mathbf{l}_{j_k}) \tag{2.1}$$

where $p(\mathbf{x}_0)$ is the prior over the initial pose of the robot, $p(\mathbf{x}_i \mid \mathbf{x}_{i-1}, \mathbf{u}_i)$ represents the process model or the motion model that gives the next pose x_i based on the control input $\mathbf{u}_i$ and $p(\mathbf{z}_k \mid \mathbf{x}_{i_k}, \mathbf{l}_{j_k})$ represents the measurement model. The measurement model is parametrized by the current pose $\mathbf{x}_i$ that is obtained from the motion model. Throughout this explanation, I assume known correspondence between the landmark and measurement (i_k, j_k). However, in our experiments this correspondence is obtained from data association as described in Chapter 6.1.

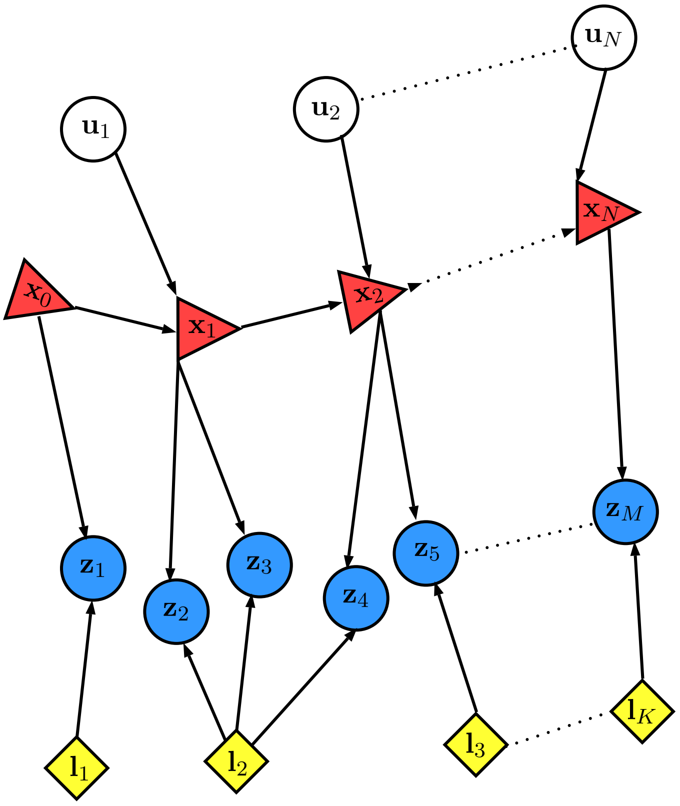


Figure 2.1: The belief network showing the cause-effect relationship of a robot navigation scenario. The robot (red) is given action inputs (plain) to navigate and collect landmark (yellow) measurements (blue). Same landmarks detected at multiple observations from a single pose indicates duplication which will be solved by data association.

I consider Gaussian noise in the process and measurement models following the standard assumption in SLAM literature [11]. The below formulation differs slightly from the typical way it is done in the SLAM papers as needed for our implementation and walking through it would provide a clear picture. The process model relating the previous pose $\mathbf{x}_{i-1}$ and the control input $\mathbf{u}_i$ with the ground truth value of current pose $\mathbf{x}_i$ is:

$$\mathbf{x}_i = f(\mathbf{x}_{i-1}, \mathbf{u}_i) + \mathbf{w}_p \tag{2.2}$$

where $f(.)$ is the odometry model and $\mathbf{w}_p$ is the normally distributed zero mean process noise with variance Λ. The previous pose $\mathbf{x}_{i-1}$ is obtained from recursion and a prior over the very first pose is provided as a starting point. The equation can be understood as expressing the gap between the model and the ground truth with noise $\mathbf{w}_p \sim \mathcal{N}(0, \Lambda)$. Since this noise parameter only captures the local variation between two poses it is a valid approximation owing to data coming from sensors at a high frequency. We also use point-to-line metric based Iterative Closest Point (ICP) [12] for laser scan-matching which is another source for obtaining local transform measurement between every successive pose of the robot. The laser scan matching model is given by:

$$\mathbf{x}_i = \mathbf{x}_{i-1} + s(\mathbf{d}_{i-1}, \mathbf{d}_i) + \mathbf{w}_s \tag{2.3}$$

where $s(.)$ is the laser scan-matching model, $\mathbf{d}_{i-1}$ and $\mathbf{d}_i$ are the scan ranges obtained at $\mathbf{x}_{i-1}$ and $f(\mathbf{x}_{i-1}, \mathbf{u}_i)$ respectively and $\mathbf{w}_s$ is the normally distributed zero mean process noise with variance Ω. A curious reader can refer Appendix 2 [update] for a better description of motion and scan-matching models. The measurement model relating the current pose obtained from the process model, the landmark pose $\mathbf{l}_j$ and the ground truth landmark pose $\mathbf{z}_k$ is:

$$\mathbf{z}_k = h(f(\mathbf{x}_{i-1}, \mathbf{u}_i) + \mathbf{w}_p, \mathbf{l}_k) + \mathbf{w}_m \tag{2.4}$$

where $h(.)$ is the measurement model, $\mathbf{w}_m$ is the zero mean Gaussian measurement noise with variance Γ. Using the above Equations 2.2, 2.3 and 2.4 to express the probability distributions in Equation 2.1:

$$p(\mathbf{x}_i \mid \mathbf{x}_{i-1}, \mathbf{u}_i) \propto \exp\left(-\frac{1}{2}\left(\frac{1}{8}\,\|\mathbf{x}_{i-1} + s(\mathbf{d}_{i-1}, \mathbf{d}_i) - f(\mathbf{x}_{i-1}, \mathbf{u}_i)\|^2_{\Xi} + \frac{1}{2}\ln\frac{\mid \Xi \mid}{\sqrt{\mid \Lambda \mid\mid \Omega \mid}}\right)\right) \tag{2.5}$$

where

$$\Xi = \frac{\Lambda + \Omega}{2}$$

$$p(\mathbf{z}_k \mid \mathbf{x}_{i_k}, \mathbf{l}_{j_k}) \propto \exp\left(-\frac{1}{2}\left(\frac{1}{8}\,\|\mathbf{z}_k - h(f(\mathbf{x}_{i-1}, \mathbf{u}_i) + \mathbf{w}_p, \mathbf{l}_j)\|^2_{\Psi} + \frac{1}{2}\ln\frac{\mid \Psi \mid}{\sqrt{\mid \Lambda \mid\mid \Gamma \mid}}\right)\right) \tag{2.6}$$

where

$$\Psi = \frac{\Lambda + \Gamma}{2}$$

The distance measure raised by the exponential term in the above equations are given by the Bhattacharyya distance [13] between the noise distributions of different sensor models. It is more reliable than the usual Mahalanobis distance formulation for finding the distance between distributions of different standard deviations.

Remark: We denoted $\mathbf{x}_i$ as the ground truth value of the current pose. However, in practise ground truth is either not available or extremely difficult to obtain. But it has to be noted that none of the above equations in this section has $\mathbf{x}_i$ term on the right hand side. This means that we do not require the ground truth for any of our calculations and only require the previous pose $\mathbf{x}_{i-1}$ which is obtained by forward simulating the process model during the last iteration. Effectively, all that is represented by the Equation 2.5 and 2.6 are the distributions of error between various sensor models.

2.1.1 Formulating as Optimization

The optimal estimate of the unknown variables is obtained by minimizing the error distribution. The error distribution that is obtained in the previous subsection eventually becomes the cost function to be optimized. The error magnitude given by $\|.\|^2$ in Equation 2.5 and 2.6 has to be minimum for the probabilities to attain maximum. Note that the constant natural logarithm term is immaterial in the optimization. This is called as Maximum a Posteriori (MAP) estimate of the unknown variables. The problem could easily be converted in to non-linear least squares optimization to take advantage of the state-of-the-art sparse solvers based on Gauss-Newton or the Levenberg-Marquardt algorithm [14]. The estimate of the unknown variables at peak probability from Equation 2.1:

$$\Theta^* = \arg\max_{\Theta} P(X, U, Z, L) \tag{2.7}$$

where $\Theta = [X; L]$ and $\Theta^* = [X^*; L^*]$ is the augmented state vector containing all the state and landmark unknowns. Maximizing the above function is equal to minimizing its negative log likelihood:

$$\Theta^* = \arg\min_{\Theta} -\ln\, P(X, U, Z, L) \tag{2.8}$$

This gives the most famous non-linear Least Squares formulation of the SLAM problem:

$$\begin{aligned}\Theta^* = \arg\min_{\Theta}\Bigg\{ & \frac{N}{16}\sum_{i=1}^{N} \|\mathbf{x}_{i-1} + s(\mathbf{d}_{i-1}, \mathbf{d}_i) - f(\mathbf{x}_{i-1}, \mathbf{u}_i)\|_{\Xi}^2 + \\ & \frac{M}{16}\sum_{k=1}^{M} \|\mathbf{z}_k - h(f(\mathbf{x}_{i-1}, \mathbf{u}_i) + \mathbf{w}_p, \mathbf{l}_j)\|_{\Psi}^2 + \frac{1}{4}\ln\left(\frac{\mid \Xi \mid\mid \Psi \mid}{\mid \Lambda \mid \sqrt{\mid \Omega \mid\mid \Gamma \mid}}\right)\Bigg\} \qquad (2.9)\end{aligned}$$

where $\|\mathbf{e}\|_{\Sigma} = \mathbf{e}^{\top}\Sigma^{-1}\mathbf{e}$. Although the constant natural logarithm term does not matter during optimization it is necessary to include it in case of time varying covariance as used by us.

2.1.2 Optimization for Multiple robots

Formulating the optimization objective for multiple robots is simple once the Equation 2.9 is given. However, the difficulty arises during real-time implementation as it involves online fusing of multiple factor graphs and initializing the relative pose graphs. The next two chapters exclusively contributes towards that by introducing smart numerical techniques.

In SLAM, loop closures play an important role in significantly improving the overall estimate of all the unknown variables. A loop introduces correlations between the current pose and previously observed landmarks, which themselves are connected to earlier parts of the trajectory. During an update after the loop closure, the belief is propagated all the way through the trajectory to improve the estimate based on the rich set of information obtained. In multi-robot scenario, such belief propagations can be very non-trivial. A robot-robot encounter could give rise to interesting correlations between same portions of map previously visited by both the robots. In such cases, propagating the belief across both the trajectory increases the confidence of the overall map. Figure 2.2 shows a belief network of a multi-robot encounter. It can be seen that there are two types of interactions - 1) robot-robot and 2) robot-landmark-robot.

Different robots have different sensor models. This is due to variations in the wheel diameter, LIDAR manufacturer *etc.* Upgrading the above symbols to multiple robots, let

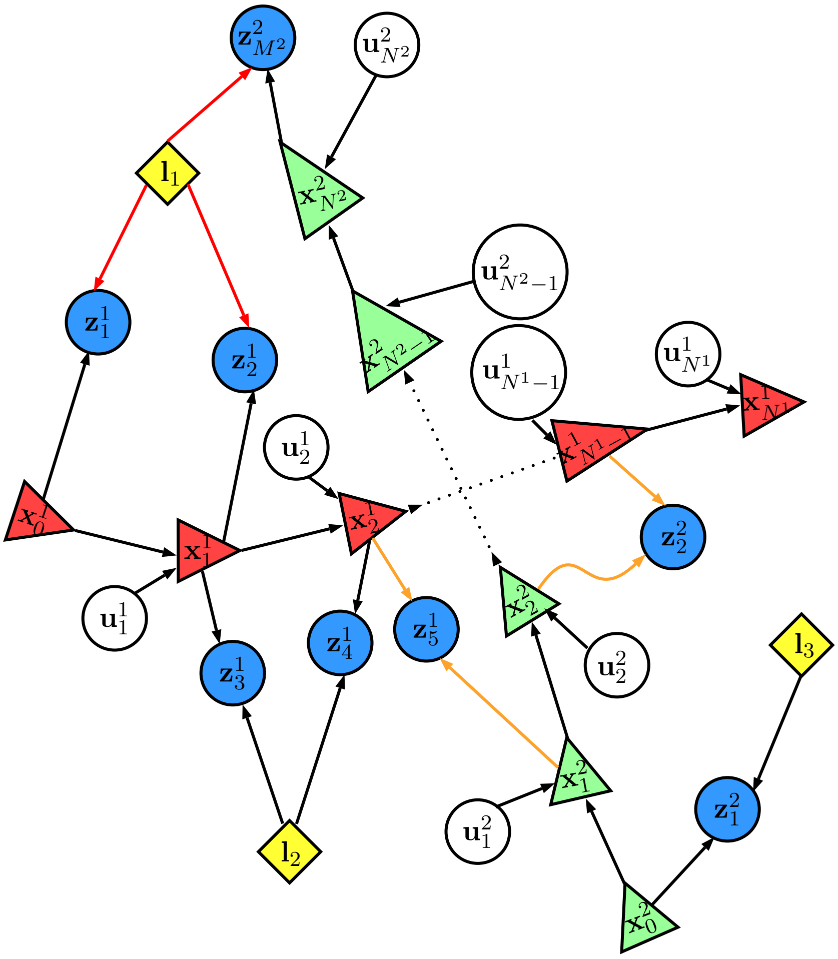


FIGURE 2.2: A belief network showing direct and indirect encounter in a multi-robot scenario. Robot-robot encounters (direct) are shown by orange arrows and a robot-landmark-robot encounter (indirect) are shown by red arrows.

$X_r = \{\mathbf{x}^r_i\}^{r=R,i=N^r}_{r=1,i=0}$, $U_r = \{\mathbf{u}^r_i\}^{r=R,i=N^r}_{r=1,i=1}$ and $Z_r = \{\mathbf{z}^r_k\}^{r=R,k=M^r}_{r=1,k=0}$ represent the robot's position, control inputs and observations for different robots $r \in 1...R$. The landmarks L need not be redefined as they depend on the environment and not on the number of robots. Then the Equation 2.9 for multi-robot scenario becomes:

$$\begin{aligned}\Theta^*_r = \arg\min_{\Theta_r}\Bigg\{ & \frac{N^r}{16}\sum_{i=1}^{N^r}\left\|\mathbf{x}^r_{i-1} + s_r(\mathbf{d}^r_{i-1},\mathbf{d}^r_i) - f_r(\mathbf{x}^r_{i-1},\mathbf{u}^r_i)\right\|^2_{\Xi^r} + \\ & \frac{M^r}{16}\sum_{k=1}^{M^r}\left\|\mathbf{z}^r_k - h_r(f(\mathbf{x}^r_{i-1},\mathbf{u}^r_i) + \mathbf{w}_p, \mathbf{l}_j)\right\|^2_{\Psi^r} + \\ & \frac{1}{4}\ln\left(\frac{\mid \Xi^r \mid\mid \Psi^r \mid}{\mid \Lambda^r \mid \sqrt{\mid \Omega^r \mid\mid \Gamma^r \mid}}\right)\Bigg\} \end{aligned} \tag{2.10}$$

where $f_r(.)$, $s_r(.)$, $h_r(.)$, w^r_p, w^r_s, w^r_m, Λ^r, Ω^r, Γ^r are the process, scan-matching and measurement model, noise and their covariance respectively. Similarly, $\Xi^r = \frac{\Lambda^r+\Omega^r}{2}$ and

$\Psi^r = \frac{\Lambda^r + \Gamma^r}{2}$. The optimization is very similar to Extended Kalman Filter (EKF) in [15] except that they are iterated continuously until convergence.

2.2 Solving the Multi-robot Least-Square SLAM

Both Gauss-Newton and Levenberg-Marquardt proceeds by finding the step value to be added to all the unknowns at every iteration using the approximate linear system at a particular linearization point. To begin with, this linearization point is supplied as initial guess on the decision variables. In other words, we work on the linearized form of the objective function. By using first-order Taylor series approximation on the first term of the objective function in Equation 2.10 representing the difference odometry and scan-matching model,

$$\mathbf{x}^r_{i-1} + s_r(\mathbf{d}^r_{i-1}, \mathbf{d}^r_i) - f_r(\mathbf{x}^r_{i-1}, \mathbf{u}^r_i) \tag{2.11}$$

$$\approx {}^0\mathbf{x}^r_{i-1} + \delta^0\mathbf{x}^r_{i-1} + \{s_r({}^0\mathbf{d}^r_{i-1}, {}^0\mathbf{d}^r_i) + S^{i-1}_r \delta\mathbf{d}^r_{i-1} + S^i_r \delta\mathbf{d}^r_i\} - \{f_r({}^0\mathbf{x}^r_{i-1}, \mathbf{u}^r_i) + F^{i-1}_r \delta\mathbf{x}^r_{i-1}\} \tag{2.12}$$

$$= \{\delta^0\mathbf{x}^r_{i-1} + S^{i-1}_r \delta\mathbf{d}^r_{i-1} + S^i_r \delta\mathbf{d}^r_i - F^{i-1}_r \delta\mathbf{x}^r_{i-1}\} + \{{}^0\mathbf{x}^r_{i-1} + s_r({}^0\mathbf{d}^r_{i-1}, {}^0\mathbf{d}^r_i) - f_r({}^0\mathbf{x}^r_{i-1}, \mathbf{u}^r_i)\} \tag{2.13}$$

$$= \{\delta^0\mathbf{x}^r_{i-1} + S^{i-1}_r \delta\mathbf{d}^r_{i-1} + S^i_r \delta\mathbf{d}^r_i - F^{i-1}_r \delta\mathbf{x}^r_{i-1}\} + \boldsymbol{\alpha}^r_i \tag{2.14}$$

where F^{i-1}_i is the Jacobian of the process model with respect to linearization point, $\boldsymbol{\alpha}^r_i = {}^0\mathbf{x}^r_{i-1} + s_r({}^0\mathbf{d}^r_{i-1}, {}^0\mathbf{d}^r_i) - f_r({}^0\mathbf{x}^r_{i-1}, \mathbf{u}^r_i)$ is the difference in odometry and scan-matching prediction and a superscript on the left side of the variable indicate the index number of optimization iteration, a "0" indicates initial guess.

$$F^{i-1}_r := \left.\frac{\partial f_r(\mathbf{x}^r_{i-1}, \mathbf{u}^r_i)}{\partial \mathbf{x}^r_{i-1}}\right|_{{}^0\mathbf{x}^r_{i-1}} \tag{2.15}$$

$$S^{i-1}_r := \left.\frac{\partial s_r(\mathbf{d}^r_{i-1}, \mathbf{d}^r_i)}{\partial \mathbf{d}^r_{i-1}}\right|_{({}^0\mathbf{d}^r_{i-1}, {}^0\mathbf{d}^r_i)} \tag{2.16}$$

$$S^i_r := \left.\frac{\partial s_r(\mathbf{d}^r_{i-1}, \mathbf{d}^r_i)}{\partial \mathbf{d}^r_i}\right|_{({}^0\mathbf{d}^r_{i-1}, {}^0\mathbf{d}^r_i)} \tag{2.17}$$

The first-order Taylor expansion of the second term in the objective function Equation 2.10 scan-matching and measurement model is as follows:

$$\mathbf{z}_k^r - h_r(f_r(\mathbf{x}_{i-1}^r, \mathbf{u}_i^r) + \mathbf{w}_p, \mathbf{l}_j) \tag{2.18}$$

$$\approx \mathbf{z}_k^r - \{h_r(f_r({}^0\mathbf{x}_{i-1}^r, \mathbf{u}_i^r),{}^0\,\mathbf{l}_{j_k}^r) + H_r^{i_k}\delta\mathbf{x}_{i_k}^r + J_r^{j_k}\delta\mathbf{l}_{j_k}\} \tag{2.19}$$

$$=\boldsymbol{\beta}_i^r - \{H_r^{i_k}\delta\mathbf{x}_{i_k}^r + J_r^{j_k}\delta\mathbf{l}_{j_k}\} \tag{2.20}$$

where $H_r^{i_k}$ and $J_r^{j_k}$ is the Jacobian of measurement model with respect to $\mathbf{x}_{i_k}^r$ and $\mathbf{l}_{i_k}^r$ and $\boldsymbol{\beta}_i^r = \mathbf{z}_k^r - h_k(f({}^0\mathbf{x}_{i-1}^r, \mathbf{u}_i^r),{}^0\,\mathbf{l}_{j_k}^r)$ is the measurement prediction error.

$$H_r^{i_k} := \frac{\partial h_r(\mathbf{x}_{i_k}, \mathbf{l}_{j_k})}{\partial f(\mathbf{x}_{i-1}^r, \mathbf{u}_i^r)} \cdot \frac{\partial f_r(\mathbf{x}_{i-1}^r, \mathbf{u}_i^r)}{\partial \mathbf{x}_{i-1}^r}\bigg|_{({}^0\mathbf{x}_{i_k}^r, {}^0\mathbf{l}_{j_k}^r)} \tag{2.21}$$

$$J_r^{j_k} := \frac{\partial h_r(\mathbf{x}_{i_k}, \mathbf{l}_{j_k})}{\partial \mathbf{l}_{j_k}^r}\bigg|_{({}^0\mathbf{x}_{i_k}^r, {}^0\mathbf{l}_{j_k}^r)} \tag{2.22}$$

Substituting the Taylor approximated linear system formulated in the Equations 2.14 and 2.20 back in the non-linear least squares problem in 2.10:

$$\begin{aligned}\delta\Theta_r^* = arg \min_{\delta\Theta_r}\Bigg\{ & \frac{N}{16}\sum_{i=1}^{N} \left\|G_r^{i-1}\delta^0\mathbf{x}_{i-1}^r + S_r^{i-1}\delta\mathbf{d}_{i-1}^r + S_r^i\delta\mathbf{d}_i^r - F_r^{i-1}\delta\mathbf{x}_{i-1}^r + \boldsymbol{\alpha}_i^r\right\|_{\Xi}^2 + \\ & \frac{M}{16}\sum_{k=1}^{M} \left\|\boldsymbol{\beta}_i^r - H_r^{i_k}\delta\mathbf{x}_{i_k}^r + J_r^{j_k}\delta\mathbf{l}_{j_k}\right\|_{\Psi}^2 + \\ & \frac{1}{4}\ln\left(\frac{\mid \Xi^r \mid\mid \Psi^r \mid}{\mid \Lambda^r \mid \sqrt{\mid \Omega^r \mid\mid \Gamma^r \mid}}\right)\Bigg\}\end{aligned} \tag{2.23}$$

where $\delta\Theta_r = [\delta X_r; \delta L_r]$ and $\delta\Theta_r^* = [\delta X_r^*; \delta L_r^*]$ is the step size to be added to all the unknown variables towards minimizing the cost function. This now forms a linear least squares problem in $\delta\Theta$. The length of the state vector is given as $n^r = N^r\mathbf{d}_{\mathbf{x}}^r + K^r\mathbf{d}_{\mathbf{l}}^r$ where $\mathbf{d}_{\mathbf{x}}^r$ and $\mathbf{d}_{\mathbf{l}}^r$ is the dimension of pose and landmark variables. $G_r^{i-1} = I_{\mathbf{d}_x^r \times \mathbf{d}_x^r}$ is the identity matrix of size $\mathbf{d}_x^r$. The $\delta\mathbf{d}_{i-1}^r$ and $\delta\mathbf{d}_i^r$ terms are not added to the state vector as they are estimated as a part of the scan-matching process. See [Appendix] [Appendix 2] for details. The covariance terms weighting the squared norms could be removed by multiplying the Jacobians with the transpose of inverse square-root of the covariance:

$$\|\mathbf{e}\|_{\Sigma}^{2} := \mathbf{e}^{\top}\Sigma^{-1}\mathbf{e} = (\Sigma^{-T/2}\mathbf{e})^{\top}(\Sigma^{-T/2}\mathbf{e}) = \left\|\Sigma^{-T/2}\mathbf{e}\right\|^{2} \tag{2.24}$$

By using the relation 2.24 in Equation 2.23 we can simplify it to group all the model specific Jacobians into single large Jacobian $A_r \in \mathbb{R}^{m\times n}$ where m is the total number of input measurements scaled to dimension of each measurement. The constants in the objective function including $\boldsymbol{\alpha}_i^r$, $\boldsymbol{\beta}_i^r$ and the natural logarithm term can be grouped to form $\mathbf{b}^r \in \mathbb{R}^m$. We obtain a over-determined system of linear equations that require inverting a huge matrix to find the solution:

$$\delta\Theta_r^* = arg\min_{\delta\Theta_r} \|A_r\delta\Theta_r - \mathbf{b}^r\|^2 \tag{2.25}$$

The above optimization can be minimized by setting the derivative equal to zero. With the assumption that we begin at a good linearization point (good initial guess) we are bound to slide into the global minima.

2.2.1 Incremental Optimization

An optimization problem is often supplemented by an initial guess for the decision variables. In our case, we have to supply the initial guess for the variables to be estimated whose error will then be iteratively reduced based on the measurements. However, one important aspect to be worried about is the source of the initial guess. Modern robots are mostly accompanied with more sensors than necessary for localization, and hence one of them can be utilized as the source for initial guess. But if SLAM is done as a batch process with a fixed lag and not at real-time this again becomes a problem. This is because, sensors like odometry encoder, LIDAR and Inertial Measurement Units (IMU) are very good only at providing a local estimate between nearly successive pose frames but drifts away significantly when forward integrated continuously. Figure 2.3 shows an example comparison between a robot's trajectory generated from dead reckoning odometry and SLAM reconstruction. Thus, incremental optimization is beneficial in several ways that include:

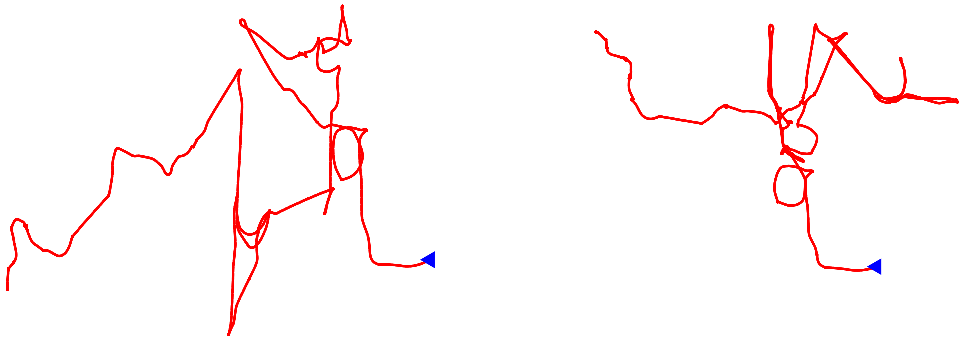


FIGURE 2.3: Left: Trajectory obtained by just forward integrating the odometry provided by robot. Right: Trajectory reconstructed after batch SLAM. It can be seen that the plain odometry trajectory looks same as the reconstructed trajectory at the beginning (blue triangle) but starts drifting away due to integral error.

1. Easier to provide an accurate initial guess based on local sensor information as we have an optimized estimate for variables processed so far.
2. A good initial guess means that it is close to the optimal solution and saves a lot of optimization time and iterations.
3. A good initial guess also significantly reduces the probability of getting trapped in a local minima. For example, in Figure 2.3, the dead-reckoning odometry trajectory as an initial guess for a batch SLAM optimizer is far away from the optimal solution and may suffer local minima.
4. Finally, an incremental system gives optimal estimates on the fly and is real-time.

For our purpose, we use ISAM2 [9] as the underlying optimization algorithm.

2.2.1.1 Jacobian Decomposition

This is a well known material but discussed mainly to differentiate linear algebra algorithms with graph theoretic view in the next section. A detailed study is available in the standard textbook [16]. Matrix decomposition is a very common technique in mathematics and engineering disciplines to factorize the matrix into product matrices so as to arrive at the solution faster than pure inversion. Among several methods available, Cholesky, QR and LDL [16] decomposition are the ones commonly used by the SLAM community. The Jacobian matrix A_r in Equation 2.25 has a column for every variable being estimated of size n and a row for every sensor measurement of size m. It is a very sparse matrix owing to the fact that different measurements are recorded at every pose variable and every measurement measures no more than a few pose variables. As there are multiple sensors, total number of measurements are very large compared to number of variables, i.e. $m \gg n$. With respect to time complexity, both Cholesky and QR factorization require $\mathcal{O}(mn^2)$ operations for dense matrices when $m \gg n$. But for dense and sparse matrices, in practice, we have seen that both Cholesky and LDL factorization outperform QR factorization at least by a factor of 2. However, mathematical tools from linear algebra and graph theory literature like Gram-Schmidt orthogonalization and Householder reflections for QR decomposition along with Givens rotations [16] for incremental update of the square-root information matrix R_r make QR decomposition as the most suitable choice for our purpose. Also, QR decomposition works directly on the Jacobian matrix A_r unlike Cholesky which needs the information matrix, $\mathcal{I} = {A_r}^\top A_r$ to be computed. It is also more accurate and numerically stable compared to Cholesky decomposition. The Q_r matrix is orthogonal and is usually not formed as a part of factorization. The QR factorization of the Jacobian matrix A_r is:

$$A_r = Q_r \begin{bmatrix} R_r \\ 0 \end{bmatrix} \tag{2.26}$$

Applying this factorization to the multi-robot linearized least-squares problem 2.25:

$$\begin{aligned}
\|A_r\Theta - \mathbf{b}^r\| &= \left\| Q_r \begin{bmatrix} R_r \\ 0 \end{bmatrix} \Theta_r - \mathbf{b}^r \right\|^2 && (2.27)\\
&= \left\| Q_r^\top Q_r \begin{bmatrix} R_r \\ 0 \end{bmatrix} \Theta_r - Q_r^\top \mathbf{b}^r \right\|^2 && (2.28)\\
&= \left\| \begin{bmatrix} R_r \\ 0 \end{bmatrix} \Theta_r - \begin{bmatrix} \mathbf{d}^r \\ \mathbf{e}^r \end{bmatrix} \right\|^2 && (2.29)\\
&= \|R_r\Theta_r - \mathbf{d}^r\|^2 + \|\mathbf{e}^r\|^2 && (2.30)
\end{aligned}$$

where R_r is the upper triangular square-root information matrix or square-root factor obtained using QR factorization, $Q_r^\top \mathbf{b}^r = [\mathbf{d}^r; \mathbf{e}^r]$, $\mathbf{d}^r \in \mathbb{R}^n$, $\mathbf{e}^r \in \mathbb{R}^{m-n}$ and $R_r^\top R_r = A_r^\top A_r$. For the above Equation 2.30 to attain minimum, $R_r \Theta_r$ should be equal to $\mathbf{d}^r$. This leaves the residual of the least squares problem given by $\|\mathbf{e}^r\|^2$. Since R_r is upper triangular, back-substitution is trivial and the estimate $\delta\Theta_r$ is added to the linearization point to update and proceed with the next iteration.

2.3 Foundations of Matrix and Graphs

A major key to the improving performance of pose graph based SLAM algorithms is rooted in tiny numerical optimizations developed over time. One has to analyze the deep connections between Linear Algebra and Graph Theory in conjunction with SLAM algorithms to understand this fact better. Due to the nature of SLAM problem, we specifically deal with sparse graphs and matrices. Both sparse linear algebra and graph theory has flourished over the past four decades with each being critical to other [17]. This section gives a brief grounding of terminologies frequently used in the upcoming chapters and then examines the relationship between matrices and graphs.

2.3.1 Graph Theory Terminology

Definition: A **graph** $G = (V, E)$ is a collection V of vertices and $E \subset V \times V$ of edges. Informally, we think of the edges as linking the pairs of vertices that they correspond to, and typically represents graphs by drawings in which we connect the endpoints by a curve.

Glossary of Terms:

- A graph is **simple** if every edge links a unique pair of distinct vertices.
- A graph is **bipartite** if the vertex set can be partitioned into two sets $V_1 \cup V_2$ such that edges only run between V_1 and V_2.

- A **clique** on n vertices, denoted K_n, is the n-vertex graph with all $\binom{n}{2}$ possible edges.
- A **complete graph** on n vertices, denoted K_n, is the n-vertex graph with all $\binom{n}{2}$ possible edges.
- A graph is **connected** if there is a path between every pair of distinct vertices.
- A **cycle** is a path for which the first and last vertices are the same.
- A **chord** is an edge that is not part of the cycle but connects two vertices of the cycle.
- A **chordal graph** is one in which all cycles of four or more vertices have a chord.
- The **degree** $d(v)$ of a vertex v is the number of edges that are incident to v.
- We say that an edge e is **incident** to a vertex v if v is an endpoint of e.
- A **path** is a sequence of distinct, pairwise-adjacent vertices.
- A graph is **planar** if it is possible to draw it in the plane without any crossing edges.
- A **tree** is a connected graph with no cycles.
- A **subgraph** of a graph G is another graph formed from a subset of the vertices and edges of G. The vertex subset must include all endpoints of the edge subset, but may also include additional vertices.
- A **supergraph** is a graph formed by adding vertices, edges, or both to a given graph. If H is a subgraph of G, then G is a supergraph of H.
- A **connected component** (or just **component**) of an undirected graph is a subgraph in which any two vertices are connected to each other by paths, and which is connected to no additional vertices in the supergraph.

2.3.2 Factor Graph and Bayes Tree

A formal introduction of factor graphs [6] and Bayes tree [18] is presented below. I will also provide details on how the formulations made in this subsection relate to ones derived previously. This section might slightly abuse the usage of symbols and hence should not be confused with the usage in the previous sections.

2.3.2.1 Factor Graph

A factor graph is a bipartite graph $\mathcal{G} = (\mathcal{C}, \Theta, \mathcal{E})$ with two node types: factor nodes $c^p \in \mathcal{C}$ and variable nodes $\theta^q \in \Theta$. Edges $e^{pq} \in \mathcal{E}$ are always between factor nodes and variable nodes. A factor graph $\mathcal{G}$ represents the factorization of a function:

$$c(\Theta) = \prod_p c^p(\Theta_p) \tag{2.31}$$

where Θ^p is the set of variables θ^q adjacent to the factor c^p. The independence relationship between the factors and variables are encoded by the edges e^{pq}. Every factor c^p is a function of variables contained in Θ^p. Assuming Gaussian measurement models:

$$c^{pq} \propto \exp\left(-\frac{1}{2}\left\|\mathcal{H}^p(\Theta^p) - \mathcal{Z}^p\right\|^2_{\Sigma^p}\right) \tag{2.32}$$

The above equation is a consolidation of detailed objective function listed in 2.23 to minimize the net difference between prediction $\mathcal{H}^p(\Theta^p)$ and measurement $\mathcal{Z}^p$ weighted by the covariance Σ^p.

2.3.2.2 Bayes Network

A Bayes net is obtained as an intermediate structure when *eliminating* a factor graph into Bayes tree. Eliminating a factor graph into a Bayes tree uses a variable elimination procedure which has roots dated 2000 years ago in Chinese and Indian literature [19]. In the modern times, variable elimination was first implemented by C. F. Gauss in 1809 [20]. Factor graphs elimination is done using bipartite elimination game that was proposed by

Heggernes *et al.* [21]. Variable elimination is a procedure of expressing a variable in terms of other, one at a time based on sequence given by variable ordering. On eliminating every variable a corresponding node is introduced in the Bayes net that has conditional variables directed towards it. The conditional variables are the independent variables used to express the eliminated variable at every iteration. This process continues until all the variables are eliminated in the factor graph. Thus, the structure of the Bayes net depends on the choice of variable ordering and any operation carried out on it is specific to that ordering. The pseudo code for a quick understanding of eliminating the factor graph into Bayes net is present in [18]

2.3.2.3 Bayes Tree

The Bayes net resulting from the elimination of variables is chordal. This chordal graph can be converted into a Bayes tree by identifying the cliques in the graph. This was first intoduced by Kaess *et al.* in [18]. A Bayes tree is a directed tree where the nodes represent cliques of the underlying chordal Bayes net. In this respect, Bayes trees are similar to clique trees, but a Bayes tree is directed and is closer to a Bayes net in the way it encodes a factored probability density. Every chordal Bayes net can be transformed into a tree by discovering its cliques. Discovering cliques in chordal graphs is done using the maximum cardinality search algorithm by Tarjan and Yannakakis [22], which proceeds in the *reverse elimination order* to discover cliques in the Bayes net. In this regard, as a Bayes tree is generated out of the Bayes net, the structure of the Bayes tree is also specific to the variable ordering. From the Bayes tree it is not possible to retrieve the order in which the variables were measured.

2.3.3 Matrix vs. Graph

In the following chapters, this thesis revolves around two main types of graphical models and a type of tree data structure. They are central in understanding the contributions of the work and gaining sufficient intuition on how they are related to their equivalent matrix representations is imperative. The two types of graphical models include factor graphs [6] and Bayesian Network. Bayes Tree [18] is the tree data structure used widely in this work.

2.3.3.1 Jacobian vs. Factor Graph

The measurement Jacobian A_r is the matrix of the factor graph associated with SLAM. This statement can be understood in two stages:

1. Every block of A_r corresponds to one term in the least-squares criterion 2.23, either a landmark measurement or an odometry/scan-matching constraint, and every block-row corresponds to one factor in the factor graph. Within each block-row, the sparsity pattern indicates which unknown poses and/or landmarks are connected to the factor. Hence, the block-structure of A corresponds exactly to the adjacency matrix of the factor graph associated with SLAM.

2. At the scalar level, every row A^j in A corresponds to a scalar term $\left\| A_r^j \delta - \mathbf{b}^r \right\|_2^2$ in the sparse matrix least squares criterion:

$$\|A_r \delta - \mathbf{b}\|_2^2 = \sum_j \left\| A_r^j \delta - \mathbf{b}^r \right\|_2^2 \tag{2.33}$$

   The reordering is not done across all the scalar parts of the pose and landmark variables as done by the standard linear algebra literature. But for our case, such a fine and granular ordering is not needed for two reasons 1) We would then be operating on a larger matrix and hence takes longer time for finding a variable ordering. 2) A single pose constraint or landmark measurement introduces a block and all the scalars in them are generally non-zero. So it is better to operate on these blocks rather than those fine scalar variables [10].

2.3.3.2 Square-root Information vs. Bayes Network

The square-root information matrix is more or less the adjacency matrix of the Bayes net. Since it is a directed graph, the adjacency matrix is not going be symmetric.

Nevertheless, the square-root factor R_r is upper triangular and not a symmetric matrix. The arrows pointing towards any node in the Bayes net denoting the variables on which the node is conditioned upon, is analogous to a non-zero present other than the main diagonal in the row corresponding to that node. The Bayes net also indicates conditional independence relationship between variables as that of a factor graph. But the relation follows a causal direction implied by the choice of variable ordering. By definition, although a Bayes net does not have any cycles they contain loops. Storing and updating the Bayes net by attaching new factors and variables to a node in these loops is intractable because 1) Changes to the direction of arrows within a clique is not understood [18] 2) Additional variables that get affected is not elegantly traceable.

2.3.3.3 Square-root Information vs. Bayes Tree

One of the finest outcomes from The Borg Lab at Georgia Tech was incremental smoothing and mapping using the Bayes tree (ISAM2) [9]. The increase in performance over other pose graph SLAM solutions comes from using the Bayes tree [18] to represent the cliques of the square-root factor or the Bayes net in the incremental smoothing and mapping (iSAM). Updating the square root factor with a new measurement removes its upper triangularity. It is made upper triangular again by using Givens rotations [16]. During this process, the new measurement row is multiplied with the Givens rotation matrix producing new non-zeros replacing zeros in the upper triangle. The pattern in which these non-zeros are generated is unintuitive and was not understood in the matrix form. The new non-zero elements that are created during variable elimination is referred as fill-in. These blocks of non-zeros relate to the cliques in the Bayes tree. The row in R_r corresponding to the first eliminated frontal variable of any clique has non-zeros in the positions corresponding to the rest of the variables of that clique. However, by representing the square root factor R_r as a Bayes tree an independence relation conditioned on the *variable ordering* is established across variables that allows us to find the subset of variables that gets affected on adding a new measurement. This also led to an interesting concept of just-in-time fluid relinearization that updates the linearization point incrementally and efficient access to marginal covariances [23].

2.3.4 Factor Graph → Bayes Network → Bayes Tree

Despite the attempts to simplify the understanding of relevant matrix-graph relation, one may still wonder how to eliminate any arbitrary factor graph into a Bayes net and in turn convert a Bayes net into a Bayes tree. This portion of research has spanned over several decades and intersects different areas of study including sparse linear algebra, graph theory, probabilistic and statistical inference, finite element analysis and numerical analysis. It is therefore beyond the scope of this piece of work to provide a step-by-step explanation. However, a brief visual treatment is attempted to expedite the process of understanding in Figure 2.4 and 2.5. Some seminal works that could be referred include [18, 19, 21, 22, 24, 25]

2.4 Variable Reordering

This is the most important section of this chapter as it is the gateway towards the set of contributions. The most dramatic improvement in performance comes from choosing a good variable ordering when factorizing a matrix. This variable ordering is used by the graph theoretic algorithm called *variable eliminiation*, in which each variable is expressed in terms of other variables based on the order of elimination. The order in which the variables are eliminated has a large impact on the running time and storage of matrix factorization algorithms such as QR and Cholesky factorization. Finding an optimal ordering is an NP-complete problem [26], but there are numerous ordering heuristics and approximate algorithms that perform well on general problems [21, 27]. While a great deal of improvement in terms of storage and time complexity is obtained by using these general purpose orderings, yet another order-of-magnitude improvement is obtained by exploiting the structure of SLAM problem. Ordering the variables depending on the nature and structure of the problem is not unprecedented and has been a trend in linear algebra [28]. It is therefore valid to believe that more efficient and sophisticated algorithms can be developed by viewing the problem as one of the computation on a SLAM pose graph. While a large body of work is available for delivering a comprehensive

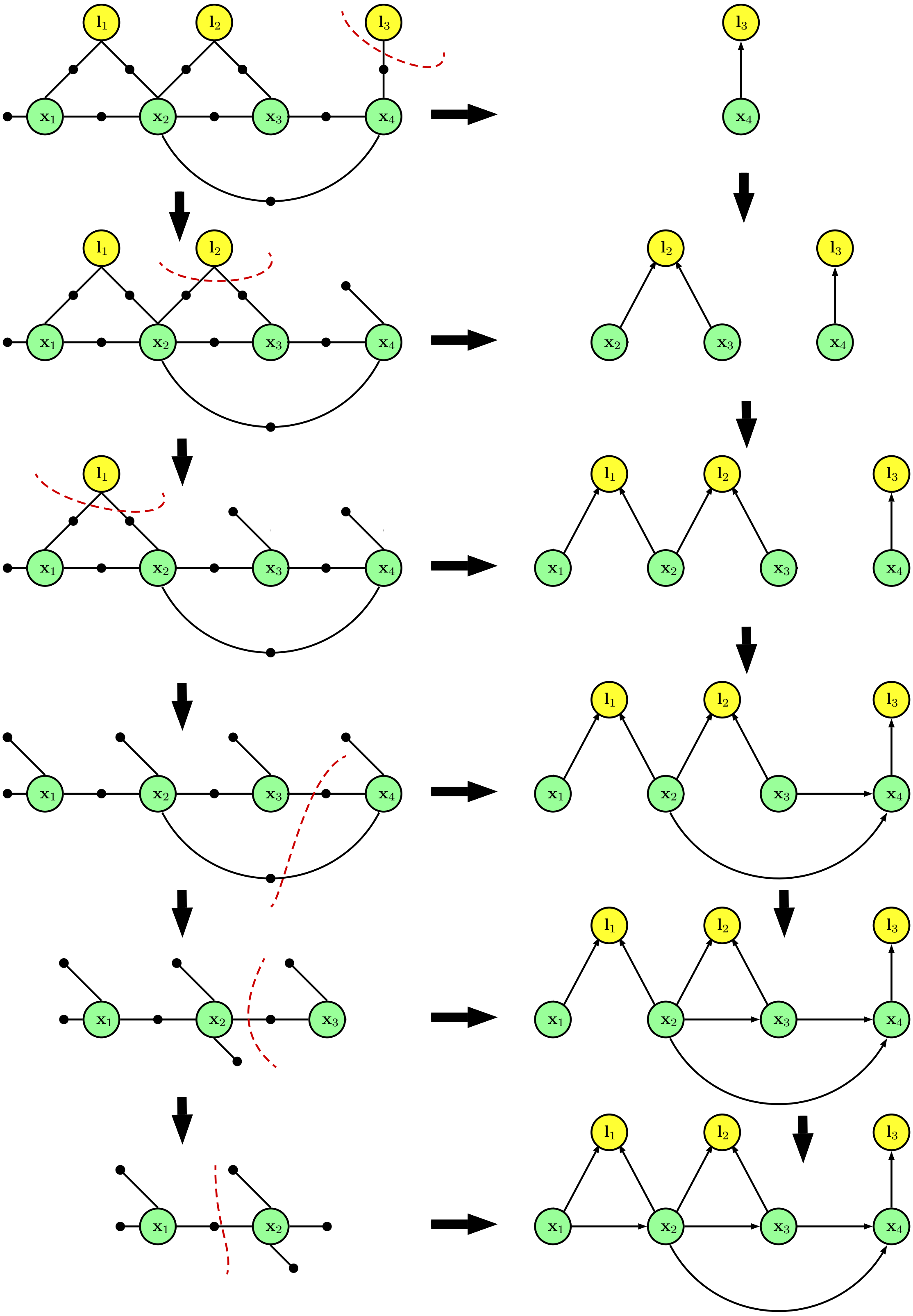


Figure 2.4: Eliminating a factor graph into a Bayes net. The elimination order used for converting is $O = [l_3, l_2, l_1, x_4, x_3, x_2, x_1]$. The vertex of the factor graph that is removed at every step is shown by a red dotted separator. The horizontal arrows indicate the equivalent Bayes net on removing a factor graph vertex. The vertical arrows point the progress within the factor graph and Bayes net.

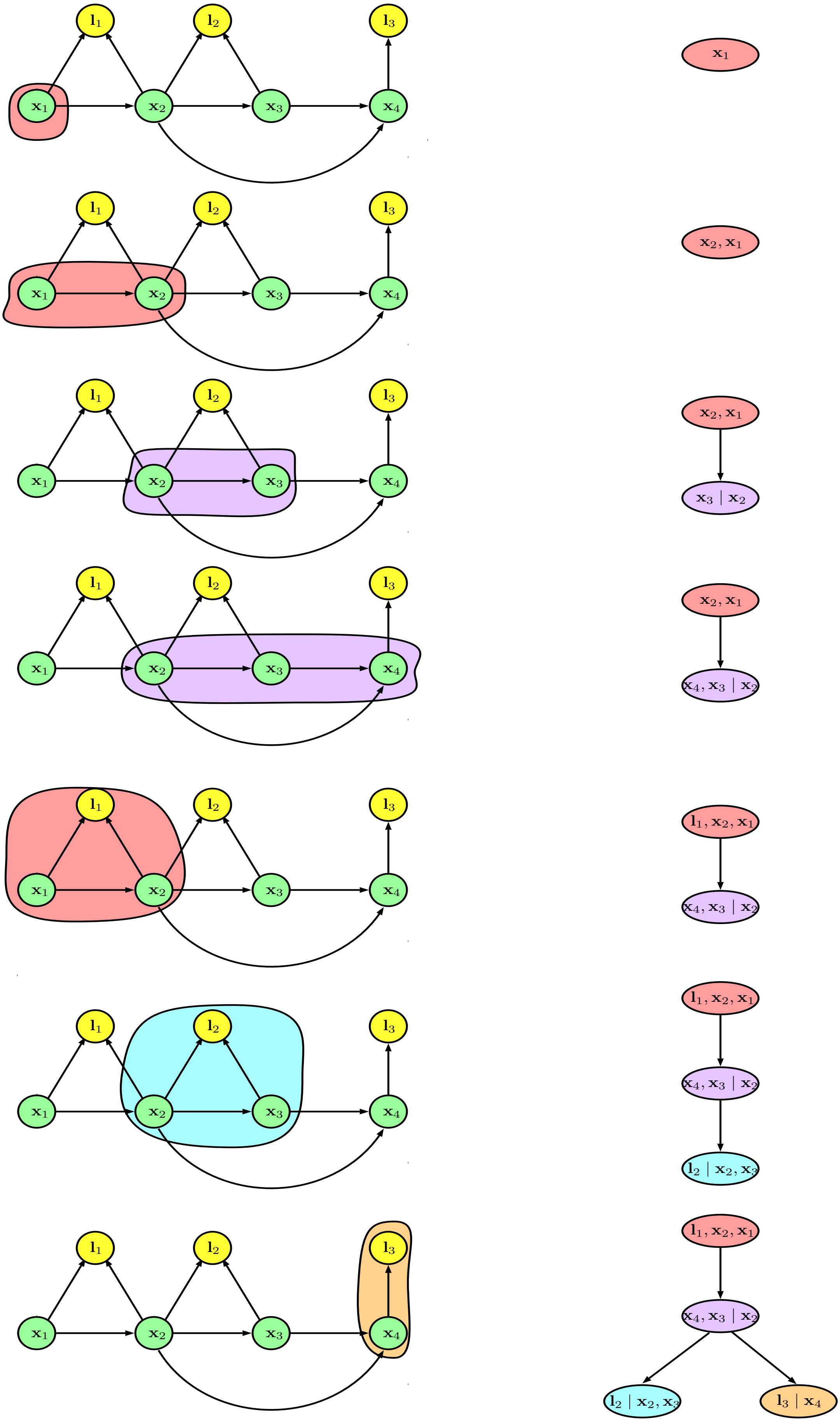


FIGURE 2.5: Converting a Bayes net into a Bayes tree by clique factorization. The bubble around the node(s) in the Bayes net indicate the cliques found at every iteration based on reverse elimination ordering $x_1, x_2, x_3, x_4, l_1, l_2, l_3$. The color of the bubble denotes the node being added to the same colored clique.

or esoteric explanation about the need for variable ordering, in my thesis, I will provide a quick overview using a practical example and simple language.

2.4.1 Variable Ordering for QR Factorization

Given a variable ordering O, the QR factorization using Gram-Schmidt process works as follows: At every iteration, a column vector as per the sequence in variable ordering is orthogonalized with respect to all the previously orthogonalized column vectors (the first column vector is taken as-is). These orthogonalized vectors are unit normalized and stacked horizontally to form the orthonormal Q matrix. The order of columns in Q follows the same ordering as O. Given a full column rank matrix $A = [\mathbf{a}_1, \cdots, \mathbf{a}_n]$ with inner product defined as $\langle \mathbf{v}, \mathbf{w} \rangle = \mathbf{v}^\top \mathbf{w}$ then $\mathbf{u}_k$ is:

$$\mathbf{u}_k = \mathbf{a}_k - \sum_{j=1}^{k-1} \mathrm{proj}_{\mathbf{u}_j} \mathbf{a}_k, \qquad \mathbf{e}_k = \frac{\mathbf{u}_k}{\|\mathbf{u}_k\|} \qquad \forall\ k = 1 \cdots n \tag{2.34}$$

where

$$\mathrm{proj}_{\mathbf{e}} \mathbf{a} = \frac{\langle \mathbf{e}, \mathbf{a} \rangle}{\langle \mathbf{e}, \mathbf{e} \rangle} \mathbf{e} \tag{2.35}$$

Then Q is given as,

$$Q = [\mathbf{e}_1, \cdots, \mathbf{e}_n] \tag{2.36}$$

As Q is an orthonormal matrix $Q^\top Q = 1$. The R matrix of QR factorization is now obtained using the below step:

$$R = Q^\top Q\, R = Q^\top A; \tag{2.37}$$

Expanding the value of R with $A = [\mathbf{a}_1, \cdots, \mathbf{a}_n]$ and Equation 2.36:

$$R = \begin{pmatrix} \langle \mathbf{e}_1, \mathbf{a}_1 \rangle & \langle \mathbf{e}_1, \mathbf{a}_2 \rangle & \langle \mathbf{e}_1, \mathbf{a}_3 \rangle & \ldots \\ 0 & \langle \mathbf{e}_2, \mathbf{a}_2 \rangle & \langle \mathbf{e}_2, \mathbf{a}_3 \rangle & \ldots \\ 0 & 0 & \langle \mathbf{e}_3, \mathbf{a}_3 \rangle & \ldots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix} \tag{2.38}$$

The inner product $\langle \mathbf{e_i}, \mathbf{a_j} \rangle = 0;\ \forall\ j < i$ because at every iteration k, $\mathbf{e}_k$ is the orthogonalized version of $\mathbf{a}_k$ with $\mathbf{e}_l;\ \forall\ l < k$. To retain the sparsity of the square root factor it is necessary to have as many inner products, $\langle \mathbf{e_i}, \mathbf{a_j} \rangle$, equalling 0 as possible. The inner product of two vectors are zero if they are orthogonal or if the intersection of their components forms a nullset. The latter is a specific case of the former. From the general representation of R in Equation 2.38 it can be seen that, the lower the value of subscript of $\mathbf{e}$ higher is its presence. So lesser the dimension of a $\mathbf{e}_i$ for lesser values of i higher is the probability that it does not share a component with $\mathbf{a}_j$. The vector $\mathbf{e}_i$ with the lowest dimension corresponds to a column i in A with maximum zeros. In other words, as the columns of the Jacobian matrix A is same as the nodes of the factor graph, the previous statement refers to node with the lowest degree. Based on the above explanation, an example matrix and its equivalent factor graph with two different orderings are factorized to show the difference in the non-zero pattern in R.

Consider a measurement Jacobian A as given below with the column number annotated on top of each column:

$$A = \begin{array}{c} \begin{array}{ccccc} 1 & 2 & 3 & 4 & 5 \end{array} \\ \begin{pmatrix} 2 & 0 & 0 & 4 & 0 \\ 0 & 4 & 8 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 10 \\ 0 & 0 & 6 & 6 & 0 \end{pmatrix} \end{array}$$

Let us assume the column elimination ordering as $O = [x_2, x_3, x_4, x_1, x_5]$. The transformed matrix A^O based on the ordering O and the structure of R^O on factorizing the transformed matrix is given below:

$$A^O = \begin{array}{c} \begin{array}{ccccc} x_2 & x_3 & x_4 & x_1 & x_5 \end{array} \\ \begin{pmatrix} 0 & 0 & 4 & 2 & 0 \\ 4 & 8 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 10 \\ 0 & 6 & 6 & 0 & 0 \end{pmatrix} \end{array}$$

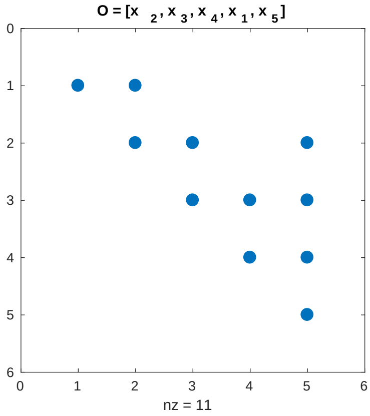


FIGURE 2.6: Left: Matrix in which the lower degree node x_2, $d(x_2) = 1$, is ordered before the higher degree node 3, $d(x_3) = 3$. The structure of square root factor R^O with 11 non-zero elements.

The above ordering is also performed in the factor graph and factorized into a Bayes tree as shown in Figure 2.7

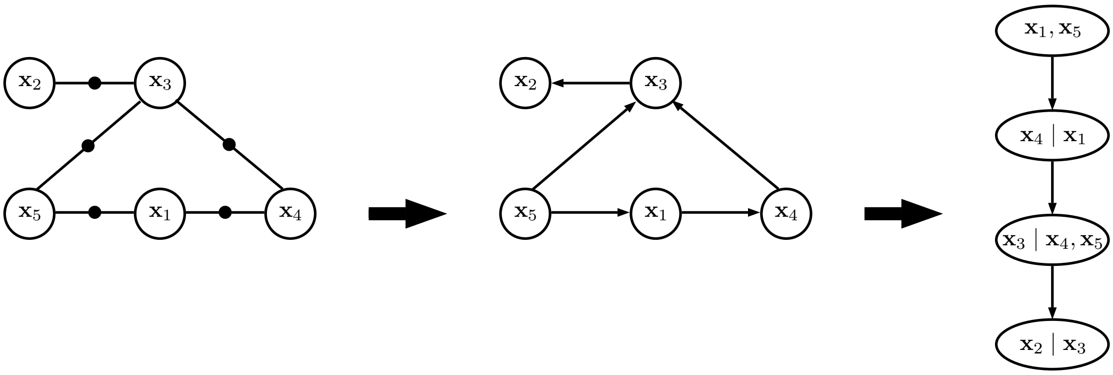


FIGURE 2.7: Eliminating a factor graph into a Bayes net and in turn into a Bayes tree. A good ordering has resulted in smaller cliques or reduced fill-in.

Whereas if the matrix is ordered by placing the vector with highest dimension or the node with the highest degree at the first place, for example $O = [x_3, x_4, x_1, x_2, x_5]$, then the structure of square root factor R^O is as follows:

$$A^O = \begin{array}{c} \begin{array}{ccccc} x_3 & x_4 & x_1 & x_2 & x_5 \end{array} \\ \begin{pmatrix} 0 & 4 & 2 & 0 & 0 \\ 8 & 0 & 0 & 4 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 10 \\ 6 & 6 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

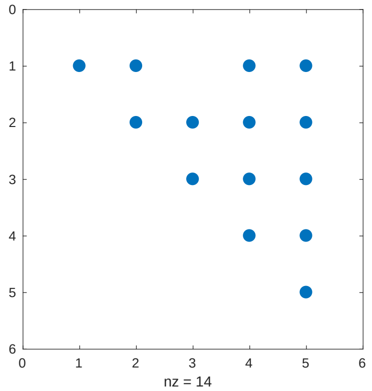


FIGURE 2.8: Left:Jacobian matrix with highest degree variable x_3 of degree $d(x_3) = 3$ in the first place. Right: The square-root factor with higher fill-in when compared the ordering $O = [x_2, x_3, x_4, x_1, x_5]$. Although the increase in non-zeros is only 4, it is very large given the original fill-in and the size of the matrix.

Applying the above ordering in the factor graph yields a Bayes tree with a large clique representing the fill-in as shown in Figure 2.9,

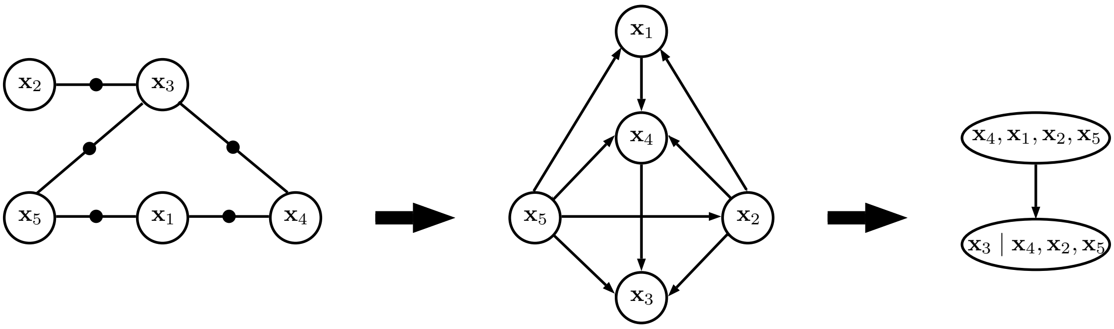


FIGURE 2.9: Eliminating a factor graph into a Bayes net and in turn into a Bayes tree. Eliminating nodes of higher degree at the beginning has resulted in larger cliques or high fill-in.

Although ordering the variables based on the minimum degree seems to work for small examples like the above, it quickly plummets in terms of time and storage efficiency as the degree of the remaining nodes get affected on eliminating a variable. This change in the degree has to be tracked separately and accounted in the algorithm. Several previous works have showed that keeping track of the changes to the degree of the node is the most expensive part of the algorithm [29–31]. So the above ordering is a greedy solution that has not taken into account the changes to the degree of the neighbouring nodes. But I hope that the magnitude of improvement achieved by such a tiny greedy alteration serves as a motivation for the need for fast and efficient variable ordering. In the next chapter, I will address the idea of efficiently reusing the parent ordering when fusing the graphs of multiple robots.

Chapter 3

Efficient Factor Graph Fusion

In the previous chapter, I described how the multi-robot smoothing and mapping is formulated as a least squares problem with a flavor that is specific to our implementation. Following that, I discussed the options to incrementally solve the least squares optimization and the critical requirement for fast, efficient and incremental variable reordering. A good ordering is necessary to reduce the amount of fill-in which refers additional non-zeros introduced during elimination. However, it has been shown that finding an optimal ordering for an arbitrary factor graph is NP-complete [26]. An optimal ordering called perfect elimination ordering [32] exists only if the factor graph is chordal. SLAM graphs are generally *not* chordal mainly due to revisiting the landmarks after a long time and loop closures that connect two far away nodes. Loops in the trajectory can result in a significant increase in computational complexity through a large increase of non-zero entries in the factor matrix. In addition to the typical loop closures in the SLAM problem, a multi-robot scenario could introduce several non-trivial loop closures. This is because for an indirect encounter between the robots, a big chunk of graph with several measurements from one robot has to combined with a far away node of another robot. Such an update is equivalent to loop closure in terms of computation and storage. Despite that, it is important in a multi-robot scenario to fuse the factor graphs of individual robots to improve the overall estimate. This requires finding the variable ordering of the fused factor graph.

To combat this scenario, this chapter comes up with a methodology to quickly find the variable ordering of the combined graph using the ordering of the participating factor graphs. The worst case here would be to do a complete reordering of the fused graph. We provide a complexity analysis to compare the time performance of both the options. The outline of this chapter is as follows: In the next section, I will provide a relevant literature survey from linear algebra and graph theory community. Following that, I will dive deep in to Bayes tree data structure that was introduced in the previous chapter to establish incremental variable ordering. Then I will formally verify that the proposed ordering does not violate any of the standard rules to be obeyed. Finally, I will explain the proposed approach and display results on a standard dataset from sparse linear algebra community.

3.1 Related Work

Solving the least-squares and linear programming problem is central to several scientific and engineering applications. Our focus in on sparse least-squares optimization with primary applications to SLAM. Smoothing formulation of SLAM as a least-squares with sparse graphs was first done by Lu and Milios [33]. Their method provides both batch and sequential procedure but performs the expensive inversion of the information matrix for updates. Although several smoothing based SLAM solutions have been developed based on conjugate gradient [34], gradient descent [35], relaxation [11] and multi-level relaxation [36], we only deal with the ones that derive performance improvements from information matrix decomposition. $\sqrt{SAM}$ by Dellaert [37] was the first work to replace expensive matrix inversion with sparse matrix factorization for the SLAM problem. It mentioned the dramatic performance improvements that could be derived from good variable ordering but is done only on a batch setting. The two key algorithms that improved on the least-squares formulation of SLAM using the premise set by [37] include ISAM [8] and ISAM2 [9]. Both of these works combine the formulation in [33] with the interchangeable linear algebra and graph theory flavor introduced in [37] and provide some additional improvements in terms of incremental optimization. Recently, Agarwal and Olson [38] provided different variable reordering strategies to SLAM and explained the critical role played by variable ordering in different solutions to SLAM.

In our work, we use ISAM2 using the Bayes tree [18] [9] as the underlying state estimation engine because it is exact, incremental and solves the full non-linear problem. While tremendous amount of work is done to extend the general SLAM algorithms to multiple robots, smoothing and mapping for multi-robot SLAM is not explored much. The most relevant ones include [39] by Kim *et al.*, C-SAM [40] and Tectonic SAM [41]. Although these algorithms are based on Smoothing and Mapping, Tectonic SAM addresses only single robot, batch mapping and the other two algorithms only address the fundamental multi-robot mapping issues like map-aligning and relative pose initialization. Important graph based SLAM approaches that build independent sub-graphs and merge those graphs include [35], [42] and [43]. Folkesson's [35] reduces the graph complexity by collapsing parts of the robot trajectory into a cluster called star-nodes. Frese's [42] and [43] gives a hierarchical approach by exploiting the square root information and representing the Cholesky factors as a tree data structure. They provide a highly efficient algorithm but employ numerous approximations and none of these partition based algorithms are extended to multi-robots. Also, several recent developments in the algebraic graph theory community like hypergraph nested dissection ordering [44] and exact graph partitioning algorithm [45] have not been utilized in solutions for SLAM. Clearly from the above discussion, previous research has either been done in graph merging for single robot SLAM or, multi-robot SLAM that does not deal with combining the graphs.

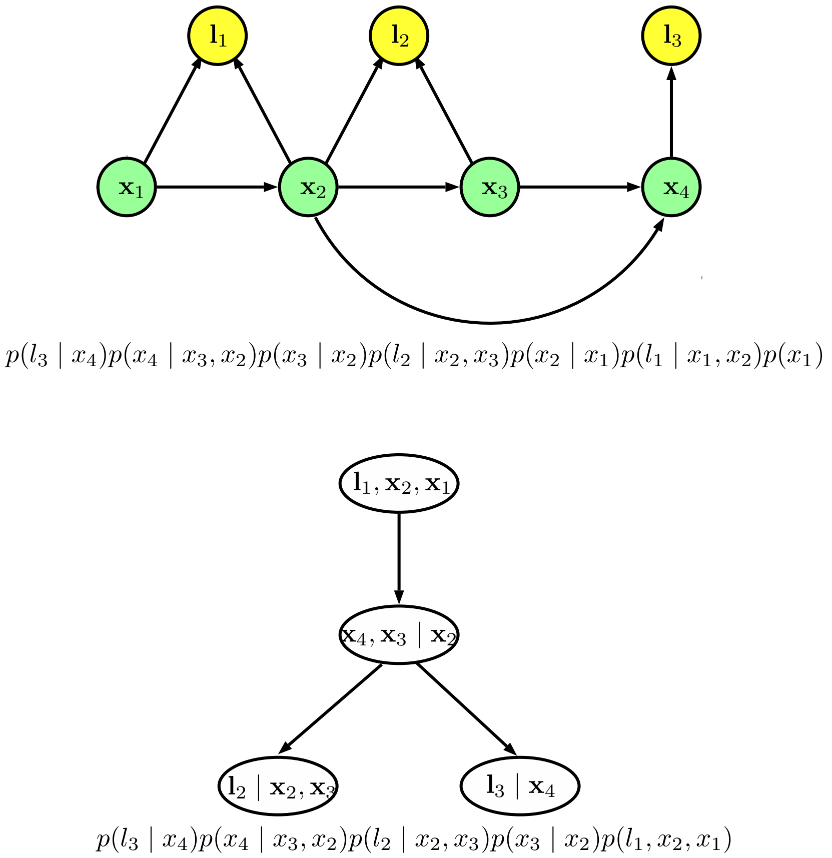
While solving large linear systems, it is a very common preprocessing step to order the columns of the matrix A to be factorized to keep the factorization as sparse as possible. Finding the variable ordering involves finding the permutation matrix P which is right-multiplied with A to obtain the ordered matrix AP. Cholesky or QR factorization of AP is more sparser and requires less storage than factorizing A. Ordering schemes have been proposed for different class of problems that include - 1) A being symmetric 2) A being unsymmetric. As finding the optimal ordering is NP-complete [26] various heuristics have been developed. The common aspect across all the approaches is to eliminate the nodes in the ascending order of their degrees. This is called as the minimum degree algorithm which is derived from a method first proposed by Markowitz in 1957 [46] for non-symmetric linear programming problems. The symmetric matrix version was formalized by Tinney *et al.* [47]. The ordering of the nodes directly on the graph data structure was derived by Rose [48]. Initial algorithms for symmetric matrices also include approximate minimum degree (AMD) [49] and nested dissection [50]. In case of the unsymmetric matrix such as Jacobian A, it is converted to symmetric information form $A^\top A$ to be used by these ordering schemes. State-of-the-art algorithms like column approximate minimum degree ordering (COLAMD) [27] works directly on the non-zero pattern of A without explicitly calculating $A^\top A$. A brief survey of the evolution of minimum degree ordering is consolidated by George [51]. Nested dissection is a divide and conquer heuristic based on graph partitioning that has the advantage of reordering the matrix into a form suitable for parallel execution. Very recently, nested dissection has been extended to unsymmetric hypergraphs in [44]. Although there is a significant amount of research in developing a near to optimal variable ordering tailored for various graphical models, to the best of our knowledge, there is no work in finding an ordering for the graph obtained by fusing ordered graphs.

3.2 Bayes Tree for Variable Ordering

Topologically, the Bayes net described in Section 2.3.2.2 is a chordal directed acyclic graph. By identifying cliques (groups of fully connected variables), the Bayes net may be rewritten as a Bayes tree. For full details about the clique-finding algorithm, see Kaess et al. [18]. Within the Bayes tree, each node represents the conditional density of the clique variables (also called as frontal variables), Θ_j, given all of its neighbors (also called as separators), $N(\Theta_j)$:

$$p(\Theta) = \Pi(\Theta \mid N(\Theta_j)) \tag{3.1}$$

During elimination of the factor graph (assuming the Bayes net is formed), the leaves of the tree are built first, and factors on the conditional variables are passed up the tree to



$p(l_3 \mid x_4)p(x_4 \mid x_3, x_2)p(x_3 \mid x_2)p(l_2 \mid x_2, x_3)p(x_2 \mid x_1)p(l_1 \mid x_1, x_2)p(x_1)$

$p(l_3 \mid x_4)p(x_4 \mid x_3, x_2)p(l_2 \mid x_2, x_3)p(x_3 \mid x_2)p(l_1, x_2, x_1)$

Figure 3.1: Bayes network and Bayes tree representations of the factor graph example used in last chapter with the elimination order $O = [l_3, l_2, l_1, x_4, x_3, x_2, x_1]$. The factorization of the factor graph joint probability density is mentioned for both the representations.

their parents. Back-substitution then proceeds top-down from the root clique, which is eliminated last, as it has no external dependencies. The solution of the frontal variables of the parent clique are passed down the tree to the children, which are guaranteed to depend only on the frontal variables of their ancestors.

Like the Bayes net, the structure of the Bayes tree is affected by the selected variable ordering. The Bayes net and Bayes tree representations are interchangeable. This is shown in the Figure 3.1 with the factor graph example used in Section 2.3.4. The terms $p(l_3 \mid x_4)$, $p(x_4 \mid x_3, x_2)$, $p(x_3 \mid x_2)$, $p(l_2 \mid x_2, x_3)$ are present in both the factorizations. It can be shown using the chain rule in Bayes theorem that $p(l_1, x_2, x_1) = p(l_1 \mid x_1, x_2)p(x_2 \mid x_1)p(x_1)$. Although both the representations are same given the variable ordering, modeling the inference using a tree structure is often more convenient and intuitive: elimination passes information up the tree, while back-substitution propagates information down the tree.

At every iteration, the update can be a new variable to be added to the graph or a new measurement factor connecting already existing variables. In both the cases, the square-root information of several variables other than the new variable in different cliques of the Bayes tree has to be updated. This could either be done by 1) Going back to the factor graph, adding a new measurement and/or a variable, reordering all the variables and completely factorizing the graph from scratch or 2) Partially recovering the portion of factor graph to which the new measurement and/or a variable should be added, reordering the partial graph and factorizing it. The second option is incremental, efficient and hence preferable, but would require the knowledge of subset of variables and factors that gets affected on touching a variable. This is exactly obtainable from the formulation of the Bayes tree.

A new variable is always accompanied by a measurement. When a new measurement is added, for example a factor $c'(x_j, x_{j'})$, only the paths between the cliques containing x_j and $x_{j'}$ (respectively) and the root are affected. The sub-trees below these cliques are unaffected, as are any other sub-trees not containing x_j or $x_{j'}$. The example in Figure 3.2 demonstrates the addition of a new pose node to the factor graph and Bayes tree example used in the previous chapter in Figure 2.5.

This in turn may move the estimates far off the current linearization point or add new non-zero components to the square-root factor. The former would require updating the linearization point and the latter requires reordering the variables. But as the set of variables that will be affected is known in advance it could be leveraged by changing the linearization point or reordering and factorizing only those variables. This relationship is understood by the construction of Bayes tree and was not obvious within the matrix framework [18].

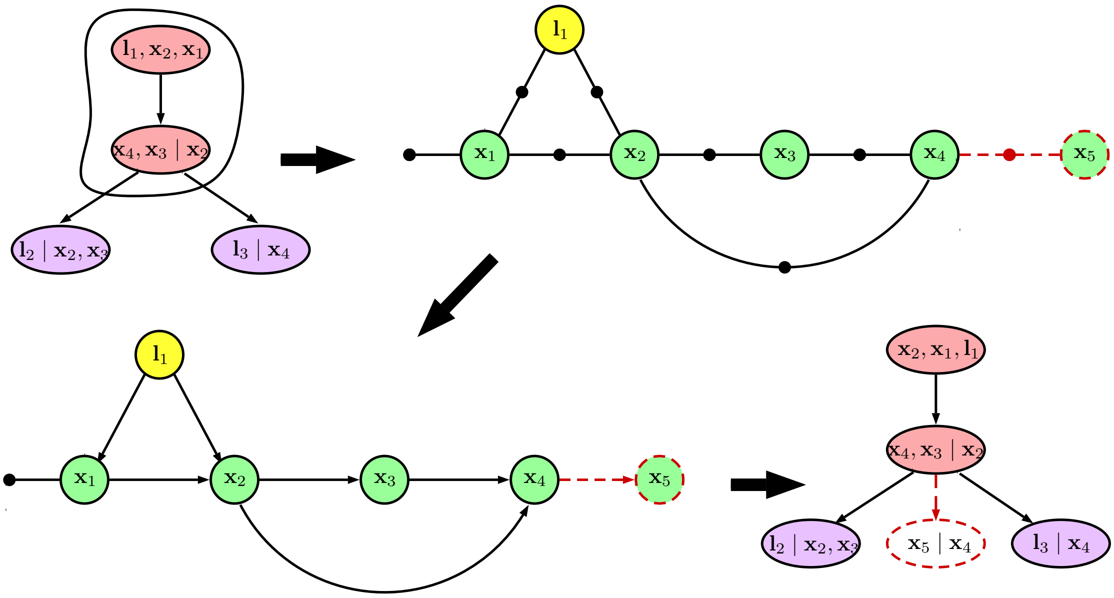


Figure 3.2: Adding a new variable with measurement to a factor graph example described in the previous chapter. It should be noted that only a part of the Bayes tree (red filled nodes) to which the new variable is added is recovered, reordered and factorized. The unaffected cliques (purple filled nodes) are attached back to the new tree. The new variable added is shown used a red broken circle.

3.2.1 Multi-robot Pose Graph Fusion

From the construction of the Bayes tree discussed so far and the example in Figure 3.2 it is clear that the Bayes tree is suitable for updating the square-root information incrementally. With ISAM2 [9] as the SLAM algorithm, a centralized cooperative mapping system will require fusing the maps from individual robots represented as the Bayes tree. In case of centralized mapping it could be assumed that every robot's factor graph is accessible always with no restrictions on bandwidth or that the robots share information only when they encounter. For our experiment, we assume that the robots share information only when they encounter. Also, our procedure does not require both the robots to identify each other when they encounter and work even if one robot identifies the other. Knowing the subset of variables using the Bayes tree which alone can be reordered could easily be extended to the situation of fusing multiple robots' pose graphs.

Direct encounters between robots introduce a constraint between the cliques of two different Bayes tree with pose nodes as the frontal variables. As a direct encounter relates the recent pose of the robots, the constraint joins the nodes near the root of the tree. The argument that the recent poses of the robot need not necessarily end up near the root of the tree after variable ordering will be addressed later. Since a connection is made between the nodes near the root of the tree the total number of affected variables is minimal. Whereas an indirect encounter occurs when two robots observe the same part of an environment (not necessarily at the same time), allowing a constraint of type robot-landmark-robot between the robot poses pivoted by the landmark pose, to be estimated. They could also be transformed into a constraint between the positions of two robots at the respective time steps. This type of constraint connects the recent pose of one of the robots from which the landmark is observed with a older pose of another robot from which the same landmark was observed. Figure 3.3 gives a pictorial step-by-step procedure to combine two Bayes trees.

In Figure 3.3, the factor graph of two robots exploring the environment is considered. Row 3 indicates the equivalent Bayes tree based on some variable ordering O_1 and O_2. It can be seen from the common variable among the factor graphs that both the robots have visited the same landmark l_1. To fuse those Bayes trees, only the path between the clique containing the landmark frontal variable and root clique is recovered as factor graph. The factor graph centred over the common variable is fused, reordered and eliminated back into a Bayes tree. The unaffected portions of the individual Bayes tree is merged back. These unaffected portions are shown as colored cliques in row 3 of Figure 3.3. While merging back the unaffected portions, the clique containing the earliest eliminated variable out of all the conditional variables of the unaffected clique is considered as the parent clique. For example, in Figure 3.3, it can be seen in the second Bayes tree that the unaffected green colored clique with frontal variable $\mathbf{x}_1^2$ has two conditional variables $\mathbf{x}_2^2$ and $\mathbf{x}_4^2$. It is attached to the clique containing the earliest eliminated variable, according to O_2, as the frontal variable, $\mathbf{x}_2^2$. However, in row 5 containing the fused Bayes tree, the same unaffected clique is attached to the clique with frontal variable $\mathbf{x}_4^2$ as it is eliminated before $\mathbf{x}_2^2$ according to O_{fused}. The idea here is that, during elimination information is propagated up the tree and every eliminated

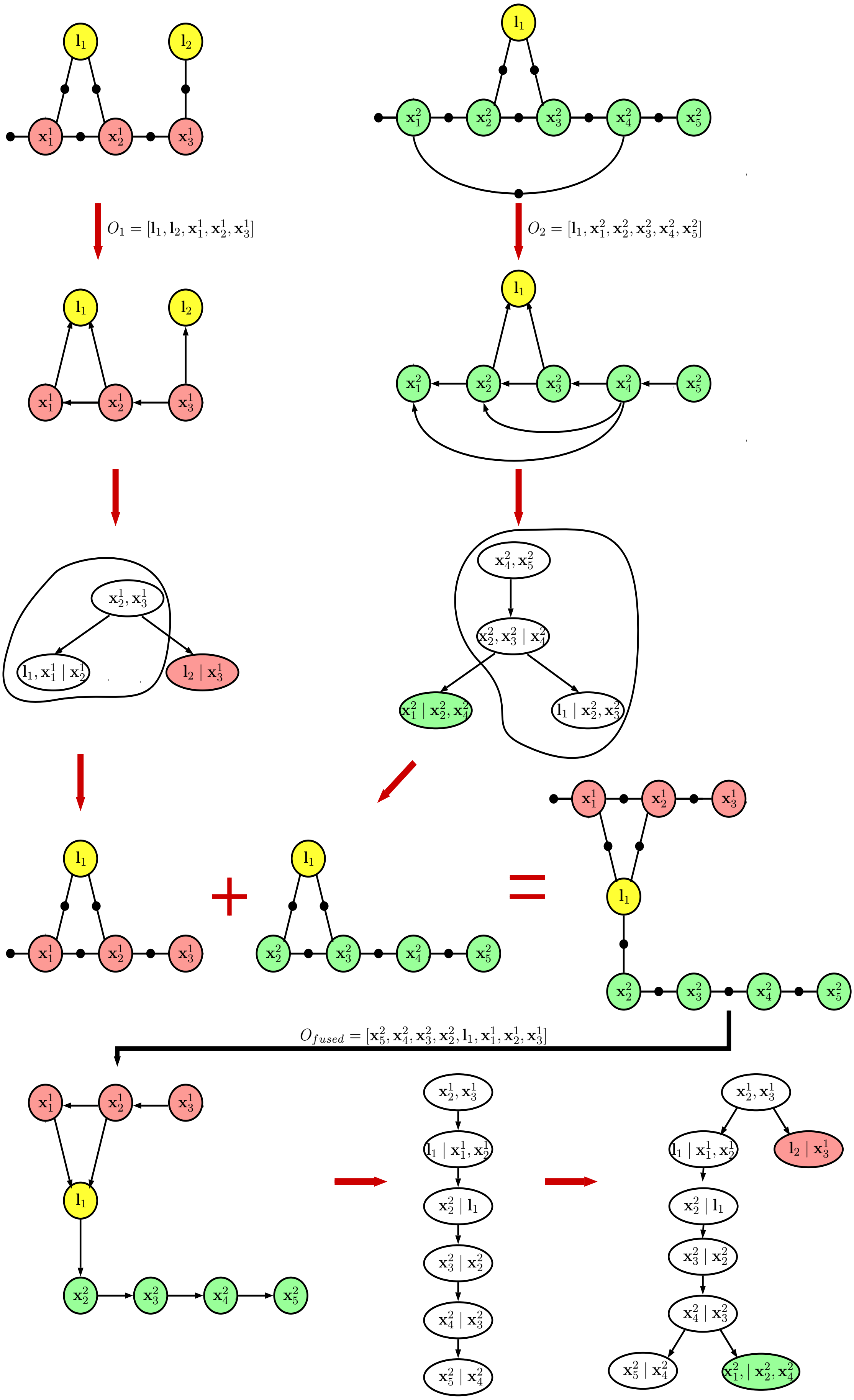

$O_1 = [l_1, l_2, x_1^1, x_2^1, x_3^1]$
$O_2 = [l_1, x_1^2, x_2^2, x_3^2, x_4^2, x_5^2]$
x_2^1, x_3^1
$l_1, x_1^1 \mid x_2^1$
$l_2 \mid x_3^1$
x_4^2, x_5^2
$x_2^2, x_3^2 \mid x_4^2$
$x_1^2 \mid x_2^2, x_4^2$
$l_1 \mid x_2^2, x_3^2$
$O_{fused} = [x_5^2, x_4^2, x_3^2, x_2^2, l_1, x_1^1, x_2^1, x_3^1]$
x_2^1, x_3^1
$l_1 \mid x_1^1, x_2^1$
$x_2^2 \mid l_1$
$x_3^2 \mid x_2^2$
$x_4^2 \mid x_3^2$
$x_5^2 \mid x_4^2$
$l_2 \mid x_3^1$
$x_1^2, \mid x_2^2, x_4^2$

FIGURE 3.3: Fusing multiple Bayes trees to calculate the best estimate. Only the affected part (inside blob in row 3) of both the Bayes trees are recovered and refactorized. O_1, O_2 and O_{fused} are the variable ordering of first, second and the fused factor graph.

variable will have the variables yet to be eliminated as its ancestor if there is a fill-in between them.

3.3 Formal Verification

Before proceeding to the proposed algorithm, it is important to formally validate that the operations carried out and the utilization of Bayes tree does not violate any of the standard assumptions, works for any general class of factor graph problems and is backward compatible. Firstly, when multiple robots visit several same regions of the environment the number of common landmarks between the graphs go up. The Bayes tree would be used to extract the affected subset of variables when combining the graphs with multiple common variables. It has to be verified that these affected subset of variables do not form disconnected components. Secondly, it has to be ensured that the root clique always contains more than one frontal variable. This is needed both in terms of the property of the Bayes tree and the requirements of the software optimizer used by us, [52]. To ensure these the following two proofs are given: In the following write-up a variable in/of the clique usually refers to the frontal variable of the clique. Let us consider the simple case of merging two bipartite graphs $\mathcal{G}_1 = (\mathcal{C}_1, \Theta_1, \mathcal{E}_1)$ and $\mathcal{G}_2 = (\mathcal{C}_2, \Theta_2, \mathcal{E}_2)$ with their Bayes tree function given as $\mathcal{B}_i(\mathcal{C}_i)$. The Bayes tree function returns the union of set of variables in the ancestor cliques of the clique containing each and every variable in $\mathcal{C}_i$, $i = 1, 2$ here. The graph is merged only when the set of common variables $\mathcal{C}_1 \cap \mathcal{C}_2 \neq \emptyset$. The set of all affected variables obtained from the Bayes tree is given as $\mathcal{C}^{fused} = \mathcal{B}_1(\mathcal{C}_1) \cup \mathcal{B}_2(\mathcal{C}_2) \cup (\mathcal{C}_1 \cap \mathcal{C}_2)$. The fused graph is then given as $\mathcal{G}^{fused} = (\mathcal{C}^{fused}, \Theta^{fused}, \mathcal{E}^{fused})$ where $\mathcal{E}^{fused} = \{(u, w) \mid [u, w \in \mathcal{C}_1 \cap (u, w) \in \mathcal{E}_1] \cup [u, w \in \mathcal{C}_2 \cap (u, w) \in \mathcal{E}_2]\}$ and $\Theta^{fused} = \{\theta(u, w) \mid \theta \in \Theta_1 \cup \Theta_2; u, w \in \mathcal{C}_1 \cup \mathcal{C}_2\}$.

Theorem 3.1. *The set of affected variables from the Bayes tree do not form a disconnected graph.*

Proof. The common variables forming the separator when combining the factor graphs can have multiple connected components. However, the subset of affected variables obtained by recursively traversing up the Bayes tree from every common variable forms a single connected component. It follows from the fact that, although the common variables might be present at different leaf cliques or non-leaf cliques, different branches or different depths, they all have a single root clique. Recovering all the common variables with all its ancestors will eventually be connected by the frontal variables of the root clique. It is mathematically verified by showing that the multiplicity of 0 as an eigenvalue of the Laplacian matrix of the fused graph is 1. The Laplacian matrix $\mathcal{L}_{N\times N}$ of a bipartite factor graph with N variable nodes is given as:

$$\mathcal{L}^{fused} = \mathcal{D}^{fused} - \mathcal{A}^{fused} \tag{3.2}$$

where $\mathcal{D}$ is the degree matrix and $\mathcal{A}$ is the adjacency matrix of the graph. Simplifying the above equation:

$$\mathcal{L}_{j,k} := \begin{cases} \deg(v_j) & \text{if } j = k \\ -1 & \text{if } j \neq k \text{ and } v_j \text{ is adjacent to } v_k \\ 0 & \text{otherwise} \end{cases} \tag{3.3}$$

So from the above general equation it can be seen that $\mathcal{L}^{fused}$ is a square matrix of size $\mid \mathcal{C}^{fused} \mid$ with the degree, $deg(\mathcal{C}_n)$; $n = 1 \ldots N$, along its diagonals. The degree of a variable node is equal to number of edges that are incident to it from other variable nodes. This is also equal to the number of negative ones in every column other than the diagonal element as the number of adjacent variable nodes is equal to degree of the node. Therefore a row transformation such as $R_1 = R_1 + R_2 + \ldots + R_n$ will result in all-zeros in row 1. Hence, the Laplacian matrix is rank deficient and the determinant is zero. It follows from the Invertible Matrix Theorem (IMT) [53] that a $\mathcal{L}_{N\times N}$ matrix is invertible if and only if 0 is not an eigenvalue of $\mathcal{L}_{N\times N}$. But as the rank goes down by 1 (not more than one row could be zeroed by this transformation), only the constant term vanishes from the characteristic polynomial while finding the eigenvalues. Hence the multiplicity of 0 in the eigenvalue multiset is 1 which equals the number of connected components. This proves that the set of affected variables forms a connected graph. In other words, the set of affected variables from the Bayes tree do not form a disconnected graph. □

Theorem 3.2. *On fusing the graphs, the root clique of the Bayes tree will have more than one frontal variable.*

Lemma: There is always a path between two nodes in the fused factor graph.

Proof. It naturally follows from the previous theorem that the fused factor graph forms a single connected component and the well known fact that any two nodes are joined by a path in an undirected graph. □

Lemma: The edge between the last and last but one node always fills-in, if all the nodes between these two nodes are eliminated before these two nodes.

Proof. The concern here is that variable ordering an arbitrarily mixed graph should not defy the standard norms of the root clique. Let the factor graph obtained by joining that of two robots be the one obtained from a single robot exploration task itself. Structurally, the fused factor graph is not any special when compared to the individual graphs. So any variable ordering is valid but may end up with a high fill-in. In the equivalent matrix representation, every column eliminated will leave changes with the rest of the columns yet to eliminated. By proceeding this, the last but one column in the elimination order will leave the information (make changes) with the last column to be eliminated. This information left by the last variable to the last but one variable is represented by an arrow pointing to from the last variable node to the last but one variable node. Therefore there is always a factor of the form $p(\textit{last but one variable} \mid \textit{last variable})$ at the second iteration of clique-finding algorithm which forms the root clique with these nodes as the frontal variables. □

3.4 Ordering the Fused Graph

Let $\mathcal{G}_1 = (\mathcal{C}_1, \Theta_1, \mathcal{E}_1)$ and $\mathcal{G}_2 = (\mathcal{C}_2, \Theta_2, \mathcal{E}_2)$ be two bipartite factor graphs that has to be fused. The Bayes tree function $\mathcal{B}_i(\mathcal{C}_i)$ is same as that defined in the previous section. Let O_1 and O_2 be the COLAMD [27] orderings of $\mathcal{G}_1$ and $\mathcal{G}_2$ respectively. The factor graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ are eliminated based on the ordering O_1 and O_2 as explained in Section 2.3.4

and stored as the Bayes tree [18]. Let $\mathcal{C}_{common} = \mathcal{C}_1 \cap \mathcal{C}_2$ represent the set of common variables among the graphs. We inherit the definition of $\mathcal{C}^{fused}$, Θ^{fused} and $\mathcal{E}^{fused}$ from the previous section that gives the fused graph $\mathcal{G}^{fused} = (\mathcal{C}^{fused}, \Theta^{fused}, \mathcal{E}^{fused})$. The set of variables that get impacted are obtained from the Bayes tree as $\mathcal{C}_1^{affected} = \mathcal{B}_1(\mathcal{C}_{common})$ and $\mathcal{C}_2^{affected} = \mathcal{B}_2(\mathcal{C}_{common})$. It should be noted that $\mathcal{C}_1^{affected} \cap \mathcal{C}_{common} = \emptyset$, $\mathcal{C}_2^{affected} \cap \mathcal{C}_{common} = \emptyset$ and $\mathcal{C}^{fused} = \mathcal{C}_1^{affected} \cup \mathcal{C}_2^{affected} \cup \mathcal{C}_{common}$.

3.4.1 Fusion Ordering

Ordering a fused graph is same as finding the permutation matrix P_1 that has to be post-multiplied with the factor graph matrix or the Jacobian matrix A. A permutation matrix is a square binary matrix that has exactly one entry of 1 in each row and each column and 0s elsewhere. The graphs of multiple robots are fused only when there are common variables among them, $\mathcal{C}_1 \cap \mathcal{C}_2 \neq \emptyset$. Let the ordering $O_1 = [o_1^{c_1^1}, o_1^{c_1^2}, \ldots, o_1^{c_1^{n_1}}]$ and $O_2 = [o_2^{c_2^1}, o_2^{c_2^2}, \ldots, o_2^{c_2^{n_2}}]$ where $n_1 =\mid \mathcal{C}_1 \mid$, $n_2 =\mid \mathcal{C}_2 \mid$, $\{c_1^1, c_1^2, \ldots, c_1^{n_1}\} \in \mathcal{C}_1$ and $\{c_2^1, c_2^2, \ldots, c_2^{n_2}\} \in \mathcal{C}_2$. Consider the functions $o_1(v)$ and $o_2(v)$ that gives the ordering value of the variable $v \in \mathcal{C}_1$ and $v \in \mathcal{C}_2$ respectively. For instance, $o_1(c_1^{n_1}) = o_1^{c_1^{n_1}}$. With these notations, the relative ordering of the fused graph, $\mathcal{G}_{fused}$ is derived below.

The relative ordering of the $\mathcal{C}_1^{affected}$ variables of the $\mathcal{G}^{fused}$ graph is given as:

$$O_1^{fused} = arg\ \text{sort}\left(\bigcup_{j \in \mathcal{C}_1} o_1(j)\right) \tag{3.4}$$

where *arg*sort gives the original position of each element in the sorted array. For example, the above operation on the array $[41, 23, 12, 8, 22]$ gives $[4, 3, 5, 2, 1]$. The relative ordering of the $\mathcal{C}_2^{affected}$ variables of the G^{fused} graph is given as:

$$O_2^{fused} =\mid O_1^{fused} \mid + arg\ \text{sort}\left(\bigcup_{j \in \mathcal{C}_2} o_2(j)\right) \tag{3.5}$$

The common variables are positioned towards the end of the ordering and are eliminated at last:

$$O^{fused}_{common} = \mid O^{fused}_{1} \mid + \mid O^{fused}_{2} \mid + arg \text{ sort}\left(\bigcup_{j \in \mathcal{C}_{common}} o_{common}(j) \right) \tag{3.6}$$

Thus, using the parent orderings of the graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ the relative ordering of G^{fused} is given by the concatenated array $O^{fused} = [O^{fused}_1, O^{fused}_2, O^{fused}_{common}]$. The permutation matrix $P^1 \in [0,1]^{|N| \times |N|}$ and $\mid O^{fused} \mid = N$ where $N = \mid O^{fused}_1 \mid + \mid O^{fused}_2 \mid +$ $\mid O^{fused}_{common} \mid$.

$$P^1_{j,k} := \begin{cases} 1 & \text{if } j = O^{fused}_k \\ 0 & \text{otherwise} \end{cases} \tag{3.7}$$

Therefore, in an event of fusing the graphs containing common set of nodes, the variables can be ordered by finding the permutation matrix and post-multiplying with the factor graph Jacobian matrix. The fusion ordered factor graph is then factorized using QR decomposition to obtain the estimate on combining the information from multiple robots. This fused graph is then converted to the Bayes tree using the proposed ordering and attached back to the unaffected portion of the Bayes tree. In this way, the incremental updates could be generated by reusing the parent ordering.

3.4.2 Relation with Nested Dissection

The idea of reusing the variable ordering by finding the relative ordering was inspired from the working principle of nested dissection [50]. Although the principle of nested dissection has not been employed for reusing the variable ordering, in the SLAM context, it has been used for recursively partitioning the graph into multi-level submaps in [54]. Nested dissection is a fill-reducing ordering method based on the divide-and-conquer principle. It is a recursive algorithm that finds a graph separator at every iteration, which on removal splits the graph into multiple connected components. This process continues until a point at which the size of the connected components are small and ordering them is trivial. These simpler graphs in the leaf nodes are ordered using some standard variable ordering technique like COLAMD [27]. The backtracking starts by ordering the leaf nodes containing these smaller graphs first and continues till it reaches root node. By this process the nodes in the first separator are placed towards the end of the variable ordering.

Reusing the variable ordering is similar to depth 1 nested dissection. Evidently, the nodes in $\mathcal{C}_{common}$ forms a separator that divides the graph $\mathcal{G}_{fused}$ into child graphs $\mathcal{G}_1^{affected}$ and $\mathcal{G}_2^{affected}$ where $\mathcal{G}_i^{affected} = (\mathcal{C}_i^{affected}, \Theta_i^{affected}, \mathcal{E}_i^{affected})$, $\Theta_i^{affected} \mid \{\theta(u,w) = \theta \in \Theta_i; u, w \in \mathcal{C}_i\}$, $\mathcal{E}_i^{affected} = \{(u,w) \mid u, w \in \mathcal{C}_i \cap (u,w) \in \mathcal{E}_i\}$. If the ordering of the graphs $\mathcal{G}_1^{affected}$ and $\mathcal{G}_2^{affected}$ could be reliably estimated using the parent ordering, the divide and conquer loop could be stopped and backtracked from depth 1 itself. This is exactly what is achieved by coming up with an ordering as per Equation 3.6 and post-multiplying the Jacobian with the permutation matrix P^1 as given in Equation 3.7.

3.4.3 Numerical Stabilization

Before explaining the approach that has been used to ensure numerical stability for matrix operations a few terminologies are introduced:

Pivoting is a common practise in matrix algorithms to add numerical stability to the final result. In the case of matrix algorithms, a pivot entry is usually required to be at least distinct from zero, and often distant from it; in this case finding this element is called *pivoting*. Pivoting may be followed by an interchange of rows or columns to bring the pivot to a fixed position and allow the algorithm to proceed successfully, and possibly to reduce round-off error. Usually the element which has the largest absolute value in the pivot row or column is chosen as the pivot element. This is because the percentage error that accrues on using a d-digit arithmetic precision is as much lower as the absolute value is higher. Thus pivoting on the largest element propagates the smallest round-off errors possible.

Given a factor graph $\mathcal{G} = (\mathcal{C}, \Theta, \mathcal{E})$, a subset $\mathcal{M} \subseteq \mathcal{E}$ is defined as matching or assignment if no two edges of $\mathcal{M}$ are incident to the same node. In other words, it is a set of pairwise non-adjacent edges, *i.e.* no two edges share a common vertex.

The proposed ordering described in the previous section only takes the structure into account, and not the numerical values. To stabilize the factorization and minimize pivoting, we wish to permute large entries to the diagonal. A standard approach is to model this as matching in the bipartite graph [55]. We use the matching permutation P^2 to permute the rows such that the large entries in every column resides on the diagonal. The remaining rows in the originally rectangular blocks have been pushed down. All the permutations applied on the matrix after this step should be symmetric. This permutation step for obtaining a strong diagonal is helpful for dynamic (partial) pivoting methods, since the number of row swaps is significantly reduced, thereby speeding up the factorization process [55]. It is essential for static pivoting methods, because it decreases the probability of encountering small pivots during the factorization.

In summary, the proposed fusion ordering and the numerical stabilization is applied on the fused graph or the fused Jacobian by post-multiplying and pre-multiplying the fusion ordering permutation matrix P_1 and the matching permutation matrix P_2.

3.5 Experimental Results

In this section, we present the experimental results of the proposed ordering algorithm when applied to the real world matrices. The algorithm was tested on a 2.3 GHz core i5 processor with 15.1 Gigabytes of memory. Figure 3.4 and 3.5 shows the non-zero structure of the square root factor for a bunch of real-world matrix datasets which are split and fused again using the standard COLAMD ordering [27], the relative fusion ordering (Section 3.4) and the default order in which nodes of the graph are retrieved from the memory respectively. It can be seen from the R matrix plot that the number of non-zeros are close to COLAMD ordering in the proposed ordering. At the same time, the gap in the number of non-zeros between the proposed ordering and the variable memory order is huge. It can be calculated that from these examples that, on an average the ratio of the difference between number of non-zeros in the proposed ordering and the variable memory order to the difference between the number of non-zeros in the proposed ordering and COLAMD ordering is 8.373. Also, it can be observed that just eliminating the variables in the order they are retrieved from the memory produces complete fill-in in some cases.

A comparison between the three types of orderings based on time taken to factorize the matrix after applying those ordering to the factor graph is performed in Figure 3.6. As mentioned earlier, the orthonormal Q matrix from QR decomposition is never explicitly formed in practice and hence the value of time is only that required to compute the square root factor R. Same experiment as before, comparing the number of non-zeros in the square root factor obtained from different ordering is also done on 120 real world matrices and plotted in Figure 3.7. It can be seen that the proposed fusion ordering takes less time for factorization in many cases. This might be because the proposed ordering is inspired from variable ordering by nested dissection and is compatible to parallel decomposition [2].

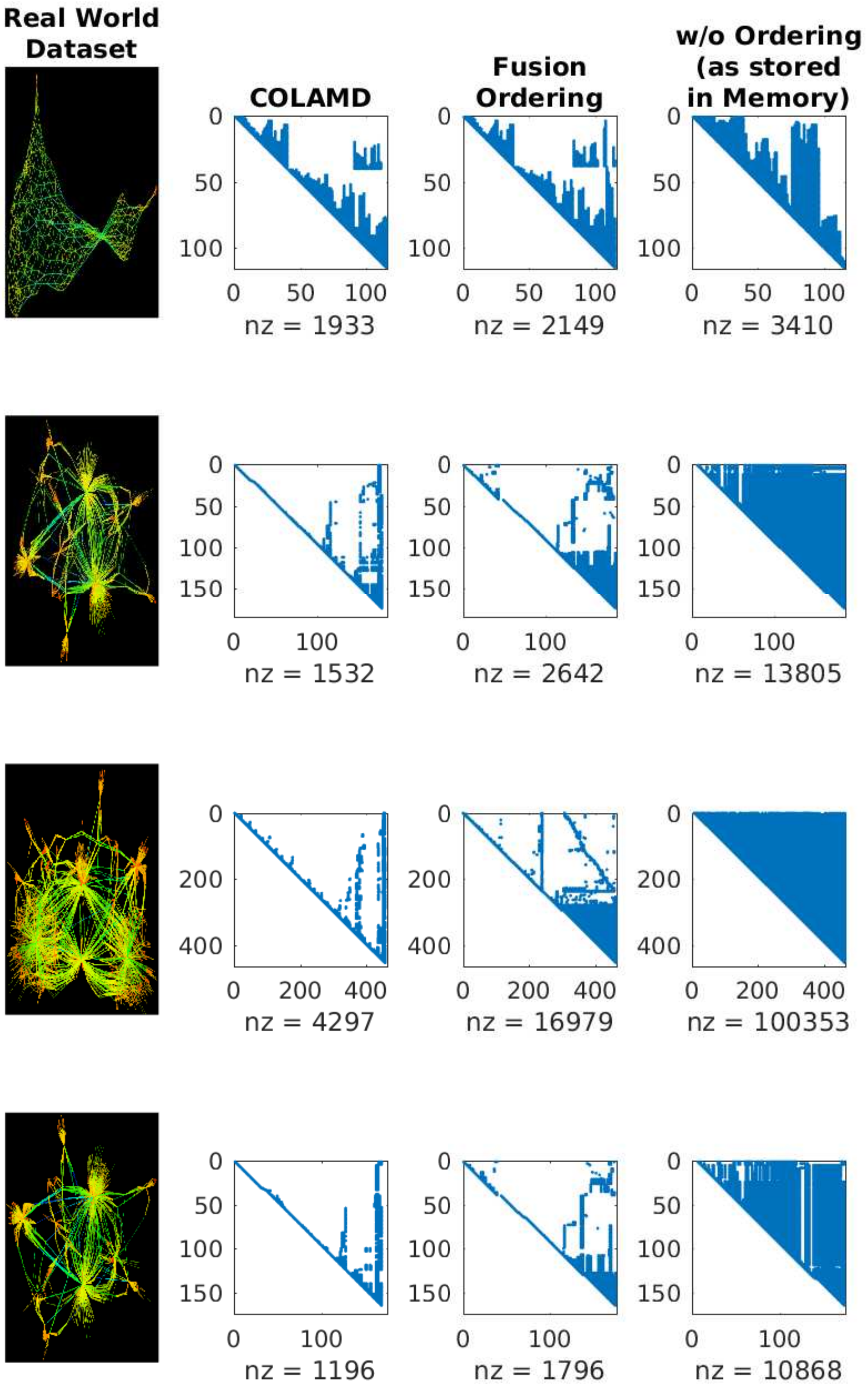


FIGURE 3.4: Comparison of number of non-zeros in the square root factor from QR factorization between the COLAMD ordering (second column), proposed ordering (third column) and the order in which the variables are retrieved (third column). The first column is the graph representation of the matrices in their lowest energy state [1].

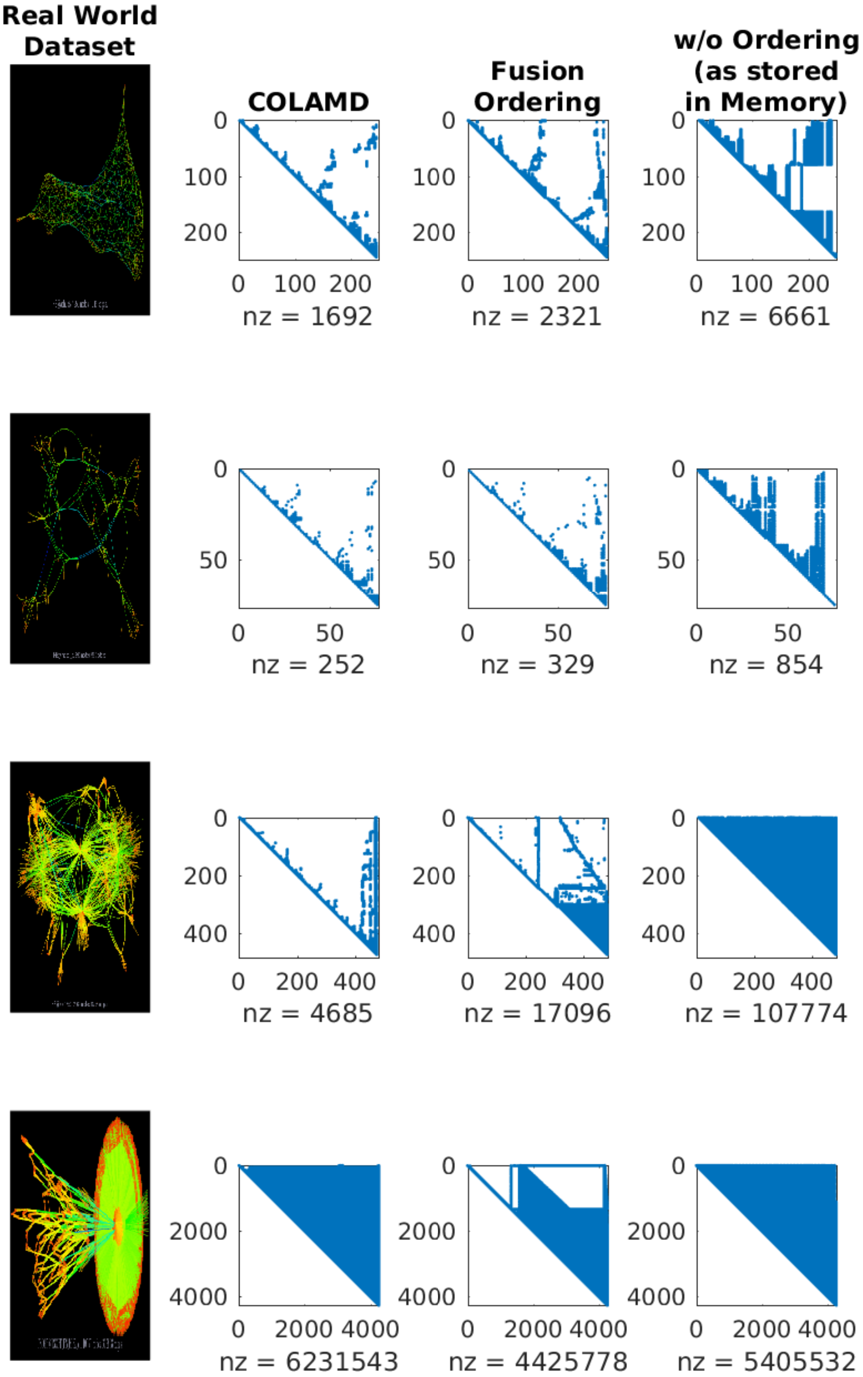


FIGURE 3.5: Comparison of number of non-zeros in the square root factor from QR factorization between the COLAMD ordering (second column), proposed ordering (third column) and the order in which the variables are retrieved (third column). The first column is the graph representation of the matrices in their lowest energy state [1].

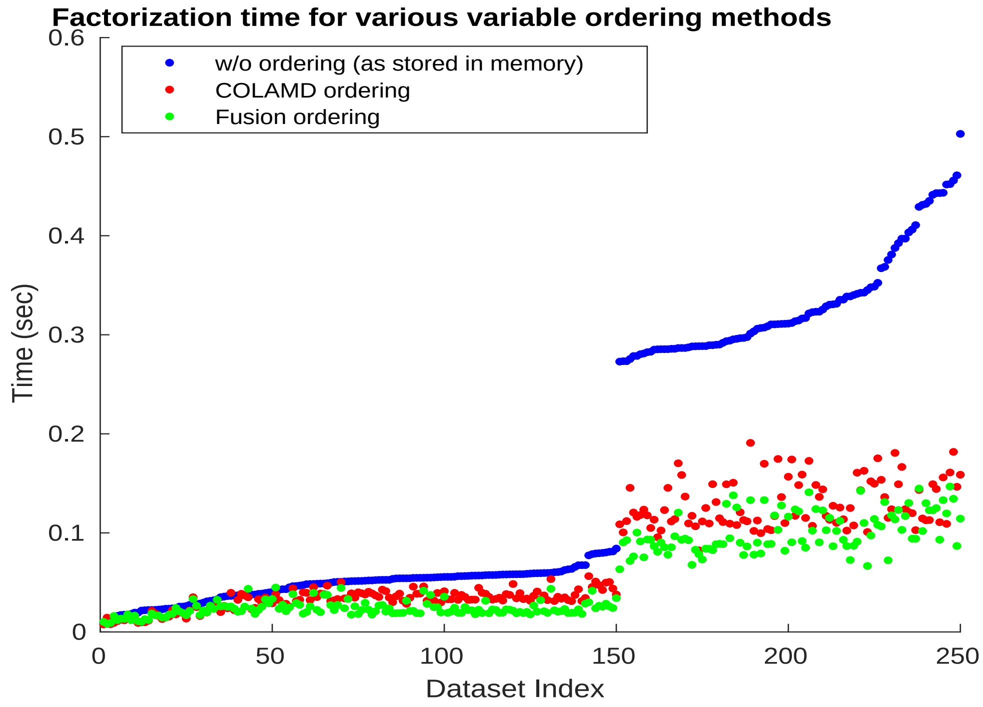


FIGURE 3.6: Comparison among different ordering schemes on the time taken to compute the square root factor R. It can be seen that the proposed fusion ordering takes less time than the COLAMD ordering as fusion ordering leaves the matrix in a state suitable for parallel QR decomposition [2].

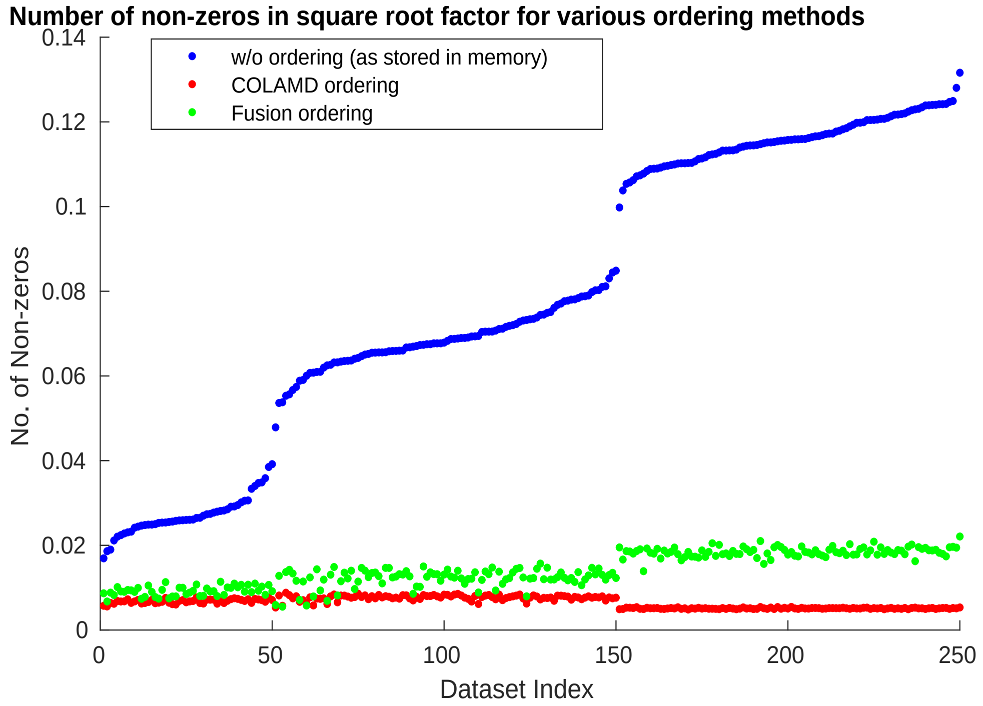


FIGURE 3.7: Comparison among different ordering schemes on the number of non-zero fill-in produced. It can be seen that the COLAMD produces the least and is closely followed by fusion ordering. On the other hand, the variable memory order produces huge fill-in.

Chapter 4

Multi-robot Relative Pose Initialization

Chapter 5

Improved Viola-Jones Object Detection for Landmark Extraction

Viola-Jones object detection algorithm [7] is a gold-standard in computer vision to detect trained objects at a very high speed using boosted classifiers. Because of their speed, they are used for object detection based landmark extraction in SLAM. In our multi-robot evaluation dataset, all the robots are fitted with a unique fiducial to represent their identity. Apart from this, the environment also contains fiducial markers that can be considered as landmarks in the map. In order to detect these with at a much lower false positive rate, at a lower resolution and eventually at a higher speed we improve the algorithm to utilize all the color channels in the input image. The original version works only on grayscale images. The key requirements of detection for our scenario include high detection percentage with lower false positive rate and more importantly, the detection of the fiducial markers at various scale. As the robots are navigating, the target fiducial may come near or move away resulting in change in the size of fiducial on the captured image over continuous frames. The Haar features in Viola-Jones algorithm are perfectly suitable for this as they are scale invariant. We retain this property of the algorithm and increase the detection percentage along with the detection speed by leveraging the additional information obtained by using all the channels of the image.

The improved version uses the same metric used in the original version to score a Haar feature but does it for all the channels of the input image. Thus the score is represented as a vector as opposed to a scalar value in the original version. These vectors are then ranked using linear discriminant analysis. We train a classifier to detect these fiducial markers using this improved version of Viola-Jones algorithm. The increase in the time complexity is matched by the ability to achieve better detection rates at lower resolution thereby producing a lower detection time overall.

The detection algorithm is very sensitive to various tunable parameters and hence it is necessary to understand how different parameters affect the detection time. This chapter focuses on the complexity analysis of the multi-scale and multi-channel decision tree based detector. The algorithm learns a decision tree during the training stage which is usually evaluated on the test image set using a sliding window approach for multiple sizes of the window (scale invariance). In the next section, I would shed some light on few relevant works that have extended the original work of Viola-Jones in different contexts such as improving false alarm rate, modifications in Haar features, weighting schemes in Adaboost etc. There is no particular work that specifically deals with a detailed theoretical complexity analysis of the final detector although there has been empirical analysis both in the original paper [7] and [56]. These theoretical results would be followed by some of the experiments and new directions that summarize the planned future work.

5.1 Related Work

In Viola-Jones object detection algorithm [7], the integral image for feature computation, Adaboost for feature selection and an attentional cascade for efficient computational resource allocation are the three key components behind achieving very high processing speed although the performance is same as the previous complex single stage classifiers. Following that, Lienhart *et. al,* [56] did an empirical analysis of the original work and also introduced 45^o rotated HAAR features and verified that the Gentle Adaboost performs better than Real and Discrete Adaboost in terms of detection accuracy. A complete algorithmic description that explains the implementation logic behind Open Computer Vision library [57] is provided by [58]. Almost all the previously existing analysis deal only with the complexity of training the Cascade Classifier and just mention that the detection rate is very fast. To the best of my knowledge, this is the first analysis of the computational complexity of the Cascade Classifier object detection as a function of various tunable parameters.

5.2 Theoretical Results

The trained classifier performs a sliding window search over the target image for a fixed window size during every iteration. The set of tunable parameters for the Cascade Classifier Detector are:

- The minimum window width and height of search, $W_{min} = \{w_{min}, h_{min}\}$
- The maximum window width and height of search, $W_{max} = \{w_{max}, h_{max}\}$
- The scale factor by which the window size grows from minimum to maximum, γ_0
- The minimum number of detections in the neighborhood to be selected as a true positive, β

The classifier is a cascaded set of decision trees and the number of such decision trees and the depth of each tree also influences the total time. However, since this value is constant for a given classifier it cannot be considered as a part of tunable parameters. Nevertheless, I have mentioned an upper bound on the number of evaluations of the entire decision tree over a given image.

REMARK: The classifier cannot detect target objects smaller than the trained window size.

5.2.1 Search window size and scale factor

The maximum and minimum window sizes and the scale factor are interrelated and hence considered together for analysis. The size of the window along both the width and height is increased from the minimum to maximum value at the scale factor ratio. This is done to stick to the constant aspect ratio at which the classifier is trained. Let w_t and h_t be the training width and height. The set of all scale factor values is given by

$$\Gamma = \{\gamma \mid w_t.\gamma \in [w_{min}, w_{max}] \cap h_t.\gamma \in [h_{min}, h_{max}], \gamma = \gamma_0, \gamma_0^2, ..., \gamma_0^n\} \ \forall \ n \in \mathbb{N} \tag{5.1}$$

where γ_0 is the initial value of scale factor which is also equal to the one input by the user. Generally, it is advised that γ_0 be in the positive neighborhood of 1, $\gamma_0 \rightarrow 1_+$ to get a finer search window size. The set of scale factors form a geometric progression with γ_0 being the common ratio between subsequent values. Now the set of image resolutions for which the integral image is to be calculated is given by

$$I = \{(x, y)|x = \frac{w_t}{r}, y = \frac{h_t}{r}\} \ \forall \ r \in \Gamma \tag{5.2}$$

The integral image is calculated for $\mid I \mid$ number of times. The number of elements in the set I is actually the number of terms in the G.P represented by Γ which can be calculated as follows.

Let i denote the index of the terms in a G.P formed by initial value 1 and common ratio γ_0. Then the least and greatest value of i that scales w_t to the range $[w_{min}, w_{max}]$ are

$$i_l^w = \left\lceil log_{\gamma_0}\left(\frac{w_{min}}{w_t}\right)\right\rceil \tag{5.3}$$

$$i_g^w = \left\lfloor log_{\gamma_0}\left(\frac{w_{max}}{w_t}\right)\right\rfloor \tag{5.4}$$

where the subscripts l and g represent the least and greatest index values respectively and $\lfloor . \rfloor$ and $\lceil . \rceil$ are the least and greatest integer functions.

Similarly, the least and greatest value of i that scales h_t to the range $[h_{min}, h_{max}]$ are

$$i_l^h = \left\lceil log_{\gamma_0}\left(\frac{h_{min}}{h_t}\right)\right\rceil \tag{5.5}$$

$$i_g^h = \left\lfloor log_{\gamma_0}\left(\frac{h_{max}}{h_t}\right)\right\rfloor \tag{5.6}$$

To stick with the aspect ratio, only the terms that are common between the set of width and set of height ratios are considered. The number of common scale factors K which is same as $\mid \Gamma \mid$ and $\mid I \mid$ is in turn given by

$$K =\mid \Gamma \mid=\mid I \mid= min\{i_g^w - i_l^w, i_g^h - i_l^h\} \tag{5.7}$$

$$K = min\left\{\left\lceil log_{\gamma_0}\left(\frac{w_{max}}{w_{min}}\right)\right\rceil, \left\lceil log_{\gamma_0}\left(\frac{h_{max}}{h_{min}}\right)\right\rceil\right\} \tag{5.8}$$

Using the change of base rule,

$$K = min\left\{\left\lceil \frac{ln\left(\frac{w_{max}}{w_{min}}\right)}{ln\ \gamma_0}\right\rceil, \left\lceil \frac{ln\left(\frac{h_{max}}{h_{min}}\right)}{ln\ \gamma_0}\right\rceil\right\} \tag{5.9}$$

All example sub-windows used for training were variance normalized to minimize the effect of different lighting conditions. Normalization is therefore necessary during detection as well. The variance of an image sub-window can be computed quickly using a pair of integral images. Recall that $\sigma^2 = m^2 - \frac{1}{N}\sum x^2$, where σ is the standard deviation, m is the mean, and x is the pixel value within the sub-window. The mean of a sub-window can be computed using the integral image. The sum of squared pixels is computed using an integral image of the image squared (i.e. two integral images are used in the scanning process). During detection the effect of image normalization can be achieved by post-multiplying the feature values rather than pre-multiplying the pixels. Thus the integral image is calculated $2K$ number of times. The integral image is a summed area table data structure which is calculated efficiently using a recursive algorithm. Though the computation of summed area table depends on the resolution of the image, the task of evaluating the intensities over any rectangular area requires only four array references. This allows for a constant calculation time that is independent of the size of the rectangular area. Also for multichannel images with M channels, the number of integral image computations is $2MK$ times. This factor M is the *only* significant difference between multichannel cascade classifier detector and grayscale detector.

5.2.2 Minimum number of neighbors

The element set containing multiple detections of differently sized objects in target images are split into equivalency classes (clustering). We use a first order logic with a predicate that relates the objects in the set based on the test of similarity of rectangles. The running time of the clustering algorithm is actually *independent* of the threshold β that corresponds to the minimum number of neighbors jointly satisfying the predicate. The logic implements an $\mathcal{O}(N^2)$ algorithm for clustering a set of N elements into one or more equivalency classes [59] as described using disjoint-set data structure representation [60]. For every element in the disjoint-set data structure of size N^2 the predicate is evaluated which returns either true or false.

The predicate evaluates similarity of rectangles by taking any two elements from the input set along with the internal parameter ϵ, $0 < \epsilon < 1$.

Two rectangles are similar if their sides are proportional.

This is checked by finding if all the sides of one rectangle is within $\Delta = \epsilon * (min(w_1, w_2), min(h_1, h_2))/2$ of the other rectangle.

Two rectangles which are within a threshold Δ on all the four sides, for a Δ formulated using minimum of width and height of the participating rectangles, are also within a threshold Δ for a Δ formulated using the maximum of their width and height.

Hence a Δ formed by the minimum of the their sides is a more restricted metric. The reason for formulating Δ is to account of uncertainty in similarity. The representative rectangle for all the equivalency class in the disjoint-set data structure is obtained by averaging all the constituent rectangles which is in turn returned as the detected rectangle.

5.2.3 Depth of the decision tree and target image resolution

Though it looks like the detector is evaluated for various window sizes from minimum to maximum size, algorithmically the target image is shrunk from its original resolution to various sizes depending upon scale factors such that the objects of interest of sizes between minimum and maximum window size falls inside the constant window size. In other words, we are decreasing the size of the target image itself rather than increasing the size of the search window and rescaling the features appropriately. We take this rather long route of rescaling the image because the detection could be done using the same window size by which the classifier is trained. Although the values of the feature learned at every level of the classifier is normalized by the area of the feature, no clear explanation about guarantees on generality of learned classifier value for arbitrary rescaling of the feature considering linear interpolation of image and truncation/round-off errors in feature rescaling is given in the original paper [7]. Obviously, by fractional rescaling the new correct positions become fractional. A plain vanilla solution is to round all relative look-up positions to the nearest integer position. However, performance may degrade significantly, since the ratio between the two areas of a feature may have changed significantly compared to the area ratio at training due to rounding. One solution is to correct the weights of the different rectangle sums so that the original area ratio between them for a given haar-like feature is the same as it was at the original size [56]. Nevertheless, by doing this reverse operation we end up linearly decimating the target image $\mid \Gamma \mid$ times and calculating integral image for each of them. From a memory perspective, the integral image for all the scales of the target image are computed and stored in a continuous serial buffer of size

$$M \times \sum_{\gamma \in \Gamma} \left(\frac{W}{\gamma} + 1\right)\left(\frac{H}{\gamma} + 1\right)$$

where $W \times H$ is the resolution, M is the number of channels of the target image and Γ is the set of scale factors.

For a decision tree of S stages and depth d_s per stage where $s \in S$, an upper bound on the number of evaluations of all the stages of the decision tree is given by

$$\leq \sum_{\gamma \in \Gamma} \left\{ \left(\frac{W}{\gamma} - w_t + 1\right)\left(\frac{H}{\gamma} - h_t + 1\right) \times \sum_{s \in S} d_s \right\} \tag{5.10}$$

where $W \times H$ is the resolution of the target image, $w_t \times h_t$ is the training resolution and Γ is the set of all scale factors.

Note: An implementation detail is that, for integral image resolutions whose corresponding scale factor is less than 2, the cascaded decision tree is only evaluated over every other pixel along width and height. Also, the entire continuous buffer of integral image across various resolutions is striped into memory chunks of size

$$\delta_i = 32 \times \frac{H}{\gamma_i W} \quad \forall \gamma_i \in \Gamma \tag{5.11}$$

where 32 is code specific. In total, there are $\sum_i \delta_i$ chunks which are processed in parallel using TBB multi-threading library.

5.3 Experiments run

The so far described Haar Cascade Classifier [?] is used to detect the product labels in an online fashion. The biggest advantage of the Cascade Classifiers are their rapid speed of detection despite very long training time. The complexity analysis summarised in this report gives idea on the sensitivity of each classifier parameter on the detection time. All the experiments are conducted using *Gaeta+*, a BossaNova Robotics proprietary robot with the images captured from the camera stack mounted on it. The camera stack contains 11 cameras arranged vertically to cover the entire span of a typically setup store aisle shelf as shown in Fig.[fig]. The camera stack is also supported by a vertical LED array to capture bright and sharp images at very high shutter speeds while the robot is moving. The images are projected on a plane fitted in the pointcloud (pointcloud data is collected by a vertically mounted LIDAR) constructed at the end of the aisle. This generates a panorama of the entire aisle. The product labels are detected on this panorama. It is imperative to detect the labels very quickly so as the solutions such as out-of-stock of store products could be reported readily. To give a context about the average size of the target object, the robot tracks the aisle sections from a fixed distance that falls in the depth of field of the camera. However, there are always cases in which the depth of various sections and product label holding pegs in the aisle are different and hence the target object size may vary significantly. A conservative search size may detect all object instances but it is time critical to have a tight estimate that is just enough to detect all such instances.

From the theoretical results, it is clear how different values of the parameters are mathematically related. For this experiment we trained a product label with a training window size 63×35 using Gentle Adaboost [61, 62]. In order to prove that the detection time is logarithmically related to the size of the search space defined by maximum and minimum window size, I ran the algorithm with multiple search search sizes decided by minimum and maximum window size. For the simplicity of understanding and denoting, the search size $\mathcal{S}$ can be represented as the length of line joining the right bottom corners of the minimum and maximum sized search rectangles.

$$\mathcal{S} = \sqrt{w_{max}^2 + h_{max}^2} - \sqrt{w_{min}^2 + h_{min}^2} \tag{5.12}$$

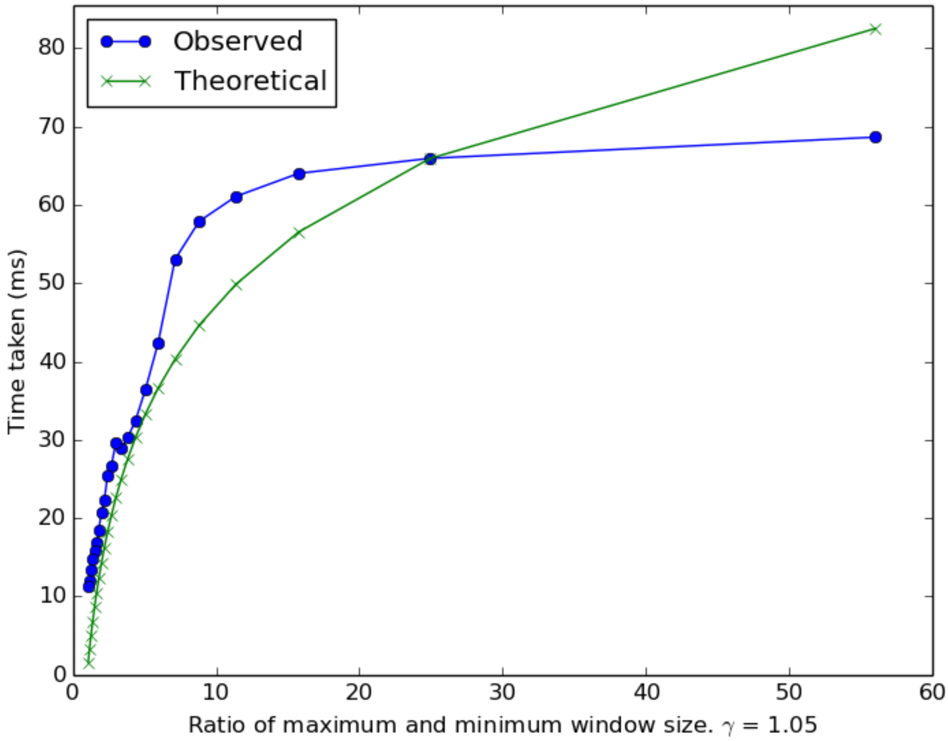


FIGURE 5.1: Comparison of ratio of maximum and minimum search window size with time taken taken for detection. It is clear from the shape of curve (blue) connecting the time taken for every search size that it varies logarithmically. The ideal curve (green) is just shown as a reference of a log curve and is not metrically accurate because we are trying to explain the theoretical complexity.

The resolution of each source image captured by the camera that form the panorama is 1920 × 2560. The average size of the target object to be detected when the tracking distance of the robot from the shelf is $75cm$ is 410 × 235. But as mentioned before, because of the various aisle section depths and camera perspective the search rectangle size has to vary across the average size. The search size characterized by the ratio of maximum and minimum window size is started from $W_{min} = (480, 274)$, $W_{max} = (518, 296)$ and broadened till $W_{min} = (63, 35)$, $W_{max} = (980, 560)$. In every iteration, the gap between the minimum and maximum window size is reduced by a step size

$s = 24$ pixels, *i.e* the minimum height is increased by 12 pixels and maximum height is decreased by 12 pixels with their corresponding widths being aspect ratio adjusted. The final time of detection for a given search gap is found by averaging the detection time over 291 source images having slightly varying target object size. Fig. 5.1 shows the various search size represented by the ratio of maximum and minimum window sizes plotted against the detection time(blue dots connected by a blue line). It can be seen from the shape of the curve that it approximates the logarithmic complexity. The green curve is the true log curve plotted with the ratio of maximum and minimum window size, *i.e.* $log_{\gamma}(\frac{W^i_{max}}{W^i_{min}})$. This experiment empirically verifies that the detection time scales logarithmically with the search size.

The second experiment is to empirically verify that the detection time varies *inverse logarithmically* with the increase in the scale factor, which is the step size from the minimum to maximum window size of detection. The experiment was conducted with the constant minimum window size $W_{min} = (63, 35)$ and maximum window size $W_{max} = (980, 560)$. The scale factor, γ is increased in steps of 0.02 starting from $\gamma = 1.01$. It can be seen from the plot in Fig. 5.2 that the detection time decreases inverse logarithmically with the increase in scale factor as explained theoretically by Eqn. 5.9. The green curve in Fig. 5.2 is the ideal curve that is plotted with the different scale factors and constant detection window size using the formula derived in Eqn. 5.9, *i.e.* $log_{\gamma_i}(\frac{W_{max}}{W_{min}})$.

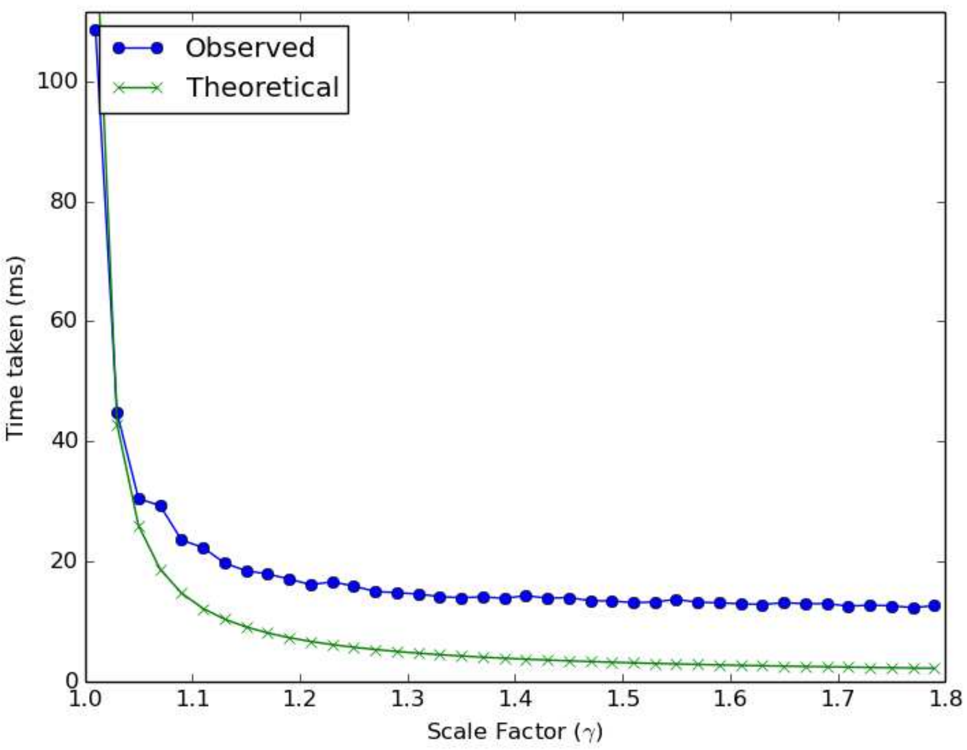


FIGURE 5.2: Comparison of scale factor with time taken taken for detection. It is clear from the shape of curve (blue) connecting the time taken for every scale factor that it varies *inverse* logarithmically. The ideal curve (green) is just shown as a reference of a inverse log curve and is not metrically accurate because we are trying to explain the theoretical complexity.

Chapter 6

Experiments and Conclusions

In this chapter, we illustrate the performance of our fusion ordering in a real-world multi-robot dataset [63] recorded at AP hill, Virginia. For our datasets, we use a canonical scan-matcher [12] that remembers a short history of scans corresponding to the key frames, fits contours to the data, and for a new scan, finds a globally optimal alignment within a given search region. The canonical scan-matcher performs another least-squares optimization that requires an initial guess about the relative transform between the scans. The relative odometry measurement coming from the poses corresponding to those scans is used as an initial guess. Larger key frame sizes are also problematic as the scan ranges vary significantly due to seeing an altogether different structure of the environment. This leads to a large uncertainty in the estimate of the relative transform. It is also difficult to provide a good initial guess for large key frame sizes as the corresponding odometry transform is likely to be drifted. In our results, we project the raw laser scans from the optimized trajectory of every robot instead of occupancy grid as the occupancy grid can sometime hide the inaccuracies in the map, such as duplication of a wall.

6.1 AP Hill Multi-robot Dataset

The publicly available AP hill multi-robot dataset [63] is recorded using 4 mobile robots with significant overlap in their trajectories. The robots travel different paths covering different blocks in the building. Each robot is equipped with an odometer, a laser range finder, an inertial measurement unit and a camera. Encounter information is also provided along with the dataset. According to the dataset, all the robots start at nearly the same position, and hence the number of encounters in the later stages are less in number. In general, we increased the number of encounters by performing multi-channel fiducial detection (see Chapter) using the data obtained from the camera mounted on the robots. Getting a good map of the environment is an engineering task that requires tuning of several optimization parameters and those related to different measurement models. In this section, we will describe the numerical values that are used for different sensor models and ISAM2 algorithm in GTSAM optimizer [52]. We will then present the timing analysis for using the fusion ordering for the combined graphs during every encounter. Finally the map alignment that is obtained by adding the relative transform between the trajectories as an estimation variable is displayed.

6.1.1 Tuning the Sensor Models and GTSAM Optimizer

Getting the optimizer up and running to provide consistent and repeatable mapping results is generally preceded by developing accurate sensor models. The AP hill multi-robot dataset contains raw wheel encoder data, dead reckoning odometry data, laser range data, IMU data and the camera feed. The raw wheel encoder data is processed using a velocity motion model given in Appendix A. The laser range data is used by the scan-matching module to convert it as pose constraint. The IMU data is also treated in a similar manner to make it as a pose factor. Apart from the velocity model factors the dead reckoning odometry data is also used. Thus, all the pose variables are constrained by a minimum of four types of factors. Apart from the fiducials on top of every robot the environment also contains fiducials that serve as the landmarks. These fiducial and encounter information in the dataset are converted into a constraint between the poses and the landmarks/other robots. We also detect the same on the camera feed to further constrain the variables and over-determine the system.

Table 6.1: My caption

Sensor Model	robot 1			robot 2			robot 3		
	x (m)	y (m)	θ (rad)	x (m)	y (m)	θ (rad)	x (m)	y (m)	θ (rad)
Velocity motion model	0.05	0.02	0.2	0.055	0.02	0.2	0.05	0.02	0.2
Odometry model	0.04	0.02	0.5	0.04	0.03	0.5	0.05	0.022	0.1
Landmark pose	0.02	0.01	0.01	0.02	0.01	0.01	0.02	0.02	0.001
IMU model	0.02	0.01	0.05	0.02	0.01	0.05	0.01	0.01	0.001
Loop closure factor	0.02	0.02	0.1	0.01	0.01	0.1	0.01	0.01	0.001
Robot-robot encounter	0.01	0.01	0.2	0.01	0.01	0.02	0.01	0.01	0.2

All these sensor models are accurate to a level that depends on the amount of uncertainty it introduces in the system. This level of uncertainty is captured by the covariance matrix that measures the joint variability of different random variables measured by the model. In other words, the sensor that is less accurate has a larger covariance. In many cases, they are provided by the sensor manufacturer. They can be used as a starting point and can then be tuned based on performance. For encoders like odometry, the error accrued is larger for larger sampling times. For such sensors, the covariance is tuned for unit time and scaled accordingly based on the time between different samples. In case of fiducial markers, the center of the detected fiducial is taken as the landmark position and therefore covariance is also defined in terms of x, y and θ.

During a direct encounter, the global time of the robot which found the other robot is used to extract the pose of the other robot. The nearest matching pose based on the time is used. Write about the possible anomalies here.

Appendix A

Sensor Models

A.1 Velocity Motion Model

The following basic physics based velocity model assumes that the robot can be controlled through two velocities, rotational and translational. The model takes these velocities as the input and gives the current pose obtained by commanding the robot with those velocities from the previous pose. The translational velocity at time t is given by v_t, and the rotational velocity by ω_t. We arbitrarily postulate that positive rotational velocities ω_t induce a counter-clockwise rotation (left turns). Positive translational velocities v_t correspond to forward motion. In reality the robot motion is subjected to noise and hence the actual velocity differ from the commanded one. Thus the actual velocity can be obtained by adding a noise term:

$$\begin{bmatrix} \hat{v} \\ \hat{w} \end{bmatrix} = \begin{bmatrix} v \\ w \end{bmatrix} + \begin{bmatrix} \epsilon_v \\ \epsilon_w \end{bmatrix} \tag{A.1}$$

where ϵ_v and ϵ_w are zero mean random variable with finite variance. At every iteration the raw wheel encoder data in terms of translational and rotational velocity is used to find the relative transform from the previous pose $(x_{t-1}, y_{t-1}, \theta_{t-1})$ to the next pose

(x_t, y_t, θ_t).

$$\begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix} = \begin{bmatrix} x_{t-1} \\ y_{t-1} \\ \theta_{t-1} \end{bmatrix} + \begin{bmatrix} -\frac{\hat{v}_{t-1}}{\hat{\omega}_{t-1}} sin\theta_{t-1} + \frac{\hat{v}_{t-1}}{\hat{\omega}_{t-1}} sin(\theta_{t-1} + \hat{\omega}_{t-1}\Delta t) \\ \frac{\hat{v}_{t-1}}{\hat{\omega}_{t-1}} cos\theta_{t-1} - \frac{\hat{v}_{t-1}}{\hat{\omega}_{t-1}} sin(\theta_{t-1} + \hat{\omega}_{t-1}\Delta t) \\ \hat{\omega}_{t-1}\Delta t + \hat{\gamma}\Delta t \end{bmatrix} \tag{A.2}$$

The above model describes the exact final location of a robot moving in a circular trajectory of radius $r = \frac{\hat{v}}{\hat{\omega}}$. However, in reality the motion is affected by control noise and the commanded circular trajectory is actually not circular. The circle constrains the final orientation which is fixed by adding an additional robot specific noise term $\hat{\gamma}$ on the final orientation in the above equation.

A.2 Iterative Closest Point based Scan Matching

Let X and P represent two laser scan measurements measured closely in time. Let k be the number of scan indices in the laser scan measurement:

$$X = \{x_1, x_2, \ldots, x_k\} \tag{A.3}$$

$$P = \{p_1, p_2, \ldots, p_k\} \tag{A.4}$$

The model takes two laser scan data as input and gives the relative transform between the poses from which those laser scans were measured as output. The problem is formulated as a least-squares optimization that finds the translation t and rotation R which minimizes the sum of squared error:

$$E(R, t) = \frac{1}{K_p} \sum_{i=1}^{K_p} \|x_i - Rp_i - t\|^2 \tag{A.5}$$

where x_i and p_i are the corresponding points between two laser scans. Calculating the corresponding points is the most expensive stage of the ICP algorithm. If there is no uncertainty in calculating the corresponding points or if the corresponding points are given beforehand then the relative translation and rotation can be calculated in a closed form.

In our case, it is *approximately* calculated by projecting the points according to the view point [64]. As a start, the initial guess provided using the odometry model is used as the view point and then shifted till convergence. In projection based matching there are slightly bad alignments in each iteration but it is one to two orders of magnitude faster than the closest point. It also requires the point-to-line error metric. Using the point-to-line error metric the above vanilla iterative closest point formulation becomes:

$$\min_{q_{k+1}} \sum_i (n_i^T [x_i \bigoplus q_{k+1} - \Pi\{\mathcal{S}^{ref}, x_i \bigoplus q_k\}]) \tag{A.6}$$

where q_i is the robot's pose in the world frame, x_i are the points in the first scan, $\Pi\{\mathcal{S}^{ref}, .\}$ is the projection on the reference surface, n_i is the normal to the surface and $\bigoplus$ is the notation introduced by Lu and Milios for pose composition [33].

Bibliography

[1] Timothy A Davis and Yifan Hu. The university of florida sparse matrix collection. *ACM Transactions on Mathematical Software (TOMS)*, 38(1):1, 2011.

[2] F Rotella and I Zambettakis. Block householder transformation for parallel qr factorization. *Applied mathematics letters*, 12(4):29–34, 1999.

[3] Andrew Howard. Multi-robot simultaneous localization and mapping using particle filters. *The International Journal of Robotics Research*, 25(12):1243–1256, 2006.

[4] Sebastian Thrun and Yufeng Liu. Multi-robot slam with sparse extended information filers. *Robotics Research*, pages 254–266, 2005.

[5] Xun S Zhou and Stergios I Roumeliotis. Multi-robot slam with unknown initial correspondence: The robot rendezvous case. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pages 1785–1792. IEEE, 2006.

[6] Frank R Kschischang, Brendan J Frey, and H-A Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on information theory*, 47(2):498–519, 2001.

[7] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–I. IEEE, 2001.

[8] Michael Kaess, Ananth Ranganathan, and Frank Dellaert. isam: Incremental smoothing and mapping. *IEEE Transactions on Robotics*, 24(6):1365–1378, 2008.

[9] Michael Kaess, Hordur Johannsson, Richard Roberts, Viorela Ila, John J Leonard, and Frank Dellaert. isam2: Incremental smoothing and mapping using the bayes tree. *The International Journal of Robotics Research*, 31(2):216–235, 2012.

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

[11] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. 2005.

[12] Andrea Censi. An ICP variant using a point-to-line metric. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Pasadena, CA, May 2008. doi: 10.1109/ROBOT.2008.4543181. URL http://purl.org/censi/2007/plicp.

[13] Guy Barrett Coleman and Harry C Andrews. Image segmentation by clustering. *Proceedings of the IEEE*, 67(5):773–785, 1979.

[14] John E Dennis Jr and Robert B Schnabel. *Numerical methods for unconstrained optimization and nonlinear equations*. SIAM, 1996.

[15] Randall Smith, Matthew Self, and Peter Cheeseman. Estimating uncertain spatial relationships in robotics. *arXiv preprint arXiv:1304.3111*, 2013.

[16] Gene H Golub and Charles F Van Loan. *Matrix computations*, volume 3. JHU Press, 2012.

[17] Alan George, John R Gilbert, and Joseph WH Liu. *Graph theory and sparse matrix computation*, volume 56. Springer Science & Business Media, 2012.

[18] Michael Kaess, Viorela Ila, Richard Roberts, and Frank Dellaert. The bayes tree: An algorithmic foundation for probabilistic robot mapping. In *Algorithmic Foundations of Robotics IX*, pages 157–173. Springer, 2010.

[19] K. Shen, J.N. Crossley, A.W.C. Lun, and H. Liu. *[The nine chapters on the mathematical art: companion and commentary]*. Oxford University Press, 200 BCE. ISBN 9780198539360 English translation available at. URL https://books.google.com/books?id=0D1GAQAAIAAJ.

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

[21] Pinar Heggernes and Pontus Matstoms. *Finding good column orderings for sparse QR factorization.* University of Linkping, Department of Mathematics, 1996.

[22] Robert E Tarjan and Mihalis Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM Journal on computing*, 13(3):566–579, 1984.

[23] Michael Kaess and Frank Dellaert. Covariance recovery from a square root information matrix for data association. *Robotics and autonomous systems*, 57(12): 1198–1210, 2009.

[24] Jean RS Blair and Barry Peyton. An introduction to chordal graphs and clique trees. In *Graph theory and sparse matrix computation*, pages 1–29. Springer, 1993.

[25] Robert G Cowell, Philip Dawid, Steffen L Lauritzen, and David J Spiegelhalter. *Probabilistic networks and expert systems: Exact computational methods for Bayesian networks.* Springer Science & Business Media, 2006.

[26] Stefan Arnborg, Derek G Corneil, and Andrzej Proskurowski. Complexity of finding embeddings in ak-tree. *SIAM Journal on Algebraic Discrete Methods*, 8(2):277–284, 1987.

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

[28] Tim Davis and Ken Stanley. Sparse lu factorization of circuit simulation matrices. *Go online to http://www. cise. ufl. edu/ davis/techreports/KLU/pp04. pdf*, 2004.

[29] Iain S Duff. *Direct methods for sparse matrices.* Oxford: Clarendon press, 1986.

[30] Alan George and DAvID R McIntyRE. *On the application of the minimum degree algorithm to finite element systems.* Springer, 1977.

[31] Stanley C Eisenstat, MC Gursky, Martin H Schultz, and Andrew H Sherman. Yale sparse matrix package i: The symmetric codes. *International Journal for Numerical Methods in Engineering*, 18(8):1145–1151, 1982.

[32] Delbert Fulkerson and Oliver Gross. Incidence matrices and interval graphs. *Pacific journal of mathematics*, 15(3):835–855, 1965.

[33] Feng Lu and Evangelos Milios. Globally consistent range scan alignment for environment mapping. *Autonomous robots*, 4(4):333–349, 1997.

[34] Kurt Konolige. Large-scale map-making. In *Proc. 21th AAAI National Conference on AI*, San Jose, CA, 2004.

[35] John Folkesson and Henrik Christensen. Graphical slam-a self-correcting map. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 1, pages 383–390. IEEE.

[36] Udo Frese, Per Larsson, and Tom Duckett. A multilevel relaxation algorithm for simultaneous localization and mapping. *IEEE Transactions on Robotics*, 21(2): 196–207, 2005.

[37] Frank Dellaert. Square root SAM. In *Proceedings of Robotics: Science and Systems*, Cambridge, USA, June 2005. doi: 10.15607/RSS.2005.I.024.

[38] Pratik Agarwal and Edwin Olson. Variable reordering strategies for slam. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 3844–3850. IEEE, 2012.

[39] Been Kim, Michael Kaess, Luke Fletcher, John Leonard, Abraham Bachrach, Nicholas Roy, and Seth Teller. Multiple relative pose graphs for robust cooperative mapping. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 3185–3192. IEEE, 2010.

[40] Lars AA Andersson and Jonas Nygards. C-sam: Multi-robot slam using square root information smoothing. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 2798–2805. IEEE, 2008.

[41] Kai Ni, Drew Steedly, and Frank Dellaert. Tectonic sam: Exact, out-of-core, submap-based slam. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 1678–1685. IEEE, 2007.

[42] Udo Frese. Treemap: An o (log n) algorithm for indoor simultaneous localization and mapping. *Autonomous Robots*, 21(2):103–122, 2006.

[43] Udo Frese and Lutz Schroder. Closing a million-landmarks loop. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pages 5032–5039. IEEE, 2006.

[44] Laura Grigori, Erik G Boman, Simplice Donfack, and Timothy A Davis. Hypergraph-based unsymmetric nested dissection ordering for sparse lu factorization. *SIAM Journal on Scientific Computing*, 32(6):3426–3446, 2010.

[45] William W Hager, Dzung T Phan, and Hongchao Zhang. An exact algorithm for graph partitioning. *Mathematical Programming*, pages 1–26.

[46] Harry M Markowitz. The elimination form of the inverse and its application to linear programming. *Management Science*, 3(3):255–269, 1957.

[47] William F Tinney and John W Walker. Direct solutions of sparse network equations by optimally ordered triangular factorization. *Proceedings of the IEEE*, 55(11): 1801–1809, 1967.

[48] Donald J Rose. A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations. *Graph theory and computing*, 183:217, 1972.

[49] Patrick R Amestoy, Timothy A Davis, and Iain S Duff. Algorithm 837: Amd, an approximate minimum degree ordering algorithm. *ACM Transactions on Mathematical Software (TOMS)*, 30(3):381–388, 2004.

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

[51] Alan George and Joseph WH Liu. The evolution of the minimum degree ordering algorithm. *Siam review*, 31(1):1–19, 1989.

[52] Frank Dellaert. Factor graphs and gtsam: A hands-on introduction. Technical report, Georgia Institute of Technology, 2012.

[53] Christopher Stover. Invertible matrix theorem. Technical report, MathWorld–A Wolfram Web Resource.

[54] Kai Ni and Frank Dellaert. Multi-level submap based slam using nested dissection. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 2558–2565. IEEE, 2010.

[55] Iain S Du and Jacko Koster. On algorithms for permuting large entries to the diagonal of a sparse matrix. 1999.

[56] Rainer Lienhart, Alexander Kuranov, and Vadim Pisarevsky. Empirical analysis of detection cascades of boosted classifiers for rapid object detection. In *Joint Pattern Recognition Symposium*, pages 297–304. Springer, 2003.

[57] G. Bradski. Opencv. *Dr. Dobb's Journal of Software Tools*, 2000.

[58] Yi-Qing Wang. An analysis of the viola-jones face detection algorithm. *Image Processing On Line*, 4:128–148, 2014.

[59] David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035. Society for Industrial and Applied Mathematics, 2007.

[60] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction To Algorithms*. MIT Press, 2001. ISBN 9780262032933. URL https://books.google.com/books?id=NLngYyWFl_YC.

[61] Robert E Schapire and Yoram Singer. Improved boosting algorithms using confidence-rated predictions. In *Proceedings of the eleventh annual conference on Computational learning theory*, pages 80–91. ACM, 1998.

[62] Yoav Freund and Robert Schapire. A short introduction to boosting. *Journal-Japanese Society For Artificial Intelligence*, 14(771-780):1612, 1999.

[63] Andrew Howard and Nicholas Roy. The robotics data set repository (radish), 2003. URL `http://radish.sourceforge.net/`.

[64] Andrea Censi. An icp variant using a point-to-line metric. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 19–25. IEEE, 2008.