Recovering Cholesky Factor in Smoothing and Mapping

Sébastien Touchette

Thesis submitted in partial fulfilment of the requirements for the
Doctorate in Philosophy degree degree in Electrical and Computer Engineering

School of Electrical Engineering and Computer Science
Faculty of Engineering
University of Ottawa

Abstract

Autonomous vehicles, from self driving cars to small sized unmanned aircraft, is a hotly contested market experiencing significant growth. As a result, fundamental concepts of autonomous vehicle navigation, such as simultaneous localisation and mapping (SLAM) are very active fields of research garnering significant interest in the drive to improve effectiveness.

Traditionally, SLAM has been performed by filtering methods but several improvements have brought smoothing and mapping (SAM) based methods to the forefront of SLAM research. Although recent works have made such methods incremental, they retain some batch functionalities from their bundle-adjustment origins. More specifically, relinearisation and column reordering still require the full re-computation of the solution.

In this thesis, the problem of re-computation after column reordering is addressed. A novel method to reflect changes in ordering directly on the Cholesky factor, called Factor Recovery, is proposed. Under the assumption that changes to the ordering are small and localised, the proposed method can be executed faster than the re-computation of the Cholesky factor. To define each method's optimal region of operation, a function estimating the computational cost of Factor Recovery is derived and compared with the known cost of Cholesky factorisation obtained using experimental data. Combining Factor Recovery and traditional Cholesky decomposition, the Hybrid Cholesky decomposition algorithm is proposed. This novel algorithm attempts to select the most efficient algorithm to compute the Cholesky factor based on an estimation of the work required.

To obtain experimental results, the Hybrid Cholesky decomposition algorithm was integrated in the SLAM++ software and executed on popular datasets from the literature. The proposed method yields an average reduction of 1.9 % on the total execution time with reductions of up to 31 % obtained in certain situations. When considering only the time spend performing reordering and factorisation for batch steps, reductions of 18 % on average and up to 78 % in certain situations are observed.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The ability of unmanned vehicles to explore their environment is not only a prerequisite for more complex tasks, but a valuable end product which has seen unmanned vehicles gain a strong footing in the mapping industry. For example, underwater caves, which are a serious hazard to all but the most experienced divers, can now be mapped by robots [1] (Figure 1.1a). Mapping ground environments can also present safety challenges, during armed conflict for example, where unmanned vehicles are used for applications such as mine sweeping [2, 3] (Figure 1.1b), tunnel exploration [4] and intelligence gathering [5]. From a cost-benefit point of view, unmanned aerial systems (UAS) compare favourably to aircraft as a complement to satellite imagery and are well within reach of small companies.

The use of unmanned vehicles in search and rescue also takes full advantage of their ability to map and explore their environment. Due to their high mobility, UAS are of particular interest for this application and have been used for disaster monitoring and post-disaster information gathering operations [10, 11]. Small ground vehicles have also been designed to explore rubble, unstable buildings or contaminated areas [12]. The research interest in this field is such that multiple competitions have been organised to promote innovation [13].

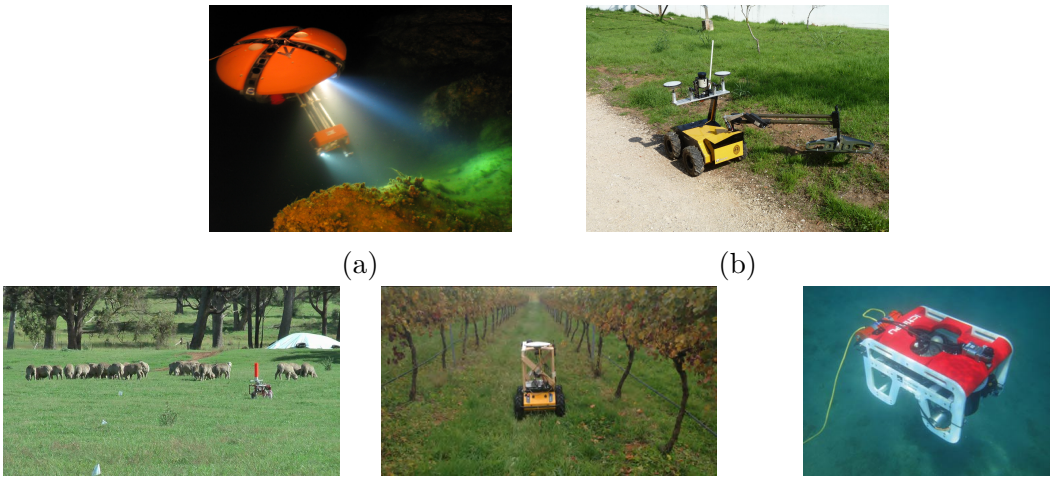


Figure 1.1: Example of unmanned vehicle uses: (a) underwater cave exploration [1] (Image courtesy of Carnegie Mellon University/NASA), (b) mine sweeping [2, 3], (c) sheep herding [6], (d) vineyard yield estimation [7], and (e) inspection of hydroelectric dams [8, 9].

Unmanned vehicles have also been designed with capabilities tailored to the needs of various industrial sectors. In the farming industry, unmanned vehicles have been used for tasks such as sheep herding [6] (Figure 1.1c) and estimating the yield of vineyards [7] (Figure 1.1d). Companies in the fields of cargo shipping, off-shore petroleum exploitation and hydroelectricity have also shown interest in using them for inspection and maintenance tasks [8, 9] (Figure 1.1e). The healthcare sector is a large potential market for unmanned vehicles where some are used for greeting and guiding patients arriving to the hospital while others are being designed for elderly care [14].

Regardless of their type or purpose, most unmanned vehicles are confronted by the same problems. One in particular, concerning the very nature of their autonomy, is how such systems navigate their environment effectively or, as is often the case, navigate while mapping their environment. In the literature, this problem is referred to as simultaneous localisation and mapping (SLAM). The field of SLAM is an active area of research with many unsolved problems that stem from the increasingly demanding applications in which unmanned vehicles are used. For example, once used only in small indoor areas,

unmanned vehicles are now expected to navigate large environments over a period of days or weeks [15]. This represents a significant challenge as computation requirements increase with the number of landmarks (Recognisable environment features saved). Moreover, the miniaturisation trend seen in some applications limits the hardware available to small autonomous vehicles. These vehicles stand to benefit most from such improvements.

In Section 1.1, the problems encountered in SLAM are presented. The objectives of this thesis and contribution to the field are discussed in Section 1.2 and 1.3 respectively. Section 1.4 concludes the chapter with an overview on the thesis structure.

1.1 Problems in SLAM

The increasing demand for longer missions and ability to **explore large areas** drive constant improvements in SLAM algorithms [16] (Figure 1.3a). To put things in perspective, the *Victoria Park* dataset of Figure 1.2 was obtained on a 4km path within a 300×300m area. After a period of 25 minutes of navigating the environment, the dataset already contained 151 landmarks. As the size of the area to be covered increases, SLAM algorithms quickly become computationally intractable. While this has been mitigated by the increasing performance of hardware in larger vehicles, the **processing capacity** of small UAS remains limited [17, 18] (Figure 1.3b). Whether to increase flight time or to leave more room for other instruments, there is a desire for navigation systems to be smaller and lighter. This imposes space and power consumption constraints on the hardware and is a driving force for the increase of algorithm efficiency beyond the existing desire for a reduced execution time. Algorithms using approximations to boost efficiency have been proposed and allow for larger maps and implementation on small size computers. In most situations, however, compromising robustness and precision is not acceptable. In fact, the demand for more accurate SLAM pushes the need for algorithms with a higher **accuracy and error robustness** in a wide variety of environments. The need for efficiency with-

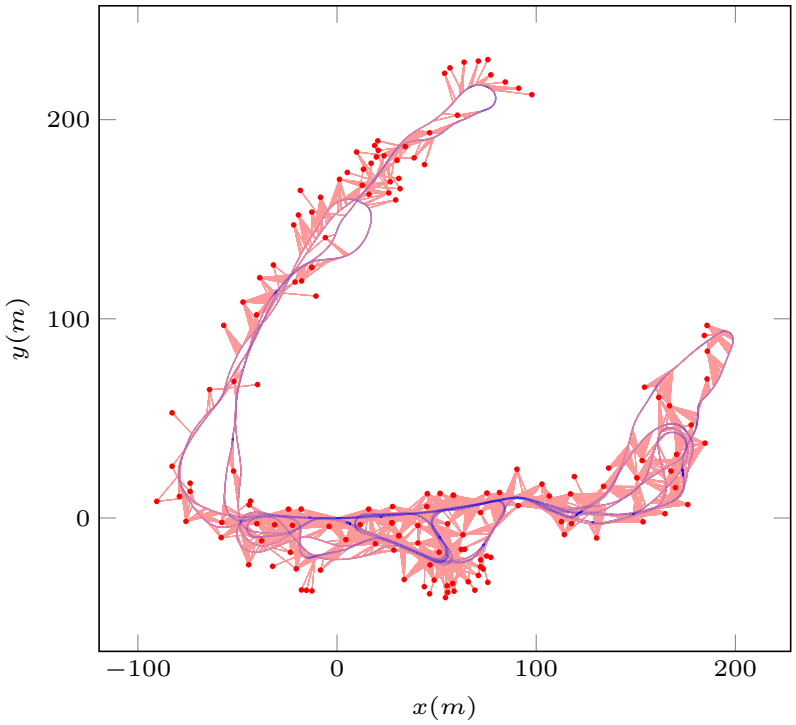


Figure 1.2: *Victoria Park* dataset

out compromising accuracy has been met by increasingly efficient SLAM algorithms that minimize the number of operations required or techniques dividing large areas into smaller regions. Such division of the environment led to multiple vehicles used together in a **collaborative mapping** effort to explore different regions simultaneously [19] (Figure 1.3c). This concept has gained traction in time sensitive applications such as search and rescue. To this end, recent research have pushed the boundaries of distributed algorithms allowing the SLAM solutions of different platforms to be merged faster for a higher combined efficiency. Autonomous vehicles involved in search and rescue and similar activities have the added difficulty of evolving in **dynamic environments**. The dynamic nature can be caused by anything from a collaborating vehicle to a lighting change. Additional knowledge can be gained by including this information instead of discarding it. In most SLAM algorithms however, moving objects are either rejected from the mapping or lead to erroneous estimations. This is true as well for environmental changes in cases where the mapping task is accomplished over multiple days or months. Recent research have been

pushing the boundaries to include methods able to recognise such changes but it is still an open problem in SLAM.

(a) (b) (c)

Figure 1.3: Three of the five main problems encountered in SLAM: (a) large environments [16], (b) small platform [18], and (c) multi-vehicle collaboration [19]

1.2 Objectives

Autonomous vehicles are often used for exploration-related tasks due to their mobility, storage convenience and affordability. Such vehicles, limited by the available computational power and their small footprint, require efficient SLAM implementations, even more so when used for long endurance missions such as mapping large areas. Recently, a class of methods called smoothing and mapping (SAM) have been proposed to address this issue. They use a smoothing based approach to SLAM and are able to incrementally update the vehicle's estimated position as it explores new areas.

Ever so often, these methods need to recompute the map and vehicle path from scratch to maintain the numerical and time performance of the SLAM algorithm as the vehicle explores the environment and the configuration of the problem changes. For example, when a vehicle moves between various areas of an environment, it is more efficient that the landmarks located in the current area be more readily accessible than the others. The operation of recomputing the entire path and map in a single step is called a batch operation. This is typically performed offline and requires significantly more time than the typical incremental step. In this thesis, the issue of the batch step performance is

addressed. The objectives are to obtain a SAM algorithm to compute the batch solution in a way that is:

- **More computationally efficient:** The proposed method should have a computational cost lower than the batch operation.
- **Equivalent:** Any solution must be mathematically equivalent to that obtained by a batch operation and have the same precision and robustness. To this end, the outcome of a batch operation is considered as the ground truth.
- **General:** The methodology must be easy to integrate in any smoothing based SLAM implementation and, if possible, extend beyond SLAM to dynamic problems in general (e.g. factorising matrices of social network relations).

1.3 Thesis Contribution

The thesis proposes a novel SLAM technique to perform the same function as a batch step but, in some conditions encountered in SLAM, it does so in a more efficient manner by reusing previously calculated results. The contributions satisfying the objectives presented in Section 1.2 can be divided as follows:

- **Factor Recovery algorithm:** When a vehicle evolving in an environment requires a batch step, this innovative algorithm allows the current map, path and intermediate results to be reused in the computation of the new map and path. This may lead to significant improvements on the computational complexity of batch steps and allow results to be obtained faster. The Factor Recovery algorithm is also applicable beyond SLAM to any similar problem whose dynamics vary with time.
- **Threshold Selection:** Depending on the environment, the Factor Recovery algorithm is not always more efficient than a batch step. This contribution aims to detect such

situations, in which case the traditional batch method is applied. This is done using information about the current map and path that are accessible to the vehicle during the mission.

- **The Hybrid Cholesky algorithm:** This algorithm consists of the integration of the two previous contributions onto an existing SLAM platform. The result is a SLAM implementation that allows a vehicle exploring an environment to forego batch operations by using Factor Recovery instead of traditional Cholesky decomposition when a threshold is satisfied. Experimental results of SLAM++ [20] with and without the Hybrid Cholesky algorithm are provided to characterise the efficiency of these contributions. They are executed on popular datasets and cover a variety of indoor and outdoor environments from real and simulated data.

1.4 Thesis Organisation/Outline

The thesis is organized as follows:

In Chapter 2, mathematical concepts necessary for the understanding of the thesis are presented. The chapter begins with a brief introduction of SLAM to set the context, followed by a description of the important datasets used in examples throughout the thesis. A detailed formal explanation of SLAM from both the probabilistic and graphical point of view is then presented after which Cholesky factorisation, elimination trees and sparse Cholesky factorisation are discussed.

In Chapter 3, a survey of the work done in the field of SLAM is presented. For each basic SLAM method, a mathematical description is provided followed by a brief overview of some important work related to the method in question. In general, the works discussed are presented in chronological order and divided on whether it relates to fundamental research or experimental implementations. Extensions to these methods addressing typical problems

encountered in SLAM are then presented. The chapter concludes with a comparison of the various basic methods and their extensions in a table format.

In Chapter 4, the contributions of the thesis are presented in detail. The chapter starts with a comparison a various SAM methods and a justification for selecting SLAM++ as a basis for this work. This is followed by background information regarding the chosen algorithm. The core of the chapter contains theoretical calculations supported by case studies detailing the contributions of the thesis.

In Chapter 5, simulation results for the algorithms derived in Chapter 4 are presented using datasets from the literature. The results are discussed and recommendations are made to the reader regarding usage and performance.

In Chapter 6, the thesis outcome and contributions are reiterated. A few possible future research avenues are suggested.

Chapter 2

Background

In this chapter, the necessary background information for a better understanding of the thesis is presented. Readers familiar with the topic at hand may choose to read only Section 2.2 regarding the datasets used and refer to the rest of the chapter as needed.

In Section 2.1, a brief introduction to SLAM is provided. The example datasets (*Victoria_Park_7119*, *Victoria_Park_500* and *Artificial_11*) are introduced in Section 2.2, along with their content and scope.

The mathematical formulation of SLAM is discussed in Section 2.3, starting with the various types of graphical representations in Section 2.3.1. Particular attention is given to the Markov random field (MRF) since it is the graph type that is used throughout the thesis. The information matrix associated with a MRF is presented and, from this, sparse matrices are introduced. Important concepts of graph theory relating a MRF to a probability density and the concept of marginalisation are also discussed. The probabilistic representation of SLAM is then presented (Section 2.3.2), starting with the basic concepts such as data association and measurement probabilities, which are linked to MRFs. The two possible SLAM formulations, Classical SLAM and full SLAM, are then discussed.

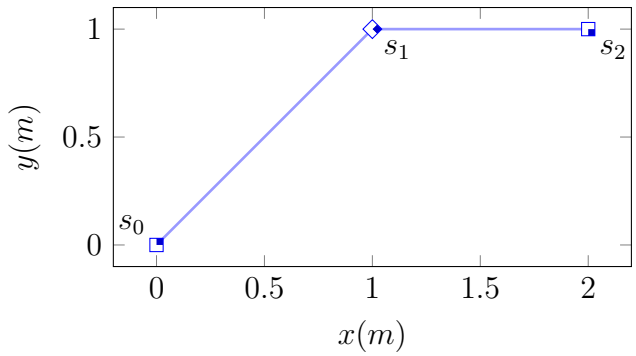
The next three sections present key mathematical concepts that are fundamental to the thesis. In Section 2.4, a refresher on the Cholesky decomposition is presented, followed by an overview of the elimination tree and its properties in Section 2.5. Section 2.6 introduces sparse Cholesky factorisation as well as factor modification and column ordering operations.

Finally, Section 2.7 concludes the chapter with a summary of the concepts presented.

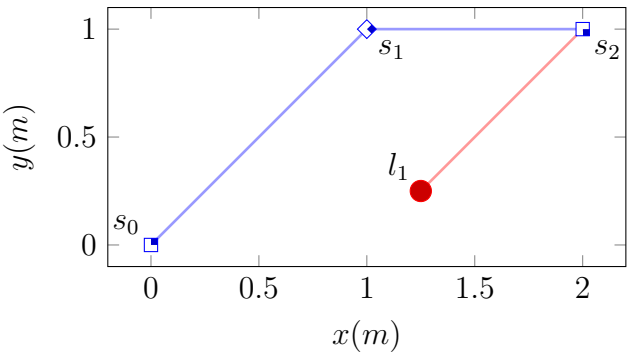
2.1 Simultaneous Localisation and Mapping (SLAM)

Simultaneous localisation and mapping is the process of recursively building a map of the world explored by a vehicle and, using this map, to estimate the vehicle's position. Typically, the relative motion of the vehicle from one time instant to another is captured with a dead reckoning sensor, defined as any sensor that integrates state changes over time to find a current state. These relative motion measurements are called odometry measurements. In Figure 2.1a, an example of a dead reckoning trajectory with three poses is displayed. In the example and throughout this thesis, vehicle poses are represented by a blue diamond-shaped outline centred at the vehicle's position and oriented such that the coloured tip of the diamond indicates the forward direction. An odometry measurement takes the form of a relative displacement vector from an initial position and is represented by a blue line between the original and final positions. A GPS position measurement takes the form of a relative displacement vector between the vehicle and a fixed reference frame.

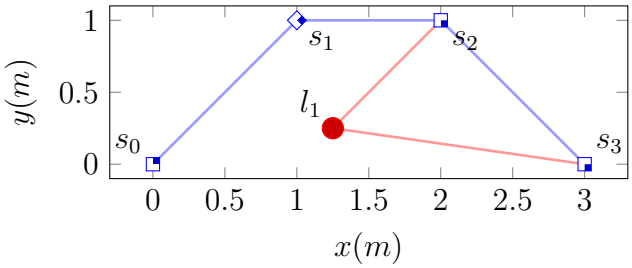
Landmarks are observed and often consist of notable features of the environment extracted from camera or laser data. The position of these landmarks with regards to the current vehicle position is called a telemetry measurement or an observation. In a process called data association (DA), a database of known landmarks is searched to find if the landmark was already observed. The decision of whether or not a landmark is already in the landmark database is based on the position of the vehicle, the telemetry measurement obtained as well as the characteristics of the landmark itself. While data association is



(a) Trajectory with three vehicle poses.



(b) Trajectory with three vehicle poses and a landmark measurement.



(c) Trajectory with four vehicle poses and two landmark measurements illustrating a loop closure.

Figure 2.1: An autonomous vehicle navigates three poses using dead reckoning (Figure 2.1a) and a landmark is observed (Figure 2.1b). It then navigates to another pose and observes the same landmark (Figure 2.1c). Vehicle poses are represented by a diamond outline centred at the $x - y$ coordinate of the vehicle and oriented such that the coloured tip represents the forward facing direction. Odometry measurements are represented by blue lines while telemetry measurements are red lines. Landmarks are represented by red circles centred at the landmark's position.

closely coupled with SLAM, it is usually treated as a separate problem to reduce complexity. Further discussion on data association and the coupling between it and SLAM will be discussed in Sections 2.3.1 and 2.3.2.1 respectively.

In the case where the landmark has not been previously observed, it is added to the landmark database. In Figure 2.1b, a landmark l_1 has been added to the example. Throughout this thesis, landmarks are represented by a red circle centred at the landmark's position. A telemetry measurement, represented by a red line, is a relative position vector between a vehicle position and a landmark. A practical example of such a measurement would be an object observed by a laser range-finder. Note that although telemetry and odometry

are both relative position measurements, telemetry is obtained from a single pose by observing the environment while odometry is obtained by monitoring the vehicle's position as it travels between two poses.

In cases where the landmark already exists in the database, an observation is added linking it to the current vehicle pose. An example of such as case is displayed in Figure 2.1c, where a new vehicle pose and a landmark observation have been added to the previous example. Mathematically, this addition introduces a loop in the diagram since it is possible to go from s_3 to s_2 using two different paths: using the telemetry measurements (red edges) involving l_1 and using the odometry measurement (blue edge) between s_2 and s_3. In the field of SLAM, this is called loop closing and it is a fundamental concept as this redundant information can be merged to obtain a more precise estimate of the relative position between landmarks or vehicle poses (in our case, s_2 and s_3).

The SLAM algorithm combines the odometry and telemetry measurements to form a map of the environment travelled by the vehicle. A block diagram of how SLAM integrates with the various components of a navigation system is shown in Figure 2.2 for a vehicle equipped with GPS, inertial navigation system (INS), laser scanner and camera. In a typical unmanned vehicle, sensor data is preprocessed to remove noise from high rate sensors or identify areas of interest in environment sensors (camera, laser). For environment sensor information, data association is then performed to convert areas of interest to landmarks (lm), which are used by SLAM to track what is being observed. The SLAM module takes inputs from the sensor preprocessors and data association module. This information is used to maintain the landmark database and return an accurate estimate of the vehicle's position (pos) and position of landmarks (map) to the path planner. SLAM modules do not necessarily maintain an image of the map for the user. In many cases, only a descriptor (visual fingerprint) of each landmark is stored.

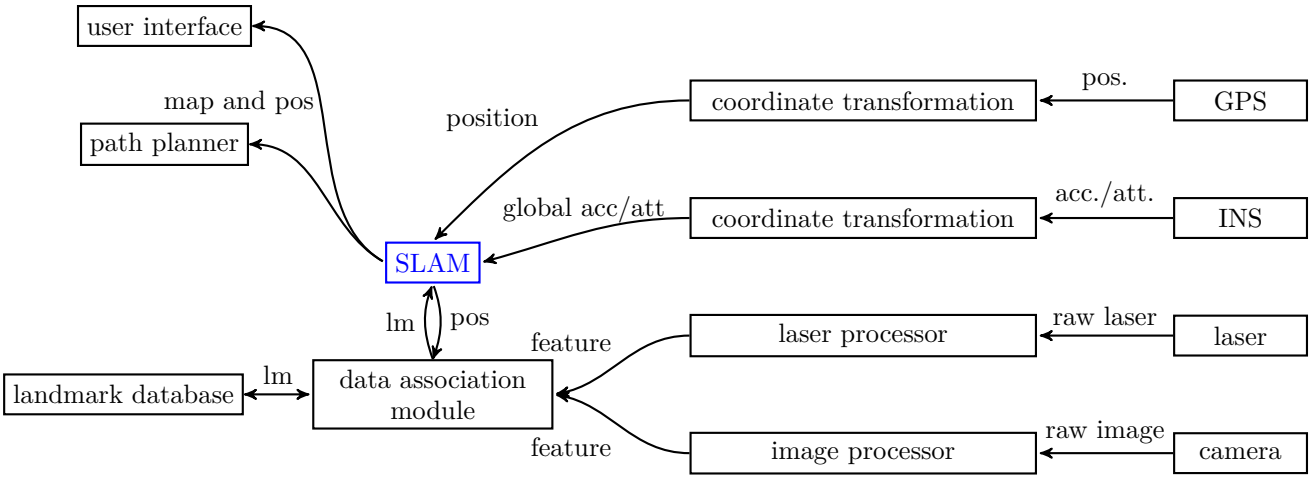


Figure 2.2: Diagram of navigation software. The SLAM module is coloured in blue.

2.2 Demonstration Datasets

In this section, the datasets used to explain concepts and illustrate algorithms are introduced. They are presented in decreasing order of complexity and summarised in Table 2.1.

Table 2.1: Example Datasets

Dataset Name	Number of Poses	Number of Landmarks
Victoria_Park_7119	6968	151
Victoria_Park_500	312	29
Artificial_11	9	2

Real and artificial datasets used to evaluate the performance of the contribution will be presented later in Section 5.1.1.

2.2.1 *Victoria_Park_7119*

Due to its popularity in the field and widespread availability, the Victoria Park dataset has been selected for the purpose of explaining SLAM concepts. This dataset is also well suited as a benchmark since it consists of experimentally obtained sensor information from an outdoor trajectory, which is representative of the need to map large areas with

sensor uncertainty. The dataset has been obtained using a vehicle mounted with a dead reckoning sensor, GPS and laser scanner. The dead reckoning sensor consists of a steering angle encoder and velocity encoder, recording the vehicle's speed and change in heading at 40 Hz. The SICK laser sensor has a range of 80 m and scans the environment every 214 ms. Each scan consists of 361 individual range measurements at 0.5° intervals over a 180° arc centred in front of the sensor. The vehicle followed a 4 km path in Victoria Park and the lightly wooded area provided multiple landmarks detectable with the laser scanner. The set contains a total of 4466 GPS points, 61945 steering angle and velocity samples and 7249 laser scans.

To remove the effect of feature detection and data association from the navigation and mapping problem and focus on the SLAM algorithm, a processed version of the original dataset is used. The set has been modified such that landmark observations are given by landmark IDs and do not require processing the laser scans. Furthermore, consecutive dead reckoning points were combined to reduce the size of the dataset and GPS data points were omitted. A typical odometry and landmark observation found in the modified dataset file consists of:

ODOMETRY 3 4 0.0640488 -2.22135e-06 -8.11689e-05 100 0 0 500 0 500
LANDMARK 4 5 11.5387 -3.2007 1.581139 0 1.581139

where the first entry identifies the type of measurement, the second and third identify the nodes involved, and the following information is the measurement data. For example, in the case of the landmark observation, node 5 (Landmark) is located at (11.5387, -3.2007) when observed from node 4. The other values give the upper diagonal entries of

the precision matrix as follows

$$\begin{bmatrix} 1.581139 & 0 \\ - & 1.581139 \end{bmatrix}$$

the importance of which will be discussed later.

The modified dataset, *Victoria_Park_7119*, has 6968 poses and 151 landmarks for a total of 7119 points and is displayed in Figure 2.3. The dataset contains 10608 observations, of which 151 are new landmark observations, 3489 are loop closings and 6968 are odometry measurements.

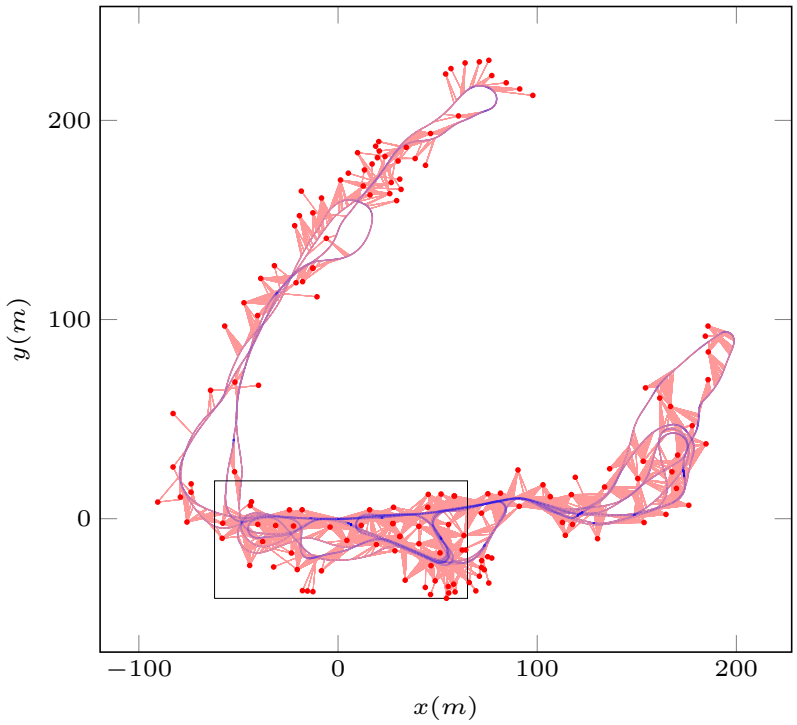


Figure 2.3: *Victoria_Park_7119* dataset. The framed section represents *Victoria_Park_500* as described in Section 2.2.2.

2.2.2 *Victoria_Park_500*

Due to the large amount of data it contains, only a subsection of *Victoria_Park_7119* is used to illustrate the theory. This reduced dataset consists of the first 500 poses and landmarks, along with their associated observations. This subsection of the dataset can be seen in Figure 2.4.

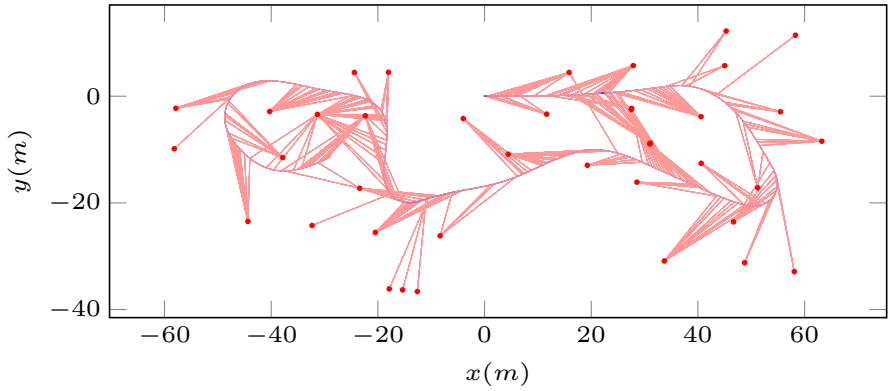


Figure 2.4: *Victoria_Park_500* is a subset of *Victoria_Park_7119* used for explanations.

2.2.3 *Artificial_11*

Finally, in sections where equations are supported by illustrative examples, the artificial dataset shown in Figure 2.5 is used. It consists of nine vehicle poses and five telemetry observations, three of which are loop closings.

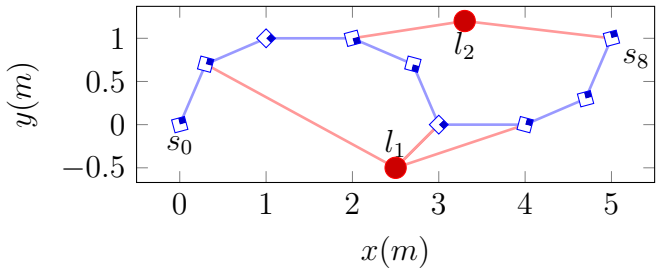


Figure 2.5: *Artificial_11* dataset. Used for visualising calculations.

2.3 Mathematical Formulation

In this section, the mathematical formulation of SLAM is introduced. First, the SLAM problem is represented using graphical models since graphs can be drawn without in-depth knowledge of statistics and these models can be intuitively related to the practical aspects of mapping.

2.3.1 Graphical Representation of SLAM

Graphical models are useful tools for representing probabilistic problems in terms of the conditional dependence between variables. They consist of using the advantages of graph theory to represent complex interactions between variables and provide rigorous methods to solve the problems they represent. Depending on the characteristics of the graph used, the resulting model is referred to as a Bayesian network (BN), Factor graph (FG) or Markov random field (MRF).

2.3.1.1 Bayesian network

The Bayesian network representation of statistical problems can be obtained by representing each variable and constraint of the problem as a node in a graph and tracing edges from each node to the nodes that depend on it. For the SLAM problem, drawing one node for each variable (poses, landmarks) or constraint (measurements) will yield a graph containing k odometry measurement nodes ($\mathbf{u}$), n_z telemetry measurement nodes ($\mathbf{z}$), n landmark nodes ($\mathbf{l}$) and $k+1$ pose nodes ($\mathbf{s}$).

Each pose node depends on the previous pose and the odometry measurement of the relative displacement between them. Following this rule, two edges are directed towards each pose node: one from the previous position and one from the odometry measurement. In the simple case where data association is known, each telemetry measurement depends

on the observed landmark and the vehicle pose it was observed from. This introduces two incident edges to the telemetry measurement node: One from the vehicle pose and one from the observed landmark.

Using the *Artificial_11* dataset of Figure 2.5, the BN representation illustrated in Figure 2.6 is obtained. The Bayesian network is required to be a directed acyclic graph (DAG), which means that it is not possible, from any given node, to follow the edges along the designated direction and return to said node. BN are thus unable to represent cases where data association is not known since in such case, the data association and telemetry measurement are interdependent as illustrated in Figure 2.7. In this Figure, the telemetry measurement $\mathbf{z}$ depends on which landmark is observed ($\mathbf{n}$) but obtaining $\mathbf{n}$ requires that the current measurement ($\mathbf{z}$) be compared with what would be measured if the existing landmark ($\mathbf{l}$) was observed at the current vehicle pose ($\mathbf{s}$). This is discussed further in Subsection 2.3.2.

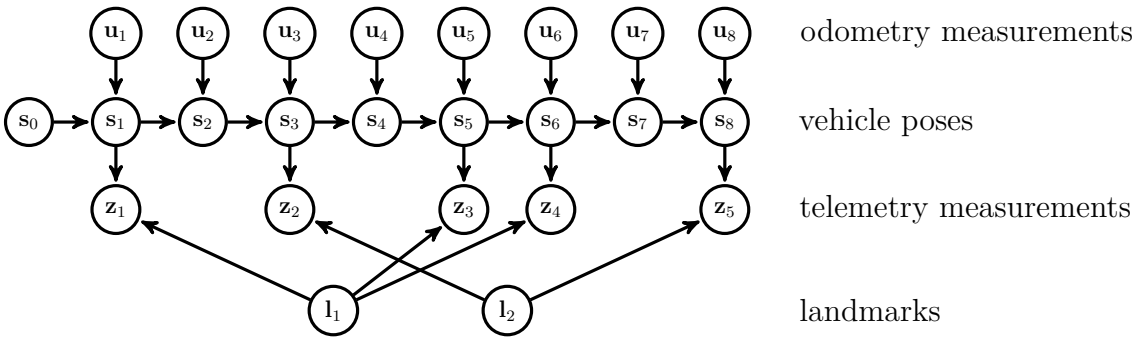


Figure 2.6: Example of a SLAM problem as a Bayes network

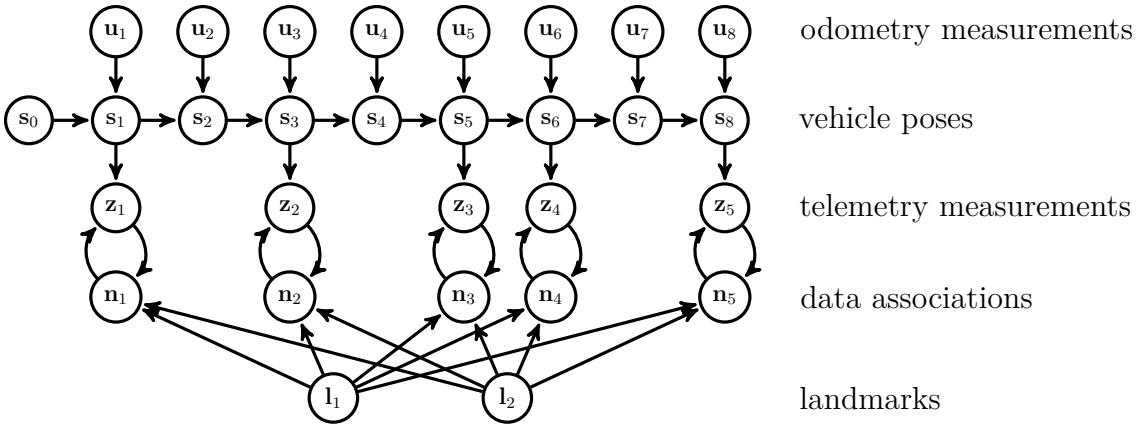


Figure 2.7: Graph of *Artificial_11* with unknown data association. Due to the cycle introduced between **n** and **z**, this graph is no longer a BN.

2.3.1.2 Factor Graph

The factor graph is another way of representing statistical problems and offers additional tools for efficient computation of results but requires the graph to be bipartite: the nodes of the graph must be dividable in two groups such that any given node is only connected to nodes of the other group. The SLAM problem with known data association is one such problem.

Using the *Artificial_11* Dataset, the FG representation of Figure 2.8 is obtained. The odometry (**u**) and telemetry (**z**) measurements are constraint nodes, drawn as dark squares, that are used to link vehicle poses (**s**) or a vehicle pose and a landmark (**l**). Note that an extra constraint node is present at the beginning of the path to represent the constraint of the initial position. As with the BN, the FG can not represent unknown data association since the resulting graph will no longer be bipartite.

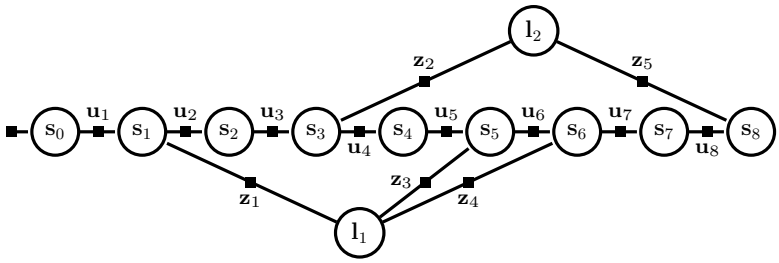


Figure 2.8: Example of a SLAM problem as a factor graph. Constraint nodes are drawn as dark squares while variable nodes are drawn as circles.

2.3.1.3 Markov Random Fields

Markov random fields are also popular for representing statistical problems. The MRF representation of a statistical problem can be obtained by representing each variable of the problem as a node of a graph. For a SLAM problem, drawing one node for each variable yields a graph containing n landmark nodes ($\mathbf{l}$) and $k+1$ pose nodes ($\mathbf{s}$). The constraints between the nodes, represented by the edges, are obtained from the measurement functions. Namely, for the odometry measurement function,

$$\mathbf{s}_n = \mathbf{s}_{n-1} + \mathbf{u}_n \tag{2.1}$$

indicates how any given pose is related to the previous one. For the telemetry measurement function,

$$\mathbf{l}_k = \mathbf{s}_n + \mathbf{z}_{n,l_k} \tag{2.2}$$

indicates that a landmark is related to the pose from which it was observed. Applying this method to the *Artificial_11* dataset, the MRF representation illustrated in Figure 2.9 is obtained. Note that the cause-effect relationship given by the directions of the edges in Figure 2.6 is not present in the graph. Contrary to the BN and factor graph, a MRF contains only the variables of the problem ($\mathbf{s}$ and $\mathbf{l}$) and allows for the representation of cycles.

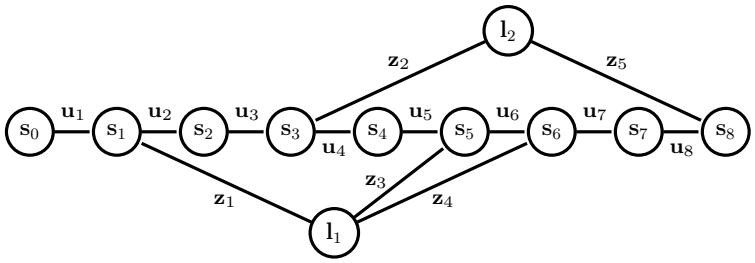


Figure 2.9: Example of a SLAM problem as a Markov random field

The MRF representation will be used throughout the remainder of this thesis to illustrate the theory and examples.

2.3.1.4 Markov Random Fields: Sparse Matrices

Since there is an equation of the form of (2.1) or (2.2) associated with each edge, they form a system of equations that can be represented in the form

$$\mathbf{A}\mathbf{x} + \mathbf{b} = 0, \tag{2.3}$$

where $\mathbf{x}$ is a vector of the pose and landmark nodes ($\mathbf{s}$ and $\mathbf{l}$), $\mathbf{b}$ is a vector of the measurements ($\mathbf{z}$ and $\mathbf{u}$) and $\mathbf{A}$ is a Jacobian matrix. For the graph of Figure 2.9, the terms of (2.3) are represented in matrix form in Figure 2.10. The dark entries represent non-zero values and e refers to the edge number while s, l, u and z are the corresponding nodes in Figure 2.9.

In the case of a MRF, it is desired to represent how nodes relate to one another rather than the individual constraints. To do so, the normal equation of (2.3) is considered. The new system obtained is

$$\mathbf{C}\mathbf{x} = -\mathbf{A}^T\mathbf{b} \tag{2.4}$$

where $\mathbf{C} = \mathbf{A}^T\mathbf{A}$ is, by definition, a real positive definite matrix. The matrix $\mathbf{C}$, called the information matrix, is shown in Figure 2.11.

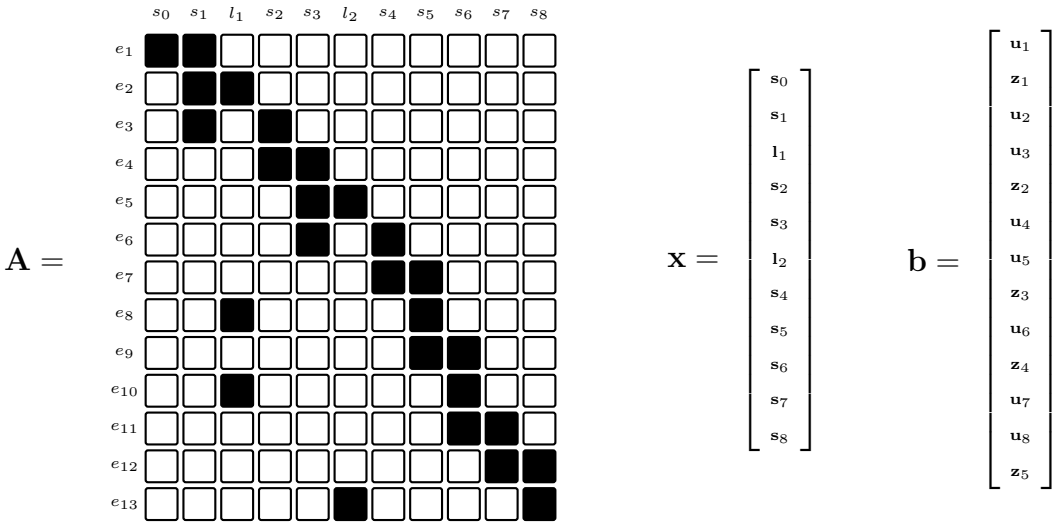


Figure 2.10: Matrices of (2.3) for the MRF representation of *Artificial_11*

Note that because of the lack of directionality in the MRF compared to the BN of Figure 2.6, the edge between two nodes is the same regardless of direction and thus the matrix is symmetric. Due to the simple nature of the equations relating the nodes, the information matrix contains many zero values. Such matrices are called sparse matrices and the low density of non-zero values may be leveraged to greatly reduce the memory required to store them. To do so, the zero entries are omitted and the values are stored sequentially, column by column. A list of column boundaries and element row indices are also stored to keep track of where the values are in the matrix. These matrices are thus represented as sets, where each column entry contains a list of values and associated indexes. As such, matrices expressed in this form will be represented by cursive letter of the set notation such as $\mathcal{A}$ for Figure 2.10 or $\mathcal{C}$ in the case of Figure 2.11.

$\mathbf{C} =$

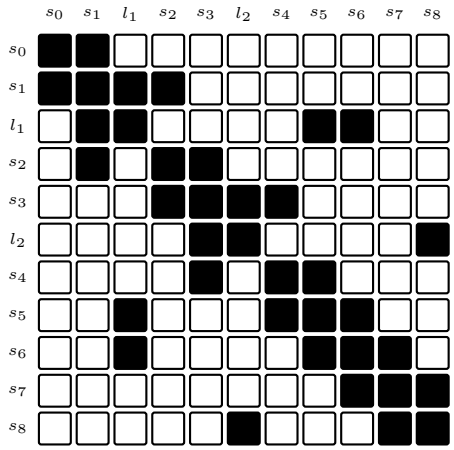


Figure 2.11: Matrix $\mathbf{C} = \mathbf{A}^T\mathbf{A}$ of (2.4) for the MRF representation of *Artificial_11*

2.3.1.5 Markov Random Fields: Graph Theory

Before going into more details, several graph concepts must be introduced. More specifically, the concept of complete graph and cliques, followed by the equation for calculating the probability of a certain graph configuration.

Complete graphs are graphs where any given node is connected to any other node in the graph. An example of a graph that is non-complete and one that is complete is illustrated in Figure 2.12

Figure 2.12: (a) a non-complete graph example; (b) a complete graph example.

A Clique is a subset of nodes of a graph that form a complete subgraph. A clique is called a maximal clique when it is not possible to include neighbours of clique nodes in the subset without making the resulting subgraph non-complete. In the MRF representation of *Artificial_11* displayed in Figure 2.13, the nodes in red and blue are examples of cliques. It can be seen that the clique formed by the blue nodes is a maximal clique since the

addition of any neighbour node (s_1, s_4 or s_7) makes the resulting subgraph non-complete and thus no longer a clique. The nodes s_5 and l_1 form a two node clique similar to the red clique. The red clique however, is a maximal clique while node s_6 can be added to node s_5 and l_1 to form a larger clique (in blue).

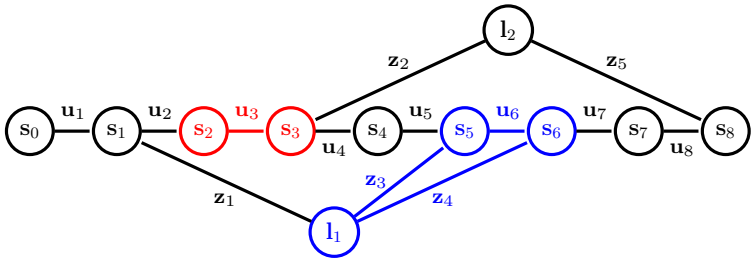
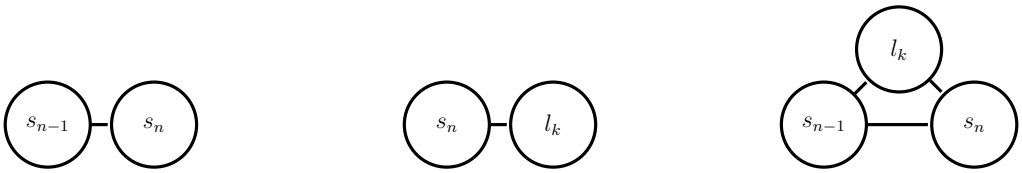


Figure 2.13: Two examples of maximal cliques (in blue and red) in the MRF of *Artificial_11*.

Due to the constraints of the SLAM problem, only successive poses are connected and landmarks are not connected to each other. As such, cliques in a MRF representing a SLAM problem can be formed of two or three nodes. As illustrated in Figure 2.14, the possible cliques encountered in SLAM are

- Two pose nodes that do not share a landmark
- A pose node and a landmark node that has not been observed at the previous or next pose
- A landmark and two consecutive poses from which it was observed



(a) (b) (c)

Figure 2.14: In SLAM, the cliques can consist of (a) two consecutive pose nodes, (b) a pose and a landmark node or (c) two consecutive pose nodes and one landmark node

Identifying the cliques of a graph is an essential aspect of graph theory as it is used to calculate the probability of the graph's random variables taking a given set of values, referred to as a graph configuration. In the case of SLAM, this corresponds to the probability that given values $X = \{\mathbf{s}_{0:k}, \mathcal{M}\}$ are the vehicle poses and landmark positions. The probability that a MRF has configuration X is given by

$$P(X) = \frac{1}{Z} \prod_{C \in cl(G)} \Phi_c(X_c) \tag{2.5}$$

where

$$Z = \sum_{x} \prod_{C \in cl(G)} \Phi_c(X_c) \tag{2.6}$$

C is the index variable for the cliques of graph G, X_c represents the graph nodes that are in clique C and $\Phi_c(X_c)$ is called the clique potential, which can be any positive function of the variables in the clique. This is discussed further in Section 2.3.2.2.

2.3.1.6 Markov Random Fields: Marginalisation

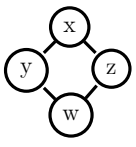
An important concept in graphical models is node marginalisation (or elimination). This process involves removing a node and its edges from the graph while adding edges between all neighbours of the eliminated node. This is analogous to variable elimination in linear algebra, where a variable (node) is replaced by it's expression (edges to neighbours) in the expressions of variables depending on it (neighbour nodes) to remove the variable from the system. For example, if we have the following symmetric system and it's associated MRF:

$$\text{x:} \quad 2x - 1y - 1z + 0w = 0 \tag{2.7}$$

$$\text{y:} \quad -1x + 2y + 0z - 1w = 0 \tag{2.8}$$

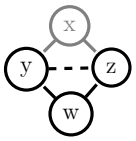
$$\text{z:} \quad -1x + 0y + 2z - 1w = 0 \tag{2.9}$$

$$\text{w:} \quad 0x - 1y - 1z + 2w = 0 \tag{2.10}$$



Eliminating x by substituting $x = 0.5y + 0.5z$ in the other equations yields the following system:

$$\begin{aligned} \text{y:} \quad & 1.5y + -0.5z - 1w = 0 \quad (2.11) \\ \text{z:} \quad & -0.5y + 1.5z - 1w = 0 \quad (2.12) \\ \text{w:} \quad & -1y - 1z + 2w = 0 \quad (2.13) \end{aligned}$$



where removed node and edges are grey and the new edge is dashed. The variable x is no longer part of the system and is no longer present in the equations for y and z (edges $x-y$ and $x-z$ are removed). However, due to this substitution, the equation of y now has a dependence on z and vice-versa (edge added between y and z).

2.3.2 Probabilistic Representation of SLAM

In this section, the SLAM problem is discussed from a probabilistic point of view. The section begins by discussing data association in details, followed by the basic description of the SLAM problem. The development of the solution to SLAM ties in to what was covered with regarding MRFs in Section 2.3.1.5 and presents the two basic SLAM formulations.

2.3.2.1 Data Association (DA)

When a telemetry measurement ($\mathbf{z}$) of a landmark is obtained, a decision must be made on whether the landmark has already been observed and, if so, which of the stored landmarks it corresponds to. This process has been briefly introduced as data association in previous sections. As it is an important SLAM concept and used in the derivations that follow, it is discussed here in more details.

Before performing data association, landmarks are extracted from the raw sensor data and characterised in a meaningful way. In the case of the Victoria Park dataset, it is known that the landmarks observed are trees, and thus their observed position and trunk

diameter are used as identifiers. In the case of visual images, more complex techniques using feature recognition can be used. For an example of a data association algorithm, the reader is referred to the joint compatibility branch and bound algorithm [21]. Once a landmark is obtained, data association is performed by comparing the characteristics of the extracted landmark (position and trunk diameter in the case of Victoria Park) with that of landmarks already encountered. Due to proximity of landmarks and measurement noise, there is uncertainty involved in the data association process and many existing landmarks may be possible matches. The landmark observed may also be new. This uncertainty results in data association returning a discrete probability density of matches in the form of

$$p(\mathbf{n}_{z_i}|\mathbf{s}_{z_i}, \mathbf{z}_i, \mathcal{M}) \tag{2.14}$$

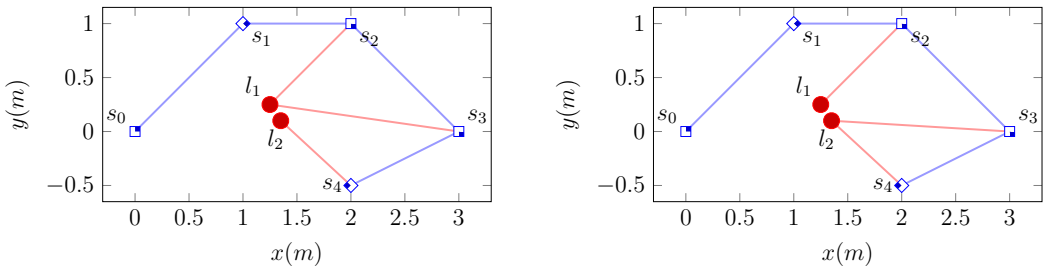
where the map ($\mathcal{M}$) is the set of all landmarks and is defined as

$$\mathcal{M} = \{\mathbf{l}_1, \mathbf{l}_2, \dots\} \tag{2.15}$$

Due to it's discrete nature, equation (2.14) can't be represented in such a way as to be solvable using the efficient methods typically used for SLAM. Furthermore, cyclic dependencies are introduced as seen in Figure 2.7. The most likely association is usually assumed to be known with certainty, yielding

$$p(E(\mathbf{n}_{z_i}|\mathbf{s}_{z_i}, \mathbf{z}_i, \mathcal{M})) = 1 \tag{2.16}$$

where $E()$ is the expectation. As more information is obtained, $E(\mathbf{n}_{z_i}|\mathbf{s}_{z_i}, \mathbf{z}_i, \mathcal{M})$ may change and it may be desirable to re-evaluate the association. For example, in Figure 2.15, a new landmark l_2 is discovered from pose s_4. In light of this information, it is found that a past observation from pose s_3 would have been a closer match to the new landmark l_2 than l_1. The association is thus changed and the solution is recomputed.



(a) New landmark observation (b) DA change for observation from s_3

Figure 2.15: In (a), a new landmark is discovered. In retrospect, the observation from s_3 matches more closely the newly discovered landmark and the DA is changed (b).

Finding such desirable DA changes can be computationally intensive on large maps and doing so is still an open problem in SLAM. In cases when the landmarks can be uniquely identified, such as with QR codes or RFID, this process is bypassed as the data association is given by the landmark itself in the form of a unique ID.

2.3.2.2 Probabilistic SLAM

As proposed by Murphy [22], the SLAM problem consists of finding the most likely vehicle path ($\mathbf{s}_{1:k}$) and landmark positions ($\mathcal{M}$) at time k given one or more measurements for each pose and landmark.

In the present case, the measurements consist of odometry observations ($\mathbf{u}_{1:k}$) at times 1 to k, n_z landmark observations ($\mathbf{z}_{1:n_z}$), n_z data associations ($\mathbf{n}_{1:n_z}$) and an initial pose ($\mathbf{s}_0$). Note that data associations are considered unknown for the next few equations for the sole purpose of illustrating how DA affects the SLAM problem. Compared to the case where DA is known, variables $\mathbf{n}_{1:n_z}$ are introduced and are part of the solution, along with $\mathbf{s}$ and $\mathbf{l}$.

In probabilistic terms, the solution to the SLAM problem is the values of $\mathbf{s}_{1:k}$, $\mathbf{n}_{1:n_z}$ and $\mathcal{M}$ that maximise

$$p(\mathbf{s}_{1:k}, \mathcal{M}, \mathbf{n}_{1:n_z} | \mathbf{z}_{1:n_z}, \mathbf{u}_{1:k}, \mathbf{s}_0) \tag{2.17}$$

As discussed in Section 2.3.2.1, (2.17) can not be solved efficiently due to the discrete nature of DA. Following the same methodology, it will thus be assumed that the data association probability in (2.14) is 1 for the most likely association as done in (2.16). The SLAM formulation of (2.17) then becomes

$$p(\mathbf{s}_{1:k}, \mathcal{M}|\mathbf{z}_{1:n_z}, \mathbf{u}_{1:k}, \mathbf{s}_0) \tag{2.18}$$

which is referred to in the literature as the full SLAM formulation. Using the theorem of total probability, and given that

1. the odometry measurement at time k, designated by $\mathbf{u}_k$ relates the pose at $k-1$ to the pose at time k

$$p(\mathbf{s}_k|\mathbf{s}_{k-1}, \mathbf{u}_k) \tag{2.19}$$

2. an arbitrary telemetry measurement $\mathbf{z}_i$ obtained at time k depends on the map, data association and position at that time

$$p(\mathbf{z}_i|\mathbf{s}_{z_i}, \mathbf{n}_{z_i}, \mathcal{M}) \tag{2.20}$$

   which, if the DA is known, can be simplified to

$$p(\mathbf{z}_i|\mathbf{s}_{z_i}, \mathbf{l}_{z_i}) \tag{2.21}$$

the full SLAM formulation of (2.18) can be factorised as

$$p(\mathbf{s}_0) \prod_{i=1}^{k} p(\mathbf{s}_i|\mathbf{s}_{i-1}, \mathbf{u}_i) \prod_{j=1}^{n_z} p(\mathbf{z}_j|\mathbf{s}_{z_j}, \mathbf{l}_{z_j}) \tag{2.22}$$

where the constant factor $p(\mathbf{s}_0)$ can be normalised out of the equation. The full SLAM formulation of (2.18) and its factorisation (2.22) are the basis of smoothing and mapping solutions to SLAM.

The result in (2.22) can also be obtained using the MRF representation of the problem as discussed in Section 2.3.1.5. By choosing the values of Φ_c to be the product of the probability densities represented by each edge related to nodes in the clique (ensuring edges are included only once), the normalisation constant Z in (2.5) is not needed. The probability distribution are selected as (2.19) and (2.21) for odometry and landmark observation edges, respectively. Taking for example the *Artificial_11* dataset and the clique elements described in Figure 2.14, the following cliques and functions Φ_c can be chosen:

- clique 1: $\{s_0, s_1\}$, $\Phi_1 = p(\mathbf{s}_1|\mathbf{s}_0, \mathbf{u}_1)p(\mathbf{s}_0)$
- clique 2: $\{s_2, s_3\}$, $\Phi_2 = p(\mathbf{s}_2|\mathbf{s}_1, \mathbf{u}_2)p(\mathbf{s}_3|\mathbf{s}_2, \mathbf{u}_3)$
- clique 3: $\{s_4\}$, $\Phi_3 = p(\mathbf{s}_4|\mathbf{s}_3, \mathbf{u}_4)$
- clique 4: $\{s_5, s_6, l_1\}$, $\Phi_4 = p(\mathbf{s}_5|\mathbf{s}_4, \mathbf{u}_5)p(\mathbf{s}_6|\mathbf{s}_5, \mathbf{u}_6)p(\mathbf{z}_1|\mathbf{s}_1, \mathbf{l}_1)p(\mathbf{z}_3|\mathbf{s}_5, \mathbf{l}_1)p(\mathbf{z}_4|\mathbf{s}_6, \mathbf{l}_1)$
- clique 5: $\{s_8, l_2\}$, $\Phi_5 = p(\mathbf{s}_8|\mathbf{s}_7, \mathbf{u}_8)p(\mathbf{z}_5|\mathbf{s}_8, \mathbf{l}_2)p(\mathbf{z}_2|\mathbf{s}_3, \mathbf{l}_2)$
- clique 6: $\{s_7\}$, $\Phi_6 = p(\mathbf{s}_7|\mathbf{s}_6, \mathbf{u}_7)$

Before efficient methods of solving (2.22) were known, the SLAM problem was stated as finding the most likely vehicle pose ($\mathbf{s}_k$) and landmark positions ($\mathcal{m}$) at time k given one or more measurement for each pose and landmark. As was the case in the full SLAM formulation, the measurements consist of odometry observations ($\mathbf{u}_{1:k}$) at times 1 to k, n_z landmark observations ($\mathbf{z}_{1:n_z}$), n_z data associations ($\mathbf{n}_{1:n_z}$) and an initial pose ($\mathbf{s}_0$). This differs from the full SLAM formulation in that only one vehicle pose is included in the solution. While the full SLAM formulation solved for the vehicle path ($\mathbf{s}_{1:k}$), here only the

last pose ($\mathbf{s}_k$) is considered of interest, the goal being to reduce the number of variables in the problem. The SLAM problem can then be stated as

$$p(\mathbf{s}_k, \mathcal{M}|\mathbf{z}_{1:n_z}, \mathbf{u}_{1:k}, \mathbf{n}_{1:n_z}, \mathbf{s}_0) \tag{2.23}$$

Assuming DA is certain, the observation equation is given in (2.21) and the SLAM formulation of (2.23) becomes

$$p(\mathbf{s}_k, \mathcal{M}|\mathbf{z}_{1:n_z}, \mathbf{u}_{1:k}, \mathbf{s}_0) \tag{2.24}$$

The SLAM formulation in (2.24) will be referred to in this document as classical SLAM. The difficulties encountered with this formulation can be easily visualised using a MRF. For example, starting with the MRF of the *Artificial_11* dataset, the MRF represented in the form of (2.24) can be obtained by consecutively eliminating (marginalising out) the previous position nodes. This process of removing (eliminating) a node is done by removing the node and its edges from the graph and adding edges between all neighbours of the eliminated node (see Section 2.3.1.6). The resulting MRF can be seen in Figure 2.16 where grey nodes and edges are those that have have been removed from the graph while dotted lines represent edges have been added by the elimination process previously described. For clarity, edges created and subsequently removed by the elimination process described below (such as edge s_2 - l_1 added in step 2 and removed in 3) are not shown. The graph of Figure 2.16 is obtained by eliminating pose nodes in ascending order as described in the following process:

1. The elimination of s_0 removes the edge s_0 - s_1.
2. The elimination of s_1 removes the edge s_1 - s_2 but adds the edge s_2 - l_1.
3. The elimination of s_2 removes the edges s_2 - s_3 and s_2 - l_1 but adds the edge s_3 - l_1.
4. The elimination of s_3 removes the edges s_3 - s_4, s_3 - l_1 and s_3 - l_2 but adds the edges s_4 - l_1, s_4 - l_2 and l_1 - l_2.

5. The elimination of s_4 removes the edges s_4 - s_5, s_4 - l_1 and s_4 - l_2 but adds the edge s_5 - l_2.

6. The elimination of s_5 removes the edges s_5 - s_6, s_5 - l_1 and s_5 - l_2 but adds the edge s_6 - l_2.

7. The elimination of s_6 removes the edges s_6 - s_7, s_6 - l_1 and s_6 - l_2 but adds the edges s_7 - l_1 and s_7 - l_2.

8. The elimination of s_7 removes the edges s_7 - s_8, s_7 - l_1 and s_7 - l_2 but adds the edge s_8 - l_1.

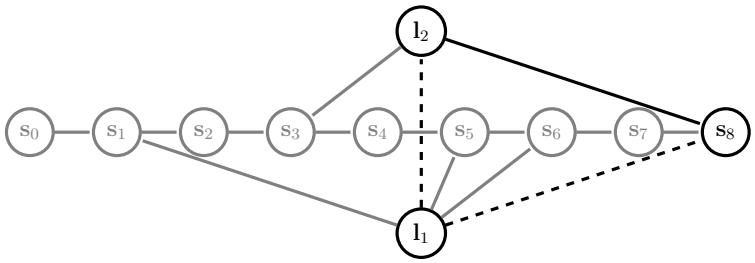


Figure 2.16: Sequential marginalisation on the MRF of *Artificial_11* . Grey nodes and edges represent marginalised information while dotted edges are added by the marginalisation process. Edges that have been added and subsequently removed are not shown.

Note that after marginalising past poses, each of the nodes remaining are connected to all the other nodes of the graph. In the field of graph theory, this is called a complete graph. While this does not constitute a problem with a dataset containing two landmarks such as *Artificial_11*, the number of edges in the complete graph obtained grows with the square of the number of landmarks. Furthermore, due to the absence of past poses (s_0 to s_7) and associated measurements (edges) in the solution, this elimination process must be restarted from the beginning every time a new measurement having repercussions on one of these eliminated nodes is added. Combining equations (2.19), (2.21) and (2.24) using

Bayes' rule and the theorem of total probability lead to the Bayes filter formulation,

$$p(\mathbf{s}_k, \mathcal{M}|\mathbf{u}_{1:k}, \mathbf{z}_{1:n_z}, \mathbf{z}_{s_k}, \mathbf{s}_0) = \frac{p(\mathbf{z}_{s_k}|\mathbf{s}_k, \mathcal{M})}{p(\mathbf{z}_{s_k})} \int p(\mathbf{s}_k|\mathbf{s}_{k-1}, \mathbf{u}_k) p(\mathbf{s}_{k-1}, \mathcal{M}|\mathbf{u}_{1:k-1}, \mathbf{z}_{0:n_z}, \mathbf{s}_0) d\mathbf{s}_{k-1} \quad (2.25)$$

which provide a recursive way of updating the last pose and map. This equation, with the assumption of Gaussian distribution, is the basis of filtering based solutions to SLAM. Note that in this case, the previous inputs and observations are not needed, which reduces memory usage. This, however, prevents changing the data association because the poses from which the landmarks were observed and the measurements values obtained have been eliminated from the equations (marginalised out of the MRF). Also note that the recursive nature of the algorithm renders the complexity of the solution independent of the number of observations but the problem still grows with the number of landmarks. This has pushed researchers to find alternative solutions to the SLAM problem.

Although the classical SLAM formulation requires to solve for the landmarks and last pose only (as seen in Figure 2.16), recent advances in space linear algebra are such that the large but sparsely connected MRF of the full SLAM formulation with all poses and landmarks (as seen in Figure 2.9) have become easier to solve and modify. To illustrate this, the sparsity pattern of the information matrix associated with the MRF of *Victoria_Park_500* is shown in Figure 2.17. With a size of 500×500, it is significantly larger than the 38×38 Information matrix obtained when past poses are marginalised but it contains only 1181 entries compared to 1444 entries for a dense 38×38 matrix. Large systems encountered in most practical SLAM applications are computationally challenging. However, for maps with more than 350 landmark, Full SLAM implementations have been found to be faster than implementations derived from the Bayes filter [23]. The full SLAM formulation also has the advantage that, since a history of all information is kept, past observations and data associations can be changed.

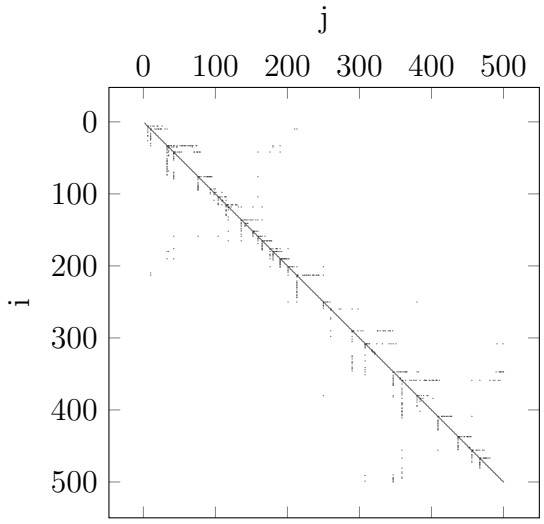


Figure 2.17: Information matrix of the MRF of *Victoria_Park_500*. Non-zero entries are represented by black squares.

2.4 Cholesky Factorisation

Cholesky factorisation is the matrix decomposition of a real positive definite matrix $\mathbf{C}$ in two factors such that $\mathbf{C} = \mathbf{L}\mathbf{L}^T$ where $\mathbf{L}$ is lower triangular. This factorisation is popular in linear algebra and can also be used in SLAM to decompose the information matrix of a MRF. A system in the form of

$$\mathbf{C}\mathbf{x} = \mathbf{b} \tag{2.26}$$

with $\mathbf{C}$ a positive definite matrix, can be expressed as

$$\mathbf{L}\mathbf{L}^T\mathbf{x} = \mathbf{b} \tag{2.27}$$

and solved efficiently by forward and backward substitution in $\mathbf{L}\mathbf{y} = \mathbf{b}$ and $\mathbf{L}^T\mathbf{x} = \mathbf{y}$ respectively. An example of (2.26) and (2.27) using the information matrix associated with the MRF of *Artificial_11* can be seen in Figure 2.18 where non-zero entries are represented by black squares.

Multiple methods exist to obtain the factorisation itself depending, amongst other things, on the memory representation used for the matrices (compressed column or com-

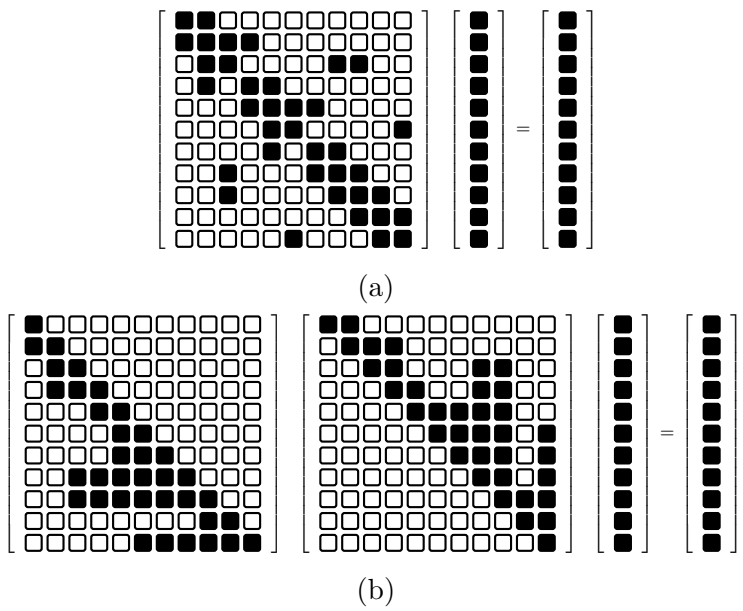


Figure 2.18: An example of (a) the system $\mathbf{Cx} = \mathbf{b}$ and (b) $\mathbf{LL}^T\mathbf{x} = \mathbf{b}$ using the MRF information matrix of *Artificial_11*. White squares indicate zero entries while non-zero entries are represented by black squares.

pressed row). The formulation most readers are familiar with is presented in Algorithm 2.1 and illustrated in Figure 2.19. In this figure, the process begins with a positive definite matrix $\mathbf{C}$ and an empty factor $\mathbf{L}$. In the first step, the lower triangular part of a column of $\mathbf{L}$ is calculated from the matching column in $\mathbf{C}$. The second step consists of propagating changes to the remaining columns of $\mathbf{C}$. The first and second steps are repeated until all the columns of $\mathbf{C}$ have a corresponding entry in $\mathbf{L}$. This calculates $\mathbf{L}$ by columns.

Algorithm 2.1 Cholesky Factorisation

Input:
$\mathbf{C}_{n\times n}$ is positive definite
Output:
$\mathbf{L}$ is the Cholesky factor of $\mathbf{C}$

1: $\mathbf{L} \leftarrow \mathbf{0}_{n\times n}$
2: **for** $i \leftarrow 1 : n$ **do**
3: $\quad a \leftarrow \mathbf{C}(i, i)$
4: $\quad \mathbf{b} \leftarrow \mathbf{C}(i+1 : n, i)$
5: $\quad \mathbf{B} \leftarrow \mathbf{C}(i+1 : n, i+1 : n)$
6: $\quad \mathbf{L}(i, i) \leftarrow \sqrt{a}$
7: $\quad \mathbf{L}(i+1 : n, i) \leftarrow \frac{\mathbf{b}}{\sqrt{a}}$
8: $\quad \mathbf{C}(i+1 : n, i+1 : n) \leftarrow \mathbf{B} - \frac{\mathbf{b}\mathbf{b}^T}{a}$
9: **end for**

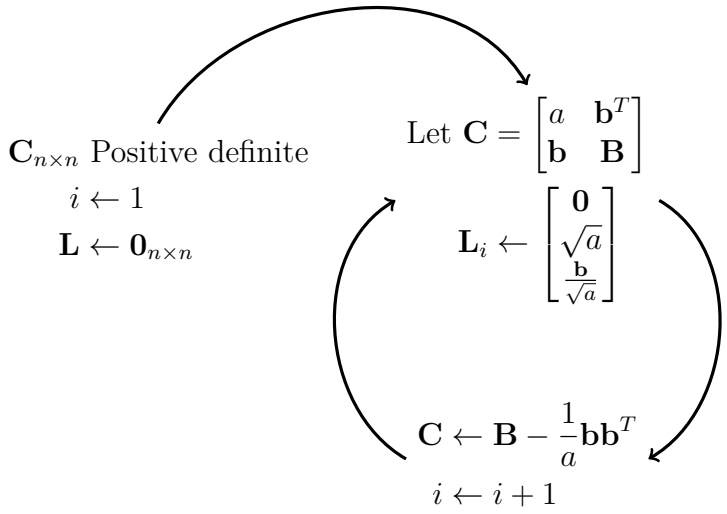


Figure 2.19: Cholesky factorisation of $\mathbf{C}$. Initially, factor $\mathbf{L}$ is empty and column index i is initialised to 1. First step: A column of $\mathbf{L}$ is calculated from the matching column in $\mathbf{C}$. Second step: propagate changes to the remaining columns of $\mathbf{C}$. Repeat first and second steps until all the columns of $\mathbf{C}$ have a corresponding entry in $\mathbf{L}$

2.5 Elimination Tree

From Algorithm 2.1 and Figure 2.19 it can be seen that calculating a given column of $\mathbf{L}$ will affect the value of subsequent columns due to $\mathbf{b}\mathbf{b}^T$. In the case where the Cholesky factor (now denoted $\mathcal{L}$) is sparse, the matrix formed by $\mathbf{b}\mathbf{b}^T$ will be sparse as well and only columns corresponding to non-zero values of $\mathbf{b}$ will be affected. These columns are said to be dependent on the column currently being calculated by Algorithm 2.1. The elimination tree is a structure that stores this dependency information in an easily accessible hierarchical format. It will become apparent in the next sections that elimination trees are key components of efficient sparse Cholesky factorisation (Section 2.6) and factor modification (Section 2.6.1), in which they are used to identify column dependence as well as to find sets of columns affected by certain operations. They are also an integral part of many algorithms proposed in this document. Elimination trees and their basic properties will be presented here for convenience using the notation used by Davis and Hager [24] whenever possible.

The elimination tree is often represented by the function $\pi(j)$, called the parent function. This function returns the parent of a given node j or, in mathematical terms, the lowest column index that depends on column j. The elimination tree is defined as

$$\pi(j) = min\_index(\mathcal{L}_j \setminus \{j\}), \tag{2.28}$$

where $\mathcal{L}_j$ is column j of $\mathcal{L}$. In other words, the parent of a given column in a factor $\mathcal{L}$ is identified by the position of the first non-zero off-diagonal entry of this column. Similarly, the parent to children relation is characterised by the children multifunction $\pi^{-1}(k)$. This represents the set of all children of k or, alternatively, the set of columns on which k depends. The children multifunction is defined as

$$\pi^{-1}(k) = \{j | \pi(j) = k\}. \tag{2.29}$$

In the case of the *Artificial_11* dataset and the Cholesky factor of its information matrix, presented in Figure 2.18, the elimination tree forms a non-branched path. A factor $\mathcal{L}$ forming a more interesting example is shown in Figure 2.20 with the associated elimination tree. Node 11 is the only node without a parent, called the root in graph terminology, because it is associated with the rightmost column. Other nodes are children of the node corresponding to the lowest index off-diagonal non-zero entry of their respective columns (in red). For example, 7 is a child of 8 while 6 and 8 are children of 9.

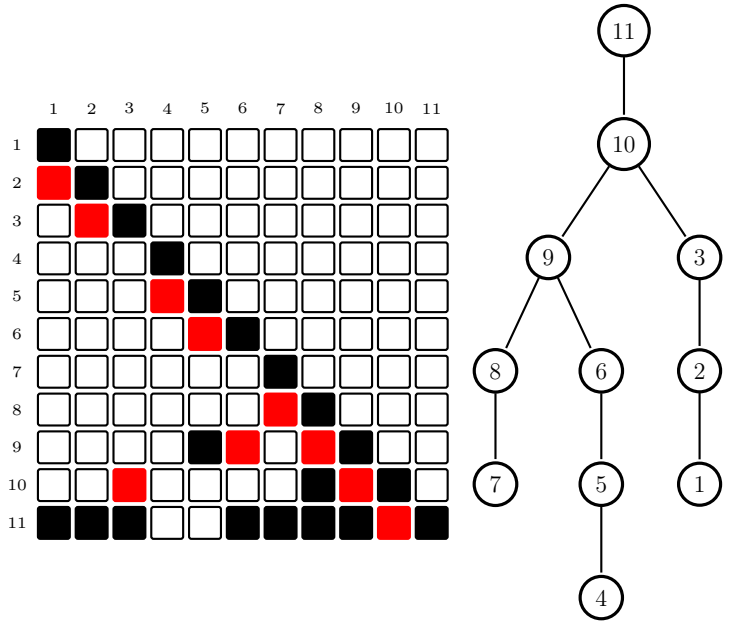


Figure 2.20: A Cholesky factor and the associated elimination tree. Node 11 is the root because it represents the rightmost column. Other nodes are children of the lowest index off-diagonal non-zero entry of their respective columns (in red).

The set of nodes between a given node x and the root is termed "path" and it consists of the recursive sequence of parents

$$Path(x) = \{x, \pi(x), \pi(\pi(x)), \dots\}. \tag{2.30}$$

For example, the path of node 3 in the elimination tree of Figure 2.20 is $Path(3) = \{3, 10, 11\}$. In matrix terms, $Path(x)$ is the set of columns that are affected by changes to column x. For example, consider the Cholesky factor of Figure 2.20 but where element 11

of column 3 in factor $\mathcal{L}$, written as $\mathcal{L}_3(11)$, is modified due to a change in $\mathcal{C}$, the sparse version of $\mathbf{C}$). The resulting repercussions on the other columns of $\mathcal{L}$ is illustrated in Figure 2.21, where it can be seen that a change in $\mathcal{L}_3(11)$ (corresponding to a change in $\mathcal{C}_3(11)$) affects the matrix $\mathbf{b}\mathbf{b}^T$ and propagates to entries (in red) of related columns while applying Algorithm 2.1. Notice that the columns affected are those in $Path(3)$. This will

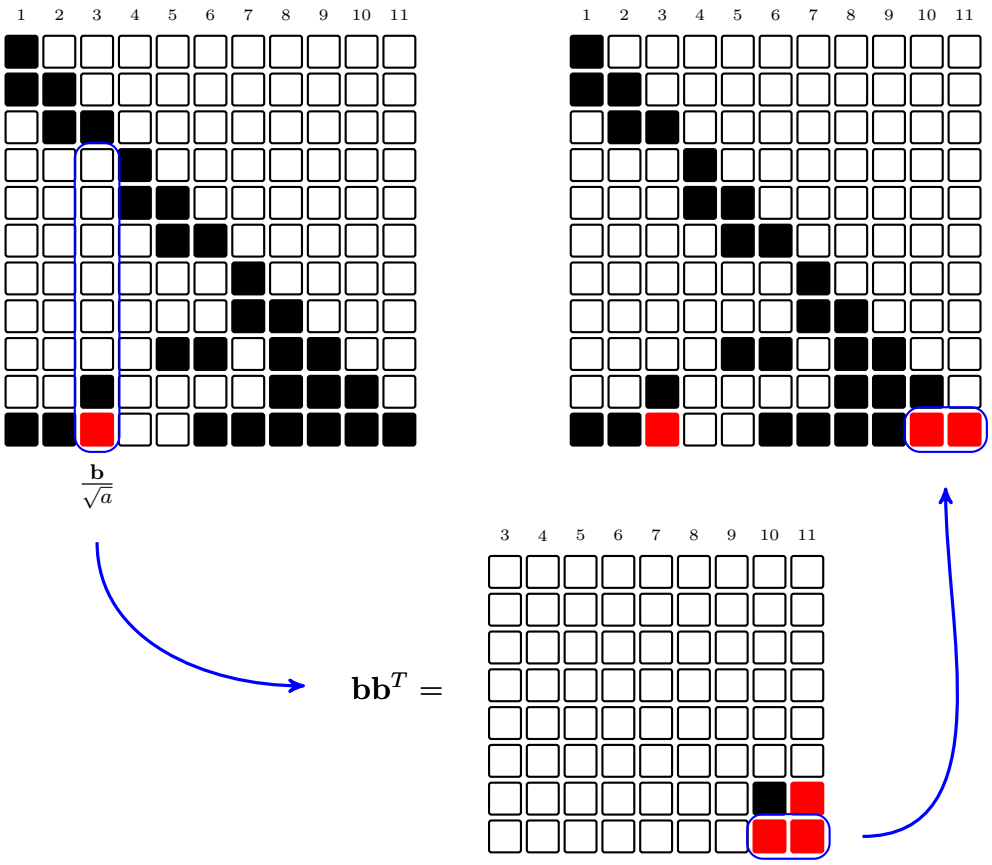


Figure 2.21: A change in $\mathcal{L}_3(11)$ (left) corresponding to a change in $\mathcal{C}_3(11)$ affects the matrix $\mathbf{b}\mathbf{b}^T$ (below) and propagates to related columns (right). Matrix elements that change are coloured in red.

be explained in greater detail in Section 2.6. For further information on the elimination tree, the reader is referred to [25].

2.6 Sparse Cholesky Factorisation

In Section 2.3.1.4, the concept of representing an MRF by its information matrix was introduced. More specifically, the sparseness of SLAM information matrices and how the use of a sparse matrix format led to memory savings was illustrated with *Artificial_11* in Figure 2.11 and discussed. An example of the sparsity matrix of *Victoria_Park_500* can be seen in Figure 2.17. Sparse Cholesky factorisation has been proposed to factorise such sparse matrices efficiently.

First used to increase the efficiency of solving finite element mesh-based problems [26, 27], sparse Cholesky decomposition is now used in all fields where efficient decomposition of large sparse matrices are required. The execution of a sparse Cholesky factorisation is done in three parts: 1) Obtaining an elimination tree (see Section 2.5); 2) executing the symbolic Cholesky factorisation (Algorithm 2.2); and, 3) calculating the factor $\mathcal{L}$. The reduction in execution time provided by sparse Cholesky factorisation lies in the symbolic factorisation step, which uses the elimination tree to determine which entries in the matrix will be non-zero. The calculations are then performed for these entries only, taking the complexity of the operation from $O(n^3/2)$ down to

$$O\left(\frac{1}{2}\sum_{i=1:n}(|\mathcal{L}_i|-1)(|\mathcal{L}_i|+2)\right), \tag{2.31}$$

where n is the number of columns in $\mathcal{L}$ and $|\mathcal{L}_i|$ is the number of non-zero entries in column i of $\mathcal{L}$ [25]. The use of symbolic Factorisation also yields improvements regarding the storage of the factor $\mathcal{L}$. Since the number of non-zero entries it contains is calculated before any factorisation takes place, the exact storage space required can be reserved.

For the purpose of this thesis, it is essential that the sparse factorisation keeps track of the multiplicity of each non-zero element where the multiplicity of a given element is the number of related elements contributing to it. The sparse factorisation using multisets (a

set that can contain multiple times the same value) is defined here with the same notation as Davis *et al.* [24], with the sharp (♯) superscript identifying a multiset. For example, a given Cholesky factor $\mathcal{L}^\sharp$ is a sparse matrix where each element (i, j) consists of a pair of values of the form $(\mathcal{L}(i, j), Multiplicity)$. When sparse Cholesky decomposition is executed on sparse matrix $\mathcal{C}$ (as in [25]) the following algorithm is obtained:

Algorithm 2.2 Symbolic Cholesky Factorisation

Input:
$\mathcal{C}$ is positive definite
Output:
$\mathcal{L}^\sharp$ is the Cholesky factor with multiplicity
π is the elimination tree

1: $\pi(j) \leftarrow \mathbf{0}_{1\times n}$
2: $\mathcal{L}_j^\sharp \leftarrow \{(i, 1)|i \in \mathcal{C}_j\}$
3: **for** $j \leftarrow 1 : n$ **do**
4: $\quad \mathcal{L}_j^\sharp \leftarrow \mathcal{L}_j^\sharp + \left(\sum_{i\in\pi^{-1}(j)} \mathcal{L}_i \setminus \{i\} \right)$
5: $\quad \pi(j) \leftarrow \min(\mathcal{L}_j \setminus \{j\})$
6: **end for**

Note that only the lower triangular part of $\mathcal{C}$ is taken into account since the matrix is symmetric. Furthermore, while this algorithm includes the calculation of the elimination tree, it is usually pre-calculated for practical implementations such as the csparse library [28]. An illustration of sparse Cholesky decomposition of the information matrix corresponding to the factor in Figure 2.20 is shown in Figure 2.22. The non zero values are identified by a thick border and multiplicity number, while solid dark cells are unused due to the symmetry of $\mathcal{C}$. This figure illustrates how the multiplicity of the non-zero values of column 9 (centre) is obtained by counting, for each row, the number of corresponding non-zero entries in $\mathcal{C}$ (right) and in child columns of the partially computed $\mathcal{L}^\sharp$ (left). The elimination tree (far right) identifies the node 9 in red and its children in blue. In recent years, sparse Cholesky decomposition has been extended to super-nodal sparse Cholesky factorisation [29] and graphs [30] for parallel computation while recent articles have detailed GPU

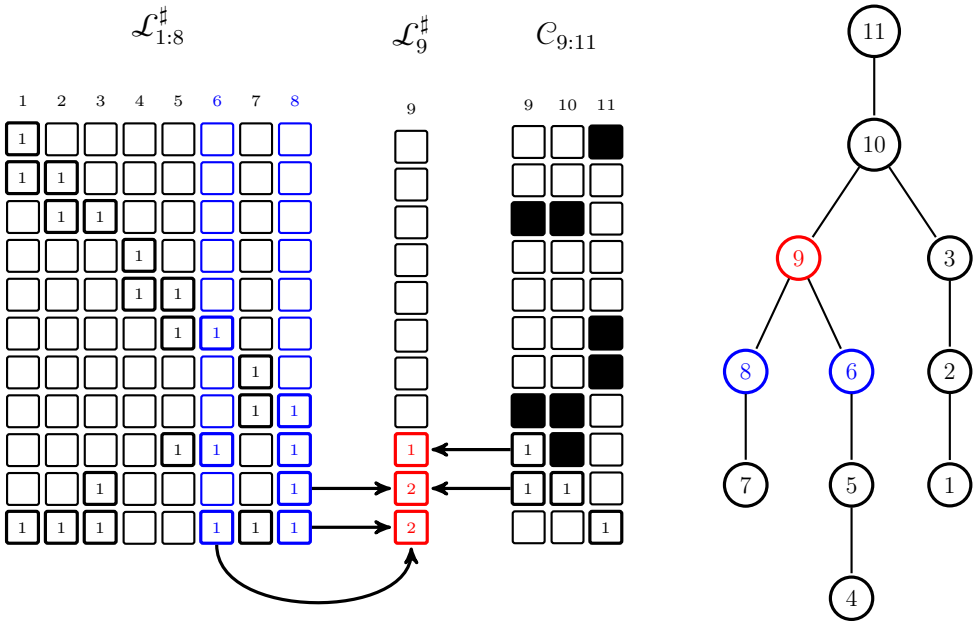


Figure 2.22: The non zero values are identified by a thick border and multiplicity number, while solid dark cells are unused. The multiplicity of the non-zero values of column 9 (centre) is obtained by counting, for each row, the number of corresponding non-zero entries in $\mathcal{C}$ (right) and in child columns of the partially computed $\mathcal{L}^{\sharp}$ (left). The elimination tree (far right) identifies the node 9 in red and it's children in blue

implementations [31, 32]. Out-of-core methods adapted to very large problems have also been proposed [33].

2.6.1 Factor modification

When a change in matrix $\mathcal{C}$ occurs, multiple methods have been proposed to update the factor $\mathcal{L}$ at a fraction of the computational cost of recalculating the Cholesky decomposition. In this section, we will introduce these algorithms for the reader's convenience. For consistency, the sparse matrix notation is used in this section however, the results presented are applicable to both dense and sparse matrices.

2.6.1.1 Row deletion

The row deletion operation described by Davis *et al.* [34] can be used to update the factor $\mathcal{L}$ in the event a row is deleted from the matrix $\mathcal{A}$, where $\mathcal{C} = \mathcal{A}^T\mathcal{A}$. If the structure of the matrices are given by

$$\mathcal{C} = \begin{bmatrix} \mathcal{C}_{11} & c_{12} & \mathcal{C}_{13} \\ c_{12}^T & c_{22} & c_{32}^T \\ \mathcal{C}_{31} & c_{32} & \mathcal{C}_{33} \end{bmatrix} \qquad \mathcal{L} = \begin{bmatrix} \mathcal{L}_{11} & & \\ \ell_{12}^T & l_{22} & \\ \mathcal{L}_{31} & \ell_{32} & \mathcal{L}_{33} \end{bmatrix} \tag{2.32}$$

where cursive upper case letters represent sparse matrices, cursive lower case represent sparse vertical vectors and regular font letters represent scalars. The removal of the second row and column gives the modified matrices $\bar{\mathcal{C}}$ and $\bar{\mathcal{L}}$ with the following structure:

$$\bar{\mathcal{C}} = \begin{bmatrix} \mathcal{C}_{11} & \mathcal{C}_{13} \\ \mathcal{C}_{31} & \mathcal{C}_{33} \end{bmatrix} \qquad \bar{\mathcal{L}} = \begin{bmatrix} \mathcal{L}_{11} & \\ \mathcal{L}_{31} & \bar{\mathcal{L}}_{33} \end{bmatrix} \tag{2.33}$$

$$\tag{2.34}$$

and the sub-matrix $\bar{\mathcal{L}}_{33}$ is obtained by

$$\bar{\mathcal{L}}_{33}\bar{\mathcal{L}}_{33}^T = \mathcal{L}_{33}\mathcal{L}_{33}^T + \ell_{32}\ell_{32}^T \tag{2.35}$$

Note that (2.35) is equivalent to a rank-1 update of $\mathcal{C}_{33} = \mathcal{L}_{33}\mathcal{L}_{33}^T$ where a rank-1 update refers to the addition of a matrix of rank equal to one, such as $\ell_{32}\ell_{32}^T$, to an existing matrix.

2.6.1.2 Row addition

The row addition operation, also described by Davis *et al.* [34] can be used to update the factor $\mathcal{L}$ in the event a row is added to the matrix $\mathcal{A}$. If the structure of the matrices are

given by

$$\mathcal{C} = \begin{bmatrix} \mathcal{C}_{11} & \mathcal{C}_{13} \\ \mathcal{C}_{31} & \mathcal{C}_{33} \end{bmatrix} \qquad \mathcal{L} = \begin{bmatrix} \mathcal{L}_{11} & \\ \mathcal{L}_{31} & \mathcal{L}_{33} \end{bmatrix} \tag{2.36}$$

The addition of a row and column between the given block matrices gives the modified matrices $\bar{\mathcal{C}}$ and $\bar{\mathcal{L}}$ whose block structure is

$$\bar{\mathcal{C}} = \begin{bmatrix} \mathcal{C}_{11} & \bar{c}_{12} & \mathcal{C}_{13} \\ \bar{c}_{12}^T & \bar{c}_{22} & \bar{c}_{32}^T \\ \mathcal{C}_{31} & \bar{c}_{32} & \mathcal{C}_{33} \end{bmatrix} \qquad \bar{\mathcal{L}} = \begin{bmatrix} \mathcal{L}_{11} & & \\ \bar{\ell}_{12}^T & \bar{l}_{22} & \\ \mathcal{L}_{31} & \bar{\ell}_{32} & \bar{\mathcal{L}}_{33} \end{bmatrix} \tag{2.37}$$

where $\bar{\ell}_{12}$ is found by solving $\mathcal{L}_{11}\bar{\ell}_{12} = \bar{c}_{12}$ and

$$\bar{l}_{22} = \sqrt{\bar{c}_{22} - \bar{\ell}_{12}^T \bar{\ell}_{12}} \tag{2.38}$$

$$\bar{\ell}_{32} = \frac{\bar{c}_{32} - \mathcal{L}_{31}\bar{\ell}_{12}}{\bar{l}_{22}} \tag{2.39}$$

$$\bar{\mathcal{L}}_{33}\bar{\mathcal{L}}_{33}^T = \mathcal{L}_{33}\mathcal{L}_{33}^T - \bar{\ell}_{32}\bar{\ell}_{32}^T \tag{2.40}$$

Note that (2.40) corresponds to a rank-1 downdate, which is similar to the rank-1 update but sees the matrix of rank equal to one subtracted from the original matrix instead of added. For a more detailed description as well as the complete proof of these operations, the reader is referred to [34].

2.6.2 Column Ordering

It is well known that the order of the columns affects the efficiency of solving linear systems. While the process is not exactly the same as solving the normal equation of (2.4), a simple linear algebra example is used to illustrate the concept.

Suppose that we have the following system of linear equations. This system can be solved using two different orders of execution.

1. $w = 1$
2. $x = 2w$
3. $y = 2x$
4. $z = 2x + 3y - 4w$

In the first case, the problem is solved using the intuitive order of 1, 2, 3 and 4.	In the second case, the problem is solved in the reverse order: 4, 3, 2 and 1.
From equation 1, $w = 1$	From equation 4, $y = \frac{z-2x+4w}{3}$
From equation 2, $x = 2w = 2$	Substituting in equation 3, $\frac{z-2x+4w}{3} = 2x$ thus $x = \frac{z+4w}{8}$
From equation 3, $y = 2x = 4$	Substituting in equation 2, $\frac{z+4w}{8} = 2w$ thus $w = \frac{z}{12}$
From equation 4, $z = 2x + 3y - 4w = 12$	Substituting in equation 1, $\frac{z}{12} = 1$ thus, $z = 12$
	back substituting in the intermediate results yields $y = 4$, $x = 2$

Note that in the second case, much more variables are present in the intermediate equations. In fact, each step has the maximum number of variables possible. These extra variables are analogous to the fill-ins, which are carried from one equation (column) to the next until they are eliminated. The analogous Cholesky factors to these situations is illustrated in Figure 2.23 where it can be seen that the matrix associated with the second case has more non-zero values.

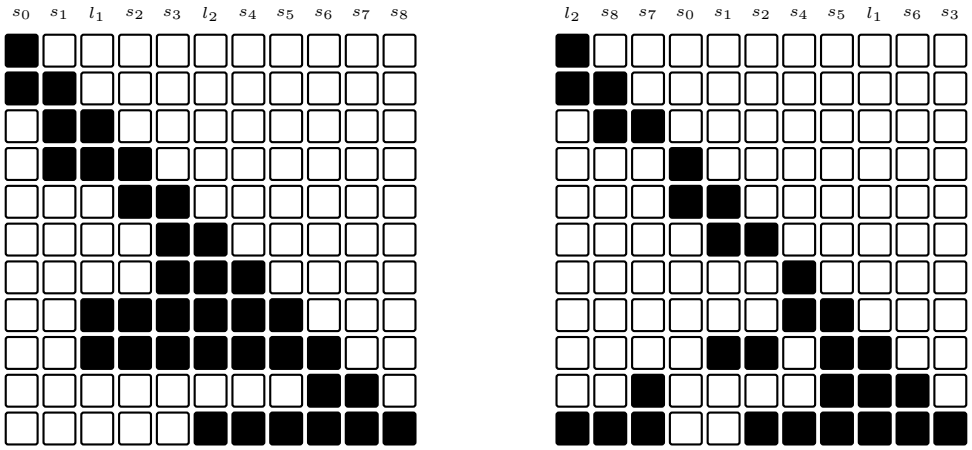
(a) case 1 (b) case 2

Figure 2.23: Example of Cholesky factors for the two cases. Case 2 (b) has more non-zero values than case 1 (a).

For simple problems, the optimal ordering can be easily discovered but this is not the case for very large problems. In fact, calculating an optimal ordering is NP-Complete [35]. Some algorithms, however, can be used to efficiently find a favourable ordering. An evaluation of various ordering strategies applied to SLAM can be found in [36]. Most of these algorithms can be classified in two categories: those based on minimum degree and those based on nested dissection. In this section, these two types of algorithms are briefly introduced and the relevant literature is presented. First, the Minimum Degree approach is presented using COLAMD as an example followed by the Nested Dissection based algorithm METIS.

2.6.2.1 Minimum Degree

Minimum degree based column ordering algorithms are currently the most popular choice for SLAM due to their efficiency and low computational cost. The original minimum degree algorithm consists of consecutively eliminating nodes with the lowest number of connected edges [37, 38]. Since maintaining the edge count is the most computationally expensive part of the algorithm, an approximate method with similar results was proposed by Amestoy *et al.* [39]. This method, called column approximate minimum Degree (COLAMD), is currently used in state of the art SLAM implementations [40]. As an example, the Cholesky factors of matrices representing the MRF of *Artificial_11* with sequential column ordering and COLAMD column ordering are shown in Figure 2.24. The factor where the ordering is calculated with COLAMD has 30 non-zero entries while the factor with sequential ordering has 36 non-zero entries.



(a) Sequential ordering (b) COLAMD ordering

Figure 2.24: Comparison between the Cholesky factor of two matrices representing the MRF of *Artificial_11*. In (a), the columns are ordered sequentially and 36 non-zero entries are present. In (b) they are ordered with COLAMD and 30 non-zero entries are present.

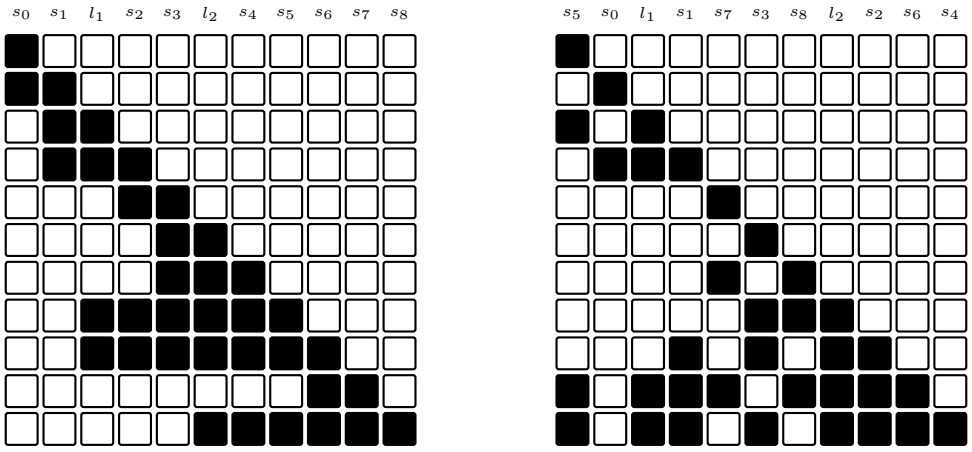
The structure of the Cholesky factors presented in Figure 2.24 is obtained by marginalizing the nodes of the graph in a particular order. For every node eliminated, the corresponding column indicates its edges to other remaining nodes. Figure 2.24a can be obtained by following the marginalisation process discussed in Figure 2.16 and associated instructions. For Figure 2.24b, a similar process would be followed but the order in which the nodes are eliminated will be such that the ones with the lowest degree are done fist.

2.6.2.2 Nested Dissection and METIS

Nested dissection is defined as the process of recursively partitioning a graph into subgraphs, where nodes at the intersection are called separators. It was first used by George [41] for column ordering on square grid problems and was later generalised to planar and almost planar grids [42]. Nested dissection is also used as a basis for the METIS algorithm as described by Karypis *et al.* [43] which was later expanded with a parallel [44] and a multi-threaded implementation [45].

METIS consists of three phases: coarsening, partitioning and uncoarsening. During the coarsening phase, vertices(nodes) of the graph are merged to obtain a graph with roughly

half the size of the original one. Each nodes resulting from the merging operation has a weight equal to the number of base nodes that were merged to create it while the weight of the edges are updated similarly. This step is repeated until the graph is sufficiently small (Less than 100 nodes [46]). The next step consists of partitioning the coarse graph into two similarly sized sets while minimizing the number of separator nodes. The graph is then gradually expanded during the uncoarsening phase and the separator is further refined. The end result is a set of nodes (the separator) that divide the rest of the graph in two.When used to calculate orderings, each of the two partitions are ordered prior to the separator. The METIS algorithm is applied recursively on the partitions until a minimum size is obtained. At this point, another algorithm such as COLAMD is used to order the smaller sub-graphs. For example, the Cholesky factors of matrices representing the MRF of *Artificial_11* with sequential column ordering and METIS column ordering are shown in Figure 2.25. The factor where the ordering is calculated with METIS has 34 non-zero entries while the factor with sequential ordering has 36 non-zero entries.



(a) Chronological ordering (b) METIS ordering

Figure 2.25: Comparison between the Cholesky factor of two matrices representing the MRF of *Artificial_11*. In (a), the columns are ordered sequentially and 36 non-zero entries are present. In (b), they are ordered with METIS and 34 non-zero entries are present.

The structure of the Cholesky factors presented in Figure 2.25a can be obtained by following the marginalisation process discussed in Figure 2.16 and associated instructions.

For Figure 2.25b, the nodes are selected such that nodes eliminated last divide the graph in nearly equal sections.

For the large matrices encountered in practice, METIS typically returns orderings comparable to those obtained with COLAMD in terms of the number of non-zeros in the Cholesky factors. Furthermore, METIS yields elimination trees that are usually more balanced and hence more suited for parallel processing.

2.7 Summary

In this chapter, it was found that the SLAM problem can be represented in terms of a sparsely connected Markov random field. In turn, it has been demonstrated that the information matrix representation of the MRF can be stored efficiently as a sparse matrix.

The introduction of sparse linear algebra, in particular sparse Cholesky factorisation, was shown to be a turning point in the field of SLAM, allowing the MRF's information matrix to be decomposed and the SLAM problem solved faster than with classical SLAM methods.

Column reordering was demonstrated to have a big impact on the size of the information matrix's Cholesky factor, with both COLAMD and METIS performing better than the typical sequential ordering. The concept of factor modification was viewed as a promising avenue for future SLAM capabilities.

In the next chapter, a broad review of the methods used in the literature to address the SLAM problem using the classical SLAM and full SLAM formulations will be presented.

Chapter 3

Literature Review

This chapter contains a comprehensive review of the literature relevant to SLAM. Various algorithms are analysed and discussed to give the reader an insight on the different strategies that have been developed over the years to solve the SLAM problem. Throughout this chapter, the *Artificial_11* dataset described in Section 2.2 is used to illustrate the various SLAM algorithms.

In Section 3.1, the basic SLAM algorithms are presented. The first algorithm discussed is the extended Kalman filter, followed by two other popular filter based algorithms: the unscented Kalman filter and the extended information filter. Algorithms based on the full SLAM formulation are introduced starting with the particle filter, followed by genetic algorithms and smoothing methods.

In Section 3.2, various extensions designed to solve problems encountered by the basic SLAM algorithms are presented. The submap methods used to map large environments are presented followed by methods addressing the issue of non-linearities in the parametrisation. Solutions to problems encountered when using multiple robots are then discussed along with methods dealing with errors in data association. Methods developed specifically for mapping dynamic environment are also presented.

In Section 3.3, various algorithms and extensions are summarised and compared with regards to a number of properties. Finally, Section 3.4 positions the contribution of this thesis within the literature.

3.1 Basic SLAM algorithms

In this section, the most common and most efficient methods used to solve the SLAM problem are discussed. The literature regarding specific implementations of these methods is reviewed with the help of applied examples and descriptive figures.

3.1.1 Extended Kalman Filter

In the state space representation of the SLAM problem, the vehicle pose, landmark positions and control inputs are represented in a state vector ($\mathbf{x}$) and input vector ($\mathbf{u}$). As the actual input to the vehicle (steering, throttle,...) that led to these poses are often unavailable or hard to model, an odometry or inertial sensor is used to obtain the effect of vehicle movement on the states and produce the input vector. The partial derivatives of a known state transition function ($f(\mathbf{x}, \mathbf{u})$) are calculated with regards to the states to obtain the Jacobian matrix of the system ($\mathbf{A}$) and give a first order approximation of the behaviour of the vehicle. The expected telemetry measurement ($\mathbf{y}$) can be calculated from the states using the Jacobian matrix ($\mathbf{C}$) of an observation function ($h(\mathbf{x})$). The state space equations of the system at time k are

$$\dot{\mathbf{x}}_k = \mathbf{A}\mathbf{x}_k + \mathbf{u}_k \tag{3.1a}$$

$$\mathbf{y}_k = \mathbf{C}\mathbf{x}_k \tag{3.1b}$$

In most cases, Kalman filter based methods make use of the extended Kalman filter (EKF) to keep track of the landmark positions and last vehicle pose. The EKF is an

efficient implementation of (2.25) obtained by assuming each random variable is completely characterised by its statistical moments up to the second order (Gaussian distributed) and that the linearised state transition and observation model is a reasonable approximation of the true model. This takes advantage of the properties of linear operations on Gaussian random variables to compute the integral in (2.25) incrementally and in closed form, allowing the EKF to solve the SLAM problem very efficiently for online applications.

The EKF is a least square estimator of the linear system represented by (3.1a) and (3.1b), taking in consideration the sensor noise ($\mathbf{R}$) and process noise ($\mathbf{Q}$). The vehicle pose and landmark positions at time k are predicted by

$$\mathbf{x}_{k|k-1} = f(\mathbf{x}_{k-1|k-1}, \mathbf{u}_{k-1}) \tag{3.2}$$

$$\mathbf{P}_{k|k-1} = \mathbf{F}_k \mathbf{P}_{k-1|k-1} \mathbf{F}_k^T + \mathbf{Q}_k \tag{3.3}$$

where $\mathbf{F} = \mathbf{I} + \mathbf{A}$ is the state transition matrix and $\mathbf{P}$ represents the state covariance matrix. The subscript $k|k-1$ indicates a variable is calculated at time k given the information at time $k-1$ (estimated value) while $k-1|k-1$ indicates the previous value of the variable. These equations, referred to as the EKF *predict* phase, account for the effect of the control inputs and elapsed time on the states and their covariance relations. Equations (3.2) and (3.3) represent an implementation of a dead reckoning navigation system where odometry measurements are accumulated over time to keep track of the increasingly uncertain vehicle pose. The sensor measurements are included in the filter by

$$\mathbf{e}_k = \mathbf{z} - \mathbf{H}_k \mathbf{x}_{k|k-1} \tag{3.4}$$

$$\mathbf{K}_k = \mathbf{P}_{k|k-1} \mathbf{H}_k^T (\mathbf{H}_k \mathbf{P}_{k|k-1} \mathbf{H}_k^T + \mathbf{R}_k)^{-1} \tag{3.5}$$

$$\mathbf{x}_{k|k} = \mathbf{x}_{k|k-1} + \mathbf{K}_k \mathbf{e}_k \tag{3.6}$$

$$\mathbf{P}_{k|k} = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k) \mathbf{P}_{k|k-1} \tag{3.7}$$

where the subscript $k|k$ indicates a variable's value at time k including all information from time k (corrected value), $\mathbf{K}$ is the Kalman gain, $\mathbf{e}$ is the observation error and $\mathbf{H}$ is the observation matrix. These equations, referred to as the EKF *update* phase, give the most likely state estimate given the sensor measurements, sensor noise, predicted state and their covariance. In *Artificial_11*, the sensor measurements refer to the telemetry measurements between a vehicle pose and one or more landmarks. Since the measurement relates (correlates) the current pose with existing landmarks, this process uses the stored knowledge of the environment to correct the pose estimate. The ever-increasing covariance of the dead reckoning situation in (3.3) is also corrected.

The EKF solution to the SLAM problem grows only with the number of landmarks in the map. If n is the number of landmarks, the EKF uses $O(n^2)$ memory due to the storage of the $n \times n$ covariance matrix. The computational complexity of the *predict* step is $O(n^2)$ while the *update* step is also $O(n^2)$ if key matrices such as $\mathbf{K}_k\mathbf{H}_k$ are sparse, which is often the case in SLAM. If they are not, the *update* step has a complexity of $O(n^3)$. To illustrate this, the matrices involved in calculating the EKF are displayed for various points of *Artificial_11* in Figure 3.1, where pose s_1 is used in (a) as an example of dead reckoning, while the telemetry measurements at pose s_1 and s_3 are used in (b) and (c) to demonstrate how the filter grows with the addition of previously unknown landmarks. Finally, (d) illustrates the effect of re-observing l_1 at pose s_5. It can be seen in (d) that matrix $\mathbf{K}_k\mathbf{H}_k$ is sparse, with white and black boxes representing respectively zero and non-zero values. In all four examples, the matrix $\mathbf{P}$ contains $(1+n)^2$ elements where n is the number of landmarks

The memory complexity of the filtering approach can also be observed from the graph representation in Figure 2.16, where the landmarks and last pose form a complete graph with dense information matrix, which translates to a dense covariance matrix $\mathbf{P}$ in the filtering equations. Although this is significantly better than solving (2.25) directly, this solution is impractical for large maps due to the size of the matrices involved. Another

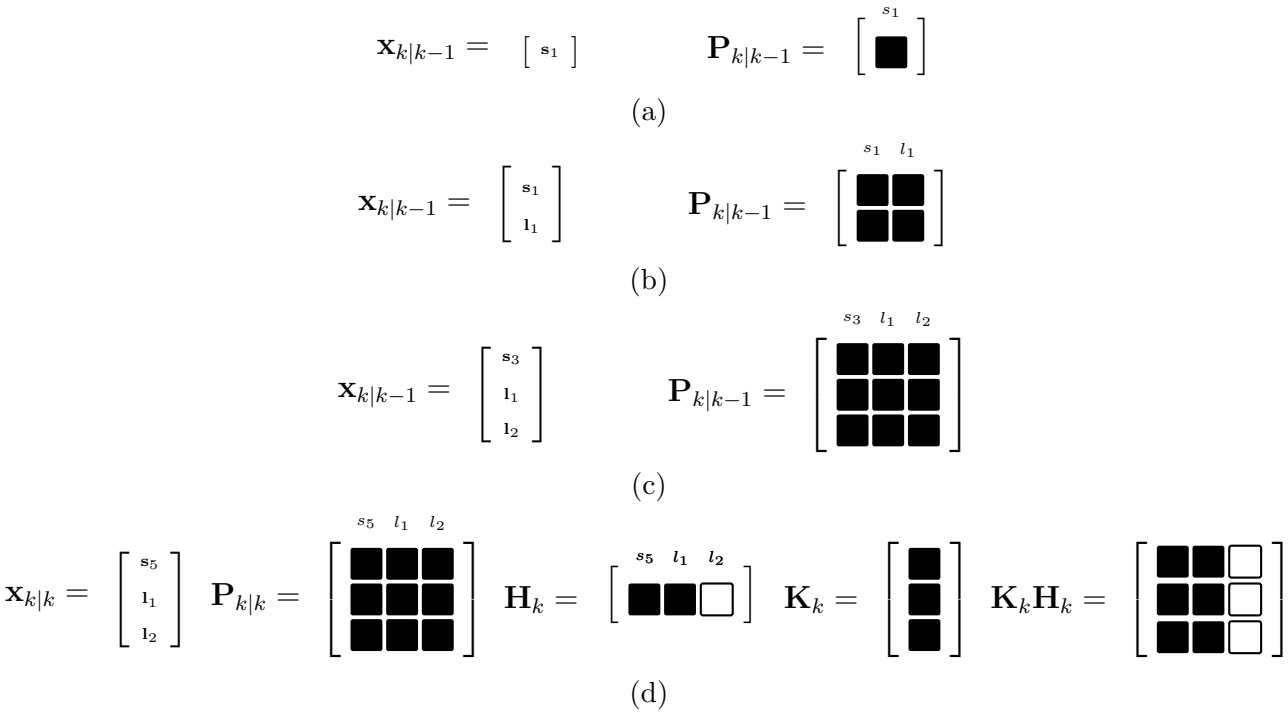


Figure 3.1: (a) Shows the state of the filter at pose s_1 before the telemetry measurement. This is a typical dead reckoning step. (b) and (c) illustrate the state of the filter at s_1 and s_3 respectively. These steps illustrate how the filter grows due to the observation of a new landmark. In (d), the observation of a known landmark at pose s_5 causes the update phase to occur. It can be seen that matrix $\mathbf{K}_k\mathbf{H}_k$ is sparse, with white and black boxes representing respectively zero and non-zero values. In all four examples, the matrix P contains $(1+n)^2$ elements where n is the number of landmarks

.

significant downfall of the EKF method is the inability to correct previous updates if the data is later found to be erroneous since odometry and telemetry measurements are not stored. The EKF also requires the system to be acceptably approximated by a linearised model and the noise to be Gaussian. In SLAM, the noise can usually be approximated by a Gaussian distribution but taking into account data association uncertainty, for example, would result in multi-modal distributions that can no longer be accurately modelled as Gaussian.

The first use of the Kalman filter for SLAM appears in a paper by Smith *et al.* [47] and it has become the default implementation of SLAM in the robotics community. It is the first computationally tractable solution to SLAM that is also exact for linear models

if the Gaussian noise assumption is satisfied. An adaptation of the Kalman filter to non-linear equations, the EKF requires that the linearisation of the model is a sufficiently good approximation and that the sensor and process noise are Gaussian. Although not exact, it has been shown that given these assumptions, the EKF based SLAM of a robot placed in an unknown environment converges and it has been demonstrated that the precision is limited only by the precision of the initial pose, the landmarks becoming increasingly correlated as observations are added [48]. It has also been demonstrated that the removal of landmarks does not affect statistical consistency and judicious selection of landmarks kept in the filter can be used to reduce computation cost [49].

Many current autonomous vehicle implementations rely on the EKF due to its speed and robustness. It has been used by the LAAS-CNRS autonomous blimp [50], a platform built to demonstrate control and mapping. An extended Kalman filter with Weiner process acceleration model has been used by Mirisola *et al.* [51] to build a dirigible capable of generating planar maps and tracking the two dimensional movement of ground objects. Ahrens *et al.* [52] present a EKF based guidance and control architecture for a quadrotor navigating an indoor GPS-denied environment. More specifically, they discuss drift-free hover and obstacle avoidance. Bath *et al.* [53] discuss a quadrotor using artificial landmarks, demonstrate a simple SLAM implementation and discuss practical issues encountered. The well known monoSLAM [54] and the recent addition allowing the tracking of mobile objects [55] are also EKF based algorithms. Kim *et al.* [56] propose an indirect EKF SLAM algorithm for an aircraft. It consists of keeping track of the error on the states instead of the states themselves and allows for a more efficient implementation.

The complexity of the EKF algorithm is still of concern. Even early on, Leonard *et al.* [57] described a non-optimal EKF implementation using decorrelated landmarks (diagonal covariance matrix) to help speed up computations. This method is, as mentioned, non-optimal since off-diagonal information (landmark-landmark edges in the marginalised graph) is discarded.

The convergence rate of a SLAM algorithm describes if and how fast the algorithm obtains an accurate estimate of the map and position. Although the EKF has been widely used and proven to converge under certain assumptions, it has been shown that the linearisation around estimated states of the EKF can yield optimistic covariance matrices and introduces error, which can ultimately lead to divergence of the vehicle pose and landmark position estimates [58]. Figure 3.2 illustrates an example given by Huang *et al.* [58] which shows that when a static vehicle with high orientation uncertainty observes a landmark multiple time, optimistic covariance on the landmark positions are obtained. The theoretical case is displayed in Figure 3.2a while Figure 3.2b shows the EKF result with the dashed theoretical case overlayed.

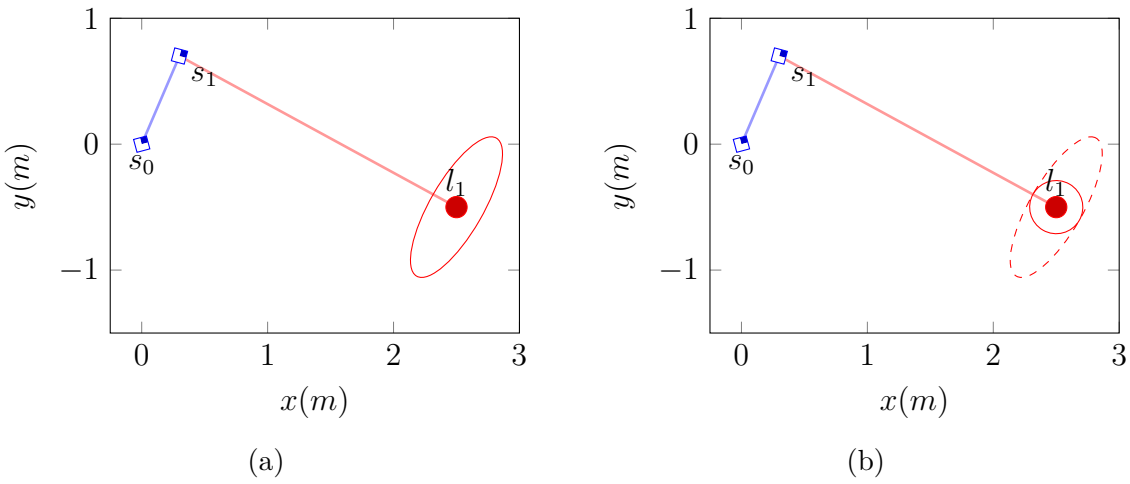


Figure 3.2: The vehicle stops at the second pose of *Artificial_11* and observes l_1. (a) The theoretical covariance ellipse of the landmark's position when the observation is made from a pose with high orientation uncertainty. (b) The EKF covariance with the theoretical covariance overlayed.

3.1.2 Unscented Kalman Filter

The unscented Kalman filter (UKF) [59] uses the unscented transform to reduce the effects of linearisation on the EKF. Instead of using the non-linear model for the state update only and the linearised model to update the covariance as seen in (3.2) and (3.3), the

non-linear model is used to recover both. First, the state distributions are augmented with the process noise

$$\mathbf{x}^a_{k-1|k-1} = \begin{bmatrix} \mathbf{x}_{k-1|k-1} \\ E[\mathbf{w}_k] \end{bmatrix} \tag{3.8}$$

$$\mathbf{P}^a_{k-1|k-1} = \begin{bmatrix} \mathbf{P}_{k-1|k-1} & 0 \\ 0 & \mathbf{Q}_k \end{bmatrix} \tag{3.9}$$

where the superscript a identifies augmented states, $E[\cdot]$ is the expectation operator, $\mathbf{w}_k$ is the sensor noise and $\mathbf{Q}_k$ is the sensor noise covariance matrix. A set of sample points is then generated from the mean and covariance

$$X^0_{k-1|k-1} = \mathbf{x}^a_{k-1|k-1} \tag{3.10}$$

$$X^i_{k-1|k-1} = \begin{cases} \mathbf{x}^a_{k-1|k-1} + \left(\sqrt{(L+\lambda)\mathbf{P}^a_{k-1|k-1}}\right)\mathbf{e}_i & \forall\ i = 1, ..., L \\ \mathbf{x}^a_{k-1|k-1} - \left(\sqrt{(L+\lambda)\mathbf{P}^a_{k-1|k-1}}\right)\mathbf{e}_{i-L} & \forall\ i = L+1, ..., 2L \end{cases} \tag{3.11}$$

where L is the length of the augmented state vector, λ is a constant chosen to control the spread of the samples points, and $\mathbf{e}_i$ is a column vector such that the i^{th} entry is one and all others are zero. In the simplified case where all variables are uncorrelated, (3.10) and (3.11) result in taking the mean and, for each variable, a sample at $\pm\sqrt{(L+\lambda)}\sigma$ from the mean, where σ is the standard deviation. An example with two random variables is illustrated in Figure 3.3, where the samples are identified by dark circles. The samples are propagated through the non-linear model

$$X^i_{k|k-1} = f(X^i_{k-1|k-1}) \tag{3.12}$$

and the Gaussian model is recovered from the transformed samples by calculating the mean and covariance of the distribution obtained. The UKF suffers from similar limitations as

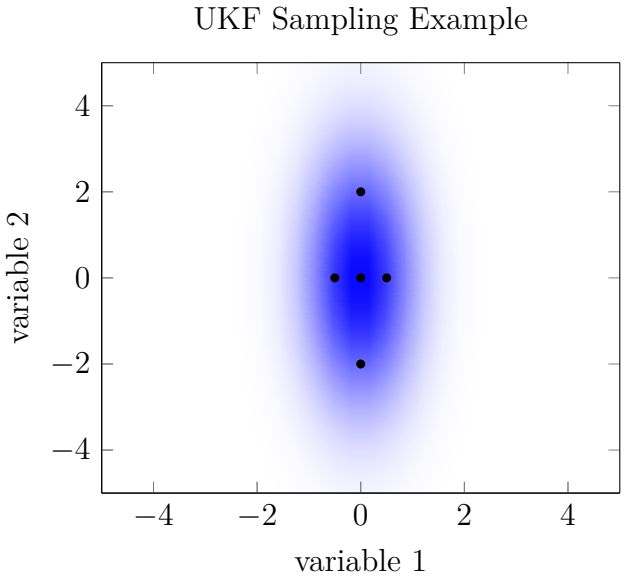


Figure 3.3: Unscented Kalman filter sampling example. A 2 variable distribution is sampled as specified in (3.10) and (3.11). The samples are identified by dark circles

the EKF, namely the Gaussian noise assumption and inability to recover from erroneous data, but is more robust to non-linearities in the model. More specifically, third order information can be recovered compared to first order for the EKF. This information is, however, still stored as a Gaussian distribution (second order statistic). Since the UKF also works on the marginalised problem (Figure 2.16) it has the same memory complexity as the EKF, which is $O(n^2)$ but has a computational complexity of $O(n^3)$ for both the *predict* and *update* steps due to the matrix square root required in (3.10) and (3.11).

Since the UKF was proposed by Julier *et al.* [59] it has been widely used to replace applications in which the EKF was found inadequate. Sunderhauf *et al.* [60] present an UKF based monocular SLAM implementation using inverse depth parametrisation to initialise landmarks observed by a single camera on the first measurement. They implement short term SLAM (landmarks are discarded when they leave the field of view) to mitigate map growth and demonstrate the UKF gives better results than the EKF but report computational complexity problems. Van Der Merwe *et al.* [61] propose the square root UKF (SR-UKF) which stores the square root of $\mathbf{P}$ instead of the full covariance matrix.

With the appropriate adaptations to the equations, this lowers the computational complexity of the *predict* step to $O(n^2)$. Holmes *et al.* [62] further study the SR-UKF in a monocular SLAM context and show that the *update* step can also be reduced to $O(n^2)$ in the case of SLAM. Their result is supported by an extensive comparison of EKF, UKF and SR-UKF. They find that the SR-UKF is faster than the UKF while achieving the same precision. They also discuss the compromise between the stability of the SR-UKF and the execution speed of the EKF. Huang *et al.* [63] propose a quadratic complexity observability-constrained unscented Kalman filter (OC-UKF). A novel sampling method is proposed which, for applications such as SLAM where a small number of states are involved in observations, reduced the complexity of the algorithm to $O(n^2)$. The OC-UKF thus has the same complexity as traditional EKF, but benefit from the UKF's tolerance to non-linearities. Furthermore, constraints related to the structure of the SLAM problem are imposed on the observations to increase the accuracy of the covariance estimate. Using theoretical and experimental data, they demonstrate that the error of the OC-UKF is significantly lower than what is obtained with the EKF and on par with iSAM, a Smoothing and Mapping algorithm presented in Section 3.1.6.

Although the computational gap between the the UKF and EKF has been reduced, UKF based algorithm still suffer from $O(n^2)$ memory and computational complexity. The unscented transform used in the UKF limits the effects of highly non-linear equations however, the Gaussian distribution assumption remains. Although some have worked on UKF-specific outlier rejection methods [64], the UKF, like the EKF, does not allow for the correction of previous updates if the data is later found to be erroneous.

3.1.3 Information Filters

The extended information filter (EIF) is equivalent to the EKF but maintains the inverse covariance matrix, called information matrix, instead. This is done to take advantage of

the fact that in cases where the EKF converges, the landmarks become highly correlated, making the value of the off diagonal elements of $\mathbf{P}$ increasingly high. By using the inverse of the covariance matrix, the off diagonal elements become asymptotically zero and the matrix becomes sparse. Both algorithms are related by

$$\mathbf{\Lambda} = \mathbf{P}^{-1}$$

$$\eta = \mathbf{P}^{-1}\mathbf{x}$$

Where $\mathbf{\Lambda}$ is the information matrix and η is the information vector. The EKF *predict* and *update* equations are adapted accordingly to perform operation on the information matrix. The update equations of the filter become

$$\mathbf{\Lambda}_{k|k} = \mathbf{\Lambda}_{k|k-1} + \mathbf{H}_k^T \mathbf{Q}_k^{-1} \mathbf{H}_k$$

$$\eta_{k|k} = \eta_{k|k-1} + \mathbf{H}_k^T \mathbf{Q}_k^{-1} (\mathbf{z} - h(\mathbf{x}_{k|k-1}) + \mathbf{H}_k \mathbf{x}_{k|k-1})$$

For the *predict* step, the information matrix and vector can be converted back to their covariance form and updated as in the EKF, or the update can be done by

$$\mathbf{\Lambda}_{k|k-1} = \mathbf{\Phi} - \mathbf{\Phi}(\mathbf{R}_{k-1} + \mathbf{\Phi})^{-1}\mathbf{\Phi}^T$$

$$\eta_{k|k-1} = \mathbf{\Lambda}_{k|k-1} f(\mathbf{u}_k, \mathbf{\Lambda}^{-1}_{k-1|k-1}\eta_{k-1|k-1})$$

where

$$\mathbf{\Phi} = \mathbf{F}_k^{-T} \mathbf{\Lambda}_{k-1|k-1} \mathbf{F}_k^{-1}$$

Aside from the information matrix itself, $\mathbf{H}$ is also sparse and leads to few changes for any given update. As $\mathbf{\Lambda}$ becomes sparse in the long term, significant computational and memory savings can be made. The EIF has a simple update step which has a complexity

of $O(n^2)$ but can be faster based on matrix structure. Both the *prediction* step and the conversion to regular parametrisation require a matrix inversion and have a complexity of $O(n^{2.4})$.

Shala *et al.* [65] compare EKF, EIF and FastSLAM (Section 3.1.4) on maps that would be encountered in planetary rover applications. They conclude that EIF provides the best results, both in terms of precision and numerical complexity. On the experimental data used, they show that the EIF takes one third the time of the EKF which itself takes about half the time of FastSLAM. A method improving on the EIF using sparse extended information filter (SEIF) is used by Thrun *et al.* [66] and consists of discarding some covariance values to make the information matrix sparse, resulting in a linear computation and storage cost due to the advantages of working with such matrices. Their result on the Victoria Park dataset show that the SEIF runs twice as fast as the EKF while using only 25% of the resources. Walter *et al.* [67] show that SEIF may give overconfident estimates and address the issue by proposing the exactly sparse extended information filter (ESEIF). Their method ignores selected measurements and obtains a more conservative estimate of the states. It can thus maintains sparsity while remaining consistent. Their results show that ESEIF performs as well as SEIF regarding memory footprint and execution time while having a normalised error consistently lower.

Ila *et al.* [68] have applied the EIF concepts to pose SLAM, which consists of using landmarks to link vehicle poses as opposed to locating the landmarks themselves. They demonstrate on experimental data that the total time spent updating the vehicle state was 50 s, which is insignificant compared to the data association time of 6265 s. Eustice *et al.* [69] propose a method of obtaining conservative covariance estimates for SLAM information filters. Their method is used on an unmanned underwater vehicle to explore the Titanic, and results show that data association using the obtained covariance estimates performs better than the Markov blanket method.

The robustness in the event of sensor failure is also of concern. Asadi *et al.* [70] implement a decentralised SLAM algorithm where INS and GPS are treated separately from the encoder and laser sensors. Each pair of sensors is updated in their own EKF and a global EIF based method is used to merge the results. This isolates the GPS information from the main filtering loop and compares advantageously to standard EKF in environments where GPS signal can be lost.

3.1.4 Particle Filters

Particle filters (PF) [71, 72] can be used to solve the full SLAM problem expressed in (2.22). The resulting distribution is approximated by a number of particles, each of which represent a particular path ($\mathbf{s}_{0:k}$) and map ($\mathcal{M}$) obtained by specific combinations of data associations and measurements. The transition from one pose to the next is done by adding samples from the odometry observation's probability density to each particle. This causes the particles to disperse as shown by the first three poses of Figure 3.4 for the *Artificial_11* dataset. This is expected in the dead reckoning scenario due to the accumulation of sensor error. Telemetry observations proceed in a similar way with samples of the observation's probability density used to update each particle's map. Since the formulation in (2.22) completely decorrelates the landmarks, they can be individually tracked. Each particle's map thus consists of multiple small sized (3x3 for 3D position) EKFs keeping track of each landmark's best position estimate as observations are added. Occasionally, the particle cloud is re-sampled with a fitness-based score such that the most likely solutions are more likely to be selected by the sampling process.

A map representation of a particle filter for the *Artificial_11* dataset is illustrated in Figure 3.4. The current position of each particle is shown in blue while past positions are shown in light blue. Landmarks associated with each particle are shown in red. The current best estimate of the landmark and vehicle positions can be obtained by averaging

each particle's estimate. For $\mathbf{l}_1$, two of the particles' estimates overlap because they share the same history up to and including the observation of $\mathbf{l}_1$. More details regarding the vehicle paths represented by each particle can be found in the MRF based representation of Figure 3.5 where each row represents a particle and each column represents a time step. The position history of a particle is indicated by blue edges, while black edges indicate particles that where eliminated by resampling. The red edges indicate landmark observations, which are linked to the corresponding pose in the particle's history. For example, at $k = 0$ and $k = 1$, each particle is kept while for $k = 2$ particle 2 and 3 are resampled from particle 1, thus at that point all particles will have their landmarks associated with s_1 from particle 1.

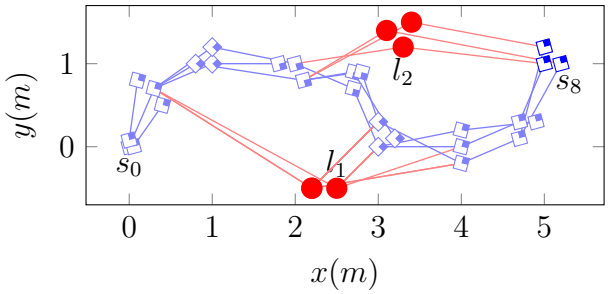


Figure 3.4: Map representation of a particle filter for the *Artificial_11* dataset. The current position of each particle is shown in blue while past positions are shown in light blue. Landmarks associated with each particle are shown in red.

A total of kn 3x3 EKFs are used, where n is the number of landmarks and k the number of particles. Although the number of EKFs required is very large, their size is fixed which leads to a fixed update cost per particle and a total update cost of $O(k)$. The original algorithm has a memory complexity of $O(kn)$ but can be made $O(k \log n)$ with proper data structures and sharing landmark positions between particles that have a common pose and observation history [73]. The complexity thus increases linearly with the number of particles and number of landmarks. The use of particle filters has the advantage of allowing for multiple hypotheses in data association but limits the ability of loop closing due to the finite number of particles.

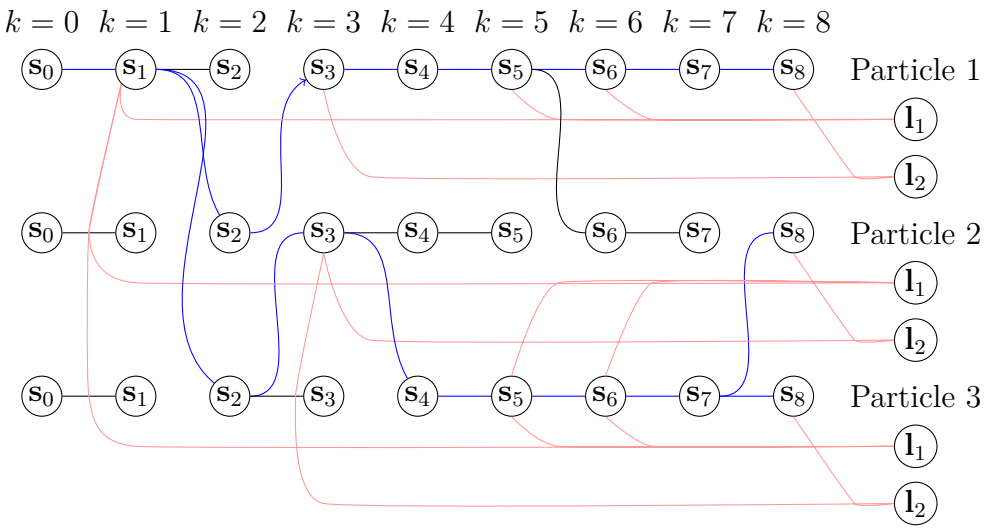


Figure 3.5: MRF based representation of a particle filter for the *Artificial_11* dataset. Each row represents a particle. The position history of a particle is indicated by blue edges, while black edges indicate particles that where eliminated by resampling. The red edges indicate landmark observations, which are linked to the corresponding pose in the particle's history.

Soon after Murphy [22] presented the full SLAM factorisation of the SLAM problem, Rao-Blackwellised particle filters (RBPF) were applied to dynamic Bayesian networks and used to solve the SLAM problem on grid maps [71]. The authors use the Rao-Blackwellisation formula to take advantage of the problem structure and limit the sampling space of the particle filter to variables that affect the solution. The use of RBPF in SLAM was later popularised by the introduction of FastSLAM and FastSLAM2 [74] using feature maps. FastSLAM offers an efficient implementation that allows, through sharing of landmark instances common to multiple particles, to solve SLAM in real time with multiple particles. It suffers, however, from the sample impoverishment problem often encountered with particle filters. This is addressed in FastSLAM2, which has also been proven to converge. The authors compared FastSLAM, FastSLAM2 and EKF on the Victoria Park dataset and found that, to obtain the same accuracy as the EKF, FastSLAM required 50 particles while FastSLAM2 required only one. The dataset is solved incrementally by FastSLAM2 in 54 s, while FastSLAM and EKF need, respectively, 315 s and 7807 s to complete the same task. Kim *et al.* [75] demonstrate that the conventional Rao-

Blackwellised unscented PF SLAM implementation uses overconfident assumptions and propose the exactly Rao-Blackwellised unscented PF to address the issue. They demonstrate on the Victoria Park dataset examples of cases where traditional PF SLAM methods lead to incorrect associations. Zhu *et al.* [76] build upon the unscented fastSLAM method and demonstrate that semantic constraints between landmarks can be included in the filter for added accuracy. They show on experimental data a case where semantic constraints are necessary for map convergence.

FastSLAM was also adapted to grid maps [77] and a method was developed to take advantage of the precision of laser scans. Schröter *et al.* [78] use a Rao-Blackwellised particle filter to build a grid map with data acquired from a sonar range sensor. The map is divided in patches such that memory is shared between particles and common map sections are stored only once. For each particle, observations are added to a queue, forming a local map that is matched to the global map to find particles' weights. Schroeter *et al.* [79] further improve on this by making the approach sensor independent and showing results for stereo and monocular systems. The sensor independence arise from the fact that the local and global maps are matched instead of features, allowing the same implementation to be used on multiple vehicle configurations. Sim *et al.* [80] use a RBPF to implement a SLAM algorithm based solely on the information from a monocular camera (odometry and SIFT landmarks). Methods are used to avoid including observations that are likely due to error and experimental results are presented with off-line execution.

Taking advantage of both feature and grid maps, Wurm *et al.* [81] design a FastSLAM2 based algorithm that can automatically switch between the type of map used. This solves the problem of robots navigating in indoor and outdoor environments, with grid maps being more efficient indoor and feature maps offering better results when used outdoor.

3.1.5 Genetic Algorithms

Genetic algorithms (GA) are an evolution inspired technique of stochastic optimisation that can be used to solve the full SLAM problem of (2.22) or to approximate the Bayes filter in (2.25). To do so, a set of possible solutions is brought closer to the optimal solution over a number of iterations. In GA terminology, a possible solution is called an individual, the set of all individuals is called a population and the population of a given iteration is referred to as a generation. The initial population is generated by sampling the solution space and the subsequent generations are obtained by re-sampling the previous generation where the probability of selecting any given solution is based on a fitness score. In the case of full SLAM, an individual would consist of the landmark positions and vehicle pose history while (2.22) is used as the fitness function. Mutation and crossover operations are also performed to ensure progress is made towards the goal. Mutations occur with predefined probability and consist of changing the value of a component of a solution, thus ensuring population diversity and allowing the algorithm to leave a local minima. On a full SLAM solution, a mutation consists of randomly displacing a landmark position or vehicle pose. Crossovers, on the other hand, attempt to merge two good solutions with the hope of producing a better one. For example, a new full SLAM solution could be generated by taking landmark positions and vehicle poses from two other solutions. The process is stopped after a number of generations or when a particular fitness score is achieved. For example, Figure 3.6 illustrates a GA based solution to the full SLAM problem on the *Artificial_11* dataset where two individuals are evolved over four generations. Randomly occurring mutations are indicated by red lines and crossovers are denoted by arrows going from one individual to the other. The next generation of each individual is build from parts or either individuals of the previous generation as well a randomly occurring mutation operators. The gradual improvement of the solution can be seen in Figure 3.7.

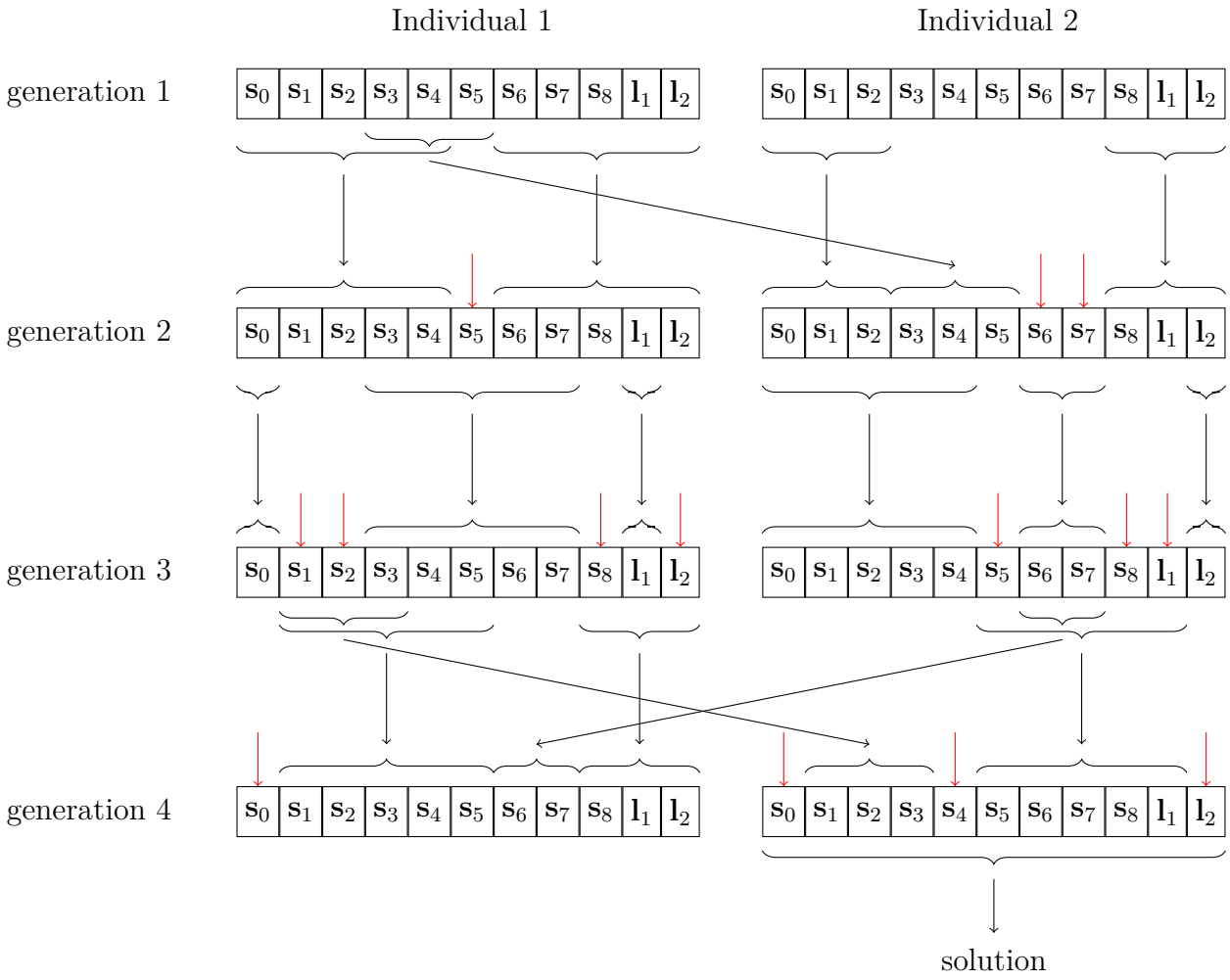


Figure 3.6: Example of GA based solution to the full SLAM problem on the *Artificial_11* dataset. Two individuals are evolved over four generations. Randomly occurring mutations are indicated by red lines and crossovers are denoted by arrows going from one individual to the other.

This avenue of research has not been widely explored, but some articles present genetic algorithm based approaches to solve SLAM. Duckett [82] proposes to treat SLAM as a global optimisation problem and uses GA to solve (2.22) using raw sensor data represented on a grid map. Possible robot trajectories defined by a number of segments as well as landmarks and data associations are included in the solution. GA is used to execute the search over the possible configurations and the configuration that obtains the highest likelihood score is selected. This method provides interesting advantages regarding loop closing and multiple data associations but the two hours runtime prohibits online testing.

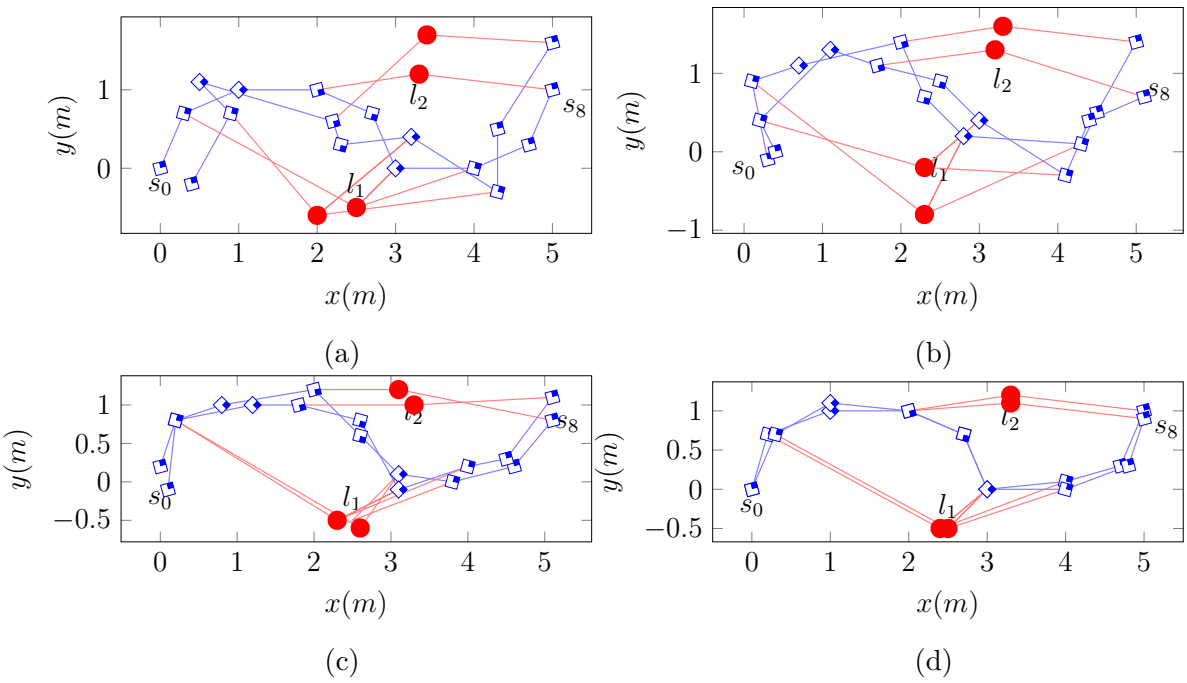


Figure 3.7: The map representation of a 4 generations, 2 individuals solution to *Artificial_11*. Subfigures (a), (b), (c) and (d) illustrate the map obtained for generations 1 to 4, respectively. It can be seen that the solution gradually improves.

The author points to the highly parallel nature of GA and combination of GA and EKF as areas of future research. Begum *et al.* [83] propose a GA based filter reminiscent of the particle filter method. It relies on the island model of genetic algorithms to encourage keeping multiple maps and data associations. Like the previous algorithm, it is based on raw sensor data represented on a gridmap. Experimental results consist of an analysis of the map quality over time as the robot explores a small environment but no comparisons are made to other algorithms. Moreno *et al.* [84] propose a genetic algorithm based solution called the evolutive localisation filter (ELF). It is based on differential evolution which consists of using the difference between two individuals as a disturbance to another. If the disturbed individual scores higher than the original one, it takes it's place. The ELF is a two step approach. The first step consist of using the maximum a posteriori (MAP) estimate in a recursive fashion to move the current generation of poses by samples from the control input probability distribution. When a loop is detected, the ELF is applied

on all poses in the loop to remove the accumulated error. Experimental results show local optimisation can be executed in 0.47 s while loop closing can take up to 15.8 s.

While most GA implementations take the full SLAM approach, Feng *et al.* [85] propose to solve the classical SLAM formulation by computing (2.25) efficiently with GA. Their algorithm, genetic algorithmic filter (GAF) uses two different GA implementations for the robot pose and map. Using experimental data, they show GAF can obtain solutions in 300 ms. Like the other GA algorithms described, they use gridmaps with raw laser sensor data.

More recently, Wu *et al.* [86] have made progress using GA in SLAM and propose a genetic based algorithm for estimating vehicle motion between two laser scans. Experimental results of a vehicle navigating a small environment are provided and show the proposed algorithm can calculate the displacement between two laser scans in 0.048 s.

3.1.6 Smoothing and Mapping (SAM)

The field of structure from motion, which studies the reconstruction of three dimensional structures from sequences of two dimensional images, makes extensive use of bundle adjustment. This technique consists of optimizing the scene structure and viewing parameters such that the error is minimized. The application of these concepts to the full SLAM problem of (2.18) has given rise to Smoothing and Mapping (SAM) . To obtain the SAM formulation, the natural logarithm of (2.22) is taken and, assuming Gaussian distribution for each random variable, the following is obtained:

$$\sum_{i=1}^{k} \|f_i(\mathbf{s}_{i-1}, \mathbf{u}_i) - \mathbf{s}_i\|^2_{\mathbf{\Sigma}_{u_i}} + \sum_{i=1}^{n_z} \|h_i(\mathbf{s}_{z_i}, \mathbf{l}_{z_i}) - \mathbf{z}_i\|^2_{\mathbf{\Sigma}_{z_i}} - b \tag{3.13}$$

where $\mathbf{\Sigma}_{u_i}$ and $\mathbf{\Sigma}_{z_i}$ are the covariance matrices of control and observation i respectively. $\|\mathbf{a}\|^2_{\mathbf{\Sigma}} = \mathbf{a}^T\mathbf{\Sigma}^{-1}\mathbf{a}$ represents the Mahalanobis norm given the covariance matrix $\mathbf{\Sigma}$ and b

represents the constant terms

$$\sum_{i=1}^{k} \ln \frac{1}{\sqrt{2\pi\boldsymbol{\Sigma}_{u_i}}} + \sum_{i=1}^{n_z} \ln \frac{1}{\sqrt{2\pi\boldsymbol{\Sigma}_{z_i}}} \tag{3.14}$$

which are ignored since they do not affect the position of the minimum. Equation (3.13) is then linearised and the $\boldsymbol{\Sigma}$ matrices are included inside the norms using

$$\|\mathbf{a}\|^2_{\boldsymbol{\Sigma}} = \mathbf{a}^T\boldsymbol{\Sigma}^{-1}\mathbf{a} = \|\Sigma^{-T/2}\mathbf{a}\|^2_2 \tag{3.15}$$

to obtain

$$\sum_{i=1}^{k} \|\hat{\mathbf{F}}_i\delta s_{i-1} - \hat{\mathbf{G}}\delta s_i - \boldsymbol{\Sigma}^{-T/2}(\mathbf{s}_i - f_i(\mathbf{s}_{i-1}, \mathbf{u}_i)\|^2_2 + \sum_{i=1}^{n_z} \|\hat{\mathbf{H}}_i\delta\mathbf{s}_{z_i} + \hat{\mathbf{J}}_i\delta\mathbf{l}_{z_i} - \boldsymbol{\Sigma}^{-T/2}(\mathbf{z}_i h_i - (\mathbf{s}_{z_i}, \mathbf{l}_{z_i}))\|^2_2 \tag{3.16}$$

where $\hat{\mathbf{G}} = -\boldsymbol{\Sigma}_i^{-T/2}\mathbf{I}$ and $\mathbf{I}$ is an identity matrix of appropriate size. $\hat{\mathbf{F}}_i = \boldsymbol{\Sigma}_i^{-T/2}\mathbf{F}$, $\hat{\mathbf{H}}_i = \boldsymbol{\Sigma}_i^{-T/2}\mathbf{H}$, $\hat{\mathbf{J}}_i = \boldsymbol{\Sigma}_i^{-T/2}\mathbf{J}$, are the new variables obtained once the covariance has been distributed and $\mathbf{F}$, $\mathbf{J}$ and $\mathbf{H}$ are such that

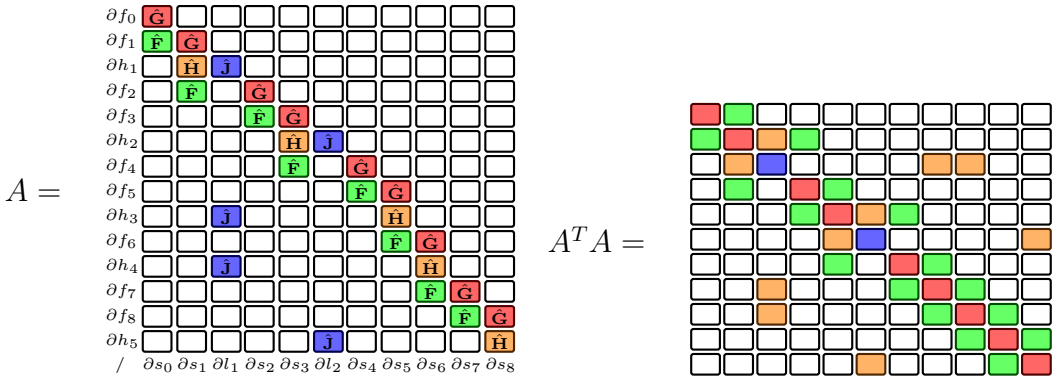
$$\frac{\partial}{\partial\mathbf{s}_{i-1}} f_i(\mathbf{s}_{i-1}, \mathbf{u}_i) = \mathbf{F}_i \tag{3.17}$$

$$\frac{\partial}{\partial\mathbf{s}_{z_i}} h_i(\mathbf{s}_{z_i}, \mathbf{l}_{z_i}) = \mathbf{H}_i \tag{3.18}$$

$$\frac{\partial}{\partial\mathbf{l}_{z_i}} h_i(\mathbf{s}_{z_i}, \mathbf{l}_{z_i}) = \mathbf{J}_i \tag{3.19}$$

Posing $\mathbf{a}_i = \boldsymbol{\Sigma}^{-T/2}(\mathbf{s}_i - f_i(\mathbf{s}_{i-1}, \mathbf{u}_i)$, $\mathbf{c}_i = \boldsymbol{\Sigma}^{-T/2}(\mathbf{z}_i - h_i(\mathbf{s}_{z_i}, \mathbf{l}_{z_i}))$ and expressing (3.16) in matrix form, the following system is obtained

$$\mathbf{A}\begin{bmatrix}\delta\mathbf{s}_{0:k}\\ \delta\mathbf{l}_{1:n_l}\end{bmatrix} = \begin{bmatrix}\mathbf{a}_{0:k}\\ \mathbf{c}_{1:n_z}\end{bmatrix} \tag{3.20}$$



(a) A matrix (b) $A^T A$ matrix

Figure 3.8: Various formulations of SLAM matrices for SAM applied to *Artificial_11*

This yields a very large but sparse matrix $\mathbf{A}$ built with the derivatives of the state transition and observation equations. This matrix is closely related to the factor graph and it has one column per variable (landmark/pose) and one row for each factor (measurement). The matrix obtained from the factor graph of *Artificial_11* (Figure 2.8) can be seen in Figure 3.8a. Depending on the method used, the system in (3.20) can be solved. For example, with successive applications of Givens rotations as follows

$$\begin{bmatrix} \mathbf{R} \\ 0 \end{bmatrix} \begin{bmatrix} \delta \mathbf{s}_{0:k} \\ \delta \mathbf{l}_{1:n_l} \end{bmatrix} = \mathbf{Q}^T \begin{bmatrix} \mathbf{b} \\ \mathbf{r} \end{bmatrix} \tag{3.21}$$

where $\mathbf{b}$ is the observation error and $\mathbf{r}$ is the least square error. The solution can also be obtained taking the Cholesky decomposition of the normal equation

$$\mathbf{A}^T \mathbf{A} \begin{bmatrix} \delta \mathbf{s}_{0:k} \\ \delta \mathbf{l}_{1:n_l} \end{bmatrix} = \mathbf{A}^T \begin{bmatrix} \mathbf{a}_{0:k} \\ \mathbf{c}_{1:n_z} \end{bmatrix} \tag{3.22}$$

which can equivalently be obtained by evaluating the norm in (3.16), taking the derivative and solving by equating to 0. The normal equation relates to the adjacency matrix of the MRF. The matrix $A^T A$ for the MRF of *Artificial_11* (Figure 2.9) can be seen in Figure 3.8b.

Taking advantage of algorithms for fill-reducing column reordering, inherent knowledge of the system, and sparse matrix operations, the solution can be recovered efficiently.

Smoothing and mapping methods have been found to be very efficient at solving the full SLAM formulation (2.18) due to the sparsity of the system. Square root Smoothing and Mapping ($\sqrt{SAM}$) by Dellaert *et al.* [23] is one such method. It approaches the SLAM problem as an over-defined system where each row corresponds to a measurement as in (3.20). $\sqrt{SAM}$ solves (3.22) for the least square solution using the **LDL** (Lower triangular-Diagonal-Lower triangular) decomposition and back substitution. This results in the best landmark positions and path for the gathered data. When new data is added, the $\mathbf{C} = \mathbf{A}^T\mathbf{A}$ matrix is incrementally updated, factored once more and solved to find the updated solution. They show that the column ordering of $\mathbf{A}$ greatly influences the computational efficiency of the algorithm and that with proper ordering, the number of non-zeros in $\mathbf{L}$ can be reduced by a factor of 20 compared to the case where $\mathbf{A}$ is unordered. Their results also demonstrate that with properly selected orderings, the linearisation is more expensive than the factorisation by a factor of 6, thus justifying the factorisation cost at every measurement addition. Their results show that the proposed solution, following a linear growth in computation time, is much faster than EKF for large environments with the EKF's computation time increasing quadratically. The authors also notice that their method is much better at solving non-linearities than traditional EKF. This is due to the fact that the linearisation point of the system can be updated since a complete history of poses and observations is kept, whereas previously included data cannot be altered in filter based algorithms. The solution obtained is exact but the processing time for such solution grows without bounds since is is proportional with the number of observations.

Kaess *et al.* [87] further improved on $\sqrt{SAM}$ with incremental Smoothing and Mapping (iSAM). This method uses Givens rotations to update the square root form directly, saving the factorisation cost. In the exploratory stage, this method takes $O(1)$ time to add an observation and a solution can be obtained by back substitution in $O(n)$. For

simple environments where only two landmarks may be observed by any three poses, this climbs to $O(n \log n)$. Kaess *et al.* [88] also present a new data structure called a Bayes tree that maps naturally to the square root information matrix used in $\sqrt{\text{SAM}}$ and introduce a new algorithm, iSAM2 that takes advantage of such data structure. The tree is build by converting the available observation information in a Bayes network and using a clique search algorithm. Future observations are iteratively included without the need for re-linearisation. The authors present simulation results with multiple datasets and results show that the algorithm can perform better or worse than iSAM depending on the environment. Huang *et al.* [89] propose U-iSAM, an algorithm based on iSAM but using the unscented transform to reduce the linearisation errors. Their formulation tackles the cooperative localisation and target tracking (CLATT) problem where a team of robots track each other as well as a dynamic target. Polok *et al.* [90] present SLAM++, an algorithm similar to iSAM but based on Cholesky decomposition and propose a method to incrementally update the factor directly without requiring re-computation as was necessary in $\sqrt{SAM}$. SLAM++ also takes advantage of the block structure of SLAM to further optimise performance. Widening the scope to include other SLAM algorithms, Strasdat *et al.* [91] compare the use of filters to bundle adjustment methods and conclude that bundle adjustment methods are preferable. Moosmann *et al.* [92] propose a least square method that is adapted specifically for the Velodyne laser scanner while McDonald *et al.* [93] build upon iSAM and appearance based method to include multi-session mapping in large scale environments.

Due to the availability of multiple high-performance implementations, SAM is an efficient solution to the SLAM problem. However, where the EKF's size is limited by the number of landmarks in a given environment, SAM methods grow without bounds as they are dependent on the number of observations. For more accuracy, SAM methods can also be adapted to work with non-linear solvers such as Gauss-Newton.

3.2 Extensions and solutions

As SLAM algorithms are tested in increasingly large environments and complex situations, methods have been proposed to allow them to cope with more exacting requirements. The most important challenges encountered by the basic SLAM algorithms discussed in Section 3.1 are listed below and presented in more details in Appendix A.

Large maps

The current methods become computationally intractable if the map is too large. Submap methods address this by dividing larger maps into tiles. See Appendix A.1 for details.

Non-linearities

The conversion from a vehicle centric coordinate system to the world coordinate system contains non-linearities that can be significant in some situations. Different parametrisations can be used to change the equation of the transformation. See Appendix A.2 for details.

Multiple vehicles

Typical use cases involve many vehicles which must share information to interact in an integrated fashion. See Appendix A.3 for details.

Wrong data association

Good data association is essential for the convergence of filter based methods. Many algorithms are available to significantly reduce the occurrence of bad matches. See Appendix A.4 for details.

Dynamic environments

Real-world applications involve dynamic environments, whether it is moving landmarks or a change in lighting conditions. Experience based methods have been proposed to address this. See Appendix A.5 for details.

3.3 Comparison and Summary

For the reader's convenience, comprehensive comparison tables are presented in this section and the literature review is summarised. Notation and definitions used in this section are described in Table 3.1. In Table 3.2, various popular SLAM algorithms are listed and their different properties and capabilities are compared. The advantages and drawbacks of the different methods are further discussed in Table 3.3. Note that these tables are the result of the author's subjective analysis of the available literature for the purpose of painting a picture of the existing methods to the reader. These tables should not be used to support objective conclusions as each algorithm's performance depends on application specific factors better captured by benchmarking.

In this chapter, popular SLAM algorithms were reviewed, with additional extensions presented in Appendix A. In Section 3.1.1 the most popular algorithm for SLAM, the EKF, was explained and various articles using it were presented. The UKF was presented in Section 3.1.2 as an extension to the EKF that is more resilient to non linearities. The EIF, working with the inverse covariance matrices, was presented in Section 3.1.3 and shown to be more efficient than traditional EKF filters in environments with highly correlated landmarks. Sections 3.1.4, 3.1.5 and 3.1.6 introduced, respectively, particle filters, genetic algorithm and SAM, as alternatives to the traditional filtering paradigm. Of these areas of research, the use of particle filters is the most mature and has been used in multiple implementations. SAM has seen a significant boost in interest in the past years and genetic algorithms, although significantly less popular, offer interesting avenues of research. Extensions to existing SLAM algorithms tackling specific problems were briefly reviewed in Section 3.2 with a more detailed discussion available in Appendix A. More specifically, the use of submaps to extend the applicability of present methods to larger areas is presented in Section A.1. Section A.2 presents parametrisations designed to be more robust to non-linearities, while Section A.3 tackles the problem of SLAM with multiple collaborating

Table 3.1: Legend and Definitions for Table 3.2

Property	Description
Type	B (Bayesian), P (Particle), G (Genetic), S (Smoothing), A (Appearance).
Complexity	Memory complexity of the algorithm. "-" complexity varies greatly based on situation, "n/a" data not available, "n" total landmarks, "n_l" local landmarks, "n_c" common landmarks, "k" particles, submaps or robots, "m" observations, "m_l" local observations, "P" global update period.
Div. resist	Resistant to divergence caused by non-linearities.
Data assoc.	Lends itself easily to data association correction methods.
Efficiency	Efficiency score relative to other methods in the same section or "offline".
Dynamic env.	The environmental changes that can be represented easily. The options used are "spd/acc" for the speed and acceleration of moving objects, "time/season" for the appearance of static objects over time, "local" if it can track dynamic objects on the local map only, and "no" if the representation of dynamic environments require more modifications.
Loop closing	Algorithms' ability to perform loop closing. The options are "yes" for regular loop closing, "limited" for cases where the ability to close loops depend on the length, "identify" in cases where loop closings are identified but the estimates are not updated optimally, and "single" for multi-vehicle algorithms that only perform loop closing locally.

robots, which is currently a very popular field of research. Various ways to tackle incorrect data associations are presented in Section A.4. Finally, various approaches to adapt SLAM to dynamic environment, including both moving obstacles and time variations such as seasons, are reviewed in Section A.5.

Table 3.2: SLAM Algorithm Performance Comparison

		Complexity		Properties				
Classical	Type	Memory	Computation	Div. Resist	Data Assoc.	Efficiency	Dynamic Env.	Loop Closing
EKF (p.51)	B	$O(n^2)$	$O(n^2)$				spd/acc	yes
UKF (p.56)	B	$O(n^2)$	$O(n^2)$	×			spd/acc	yes
EIF (p.59)	B	$O(n^2)$	$O(n^{2.4})$				spd/acc	yes
SEIF (p.61)	B	$O(n)$	$O(n)$				spd/acc	yes
FastSLAM (p.64)	P	$O(n)$	$O(k \log n)$		×		spd/acc	limited
FastSLAM2 (p.64)	P	$O(n)$	$O(k \log n)$	×	×		spd/acc	limited
ELF (p.68)	G	-	-	×	×		no	yes
$\sqrt{\text{SAM}}$ (p.72)	S	$O(m)$	$O(m)$	×	×		no	yes
iSAM (p.72)	S	-	-	×	×		no	yes
iSAM2 (p.73)	S	-	-	×	×		no	yes
SLAM++ (p.73)	S	-	-	×	×		no	yes

		Complexity		Properties				
Submap	Type	Memory	Computation	Div. Resist	Data Assoc.	Efficiency	Dynamic Env.	Loop Closing
CEKF (p.176)	B	$O(n^2)$	$O(n_l^2 + \frac{n_l(n-n_l)^2}{P})$				local	yes
CLSF (p.178)	B	$O(n^2)$	$O(n_l^2 + \frac{n_c n^2}{P})$				local	yes
NCFM (p.179)	B	$O(kn_l^2)$	$O(n_l^2)$				local	identify
T-SAM (p.180)	S	$O(m)$	n/a	×	×	offline	no	yes
T-SAM2 (p.180)	S	$O(m)$	n/a	×	×	offline	no	yes

		Complexity		Properties				
Multiple Robots	Type	Memory	Computation	Div. Resist	Data Assoc.	Efficiency	Dynamic Env.	Loop Closing
DDF-SAM (p.184)	S	$O(m_l + \frac{n_g}{k}^2)$	$O(m_l)$	×	×		no	single
DDF-SAM2 (p.185)	S	$O(m_l + \frac{n_g}{k}^2)$	-	×	×		no	yes

		Complexity		Properties				
Dynamic Environment	Type	Memory	Computation	Div. Resist	Data Assoc.	Efficiency	Dynamic Env.	Loop Closing
Rat-SAM + FAB-MAP (p.188)	A	n/a	n/a	×	incl	offline	time/season	yes
Experience based SLAM (p.188)	A	n/a	n/a	×	incl		time/season	yes

Table 3.3: SLAM Algorithm Summary

Methods	Advantages	Drawbacks
EKF (p.51)	• Covariance readily available • Widely discussed and characterised	• Sensitive to data association error • No relinearisation • Complexity of memory and computation are n^2 with number of landmarks
EIF (p.59)	• Inverse covariance readily available • Update step faster than EKF	• Sensitive to data association error • No relinearisation • Predict step more expensive then EKF
ESIF (p.61)	• Inverse covariance readily available • Complexity of memory and computation is n with number of landmark	• Sensitive to data association error • No relinearisation • Loss of information due to approximation
$\sqrt{\text{SAM}}$ (p.72)	• Faster than EKF • Allows data association correction • Allows for relinearisation	• complexity grows with the number of observation (unbounded) • Does not maintain 2^{nd} order statistics • Requires complete refactorisation each step
iSAM (p.72)	• Very fast in Practice • Exploratory computation complexity of $O(1)$ • Worst case computation complexity (for SLAM) of $O((n+m)^{1.5})$ • Relinearisation and data association correction	• complexity grows with the number of observation (unbounded) • Does not maintain 2^{nd} order statistics • Theoretical worst case computation complexity $O(n^3)$

iSAM2 (p.73)	• Very fast in Practice • Applicable to non linear systems • Exploratory complexity of $O(1)$ • Practical worst case of $O((n+m)^{1.5})$	• complexity grows with the number of observation (unbounded) • Does not maintain 2^{nd} order statistics • Worst case complexity $O((n+m)^3)$
FastSLAM (p.64)	• Applicable to non linear systems • Capable of multiple data association hypothesis • Computation complexity of $O(k \log n)$ • Best case memory complexity $O(n)$	• Loop closing abilities depend on number of particles • Suffers from sample impoverishment • Theoretical memory complexity worst case of $O(nk)$
FastSLAM2 (p.64)	• Applicable to non linear systems • Capable of multiple data association hypothesis • Computation complexity of $O(k \log n)$ • Best case memory complexity $O(n)$	• Loop closing abilities depend on number of particles • Theoretical memory complexity worst case of $O(nk)$
SLAM++ (p.73)	• Applicable to non linear systems • Faster than iSAM2 • Fast covariance recovery • Implementation lends itself to parallel execution	• complexity grows with the number of observation (unbounded) • Does not maintain 2^{nd} order statistics

Submap Methods	Advantages	Drawbacks
CEKF (p.176)	• Optional more efficient but lossy implementation • Cost amortised over time • Full update only required when map is needed or vehicle changes area • Only local map is contaminated in case of data association error	• Full SLAM cost for map update • Lower global map refresh rate
CLSF (p.178)	• Cost amortised over time • Full update only required when map is needed or vehicle changes area • Only local map is contaminated in case of data association error	• Relies on correct correspondence between local and global map • Lower global map refresh rate
NCFM (p.179)	• Reduced computation compared to EKF or other submap methods	• Suboptimal method • Global map not readily available • Loop closing is an extra step
T-SAM (p.180)	• Faster than $\sqrt{\text{SAM}}$ on same size graph • Reduces linearisation time	• Less accurate than $\sqrt{\text{SAM}}$ • Offline application only
T-SAM2 (p.180)	• Nested dissection partitioning • Significantly faster than $\sqrt{\text{SAM}}$ • Multilevel map	• Offline application only • Possible loss of accuracy

Multiple Robots	Advantages	Drawbacks
DDF-SAM (p.184)	• Allows generation of global map	• Global map does not contribute to trajectory estimation accuracy • Constraints used to generate condensed map are not ideal for factorisation
DDF-SAM2 (p.185)	• Global map is merged with each robot's local map • Demonstrated to produce consistent estimates	• Constraints used to generate condensed map are not ideal for factorisation • Extra cost to integrate other robot's map

Dynamic Env.	Advantages	Drawbacks
Rat-SAM + FAB-MAP (p.188)	• Allows loop closing at different time of day	• Increased map size compared to invariant features
Experience Based SLAM (p.188)	• Allows long term loop closing (days, seasons)	• Increased map size compared to invariant features

3.4 Contribution to the Literature

Of the SLAM algorithms and extensions presented in this chapter, Smoothing and Mapping is of particular interest for this thesis. In Section 3.1.6, the basic concepts of SAM are explained and two popular algorithms representing the current state of the art, SLAM++ and iSAM2, are presented. Based respectively on Cholesky and QR decomposition, these methods present incremental solutions to SAM and allow for partial reordering and re-linearisation. These partial calculations consist of reordering or re-linearising nodes that will be recomputed by loop closing. Doing so avoids or delays as much as possible any

reordering or re-linearisation operations on all the columns of **A**, which is a costly batch operation.

The Hybrid Cholesky algorithm proposed in this thesis applies specifically to Cholesky decomposition and allows to recover the factor **L** after reordering all columns of **A** without the need for a batch re-computation. This relaxes the need to avoid full-reordering and provides the following advantages:

- More efficient full-reordering steps
- Truly incremental SAM implementation

In order to provide a performance advantage over the batch solution, the proposed method relies on a few conditions:

- Consistent ordering algorithms (slight changes in graph connectivity do not significantly alter the ordering)
- Localised ordering changes (The Kendal-Tau distance between the previous and desired ordering is small)

Over the next chapters, the proposed algorithm will be developed and the conditions under which it is more efficient than the batch re-ordering will be discussed in greater details. Experimental results regarding the proposed method will be presented and compared against SLAM++.

Chapter 4

Hybrid Cholesky Decomposition

In this chapter, the problems addressed in this thesis and the contributions made in the field of SAM are explored in greater details. The theoretical foundations of the contributions are explained with examples in a series of three propositions. The Hybrid Cholesky algorithm is then presented and validated by experimental trials using an assortment of SLAM datasets.

Section 4.1 recapitulates the comparative study of the previous chapter for the purpose of justifying the use of SLAM++ to implement the contributions. Section 4.2 discusses the problem tackled by this thesis and Section 4.3 presents the theory required to understand the remainder of the research. The proposed solution to the problem is presented in detail in Section 4.4. The chapter ends with a summary of contributions in Section 4.5.

Throughout this chapter, many algorithms perform operations on symmetric matrices, thus affecting both rows and columns. For notational clarity, the term column is used to refer to both rows and columns of a symmetric matrix unless specified otherwise. To align with other results found in the literature, the computational complexity quantified in terms of FLOPs refers to the number of multiplication, division and square root operations required. Multiple references are made to the works of Davis *et al.*, which constitute the foundations of the theoretical side of this thesis. These works relate to the modification of

Cholesky factors [24, 34, 40, 94] explained in Section 2.6.1 and the COLAMD algorithm [40] described in Section 2.6.2.1. The reader may refer to these sections for more details.

4.1 Algorithm Selection

Smoothing-based SLAM algorithms are selected as good candidates for this thesis because they remain efficient and effective as the number of landmarks increase [23]. This is especially important for long endurance missions, such as UAVs performing SAR operations. Considering the problem as a sparse system allows smoothing algorithms to benefit from recent progress made in computational linear algebra, such as sparse matrix manipulation, factorisation and factor modification, yielding even faster algorithms [87, 88, 95, 96]. Additionally, the graph-oriented formulation of the constraints between vehicle poses and landmarks opens the possibility of applying powerful algorithms from graph theory for analysis and processing. Using this formulation inherently provides the vehicle's path updated with the latest measurements. Access to the original SLAM graph also gives SAM methods the ability to re-linearise at any time, thus avoiding inconsistencies encountered by Bayes filter based algorithms in the face of changing linearisation points [59]. Furthermore, it is possible to change the structure of the graph to correct erroneous data associations. This allows, for example, a UAV having previously identified a single feature as two distinct landmarks to merge these past observations when the error is detected.

For the purpose of this thesis, the SLAM++ program by Polok *et al.* [95] is used as a benchmark and a basis for the implementation of the methods proposed. This algorithm was selected over g2o [96] and iSAM [87] due to its higher performance in terms of execution speed. iSAM2 [88] could not be considered because the source code was not available at the time of writing this manuscript. Additionally, the underlying Cholesky factorisation used by SLAM++ is an important factor in this selection as it exhibits interesting properties well discussed in the literature which are leveraged in the contributions.

4.2 Problem Description

Although smoothing-based SLAM algorithms are very efficient, they also present some disadvantages as outlined in Table 3.3 on page 78. One such disadvantage lies in the absence of readily available second order statistics (variance and covariance) describing the accuracy of the vehicle poses and landmark information. These are available in the form of the covariance matrix $\mathbf{P}$ in EKF-based solutions and are often used in practice to determine the quality of a given landmark and detect possible data association errors. This limitation has been addressed by Ila *et al.* [97] who propose a method to efficiently recover covariance values from the graph. They demonstrate that their method is an order of magnitude faster than alternatives and allows tasks such as data association in SAM implementations to benefit from covariance information.

It is known that the order of the columns in the Jacobian matrix $\mathcal{A}$ is of prime importance for SAM efficiency (See Section 2.6.2). Since SLAM is often solved on-line, the desired column ordering varies over time as the vehicle navigates the environment and new observations are added to the graph. This requires a new column ordering to be periodically calculated which, with the current methods used in SAM, cannot be done incrementally. This, in turn, requires a computationally expensive re-computation of the solution.

Both the ordering calculation and the factor re-computation are batch algorithms that carry an increasingly high cost as the number of landmarks increases. Recent methods have proposed incremental reordering schemes to maintain matrix sparsity while avoiding batch reorderings [87, 88, 95]. These methods have succeeded in partially mitigating the cost associated with batch column ordering. However, the incrementally calculated ordering is not guaranteed to be optimal since only a subset of the problem is considered.

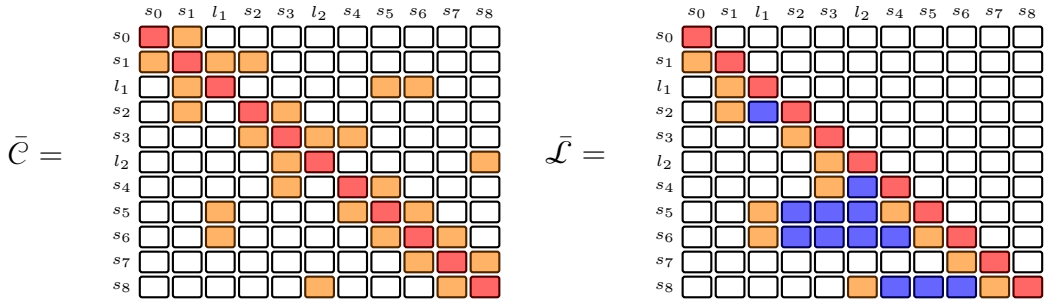
In this thesis, the computational complexity related to the batch nature of column reordering is considered and a method to provide such capability in an incremental but numerically equivalent manner is proposed.

Case Study 4.2.1. *Consider a UAV identifying survivors denoted by l_1 and l_2 as it flies over the scene of a naval accident as illustrated in the Artificial_11 dataset on page 16. Following the SAM methodology, the UAV's navigation system adds a new column to matrix $\mathcal{A}$ for each UAV position and newly observed survivors while rows are added for each UAV position and survivors observed regardless of whether they were previously observed. The graph is represented as the linear system $\mathcal{A}\mathbf{x} + \mathbf{b} = 0$. To find the least square solution to this system, the matrix $\mathcal{C} = \mathcal{A}^T\mathcal{A}$ is calculated (see Section 2.3.1.4 for details).*

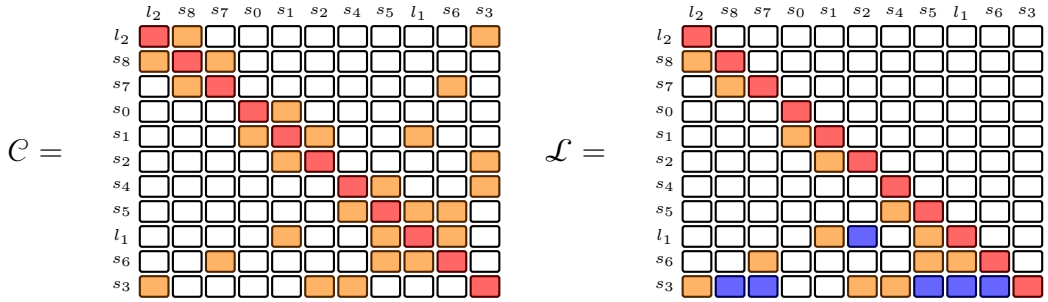
In the case where the UAV and survivor positions are stored in the order in which they are encountered, the matrix $\mathcal{C}$ obtained is illustrated in Figure 4.1a and its Cholesky factor $\mathcal{L}$ is shown to have 12 fill-ins. The factor $\mathcal{L}$ contains a total of 36 non-zero values in a 11×11 matrix which can be stored in 664 bytes on a 64-bit system if a compressed column format is used (11×64-bit column indices array, 36×64-bit row indices array and 36×64-bit value array).

At this point however, the ideal ordering obtained with COLAMD has a Cholesky factor containing 6 fill-ins as illustrated in Figure 4.1b, for a total of 30 non-zero values in a 11×11 sparse matrix. This can be stored in 568 bytes on a 64-bit system if a compressed column format is used (11×64-bit column indices array, 30×64-bit row indices array and 30×64-bit value array), corresponding to a memory saving of 14% over the chronological ordering. However, to take advantage of this reduced fill-in, the UAV must update $\mathcal{C}$ to reflect the change of ordering and the factor $\mathcal{L}$ must be recomputed. According to (2.31) This can be accomplished at the one time cost of approximately 96 FLOPs.

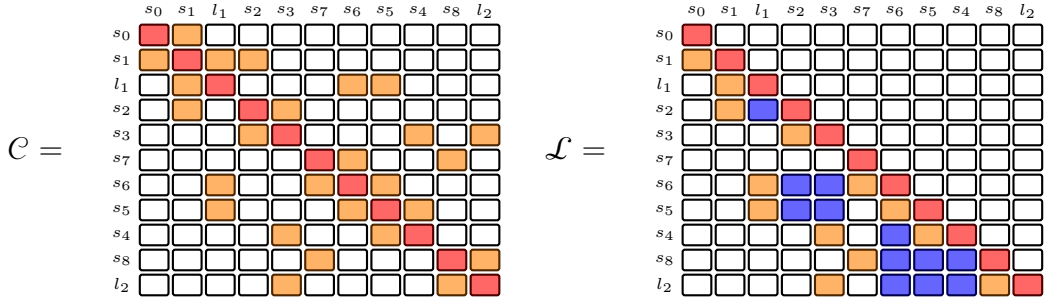
As a compromise, present methods allow the UAV's SAM algorithm to specify a section of the matrix that should remain in the current ordering while the remainder is reordered as efficiently as possible. Such ordering is obtained using constrained COLAMD (CCOLAMD). In this case, since the UAV is at s_8 and observing survivor l_2, the columns l_2 through s_8 get recomputed to process the loop closing. The navigation system will take advantage of this and reorder these column while leaving the remaining ones in the chrono-



(a) Chronological ordering: Poses and landmarks are in the order they are encountered. The Cholesky factor contains 12 fill-ins.



(b) Ideal ordering: The fill-reducing ordering obtained by COLAMD when the vehicle is at position s_8. The Cholesky factor contains 6 fill-ins.



(c) Constrained ordering: In an attempt to save time, constrained COLAMD (CCOLAMD) is used to reorder only the loop that is closed by observing l_2 from s_8. The Cholesky factor contains 12 fill-ins.

Figure 4.1: The $\mathcal{C}$ matrix and its Cholesky factor $\mathcal{L}$ for various types of ordering of the *Artificial_11* dataset. (□ - zero, ■ - diagonal, ■ - non-diagonal, ■ - fill-in)

logical ordering (Figure 4.1c). In this case, the Cholesky factor contains as many fill-ins as the chronological ordering (12) and does not allow any space savings but, in most practical situations, the number of fill-ins obtained is between that of the chronological and COLAMD orderings. Since only the columns l_2 to s_8 of the matrix are recomputed, it can be done at the reduced cost of approximately 70 FLOPs but does not benefit from the same savings as the full reordering.

4.3 SLAM++ Background

SLAM++ was proposed as a more efficient SAM algorithm than iSAM. It uses batch operations only when required instead of periodically [90]. SLAM++ also introduces three different incremental update methods which are used in different situations to maximise efficiency. Compared to iSAM's QR decomposition and QR factor update, SLAM++'s non-linear least square solver, optimised for block matrices, has a slightly lower execution time and is particularly well suited for the sparse matrix structure encountered in SLAM problems. Although it supports non-linear optimisation methods, the linear version is used in this thesis as a basis for implementing the proposed method and obtaining simulation results. Algorithm 4.1 is a simplified illustration of the SLAM++ process.

Algorithm 4.1 SLAM++

Input: new measurements
Input/Output: graph
Input/Output: $\mathcal{A}$ Measurement Jacobians
Input/Output: $\mathbf{b}$ Measurement Error
Input/Output: $\mathcal{L}$ Cholesky factor of $\mathcal{A}^T\mathcal{A}$
Input/Output: $\mathbf{x}$ Vehicle and landmark states

1: **for all** $i \in$ new measurements **do**
2: add i to the graph
3: calculate derivatives $\mathbf{G}_i$, $\mathbf{F}_i$, $\mathbf{H}_i$ and $\mathbf{J}_i$ (See (3.17), (3.18) and (3.19))
4: add derivatives to new row of $\mathcal{A}$ (See Figure 3.8a)
5: add measurement error to new row of $\mathbf{b}$
6: **if** non-zero density of $\mathcal{L} > 0.02$ **then**
7: $\mathcal{P} \leftarrow$ COLAMD($\mathcal{A}$)
8: $\mathcal{L} \leftarrow$ Cholesky($\mathcal{P}^T\mathcal{A}^T\mathcal{A}\mathcal{P}$)
9: **else**
10: $\bar{\mathcal{L}} \leftarrow$ incremental_cholesky_update($\mathcal{L}, \mathcal{A}, \mathcal{P}, \mathbf{b}$) (See Figure 4.2)
11: $\mathcal{L} \leftarrow \bar{\mathcal{L}}$
12: **end if**
13: $\mathbf{x} \leftarrow$ forward_backward_substitute($\mathcal{L}, \mathcal{P}^T\mathcal{A}^T\mathbf{b}, \mathbf{x}$)
14: **end for**

The implementation of the incremental Cholesky update function used in Algorithm 4.1 is such that different approaches are used to perform the update depending on the size and density of various sub-matrices. SLAM++ uses a value of 0.02 as a non-zero density threshold for triggering a full reordering [20]. This value was obtained experimentally by the developers as an indicator for when SLAM problems would benefit from a new column ordering. If the matrix $\mathcal{C} = \mathcal{A}^T\mathcal{A}$ and the vector $\eta = \mathcal{A}^T\mathbf{b}$ are modified such that,

$$\mathcal{C} = \begin{bmatrix} \mathcal{C}_{11} & \mathcal{C}_{12} \\ \mathcal{C}_{21} & \mathcal{C}_{22} + \Omega \end{bmatrix} \qquad \eta = \begin{bmatrix} \eta_1 \\ \eta_2 + \omega \end{bmatrix}$$

as illustrated in Figure 4.2a, the factor $\bar{\mathcal{L}}$ and the intermediate result $\bar{\mathbf{d}}$ can be written as

$$\bar{\mathcal{L}} = \begin{bmatrix} \mathcal{L}_{11} & \mathcal{L}_{12} \\ \mathcal{L}_{21} & \bar{\mathcal{L}}_{22} \end{bmatrix} \qquad \bar{\mathbf{d}} = \begin{bmatrix} \mathbf{d}_1 \\ \bar{\mathbf{d}}_2 \end{bmatrix}$$

where d is such that $\bar{\mathcal{L}}\mathbf{d} = \eta$. The submatrix $\bar{\mathcal{L}}_{22}$ can be calculated by applying Cholesky factorisation to a subset of the problem (Line 10 of Algorithm 4.1). This is called resumed Cholesky and can be done in one of two ways:

$$\bar{\mathcal{L}}_{22} = chol(\bar{\mathcal{C}}_{22} - \mathcal{L}_{21}\mathcal{L}_{21}^T) \tag{4.1}$$

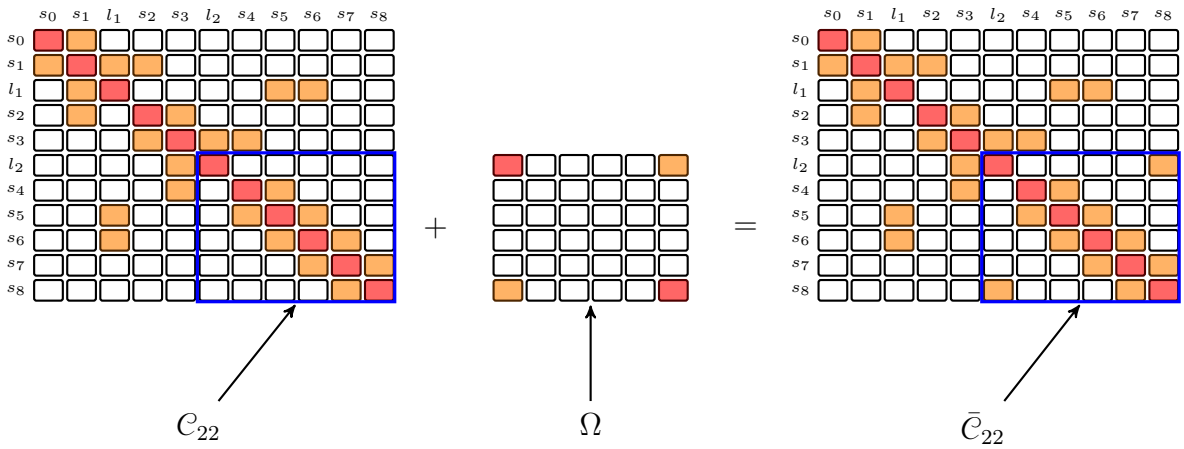
$$\bar{\mathcal{L}}_{22} = chol(\mathcal{L}_{22}\mathcal{L}_{22}^T + \Omega) \tag{4.2}$$

The choice of which equation to use depends on the size of the update. Equation (4.1) (illustrated in Figure 4.2b) is used for large updates since, in such cases, $\mathcal{L}_{21}$ is expected to be sparse while (4.2) (illustrated in Figure 4.2c) is typically used for small updates due to the lower density expected of $\mathcal{L}_{22}\mathcal{L}_{22}^T$. Note that although $\bar{\mathcal{L}}$ is used as a result at Line 10, the practical implementation of the resumed Cholesky implementation is executed in-place and overwrites the values of $\mathcal{L}$. The update to the intermediate result $\bar{\mathbf{d}}$ can be calculated by

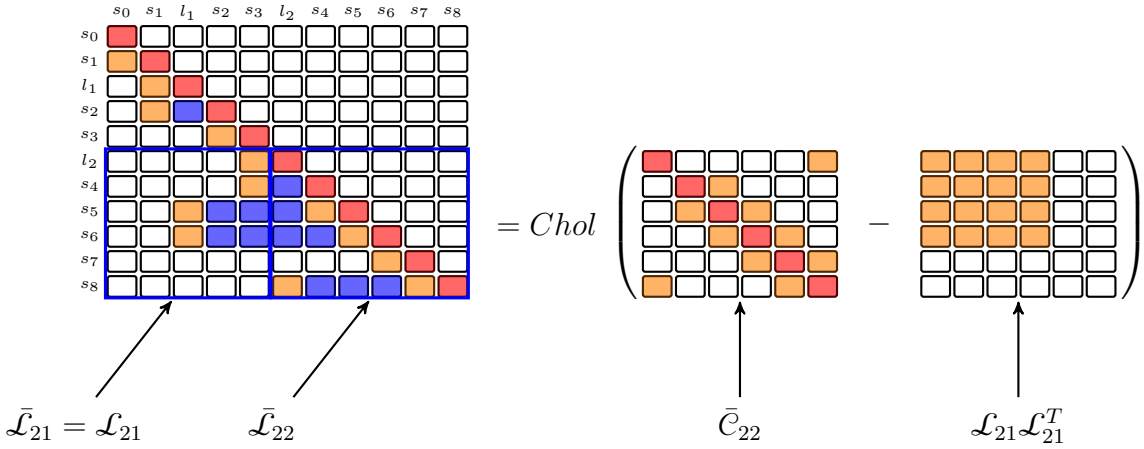
$$\bar{\mathbf{d}}_2 = \bar{\mathcal{L}}_{22}\ (\bar{\eta}_2 - \mathcal{L}_{21}\mathbf{d}_1) \tag{4.3}$$

For information regarding the optimised block matrix Cholesky factorisation, readers can refer to [95] while the incremental update scheme of SLAM++ is described in more detail in [90].

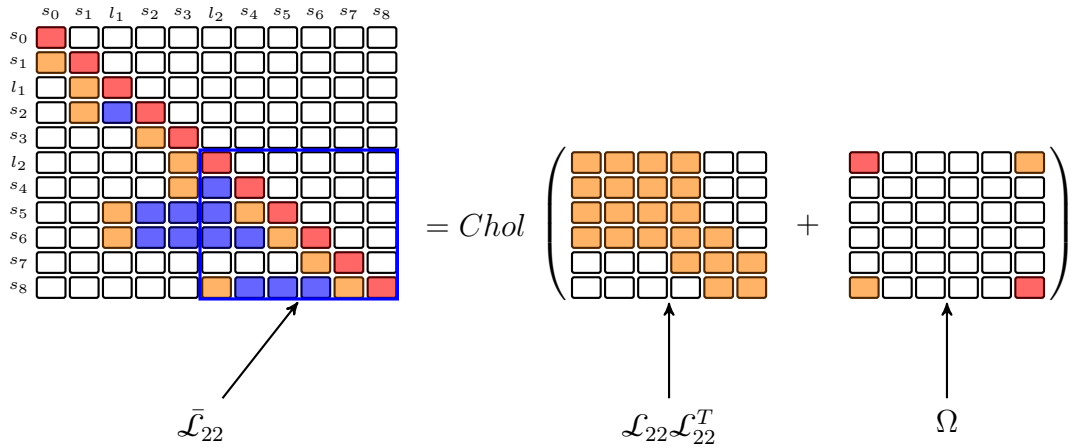
In cases where columns need to be reordered due to the density threshold being exceeded (Line 6 of Algorithm 4.1), a new ordering is obtained (Line 7) and Cholesky decomposition is performed (Line 8). Both the ordering and Cholesky decomposition are batch operations, with the latter being significantly more expensive. This thesis aims to address the



(a) The matrix $\mathcal{C}$ is updated to include the loop closing.



(b) The updated factor $\bar{\mathcal{L}}$ is obtained from $\mathcal{L}$ using the Lambda update formulation of (4.1).



(c) The updated factor $\bar{\mathcal{L}}$ is obtained from $\mathcal{L}$ using the Omega update formulation of (4.2) where $\mathcal{L}_{22}$ is lower right block of $\mathcal{L}$ illustrated in Figure 4.1a

Figure 4.2: When the robot navigates the *Artificial_11* dataset and observes landmark l_2 from pose s_8, a loop is closed. This figure illustrates this loop closure and the various options available to SLAM++ to solve the problem efficiently. (☐ - zero, ■ - diagonal, ■ - non-diagonal, ■ - fill-in)

computational challenges of calculating the Cholesky factor after a change of ordering by using a novel incremental algorithm.

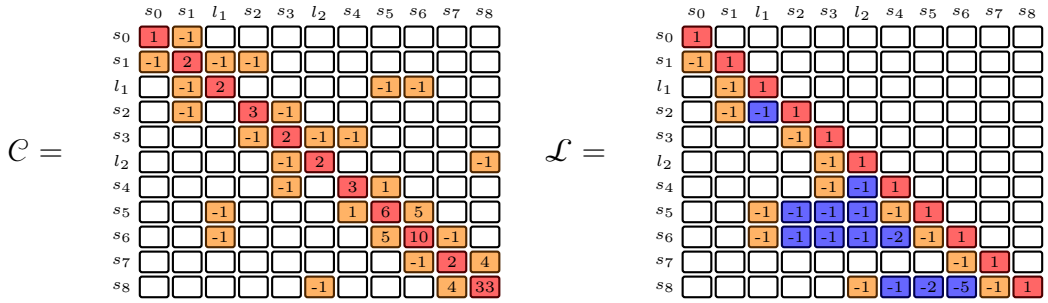
4.4 Proposed Solution: Hybrid Cholesky

As illustrated in the previous case study (page 86), a compromise is often made between the quality of the ordering and the density of the factor $\mathcal{L}$. It is often desired, however, to maintain the density below a given threshold. This may require the computation of a new ordering and new factor $\mathcal{L}$ (Line 8 in Algorithm 4.1). To circumvent the necessity of proceeding with a complete factor re-computation after reordering the columns of $\mathcal{A}$, a new algorithm able to perform incremental reordering directly on the factor $\mathcal{L}$ is proposed. The theoretical foundation of the method is presented in the form of three propositions in Section 4.4.1. Later, in Section 4.4.2, these propositions are used as building blocks for the Factor Recovery algorithm. Sections 4.4.3 and 4.4.4 show how Factor Recovery is integrated with Cholesky decomposition to obtain the proposed algorithm.

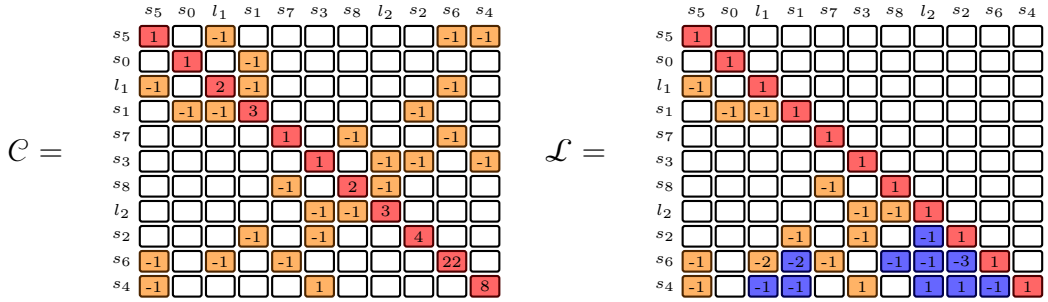
4.4.1 Column Displacement in $\mathcal{C}$

In the following section, the effect on $\mathcal{L}$ of moving a given column of the sparse symmetric positive definite matrix $\mathcal{C}$ to another position is analysed. Three cases forming a basis for arbitrary changes in the ordering of the columns of $\mathcal{C}$ and associated effect on $\mathcal{L}$ are identified and formalised into propositions. Each proposition is presented with a corresponding case study illustrated using one of two different orderings of the *Artificial_11* dataset shown in Figure 4.3a and Figure 4.3b. Note that in addition to the colour convention used in other figures, numerical values have been selected for the entries of $\mathcal{C}$ to illustrate numerical changes taking place in the Cholesky factor $\mathcal{L}$. The values themselves do not represent any application-related quantities but were rather selected to obtain integer entries in the

factor $\mathcal{L}$. The reader is also reminded that calligraphic notation is used in propositions that apply specifically to sparse matrices while bold is used when they apply to matrices in general.



(a) Chronological ordering



(b) METIS ordering

Figure 4.3: The $\mathcal{C}$ matrix representing the *Artificial_11* dataset using chronological (4.3a) or METIS (4.3b) ordering. Numerical values are used to show numerical changes in the factor $\mathcal{L}$. (☐ - zero, ■ - diagonal, ■ - non-diagonal, ■ - fill-in)

4.4.1.1 Permutation with Dependent Neighbouring Column

Proposition 4.4.1 considers the case where any two dependent neighbouring columns of $\mathbf{C}$ are permuted. More specifically, it shows that in the case where a given column of $\mathbf{C}$ exchanges position with an adjacent column (in dense matrices, all columns are dependent), the sequential application of the row/column deletion and addition operations [34] and associated rank 1 updates [98] simplify such that only the two columns involved are modified. An expression on calculating the modified columns of $\mathbf{L}$ is given.

Proposition 4.4.1 - Column Swapping. *Let* $\mathbf{c}_j$ *and* $\mathbf{c}_{j+1}$ *be two adjacent columns in a symmetric positive definite matrix* $\mathbf{C}$, $\mathbf{P} = [\mathbf{e}_1, \ldots, \mathbf{e}_{j+1}, \mathbf{e}_j, \ldots, \mathbf{e}_n]$ *a permutation matrix*

such that $\bar{\mathbf{C}} = \mathbf{P}^T\mathbf{C}\mathbf{P}$ *is* $\mathbf{C}$ *where columns* $\mathbf{c}_j$ *and* $\mathbf{c}_{j+1}$ *have exchanged positions. Let* $\mathbf{L}\mathbf{L}^T$ *and* $\bar{\mathbf{L}}\bar{\mathbf{L}}^T$ *be the Cholesky factors of* $\mathbf{C}$ *and* $\bar{\mathbf{C}}$*, respectively, where the overhead bar indicates terms that have been affected by the permutation. For notational simplicity, let matrices* $\mathbf{C}$ *and Cholesky factor* $\mathbf{L}$ *be partitioned as:*

$$\mathbf{C} = \begin{bmatrix} \mathbf{C}_{11} & \mathbf{c}_{12} & \mathbf{c}_{13} & \mathbf{C}_{14} \\ \mathbf{c}_{12}^T & c_{22} & c_{23} & \mathbf{c}_{42}^T \\ \mathbf{c}_{13}^T & c_{32} & c_{33} & \mathbf{c}_{43}^T \\ \mathbf{C}_{41} & \mathbf{c}_{42} & \mathbf{c}_{43} & \mathbf{C}_{44} \end{bmatrix} \qquad \mathbf{L} = \begin{bmatrix} \mathbf{L}_{11} & & & \\ \mathbf{l}_{12}^T & l_{22} & & \\ \mathbf{l}_{13}^T & l_{32} & l_{33} & \\ \mathbf{L}_{41} & \mathbf{l}_{42} & \mathbf{l}_{43} & \mathbf{L}_{44} \end{bmatrix} \tag{4.4}$$

such that column $\mathbf{c}_j$ *is* $\begin{bmatrix} \mathbf{c}_{12} & c_{22} & c_{32} & \mathbf{c}_{42} \end{bmatrix}^T$ *and* $\mathbf{c}_{j+1}$ *is* $\begin{bmatrix} \mathbf{c}_{13} & c_{23} & c_{33} & \mathbf{c}_{43} \end{bmatrix}^T$*. After columns* $\mathbf{c}_j$ *and* $\mathbf{c}_{j+1}$ *exchange positions, the modified partitioned matrices are:*

$$\bar{\mathbf{C}} = \begin{bmatrix} \mathbf{C}_{11} & \mathbf{c}_{13} & \mathbf{c}_{12} & \mathbf{C}_{14} \\ \mathbf{c}_{13}^T & c_{33} & c_{32} & \mathbf{c}_{43}^T \\ \mathbf{c}_{12}^T & c_{23} & c_{22} & \mathbf{c}_{42}^T \\ \mathbf{C}_{41} & \mathbf{c}_{43} & \mathbf{c}_{42} & \mathbf{C}_{44} \end{bmatrix} \qquad \bar{\mathbf{L}} = \begin{bmatrix} \mathbf{L}_{11} & & & \\ \mathbf{l}_{13}^T & \bar{l}_{33} & & \\ \mathbf{l}_{12}^T & \bar{l}_{32} & \bar{l}_{22} & \\ \mathbf{L}_{41} & \bar{\mathbf{l}}_{43} & \bar{\mathbf{l}}_{42} & \mathbf{L}_{44} \end{bmatrix} \tag{4.5}$$

The new Cholesky factor $\bar{\mathbf{L}}$ *can be recovered from* $\mathbf{L}$ *as follows:*

$$\bar{l}_{33} = \sqrt{l_{33}^2 + l_{32}^2} \qquad \bar{l}_{22} = l_{33}\frac{l_{22}}{\bar{l}_{33}}$$
$$\bar{l}_{32} = l_{32}\frac{l_{22}}{\bar{l}_{33}} \qquad \bar{\mathbf{l}}_{42} = \frac{\mathbf{l}_{42}l_{33} - \mathbf{l}_{43}l_{32}}{\bar{l}_{33}}$$
$$\bar{\mathbf{l}}_{43} = \frac{\mathbf{l}_{42}l_{32} + \mathbf{l}_{43}l_{33}}{\bar{l}_{33}}$$

where all other entries are the same in both factors.

Proof. The symmetric permutation $\mathbf{P}^T\mathbf{C}\mathbf{P}$ can be decomposed as a column deletion followed by a column addition on $\mathbf{C}$. The deletion operation [34] applied on the second column

affects the Cholesky factor such that the second row and column ($\mathbf{l}_{12}^T$ and $\begin{bmatrix} l_{22} & l_{32} & \mathbf{l}_{42}^T \end{bmatrix}$) are removed and the lower triangular section

$$\begin{bmatrix} l_{33} & \\ \mathbf{l}_{43} & \mathbf{L}_{44} \end{bmatrix} \tag{4.6}$$

is modified by a rank 1 update. The rank 1 update operation [98] affecting (4.6) is defined as

$$\begin{bmatrix} \bar{l}_{33} & \\ \bar{\mathbf{l}}_{43} & \hat{\mathbf{L}}_{44} \end{bmatrix} \begin{bmatrix} \bar{l}_{33} & \bar{\mathbf{l}}_{43}^T \\ & \hat{\mathbf{L}}_{44}^T \end{bmatrix} = \begin{bmatrix} l_{33} & \\ \mathbf{l}_{43} & \mathbf{L}_{44} \end{bmatrix} \begin{bmatrix} l_{33} & \mathbf{l}_{43}^T \\ & \mathbf{L}_{44}^T \end{bmatrix} + \begin{bmatrix} l_{32} \\ \mathbf{l}_{42} \end{bmatrix} \begin{bmatrix} l_{32} & \mathbf{l}_{42}^T \end{bmatrix} \tag{4.7}$$

where $\hat{\mathbf{L}}_{44}$ corresponds to an intermediate result. We now have

$$\bar{\mathbf{C}} = \begin{bmatrix} \mathbf{C}_{11} & \mathbf{c}_{13} & \mathbf{C}_{14} \\ \mathbf{c}_{13}^T & c_{33} & \mathbf{c}_{43}^T \\ \mathbf{C}_{41} & \mathbf{c}_{43} & \mathbf{C}_{44} \end{bmatrix} \qquad \bar{\mathbf{L}} = \begin{bmatrix} \mathbf{L}_{11} & & \\ \mathbf{l}_{13}^T & \bar{l}_{33} & \\ \mathbf{L}_{41} & \bar{\mathbf{l}}_{43} & \hat{\mathbf{L}}_{44} \end{bmatrix}$$

By evaluating the two upper blocks of (4.7) and solving the equations obtained for $\bar{l}_{33}$ and $\bar{\mathbf{l}}_{43}$, it can be shown that

$$\bar{l}_{33} = \sqrt{l_{33}^2 + l_{32}^2} \qquad \bar{\mathbf{l}}_{43} = \frac{\mathbf{l}_{42} l_{32} + \mathbf{l}_{43} l_{33}}{\bar{l}_{33}} \tag{4.8}$$

Applying the row/column addition operation [34] to put the removed column in its new position, we obtain

$$\bar{\mathbf{C}} = \begin{bmatrix} \mathbf{C}_{11} & \mathbf{c}_{13} & \mathbf{c}_{12} & \mathbf{C}_{14} \\ \mathbf{c}_{13}^T & c_{33} & c_{32} & \mathbf{c}_{43}^T \\ \mathbf{c}_{12}^T & c_{23} & c_{22} & \mathbf{c}_{42}^T \\ \mathbf{C}_{41} & \mathbf{c}_{43} & \mathbf{c}_{42} & \mathbf{C}_{44} \end{bmatrix} \qquad \bar{\mathbf{L}} = \begin{bmatrix} \mathbf{L}_{11} & & & \\ \mathbf{l}_{13}^T & \bar{l}_{33} & & \\ \bar{\mathbf{l}}_{12}^T & \bar{l}_{32} & \bar{l}_{22} & \\ \mathbf{L}_{41} & \bar{\mathbf{l}}_{43} & \bar{\mathbf{l}}_{42} & \bar{\mathbf{L}}_{44} \end{bmatrix} \tag{4.9}$$

where the bar identifies entries that may be modified. From [34], we obtain

$$\begin{bmatrix} \mathbf{c}_{12} \\ c_{32} \end{bmatrix} = \begin{bmatrix} \mathbf{L}_{11} & \\ \mathbf{l}_{13} & l_{33} \end{bmatrix} \begin{bmatrix} \bar{\mathbf{l}}_{12} \\ \bar{l}_{32} \end{bmatrix} \tag{4.10}$$

$$\bar{l}_{22} = c_{22} - \begin{bmatrix} \mathbf{l}_{12} & l_{32} \end{bmatrix} \begin{bmatrix} \mathbf{l}_{12} \\ l_{32} \end{bmatrix} \tag{4.11}$$

$$\mathbf{l}_{42} = \left(\mathbf{c}_{42} - \begin{bmatrix} \mathbf{L}_{41} & \bar{\mathbf{l}}_{43} \end{bmatrix} \begin{bmatrix} \mathbf{l}_{12}\bar{l}_{32} \end{bmatrix} \right) / \bar{l}_{22} \tag{4.12}$$

$$\bar{\mathbf{L}}_{44}\bar{\mathbf{L}}_{44} = \hat{\mathbf{L}}_{44}\hat{\mathbf{L}}_{44}^T - \bar{\mathbf{l}}_{42}\bar{\mathbf{l}}_{42}^T \tag{4.13}$$

Solving (4.10) for $\mathbf{l}_{12}$ gives

$$\bar{\mathbf{l}}_{12} = \mathbf{l}_{12}.$$

l_{22} and $\mathbf{l}_{42}$ are obtained directly by simplifying (4.11) and (4.12) respectively and thus

$$\bar{l}_{22} = l_{33}\frac{l_{22}}{\bar{l}_{33}} \qquad \bar{\mathbf{l}}_{42} = \frac{\mathbf{l}_{42}l_{33} - \mathbf{l}_{43}l_{32}}{\bar{l}_{33}}$$

Substituting the equation for $\hat{\mathbf{L}}_{44}\hat{\mathbf{L}}_{44}^T$ of (4.7) in (4.13) leads to

$$\bar{\mathbf{L}}_{44}\bar{\mathbf{L}}_{44}^T = \mathbf{L}_{44}\mathbf{L}_{44}^T + \mathbf{l}_{43}\mathbf{l}_{43}^T + \mathbf{l}_{42}\mathbf{l}_{42}^T - \bar{\mathbf{l}}_{43}\bar{\mathbf{l}}_{43}^T - \bar{\mathbf{l}}_{42}\bar{\mathbf{l}}_{42}^T,$$

from which one can obtain

$$\bar{\mathbf{L}}_{44}\bar{\mathbf{L}}_{44}^T = \mathbf{L}_{44}\mathbf{L}_{44}^T,$$

thus $\mathbf{L}_{44}$ does not change. ■

By analysing the equations presented in Proposition 4.4.1, an analysis of the computational complexity required can be done. The entries $\bar{l}_{22}$, $\bar{l}_{33}$ and $\bar{l}_{32}$, require 6 FLOPs when taking into account the shared intermediate results. The columns $\bar{\mathbf{l}}_{43}$ and $\bar{\mathbf{l}}_{42}$ can each be calculated at a cost of 3 FLOPs per non-zero entry. The total cost of swapping two dependent and neighbouring columns can be obtained by

$$3\left(|\mathbf{L}_i| + |\mathbf{L}_{i+1}| - 1\right) \tag{4.14}$$

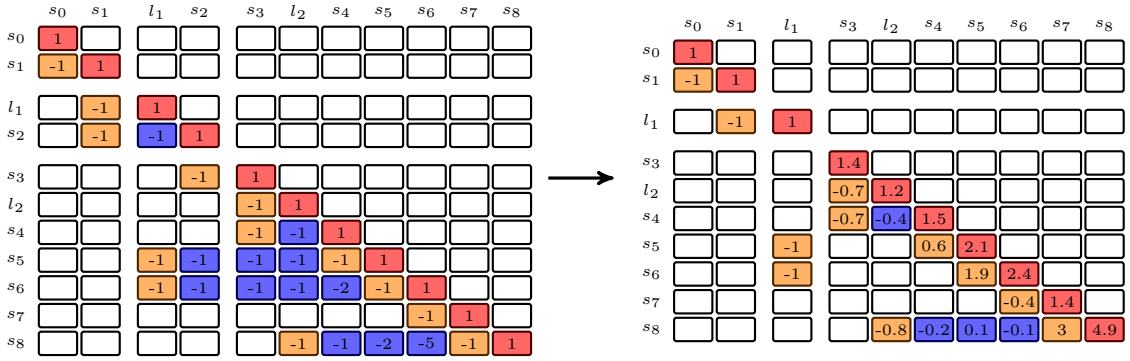
Case Study 4.4.1 - Vehicle in an Urban Environment. *Consider the case of the Artificial_11 dataset where a vehicle is navigating an urban environment using lampposts as landmarks. Assume that the vehicle's SAM algorithm is storing the poses and landmarks chronologically. After the second lamppost l_2 is observed for a second time at pose s_8, the robot stops for a moment to execute a non-navigation task.*

The state of the vehicle's SAM solver at this time is shown in Figure 4.3a. We will consider the performance of the solver's state in terms of storage space of $\mathcal{L}$ and floating point operations required to obtain the position of the poses and landmarks. As in the previous case study, the factor $\mathcal{L}$ contains 36 non-zero values in a 11×11 matrix which can be stored using 664 bytes on a 64-bit system if compressed column format is used

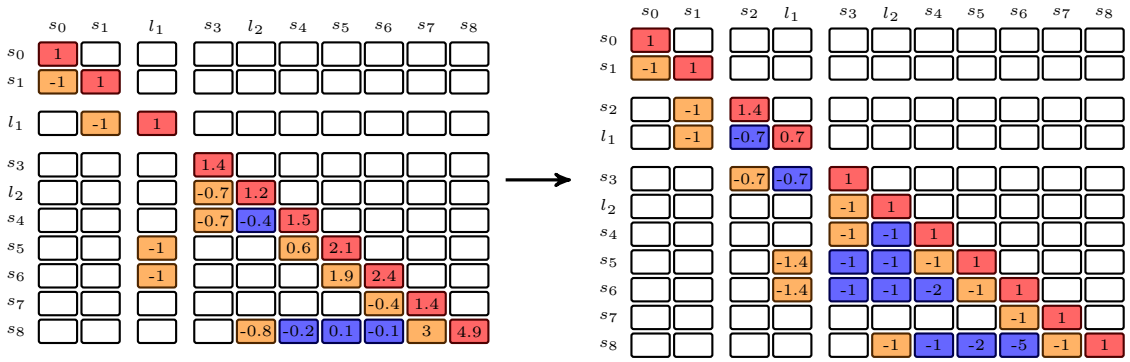
(11 × 64-bit column indices array, 36 × 64-bit row indices array and 36 × 64-bit value array). The intermediate result $\mathbf{d}$ *and the state vector* $\mathbf{x}$ *are calculated, respectively, by forward and backward substitution, which has a computational complexity linear with the number of non-zero values. More specifically, the substitution requires one floating point operation (FLOP) for each diagonal entries of* $\mathcal{L}$ *and two FLOPs for off-diagonal entries. Since the factor* $\mathcal{L}$ *contains 36 non-zero values including 11 diagonal elements, this yields a cost of 61 FLOPs for each of the forward and backward substitutions. A total of 122 FLOPs is thus required to calculate the position of the lamppost and all past poses and landmarks.*

The vehicle's SAM algorithm detects that a better order in which the variables can be solved is available. In practice, this is done by comparing the density of $\mathcal{L}$ *with a experimentally defined threshold and using COLAMD to obtain a new ordering. In this case however, let's assume the vehicle's SAM algorithm knows that savings can be made if the linear system is solved for* s_2 *before* l_1*. Current methods would execute this operation by recalculating the factor* $\mathcal{L}$ *starting at column* s_2 *which, according to* (2.31)*, would take approximately 63 FLOPs. This can also be done as a two step process by first removing* s_2 *(Figure 4.4a) and inserting it before* l_1 *(Figure 4.4b). It can be seen, however, that only column* s_2 *and* l_1 *are affected. Proposition 4.4.1, takes advantage of the local nature of the changes and proposes a method to calculate only the affected columns. By calculating the number of operations in the equations presented in the Proposition, the cost of execution is found to be 25 FLOPs, representing a saving of* 40%*.*

The state of the vehicle's SAM solver after the columns have been exchanged is shown on the right of Figure 4.4b. The factor $\mathcal{L}$ *now contains 35 non-zero values in a* 11×11 *matrix which, in compressed column format on a 64-bit system, can be stored using 648 bytes which represents memory savings of* 2%*. The substitution used to obtain the intermediate result* $\mathbf{d}$ *and the state vector* $\mathbf{x}$ *now require 59 FLOPs for each of the forward and backward substitutions. A total of 118 FLOPs is thus required, representing computational savings*



(a) Column s_2 is removed from $\mathcal{C}$. All following columns of the factor are affected.



(b) Column s_2 is reinserted in $\mathcal{C}$ before column l_1. All following columns except l_1 return to their original values. The factor has one less non-zero entry.

Figure 4.4: Changes in the Cholesky factor $\mathcal{L}$ of matrix $\mathcal{C}$ (Figure 4.3) as column s_2 is removed from its original position and inserted before l_1. (☐ - zero, ■ - diagonal, ■ - non-diagonal, ■ - fill-in)

of 4%. Note that these savings are applicable not only for pose s_8 but for all future poses as well.

4.4.1.2 Moving over Independent Columns

While Proposition 4.4.1 is valid for both dense and sparse matrices, significant savings can be obtained working exclusively on sparse matrices as will be demonstrated by Proposition 4.4.2. Due to basic properties of the elimination tree and results derived by Davis *et al.* [24], a column of $\mathcal{C}$ can be moved without causing numerical changes in $\mathcal{L}$ (columns must still be exchanged accordingly) as long as it maintains its relative position between its parent and children nodes in the elimination tree.

Proposition 4.4.2 - Sparse independent column permutation. *Let j be the index of a column in a sparse symmetric positive definite matrix $\mathcal{C}$, $\mathcal{L}$ the Cholesky factor of $\mathcal{C}$, π an elimination tree on $\mathcal{L}$, $\mathcal{P}$ a permutation matrix such that $\bar{\mathcal{C}} = \mathcal{P}^T\mathcal{C}\mathcal{P}$ is matrix $\mathcal{C}$ where column j is moved to position k, and $\bar{\mathcal{L}}$ the Cholesky factor of $\bar{\mathcal{C}}$. If $\max\{\pi^{-1}(j)\} < k < \pi(j)$, then $\bar{\mathcal{L}} = \mathcal{P}^T\mathcal{L}\mathcal{P}$ and $\bar{\pi} = \pi|\pi(\pi^{-1}(j)) = k$. That is, if column j stays between its children and parent in π, then there is no numerical or structural change in $\mathcal{L}$ and the columns are simply permuted with the appropriate index updated in the elimination tree.*

Proof. From the properties of the symbolic Cholesky factorisation [24] it is known that a numerical change in $\mathcal{L}_j$ will only affect nodes in $Path(j)$ and, in Proposition 4.4.1, it was demonstrated that permuting two adjacent columns of $\mathcal{C}$ will only modify the corresponding columns of $\mathcal{L}$. Thus, if column j and $j+1$ are permuted, numerical change will not be required if $j+1 \notin Path(j)$. This can be expressed as $j+1 \neq \pi(j)$ or, equivalently, $j \neq \max\{\pi^{-1}(j+1)\}$. By applying this observation repeatedly for values from j to $k|k > j$ (case where column j is brought to higher indexes) and from $j+1$ to $k|k < j+1$ (case where column $j+1$ is brought to lower indexes), it is demonstrated that no numerical change occurs if $\max\{\pi^{-1}(j)\} < k < \pi(j)$. ■

Case Study 4.4.2 - Mapping an Agricultural Area I. *Consider the case of the Artificial_11 dataset where a robot is mapping an agricultural area and observes fence posts l_1 and l_2. Assume the vehicle's SAM algorithm stores the poses and landmarks using METIS ordering. At this point, the state of the vehicle's SAM is shown in Figure 4.3b, which yields the elimination tree displayed in Figure 4.5.*

Suppose that pose s_1 is the position of the vehicle in a hangar and can also be observed as a landmark in the same way as l_1 and l_2. Furthermore, due to its location in the operational area, the SAM algorithm predicts that the pose s_1 will be observed often, which

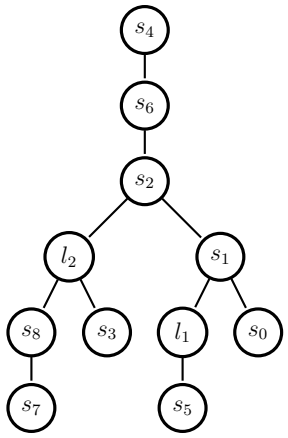


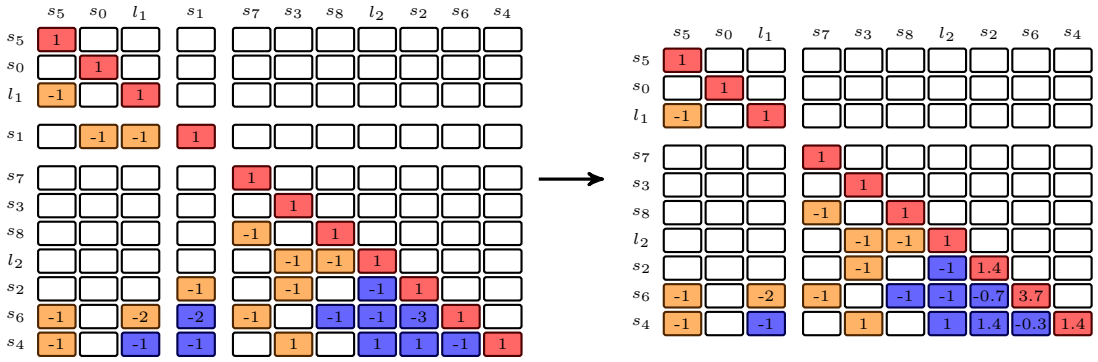
Figure 4.5: The elimination tree associated with the matrix $\mathcal{C}$ obtained when the *Artificial_11* dataset is used with METIS ordering (Figure 4.3b).

would translates to multiple loop closing operations involving s_1. According to (2.31), the current cost of updating $\mathcal{L}$ after a loop closing involving s_1 is approximately 88 FLOPs.

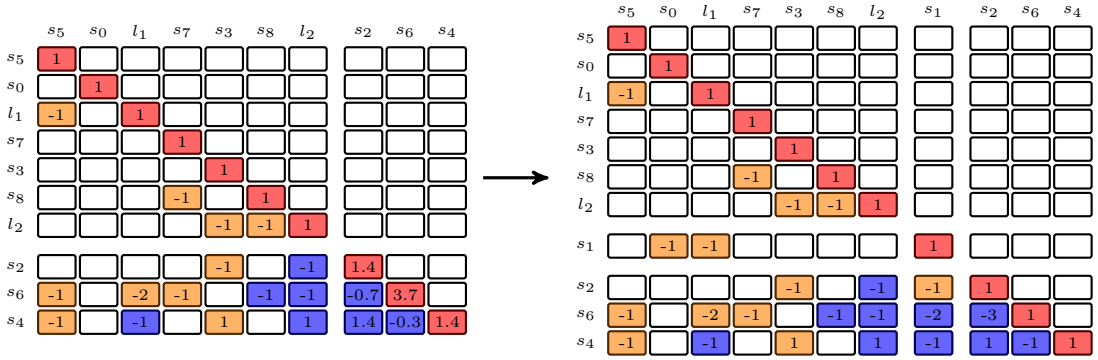
To mitigate the computational cost of these operations, the SAM algorithm moves s_1 closer to the right of $\mathcal{L}$, before pose s_2. Since no column between the current and desired position of s_1 is dependent on s_1, the move can be done at no computational cost (aside from the overhead of updating the indices). This is demonstrated in Figure 4.6 where Figure 4.6a shows that removing column s_1 affects the columns in $\mathcal{P}(s_1)$ while Figure 4.6b shows that inserting s_1 before s_2 returns the affected column to their original values.

According to (2.31), the cost of updating $\mathcal{L}$ after a loop closing involving s_1 is now approximately 32 FLOPs, representing a saving of 64%. In practice, the saving varies based on the number of loop closings and when they occur but, due to the smaller loop size, the matrices involved in the computations are typically smaller. Also, note that the elimination tree does not change since all columns involved are independent from s_1.

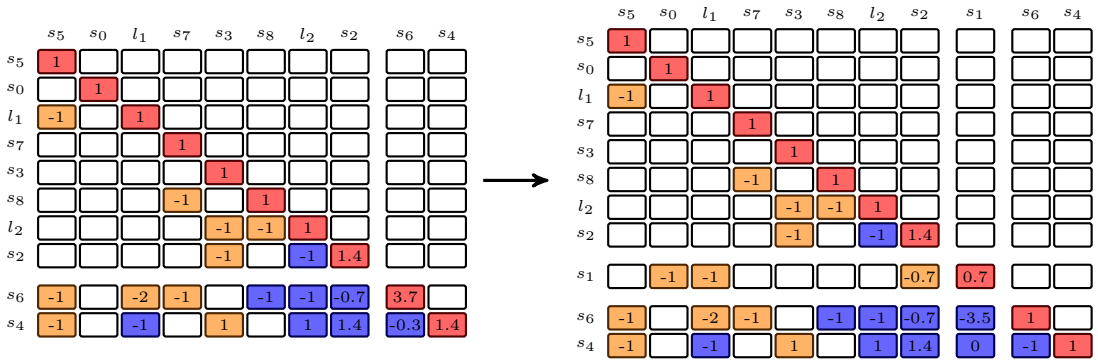
A similar result has also been demonstrated by Liu while discussing topological orderings in [25] where it is desired to find orderings that have the same fill-in but are more favourable to out-of-core execution. In our case, the problem is approached from a different direction in showing that a given manipulation does not affect the fill-in.



(a) Effect on factor $\mathcal{L}$ of removing column s_1 from $\mathcal{C}$. Columns s_2, s_6 and s_4 which are in $\mathcal{P}(s_1)$ in the elimination tree are affected.



(b) Proposition 4.4.2 example - Effect on factor $\mathcal{L}$ of inserting column s_1 before column s_2 in $\mathcal{C}$. The columns in $\mathcal{P}(s_1)$ are returned to their original values.



(c) Proposition 4.4.3 example - Effect on factor $\mathcal{L}$ of inserting column s_1 after column s_2 in $\mathcal{C}$. The columns in $\mathcal{P}(s_1)$ are returned to their original values except for s_2. The values of s_1 are also affected.

Figure 4.6: The Cholesky factor $\mathcal{L}$ of matrix $\mathcal{C}$ shown in Figure 4.3b as column s_1 is removed from its original position (a) and inserted before (b) or after (c) s_2. (☐ - zero, ☐ - diagonal, ☐ - non-diagonal, ☐ - fill-in)

4.4.1.3 Moving Over Parent/Child in the Elimination Tree

When a node is moved past its adjacent parent or child, Proposition 4.4.3 uses the basic properties of the elimination tree and multi-set representation of sparse Cholesky Factorisation [24] to show the changes required to the Cholesky Factor and Elimination tree. As described in Section 2.6, the multi-set representation of a sparse Cholesky factor, denoted $\mathcal{L}^\sharp$, contains the value and structure stored in $\mathcal{L}$ as well as a multiplicity component indicating, for a given element, how many children (dependent columns) of the element have a non-zero value at the same row.

Proposition 4.4.3 - Sparse dependent columns swapping. *Let j and $j+1$ be the indices of adjacent columns in a sparse symmetric positive definite matrix $\mathcal{C}$, $\mathcal{L}$ the Cholesky factor of $\mathcal{C}$ with multiplicity $\mathcal{L}^\sharp$, π an elimination tree on $\mathcal{L}$,*

$$\mathcal{P} = [\mathbf{e}_1, \ldots, \mathbf{e}_{j+1}, \mathbf{e}_j, \ldots, \mathbf{e}_n] \tag{4.15}$$

be a permutation matrix such that $\bar{\mathcal{C}} = \mathcal{P}^T \mathcal{C} \mathcal{P}$ is a matrix where columns j and $j+1$ are exchanged, and $\bar{\mathcal{L}}$ is the Cholesky factor of $\bar{\mathcal{C}}$ with multiplicity $\bar{\mathcal{L}}^\sharp$ and elimination tree $\bar{\pi}$.

If $\pi(j) = j+1$, then the elimination tree can be updated by

$$\bar{\pi}(l) = \begin{cases} j+1 & l \in \{\pi^{-1}(j)\} \setminus \mathcal{U}_c \\ j & l \in \{\pi^{-1}(j+1)\} \setminus \mathcal{U}_c \\ \pi(l) & \textit{otherwise} \end{cases} \tag{4.16}$$

and the structure of the new Cholesky factor can be found by

$$\bar{\mathcal{L}}^\sharp_{j+1} = \mathcal{L}^\sharp_{j+1} - \mathcal{L}_j + \sum_{i \in \mathcal{U}_c} \mathcal{L}_i \tag{4.17}$$

$$\bar{\mathcal{L}}^\sharp_{j} = \mathcal{L}^\sharp_{j} + \bar{\mathcal{L}}_{j+1} - \sum_{i \in \mathcal{U}_c} \mathcal{L}_i \tag{4.18}$$

where elements of zero multiplicity are removed and

$$\mathcal{U}_c = \{u \in \pi^{-1}(j) | \mathcal{L}_u(j+1) \neq 0\} \tag{4.19}$$

Proof. (4.16) and (4.19) can be obtained from Algorithm 2.2 where it can be seen that if $\pi(j) = j+1$, eliminating $j+1$ before j will change π from j to $j+1$ for children in $\pi^{-1}(j)$ that are also related to $j+1$ since $\mathcal{L}_l(j+1)$ will become the lowest index off-diagonal non-zero entry. Note that the index update required by exchanging columns j and $j+1$ is included in (4.16).

To compute (4.17) and (4.18), the notion of symbolic factorisation using multi-sets must be used to keep track of how many children have contributed to each non-zero element of $\mathcal{L}_{j+1}$ [24]. Since $\mathcal{L}_j$ is no longer a child of $\mathcal{L}_{j+1}$, its contribution must be subtracted while the contribution of the children inherited from $\mathcal{L}_j$ must be taken into account. This is done by (4.17) and (4.18) respectively. ■

Case Study 4.4.3 - Mapping an Agricultural Area II. *As in case study 4.4.2, consider the case where a robot produces the Artificial_11 dataset by using METIS ordering while mapping an agricultural area containing fence posts l_1 and l_2. At pose s_8, the state of the vehicle's SAM is shown in Figure 4.3b with the corresponding elimination tree displayed in Figure 4.5. Suppose that pose s_1 can also be observed as a landmark and is expected by the SAM algorithm to be observed often, which would translate to multiple loop closing operations involving s_1. According to (2.31), the current cost of updating $\mathcal{L}$ after a loop closing involving s_1 is approximately 88 FLOPs.*

To mitigate the computational cost of these operations, the SAM algorithm moves s_1 closer to the right of $\mathcal{L}$. This time, s_1 is moved after pose s_2. This can be done as a two step process by first removing column s_1 (Figure 4.6a), which affects the columns in $\mathcal{P}(s_1)$, and then insert it after s_2 (Figure 4.6b). It can be seen that all the columns affected by the first operation return to their original values except for columns s_1 and

s_2. Proposition 4.4.2 takes advantage of this to perform the operation more efficiently by directly calculating the changes on the columns. This is done in 23 FLOPs using the equations in Proposition 4.4.1. Since s_1 and s_2 are dependent, they also change position in the elimination tree (Figure 4.7).

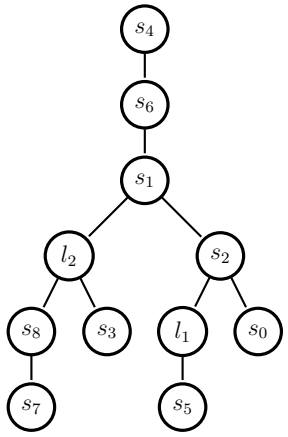


Figure 4.7: The new elimination tree associated with the matrix $\mathcal{C}$ after column s_1 is removed and inserted after s_2.

According to (2.31), the cost of updating $\mathcal{L}$ after a loop closing involving s_1 is now approximately 14 FLOPs, representing a saving of 84%. In practice, the saving varies based on the number of loop closings and when they occur but, due to the smaller loop size, the matrices involved in the computations are typically smaller.

4.4.2 Factor Recovery

In this section, the key contribution of this thesis is presented. Background information is first discussed to identify the role of the contribution in the context of the SLAM process. A mathematical explanation is then given and an Algorithm of the contribution is elaborated. This is then summarised by a case study.

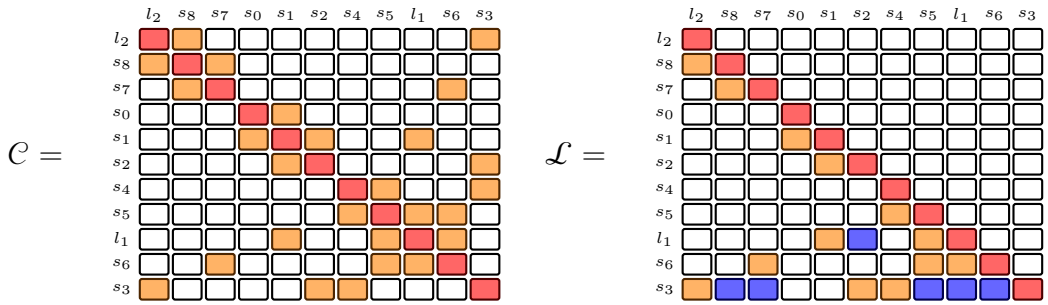
In typical dynamic least-square problems, the Jacobian matrix $\mathcal{A}$ changes every time new information is added (see Section 2.3.1.4). This translates to a change in a row/column of the information matrix $\mathcal{C}$, and requires a partial re-computation of the Cholesky factor

starting at the column that was modified (see Section 4.3 and Figure 4.2 for details). In cases where new measurements add a loop closing in the graph representing the system (observing an existing landmark), the selected column ordering may no longer yield the sparsest Cholesky factor possible since COLAMD calculates the ordering based on the number of edges incident on each node (See Section 2.6.2). At this point, a reordering of $\mathcal{C}$ and the complete re-computation of the Cholesky factor are required to obtain a sparser factor $\mathcal{L}$.

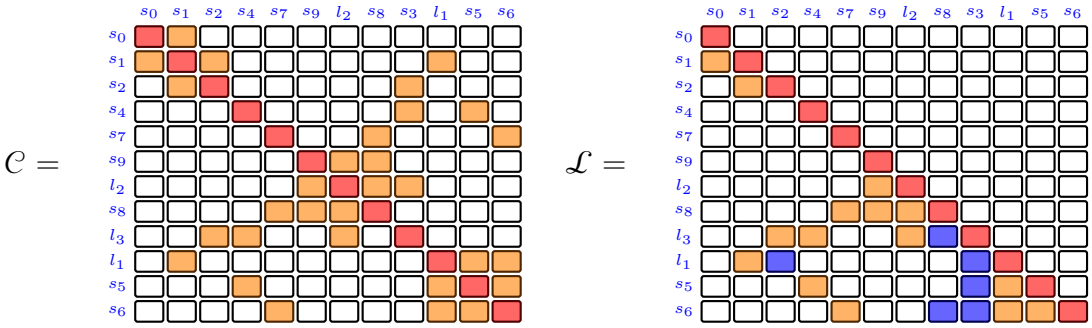
In Section 4.2, the differences between the chronological ordering and ideal ordering were explained. The constrained ordering, striking a compromise between the sparsity advantages of the ideal ordering and low computational cost of incrementing the current (chronological) ordering was discussed. When dynamic least-square problems are of a sequential nature, such as SLAM, it is also advantageous to constrain the reordering such that the last nodes remain at the end. This serves the purpose of limiting the scope of the frequent incremental changes to the most recent nodes.

Case Study 4.4.4 - Planetary Rover I. *Consider the case where a planetary rover travels along the path represented by Artificial_11 using chronological ordering and observes rock formations l_1 and l_2. While at s_8, the rover performs a variable reordering using COLAMD such that the factor $\mathcal{L}$ has the lowest possible fill-in. The state of the rover's SAM solver after the columns have been reordered is displayed in Figure 4.8a.*

The rover then continues on to a new location, say s_9, and observes l_2 again. Since s_9 is related to s_8 and the observed rock formation l_2, both on the left side of $\mathcal{L}$ (Figure 4.8a), the introduction of the new pose s_9 (added on the right side of $\mathcal{L}$) and the observation of l_2 will effectively require a re-computation of $\mathcal{L}$ since the loop to be closed involves the whole matrix. After computing yet another ordering, the state of the rover's SAM solver can be seen in Figure 4.8b. According to (2.31), the new factor is obtained at a computational cost of 74 FLOPS and, with 33 non-zero entries, can be stored in 624 bytes on a 64-bit system



(a) After pose s_8 the COLAMD ordering is computed and the factor is recalculated.



(b) When pose l_2 is observed from pose s_9 the loop closing encompases the whole factor. COLAMD is used to compute another ordering and the factor is recalculated.

Figure 4.8: Planetary rover case study. (☐ - zero, ■ - diagonal, ■ - non-diagonal, ■ - fill-in)

if a compressed column format is used (12×64*-bit column indices array,* 33×64*-bit row indices array and* 33×64*-bit value array).*

If the constrained reordering had been used, the move to pose s_9 *and observation of* l_2 *would have been significantly less computationally expensive. In fact, according to* (2.31)*, the loop closing would have been obtained at a computational cost of 14 FLOPS, representing a saving of* 90%*. However, the state of the rover's solver, illustrated in Figure 4.9 would have been worse in terms of fill ins (12 instead of 6) meaning it would require 720 bytes to store (*12×64*-bit column indices array,* 39×64*-bit row indices array and* 39×64*-bit value array) which represents an increase of* 13%*. Furthermore, the larger number of non-zero entries would lead to a longer computation time for forward and backward substitution as well as any future sparse matrix operations.*

Using constrained reordering and incremental update of the factor $\mathcal{L}$ is the preferred choice in current SLAM implementation since calculating the factor is usually the most expensive operation [90]. In doing so however, the SLAM algorithm forgoes the long term benefits brought by a sparser system.

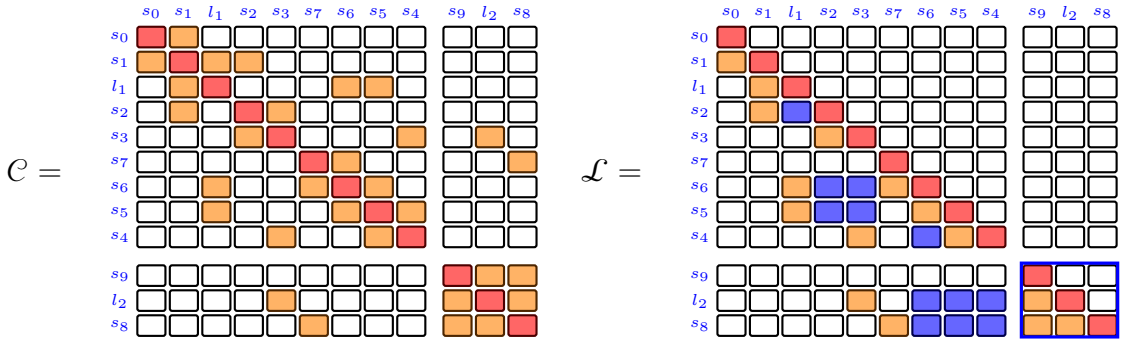


Figure 4.9: If the Constrained COLAMD ordering had been used at s_8, only a small portion of $\mathcal{L}$, outlined in blue, would have needed re-computation. (□ - zero, ■ - diagonal, ■ - non-diagonal, ■ - fill-in)

In mathematical terms, using constrained ordering with new nodes on the right side of $\mathcal{L}$ limits most changes to the top of the elimination tree or, equivalently, the lower right triangular portion $\bar{\mathcal{L}}_{22}$ [88, 90]. This is shown by the following block notation

$$\begin{bmatrix} \mathcal{C}_{11} & \mathcal{C}_{12} \\ \mathcal{C}_{12}^T & \bar{\mathcal{C}}_{22} \end{bmatrix} = \begin{bmatrix} \mathcal{C}_{11} & \mathcal{C}_{12} \\ \mathcal{C}_{12}^T & \mathcal{C}_{22} \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & \mathcal{W}_{l\times l} \end{bmatrix} = \begin{bmatrix} \mathcal{L}_{11} & 0 \\ \mathcal{L}_{12}^T & \bar{\mathcal{L}}_{22} \end{bmatrix} \begin{bmatrix} \mathcal{L}_{11} & \mathcal{L}_{12} \\ 0 & \bar{\mathcal{L}}_{22} \end{bmatrix}$$

where $\mathcal{C}_{22}$, $\mathcal{L}_{22}$, $\bar{\mathcal{C}}_{22}$ and $\bar{\mathcal{L}}_{22}$ represent, respectively, the changing part of matrices $\mathcal{C}$ and $\mathcal{L}$ before and after the change represented by $\mathcal{W}_{l\times l}$.

Information added to the SLAM solution always add edges to the graph, which requires new orderings to be computed. In existing methods, most reordering occurs when closing loops to avoid the computational cost of a separate full reordering step. In such case, an ordering and associated permutation matrix $\mathcal{P}$ is found for the columns of $\mathcal{C}$ affected by the loop closing ($[\mathcal{C}_{12}^T\ \mathcal{C}_{22}]$ and $[\mathcal{C}_{12}^T\ \mathcal{C}_{22}^T]^T$). These are reordered and a new partial factor

$\bar{\mathcal{L}}_{22}$ is obtained by applying resumed Cholesky on $\bar{\mathcal{C}}_{22}$. This is shown by

$$\begin{bmatrix} \mathcal{C}_{11} & \mathcal{C}_{12}\mathcal{P} \\ \mathcal{P}^T\mathcal{C}_{12}^T & \bar{\mathcal{C}}_{22} \end{bmatrix} = \begin{bmatrix} \mathcal{I} & 0 \\ 0 & \mathcal{P}^T \end{bmatrix} \begin{bmatrix} \mathcal{C}_{11} & \mathcal{C}_{12} \\ \mathcal{C}_{12}^T & \mathcal{C}_{22} \end{bmatrix} \begin{bmatrix} \mathcal{I} & 0 \\ 0 & \mathcal{P} \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & \mathcal{W}_{l\times l} \end{bmatrix} = \begin{bmatrix} \mathcal{L}_{11} & 0 \\ \mathcal{P}^T\mathcal{L}_{12}^T & \bar{\mathcal{L}}_{22} \end{bmatrix} \begin{bmatrix} \mathcal{L}_{11} & \mathcal{L}_{12}\mathcal{P} \\ 0 & \bar{\mathcal{L}}_{22} \end{bmatrix}$$

The ordering however, is not guaranteed to be as good as the ordering on the complete graph since only a subset of the columns are considered.

In the present section, the key contribution of this thesis is presented. It consists of an algorithm numerically equivalent to the traditional Cholesky decomposition but applying changes in ordering incrementally on the factor $\mathcal{L}$. This is done by using the results of Propositions 4.4.1, 4.4.2 and 4.4.3 to efficiently reorder $[\mathcal{L}_{11}^T\ \mathcal{L}_{12}]^T$ in conjunction with resumed Cholesky to calculate $\bar{\mathcal{L}}_{22}$. It will be assumed in the following description that reordering and loop closing events always coincide. This is usually the case in SLAM since the columns being recomputed for loop closing can be reordered at little extra cost. In cases where it is desired to change the ordering without loop closing, the resumed Cholesky step can be omitted and the proposed method is applied on the whole matrix.

The first step of the proposed reordering algorithm is to apply a constrained ordering heuristic to the full system and obtain a new reordering vector $\bar{\mathbf{p}}$. Without loss of generality, CCOLAMD [40] is used on $\mathcal{C}$ to obtain the ordering. From $\bar{\mathbf{p}}$ and the previous ordering $\mathbf{p}$, a relative ordering vector $\hat{\mathbf{p}}$ is obtained. A variant of Bubble Sort (Algorithm 4.2) is applied on $\hat{\mathbf{p}}$, with the changes duplicated on $\mathcal{L}$ using the method described in Proposition 4.4.1 to calculate new values if necessary. The results of Propositions 4.4.2 and 4.4.3 are used to maintain the non-zero structure as well as update the multiplicity and elimination tree. The sorting halts when the nodes constituting the columns $[\mathcal{L}_{11}^T\ \mathcal{L}_{12}]^T$ are in order. The remaining columns involved in the loop closing are computed by resumed Cholesky. An example is presented in Figure 4.10 and the method is summarised in Algorithm 4.3.

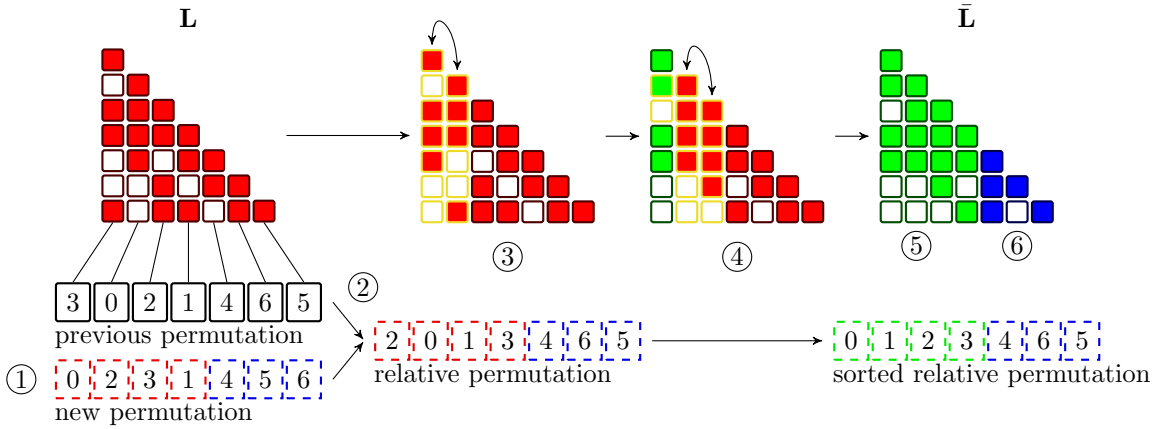


Figure 4.10: Graphical example of the Factor Recovery Algorithm. (1) A new ordering is obtained. (2) Relative ordering is obtained from previous and new ordering. (3) and (4) Perform Bubble Sort and swap columns when necessary. (5) All non-loop nodes are ordered. (6) Perform resumed Cholesky on remaining columns.

Algorithm 4.2 Bubble Sort

Input/Output: K is the array to be sorted
Input: l is the length of the array

1: $n \leftarrow 0$
2: $nn \leftarrow l - 1$
3: **while** $n < l$ **do**
4: $\quad nn \leftarrow l$
5: $\quad$ **for** $i \leftarrow l - 1 : -1 : n$ **do**
6: $\quad\quad$ **if** $K[i] < K[i-1]$ **then**
7: $\quad\quad\quad$ swap$(K[i], K[i-1])$
8: $\quad\quad\quad nn \leftarrow i$
9: $\quad\quad$ **end if**
10: $\quad$ **end for**
11: $\quad n \leftarrow nn$
12: **end while**

Algorithm 4.3 Factor Recovery Algorithm

Input: $\mathcal{L}$ is the previous Cholesky factor
Input: n_s is the number of columns in the system
Input: i_f is the index of the first column of reordered $\mathcal{C}_{22}$
Input: $\hat{\mathbf{p}}$ is a relative permutation vector of length n_s
Output: $\bar{\mathcal{L}}$ new Cholesky factor

$i_l \leftarrow 0$
while $i_l < i_f$ **do**
  $nn \leftarrow n_s - 1$
  for $i \leftarrow i_f - 1 : -1 : i_l$ **do**
    if $\mathbf{p}[i] < \mathbf{p}[i-1]$ **then**
      swap($\mathbf{p}[i], \mathbf{p}[i-1]$)
      if $\pi[i-1] = i$ **then**
        use Proposition 4.4.3
        updates non-zero values as in Proposition 4.4.1
      else
        use Proposition 4.4.2
      end if
      $nn \leftarrow i$
    end if
  end for
  $i_l \leftarrow nn$
end while
resumed Cholesky from n_f to $n_s - 1$

The use of the relatively inefficient Bubble Sort algorithm is justified in this case because the swapping operation is only defined for adjacent columns. A more efficient sorting algorithm would reduce the number of comparisons, but still require $O(n^2)$ adjacent column swaps. Therefore, it would not lead to a significant increase in the computational performance.

The part of Algorithm 4.3 responsible for reordering the columns of $[\mathcal{L}_{11}^T \mathcal{L}_{12}]^T$ has a worst case complexity of

$$O\left(\sum_{i_l=1}^{i_f-1} \sum_{i=i_f-1}^{i_l} |\hat{\mathcal{L}}_{i-1}| + |\hat{\mathcal{L}}_i|\right) \tag{4.20}$$

where $\hat{\mathcal{L}}$ is the evolving non-zero structure of the matrix. In the case where the permutations are localised, however, the permutation vector relating the new and old ordering

is close to a sorted array. For example, in the case where all columns are moved by one position the performance is given by

$$O\left(\sum_{i=i_f-1}^{1} |\hat{\mathcal{L}}_{i-1}| + |\hat{\mathcal{L}}_i|\right). \tag{4.21}$$

Between (4.20) and (4.21), one of the sum terms was removed. The cost in (4.21) is thus linear in the number of non-zeros compared to performing Cholesky on the same sub-matrix which can be seen in (2.31) to be quadratic. The relative efficiency of Algorithm 4.3 compared to Cholesky is dependent on the number of permutations required, being most efficient when $\hat{\mathbf{p}}$ is almost sorted. Case study 4.4.5 illustrates the algorithm's functionality.

Case Study 4.4.5 - Planetary Rover II. *Let us return to the situation of case study 4.4.4 where a planetary rover travels along the path represented by Artificial_11 and observes rock formations l_1 and l_2 while applying constrained reordering to the nodes involved in loop closing. After s_8, the rover continues on to a new location, say s_9, and observes l_2 again. At this point, the state of the rover's SAM system is illustrated in Figure 4.8b and can be obtained at a computational cost 90% lower than the ideal ordering requiring 74 FLOPs and stored using 13% more bytes than the ideal 624.*

To bring the number of fill-ins closer to the ideal ordering, the constrained reordering is applied such that nodes not involved in the loop closing are reordered as well. Suppose that the selected ordering algorithm returns the following order:

$$\begin{bmatrix} s_0 & s_1 & s_2 & l_1 & s_3 & s_7 & s_6 & s_5 & s_4 & \| & s_9 & l_2 & s_8 \end{bmatrix} \tag{4.22}$$

where the vertical bar separates the nodes involved in loop closing and those that are not. Algorithm 4.3 is then applied to reorder the portion of the matrix not affected by loop closing, while the remainder is updated by resumed Cholesky as in traditional SLAM implementation. The resulting matrix contains 38 non-zero entries and can be stored using

704 bytes on a 64-bit system if compressed column format is used (12×64*-bit column indices array,* 38×64*-bit row indices array and* 38×64*-bit bits value array). Algorithm 4.3 requires 21 FLOPs while the loop closing is done in 14 FLOPs, for a total of 35.*

In this case, the storage represents a 11% *increase instead of* 13% *obtained by reordering the loop variables only. The computational savings, however are* 50% *compared to* 90%*. In this particular example, the trade-off may not seem advantageous but it will be for larger SLAM matrices representing real navigation data (hundreds of landmarks and observations) as shall be illustrated later in the results section. Also note that Algorithm 4.3 may be executed in the background and its computational cost distributed over many steps (following observations), thus decreasing the computational load on the navigation system.*

4.4.3 Threshold Selection

In contrast to the Cholesky factorisation, which acts on the whole matrix $\mathcal{C}$, the Factor Recovery algorithm acts directly on $\mathcal{L}$ and only on neighbouring columns. For these reasons, their computational performance depend on different variables.

Recall from (2.31) that for Cholesky factorisation, the computational cost of computing a single column is given by

$$O\left(\frac{1}{2}\sum_{i=1:n}(|\mathcal{L}_i|-1)(|\mathcal{L}_i|+2)\right) \qquad \text{(2.31 revisited)}$$

in terms of the number of multiplications. The cost of Algorithm 4.3 however, cannot be generalised since it depends on the ordering changes required and, as swaps occur, the structure of the matrix changes due to the addition or removal of fill-ins. The computational complexity of Proposition 4.4.1 however, is known and can be calculated by (4.14).

For simplicity, let us assume a given factor $\mathcal{L}$ has a constant density d independent from the ordering such that $1/n < d < 1$ (The density is between that of a diagonal matrix

and dense matrix) and that the non-zero entries are evenly distributed in all columns. In such case, the computational cost of Cholesky factorisation is $0.5\,(d^2n^2 + dn - 2)$ FLOPs per column. Since $1/n < d$, the cost can be approximated to be $O(0.5d^2n^2)$ FLOPs. The cost of moving one column to the right position using Algorithm 4.3 can be obtained by substituting $\mathcal{L}_{i+1}$ and $\mathcal{L}_i$ with dn in (4.14) and accounting for the number of swaps. The cost would thus be

$$6rdn - 3r \tag{4.23}$$

FLOPs where r is the number of swaps required to move the column. This can be approximated to $O(6rdn)$ FLOPs. Under the assumptions made, a given column can be obtained faster using Algorithm 4.3 than Cholesky if it is displaced by r columns such that

$$r < \frac{dn}{12}. \tag{4.24}$$

The computational advantage of Algorithm 4.3 comes from the assumption that, since the structure of the SLAM problem does not change significantly over a short period of time, the ordering will also remain similar and r will be small for most columns. Although it is expected that, for SLAM problems, the new and previous orderings are closely related, this is not guaranteed as part of the COLAMD algorithm [40]. It is thus desired to obtain an estimation of the work required by both methods before their execution and compare their ratio with a threshold to decide when one should be used over the other. A similar solution is used by Supernodal Cholesky factorisation when deciding if parallel methods should be applied [29].

More details regarding the calculation of the threshold and cost functions is presented in Chapter 5. For now, lets assume that

$$\hat{u}_{Cholesky} \tag{4.25}$$

$$\hat{u}_{Recovery} \tag{4.26}$$

Represent cost functions approximating the difficulty of evaluating, respectively, the Cholesky and Factor Recovery operations. Let us further assume that a threshold α has been calculated and identifies the region where Factor Recovery is faster than Cholesky as

$$\frac{\hat{u}_{Recovery}}{\hat{u}_{Cholesky}} < \alpha \tag{4.27}$$

4.4.4 Hybrid Cholesky

This section describes how the threshold found earlier in Section 4.4.3 is used to regulate the application of the Factor Recovery algorithm and full Cholesky decomposition. The resulting algorithm, called Hybrid Cholesky, brings together the contributions of this thesis and is described here in details.

Consider a single execution step of a SLAM algorithm where a measurement is added to an $n \times n$ system represented by a symmetric positive definite matrix $\mathcal{C}$ with Cholesky factor $\mathcal{L}$ and, due to the density threshold being exceeded, a batch solution needs to be calculated. The addition of the measurement yields $\bar{\mathcal{C}}$ such that

$$\bar{\mathcal{C}}_{n\times n} = \mathcal{P}^T_{n\times n}\mathcal{C}_{n\times n}\mathcal{P}_{n\times n} + \begin{bmatrix} 0_{n-l\times n-l} & 0 \\ 0 & \mathcal{W}_{l\times l} \end{bmatrix} \tag{4.28}$$

where l is such that $n-l$ is the lowest index of values that numerically changed from $\mathcal{C}$ to $\bar{\mathcal{C}}$, $\mathcal{P}$ is the permutation matrix associated with a new ordering $\mathbf{p}$, and $\mathcal{W}$ is an $l \times l$ matrix representing the changes to $\mathcal{C}$ resulting from the new measurement added. In such a situation, $\bar{\mathcal{C}}$ can be partitioned as

$$\bar{\mathcal{C}} = \begin{bmatrix} \bar{\mathcal{C}}_{11} & \bar{\mathcal{C}}_{12} \\ \bar{\mathcal{C}}_{21} & \bar{\mathcal{C}}_{22} \end{bmatrix} \tag{4.29}$$

where numerical changes are limited to the $l \times l$ submatrix $\bar{\mathcal{C}}_{22}$ while the remainder is subject to permutations only. This is a potential case where the Factor Recovery Algorithm can be used to obtain the Factor $\mathcal{L}$ for columns 1 to $n-l$ more effectively than Cholesky decomposition. Since the computational complexity of each method depends on different variables, the cost of each method must be estimated and compared to ensure the most efficient one is used.

Since $\bar{\mathcal{C}}_{22}$ is calculated by resumed Cholesky in both cases, only the cost of factorising the first $n-l$ columns will be considered. For Cholesky, the estimated computational cost ($\hat{u}_{Cholesky}|_1^{n-l}$), is obtained by evaluating (4.25) between 1 and $n-l$. The estimated cost of Factor Recovery ($\hat{u}_{Recovery}|_1^{n-l}$), is calculated by evaluating (4.26) for the same columns. A ratio is then calculated and compared to a threshold (as in (4.27)) to select the most appropriate algorithm. If the ratio is below the threshold, then Factor Recovery (Algorithm 4.3) is used for the first $n-l$ columns and Resumed Cholesky is used for the remaining columns. Otherwise, Cholesky decomposition is used for the entire matrix $\mathcal{C}$. The resulting procedure is shown in Algorithm 4.4.

Algorithm 4.4 Hybrid Cholesky Algorithm

Input: n,$n-l$ size of $\bar{\mathcal{C}}$ and index of the last value not affected by $\mathcal{W}$
Input: α a given threshold
Input: $\mathcal{P}$ previous ordering
Input: $\bar{\mathcal{P}}$ new ordering
Input: $\bar{\mathcal{C}}$ new system matrix
Input: $\mathcal{L}$ previous Cholesky factor
Output: $\bar{\mathcal{L}}$ new Cholesky factor

1: Get $\hat{u}_{Cholesky}|_1^{n-l}$ from (4.25) using $\mathcal{L}$, $n-1$
2: Get $\hat{u}_{Recovery}|_1^{n-l}$ from (4.26) using $\mathcal{P}$, $\bar{\mathcal{P}}$, n, $n-l$
3: **if** $\frac{U_{Recovery}|_1^{n-l}}{U_{Cholesky}|_1^{n-l}} < \alpha$ **then**
4: $\quad \bar{\mathcal{L}} \leftarrow \mathcal{L}$
5: $\quad$ Update columns 1 to $n-l$ of $\bar{\mathcal{L}}$ using Algorithm 4.3
6: $\quad$ Update columns l to n of $\bar{\mathcal{L}}$ using resumed Cholesky
7: **else**
8: $\quad$ Get $\bar{\mathcal{L}}$ by regular Cholesky decomposition
9: **end if**

The Hybrid Cholesky algorithm is illustrated through the following case study.

Case Study 4.4.6 - Deep sea exploration. *Consider the case where an autonomous underwater vehicle explores the seabed along the path represented by Artificial_11 and observes wreckages l_1 and l_2 along the path. If chronological column ordering is used, the state of the SLAM solver at s_8 is represented in Figure 4.11 and the ordering used is the following:*

$$\begin{bmatrix} s_0 & s_1 & l_1 & s_2 & s_3 & l_2 & s_4 & s_5 & s_6 & s_7 & s_8 \end{bmatrix} \tag{4.30}$$

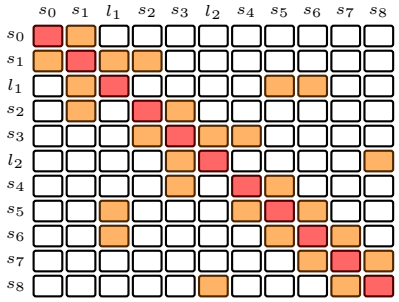
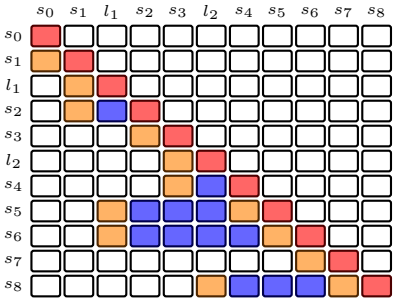
$\bar{\mathcal{C}} =$  $\bar{\mathcal{L}} =$ 

Figure 4.11: State of the SLAM solver after the underwater vehicle observes l_2 from s_8 (☐ - zero, ■ - diagonal, ■ - non-diagonal, ■ - fill-in)

While observing wreckage l_2 at s_8, the submersible performs column reordering. Assume that, using previously obtained experimental data, the threshold for the Hybrid Cholesky method has been found to be $\alpha = 0.33$. At this point, an ordering algorithm is used to obtain a column ordering that minimises the number of fill-ins in the factor $\mathcal{L}$. Suppose the ordering algorithm (See Section 2.6.2) provides the following new ordering to the Hybrid Cholesky algorithm:

$$\left[\begin{array}{ccccc|cccccc} s_0 & s_1 & s_2 & l_1 & s_3 & s_7 & s_6 & s_5 & s_4 & s_8 & l_2 \end{array}\right] \tag{4.31}$$

where columns s_7 to l_2 are computed by resumed Cholesky decomposition while columns s_0 to s_3 can be computed either with Cholesky decomposition or Factor Recovery. This new ordering is obtained from (4.30) by exchanging the position of columns l_1 and s_2.

To select the best method, a metric representative of the computational cost is estimated for both. Note that these cost metrics are unitless and do not constitute a computational cost themselves; they quantify the relative difficulty of the operation for comparison purposes. For the Cholesky decomposition, a cost metric of 70 is obtained by evaluating (4.25) *between columns corresponding to the interval from* s_0 *to* s_3 *in* (4.31)*. For Factor Recovery,* (4.26) *evaluated for the same columns yields a cost metric of 11. The ratio of the cost metrics is calculated as in* (4.27) *which yields* $11/70 = 0.157$*. In this case, since the ratio is smaller than* $\alpha = 0.33$*, the Factor Recovery method is selected.*

The choice of methods can be verified by computing the true cost of Cholesky decomposition and Factor Recovery for columns not involved in loop closing using the state of the SLAM solver after Factor Recovery illustrated in Figure 4.12.

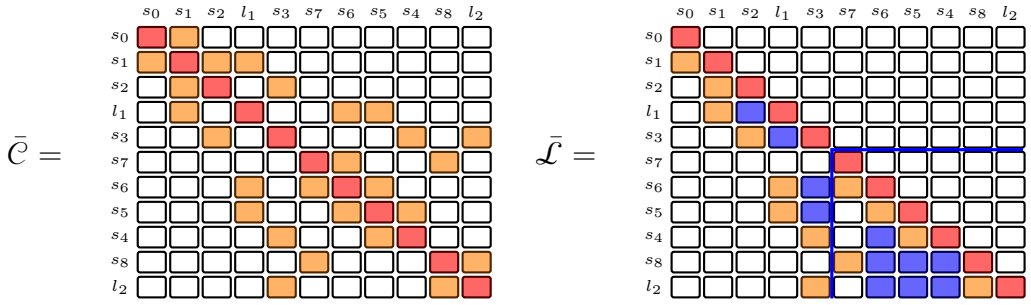


Figure 4.12: State of the SLAM solver after the columns on the left side of the blue line have been updated using Factor Recovery following the l_1 and s_2 column swapping. The columns on the right side are recomputed by resumed Cholesky. (☐ - zero, ■ - diagonal, ■ - non-diagonal, ■ - fill-in)

Using (2.31)*, the computational cost of Cholesky decomposition is found to be 35 FLOPs. On the other hand, the cost of Factor Recovery is found to be 18 FLOPs using* (4.14)*, thus confirming the right method was selected. In this case, using Factor Recovery resulted in a reduction in computational cost of* 49 % *compared to Cholesky decomposition.*

Now suppose that, instead of (4.31)*, the ordering algorithm had provided the following new ordering to the Hybrid Cholesky algorithm:*

$$\begin{bmatrix} s_0 & s_2 & s_1 & s_3 & l_1 & | & s_7 & s_6 & s_5 & s_4 & s_8 & l_2 \end{bmatrix} \tag{4.32}$$

where columns s_7 to l_2 are computed by resumed Cholesky decomposition while columns s_0 to l_1 can be computed either with Cholesky decomposition or Factor Recovery. This new ordering is obtained from (4.30) *by moving columns s_2 and s_3 respectively two and one positions towards the left.*

For the Cholesky decomposition, evaluating (4.25) *for the columns in the interval from s_0 to l_1 in* (4.32)*, yields a cost metric of 70. For Factor Recovery,* (4.26) *is evaluated for the same columns, yielding 33. The ratio of the cost metrics calculated as in* (4.27) *yields* $33/70 = 0.471$*. In this case, the ratio is higher than the threshold of* $\alpha = 0.33$ *and thus Cholesky decomposition is selected.*

The choice of methods can be verified by computing the true cost of Cholesky decomposition and Factor Recovery for columns not involved in loop closing using the state of the SLAM after the factor is recomputed illustrated in Figure 4.13.

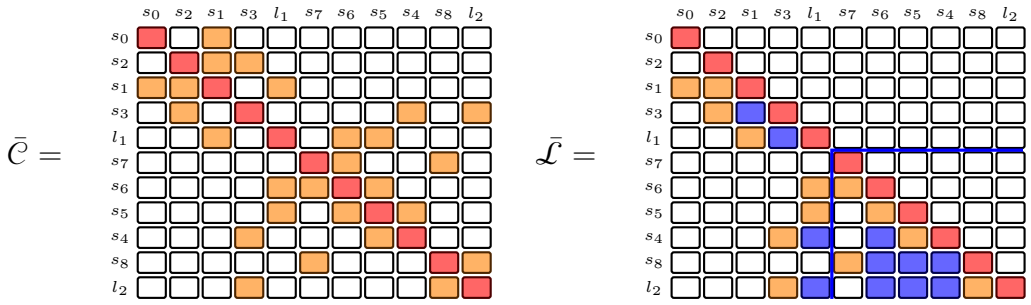


Figure 4.13: State of the SLAM solver after the underwater vehicle observes l_2 from s_8 and recomputes the factor using the regular Cholesky decomposition. The blue line delimits the columns included in the loop closing. On the left of the blue line, s_2 and s_3 moved respectively two and one positions towards the left thus requiring 5 column swappings.(☐ - zero, ■ - diagonal, ■ - non-diagonal, ■ - fill-in)

Using (2.31), the computational cost of Cholesky decomposition is found to be 35 FLOPs while the cost of Factor Recovery is found to be 57 FLOPs using (4.14). This confirms the right method was selected. In this case, avoiding the Factor Recovery algorithm prevented the computation of $\mathcal{L}$ from taking 63 % more time.

Note that $\bar{\mathcal{L}}$ in Figure 4.12 and Figure 4.13 contains the same number of fill ins. The difference in performance is due solely to the number of column swapping operations required to obtain the new ordering from the previous one.

4.5 Summary

In this chapter, the problem of Cholesky factor recovery after column reordering was discussed. First, Section 4.4.1 demonstrated that, in the case where a column of a symmetric matrix is moved, significant savings can be made in recovering the factor $\mathcal{L}$ compared to column addition and deletion operations [34]. It was shown that in such case, only the columns of $\mathcal{L}$ that lie between the original position and the final position in the elimination tree are affected instead of the entire path to the root.

Section 4.4.2 presented an algorithm that combines the propositions of Section 4.4.1 with resumed Cholesky. The proposed algorithm extends upon existing SLAM algorithms by allowing, under certain conditions, efficient full reordering during loop closing steps without the cost of recomputing $\mathcal{L}$ using the regular Cholesky factor.

Section 4.4.3 explored the conditions affecting the performance of the algorithm presented in Section 4.4.2 and compared its performance with full Cholesky factorisation for reordering steps. A way to select the fastest method was proposed. The resulting algorithm was presented in Section 4.4.4 and is expected to provide significant time savings when compared to the traditional approach as will be demonstrated in Chapter 5.

Chapter 5

Results and Discussion

In this chapter, the algorithms derived in Chapter 4 are integrated in the SLAM++ solver and tested on publicly available real and artificial datasets from the literature. The cost function and threshold, previously left undefined, are further explored and the performance in terms of execution and factorisation time are analysed and compared for multiple re-ordering periods. Finally, recommendations on the use of the Hybrid Cholesky algorithm are provided based on an analysis of the results.

The objectives of the experimentation are the following:

- Compare the efficiency of Hybrid Cholesky with traditional Cholesky in a SLAM context
- Identify the conditions in which the Hybrid Cholesky algorithm is favourable

5.1 Setup

The algorithm simulations were performed in three steps for various combinations of cost functions and thresholds to evaluate their effects on performance. The steps are as follows:

1. The threshold and cost function required for Hybrid Cholesky are identified, integrated in SLAM++, and the algorithm is run on popular SLAM Datasets.

2. The factorisation timing results are analysed to compare the Hybrid and traditional Cholesky factorisation performances in a SLAM context.

3. The total execution time is evaluated to identify potential advantages in SLAM applications.

The Hybrid Cholesky decomposition was integrated in the SLAM++ application such that, upon full reordering, Algorithm 4.4 is applied. It is desired to compare only the time spent reordering and re-factorizing based on a periodic trigger or density threshold. However, the datasets contain noisy sensor observations and, since SLAM++ solves the linearised problem, re-linearisation also causes re-factorization. To consider only re-factorisation caused by reordering, the linearisation error introduced by large loop closing must be removed. To do so, a first pass of SLAM++ is used to obtain the vehicle poses and landmark positions solution for every dataset. In light of this, the observations are corrected to remove the noise, thus ensuring that loop closing only causes small changes in pose and landmark position estimates. Since the observations are now error free, the vehicle poses and landmark positions will not require significant correction and thus changes in the linearisation point of the Jacobian matrix $\mathcal{A}$ are not required.

In the following sections, three different triggers are used for full re-computation: Density, Batch-10 and Batch-5. Density refers to SLAM++'s variable reordering routine being triggered when the density of $\mathcal{L}$ is greater than 0.02. This is the standard method used in SLAM++. Batch-10 and Batch-5 refer to the cases where reordering is triggered manually every 10 and 5 steps, respectively. This is also done in [90, 95] to compare SLAM++ with other methods offering batch solutions.

To obtain experimental results, SLAM++ is executed five times for each dataset and the timing information is written to a file. The results of the first of the five runs is discarded

as it was consistently found to be an outlier, presumably due to the extra processing the operating system must perform to start the program and allocate memory. The results of the following four runs are averaged to reduce noise. A CPU core is dedicated to the program's execution with no other processes running.

A performance gain metric is used to measure the performance of Hybrid Cholesky relative to traditional Cholesky decomposition and is defined as

$$Gain = \frac{\sum_{k \in \mathcal{V}} t_{Cholesky} - \sum_{k \in \mathcal{V}} t_{Hybrid}}{\sum_{k \in \mathcal{V}} t_{Cholesky}} \times 100\% \tag{5.1}$$

where t_{Hybrid} is the time taken by Algorithm 4.4 and $\mathcal{V}$ is the set of steps where complete reordering is performed. In the case of Density triggered reordering, this refers to the cases where Line 6 of Algorithm 4.1 on page 89 is true.

5.1.1 Datasets

The datasets used are described in this section. They have been selected to represent a variety of sizes, environments and map characteristics. The name of the datasets used as well as some of their characteristics are summarised in Table 5.1. The "Type of Environment" column indicates whether the dataset was taken indoors, outdoors or artificially created by plotting vehicle movements in a simulated environment. The "Size" column shows the number of measurements present in the dataset, while "Loop Closings" displays how many of these are linked to previously observed poses or landmarks. The "Landmarks" column contains the number of previously observed poses or landmarks in the dataset.

10K

This is an artificial dataset obtained by simulating the movement of a vehicle on a grid-based outdoor world while artificially adding noise. Loop closing observations are created

Table 5.1: Datasets Sources and Characteristics

Dataset	Size	Loop Closings	Landmarks	Type of Environment	Author	Source
10k	64311	54312	6751	artificial	G. Grisetti et al	SLAM++ [20]
City10k	20687	10688	5052	artificial	M. Kaess et al	SLAM++ [20]
CityTrees10k	14442	4343	100	artificial	M. Kaess et al.	SLAM++ [20]
CSAIL	1172	128	52	indoor	C. Stachniss	SLAM benchmarking [100]
FR079	1217	229	119	indoor	C. Stachniss	SLAM benchmarking [100]
FRH	2820	1505	1312	indoor	B, Steder et al.	SLAM benchmarking [100]
Intel	1835	895	323	indoor	D. Hähnel	SLAM++ [20]
Killian	3995	2055	720	indoor	M. Bosse and J. Leonard	SLAM++ [20]
Victoria Park	10608	3489	151	outdoor	Jose Guivant	SLAM++ [20]

when a robot comes close to a previously visited location. The process is described by Grisetti *et al.* [99]. The resulting dataset contains 64311 observations, 54312 or which are loop closings involving one of 6751 points visited more than once.

City10K

This is an artificial dataset originally produced to test iSAM. It consists of 20687 observations, 10688 of which are loop closings involving one of 5062 points visited more than once.

CityTrees10K

This is an artificial dataset originally produced to test iSAM. It differs from city10K in the way it uses landmarks for loop closing instead of previously visited points. It consists of 14442 observations, 4343 of which are loop closings involving one of 100 landmarks.

CSAIL

This dataset represents an office environment and was taken at MIT's CSAIL building using laser sensors. The pre-processed version available with SLAM++ contains 1172 observations, 128 of which are loop closings involving one of 52 points observed more than once.

FR079

This indoor dataset represents a map of building 79 at the University of Freiburg. It was obtained using a laser scanner and odometry measurements. The pre-processed version available with SLAM++ contains 1217 observations, 229 of which are loop closings involving one of 119 points visited more than once.

FRH

This dataset was obtained from indoor measurements of a floor of the Freiburg University Hospital using two SICK laser range finders, odometry and an inertial measurement unit. The pre-processed version available with SLAM++ contains 2820 observation, 1505 of which as loop closings involving one of 1312 points visited more than once.

Intel

This dataset represents an indoor office environment and was taken at the Intel research lab using laser sensors. The pre-processed version available with SLAM++ contains 1835 measurements, 895 of which are loop closings involving one of 323 points observed more than once.

Killian Court

This dataset represents the trajectory of a robot in MIT's Killian Court [101]. The approximately 2.2 km indoor path is obtained in 2.5 hours by a B21 mobile robot. The robot is equipped with a SICK PLS scanning laser, encoder based odometry and 24 ultrasonic sensors in a ring configuration. The pre-processed version available with SLAM++ contains 3995 observations, of which 2055 are loop closings involving one of 720 points observed more than once.

Victoria Park

The Victoria Park dataset represents the trajectory of a vehicle moving along a 4 km outdoor path in Victoria Park in Sidney, Australia. The vehicle is equipped with a GPS receiver for ground truth, a scanning laser for landmark observation and odometers for displacement measurement. The pre-processed version available with SLAM++ contains

10608 observations, 3489 of which are loop closings involving one of 151 landmarks. The dataset is described in more details in Section 2.2.1.

5.2 Single Threshold - Original Cost Function

In this section, initial results for the Hybrid Cholesky method on SLAM datasets are obtained. Section 5.2.1 presents the selection of parameters and cost function required by the Hybrid Cholesky algorithm. Section 5.2.2 and Section 5.2.3 discuss the factorisation time and total execution time of an Hybrid Cholesky enabled SLAM++ implementation.

5.2.1 Threshold Selection

In Section 4.4.3, a framework for estimating the computational cost of executing the Cholesky and Factor Recovery operations was proposed to support the implementation of Hybrid Cholesky. In this section, the cost function is defined and an associated threshold is obtained experimentally.

Assuming that the constant costs of the ordering and elimination tree computations can be neglected in both cases, the cost (u) of the Cholesky decomposition and the proposed Factor Recovery method are, respectively,

$$u_{Cholesky} = \sum_{i=1}^{n} |\bar{\mathcal{L}}_i|^2 \tag{5.2}$$

$$u_{Recovery} = \sum_{i \in \mathcal{S}} |\hat{\mathcal{L}}_i \cup \hat{\mathcal{L}}_{i-1}| \tag{5.3}$$

where $\mathcal{S}$ is the set of all the swaps that were done by Bubble Sort during the execution of Factor Recovery (Algorithm 4.3). To limit the overhead cost of the method, the Cholesky score is calculated from current factor $\mathcal{L}$ instead of the new factor $\bar{\mathcal{L}}$. Since it is assumed that the current ordering is replaced by a better one, this is an upper bound on the number

of operations to be done and requires less computation as $\mathcal{L}$ is already available. In the case of Factor Recovery, the cost is estimated by multiplying the length of the permutation vector with the Kendall-Tau distance (KT) between the new and current permutation vectors. $\mathrm{KT}(\mathbf{p}_2, \mathbf{p}_1)$ is defined as the number of pairwise inversions required to obtain $\mathbf{p}_2$ from $\mathbf{p}_1$. The definition of Kendall-Tau distance and the derivation of an efficient algorithm to calculate it are described in [102].

It can be seen from Algorithm 4.3 that the Factor Recovery method is an alternative for Cholesky for the calculations of $[\mathcal{L}_{11}^T \mathcal{L}_{12}]^T$ only since $\mathcal{L}_{22}$ needs to be updated for loop closing. As such, in (5.3) $\mathcal{S}$ is the set of indices swapped such that $[\mathcal{L}_{11}^T \mathcal{L}_{12}]^T$ is reordered. The cost of Cholesky for $\mathcal{L}_{22}$ is not considered in the total cost since this has to be executed in both cases. The cost of the two methods are estimated by

$$\hat{u}_{Cholesky} = \sum_{i=1}^{k-1} |\mathcal{L}_i|^2 \tag{5.4}$$

$$\hat{u}_{Recovery} = \mathrm{KT}(\bar{\mathbf{p}}_{1:k-1}, \mathbf{p}_{1:k-1})n \tag{5.5}$$

where k is the index of the leftmost column involved in the loop closing and n is the number of columns in $\mathcal{L}$. In order to have a threshold value that is independent of the matrix size and complexity, the execution time and estimated cost recorded for Factor Recovery are normalised by the execution time and estimated cost of the regular Cholesky factorisation respectively.

$$t_{ratio} = \frac{t_{Recovery}}{t_{Cholesky}} \qquad \hat{u}_{ratio} = \frac{\hat{u}_{Recovery}}{\hat{u}_{Cholesky}} \tag{5.6}$$

To obtain the threshold's value, the nine SLAM datasets presented in Section 5.1.1 are solved with a simplified version of the SLAM++ algorithm that logs the estimated computational costs (5.4) and (5.5), the execution time and the time spend in overhead tasks. Like SLAM++, the simplified version uses block columns operations to perform

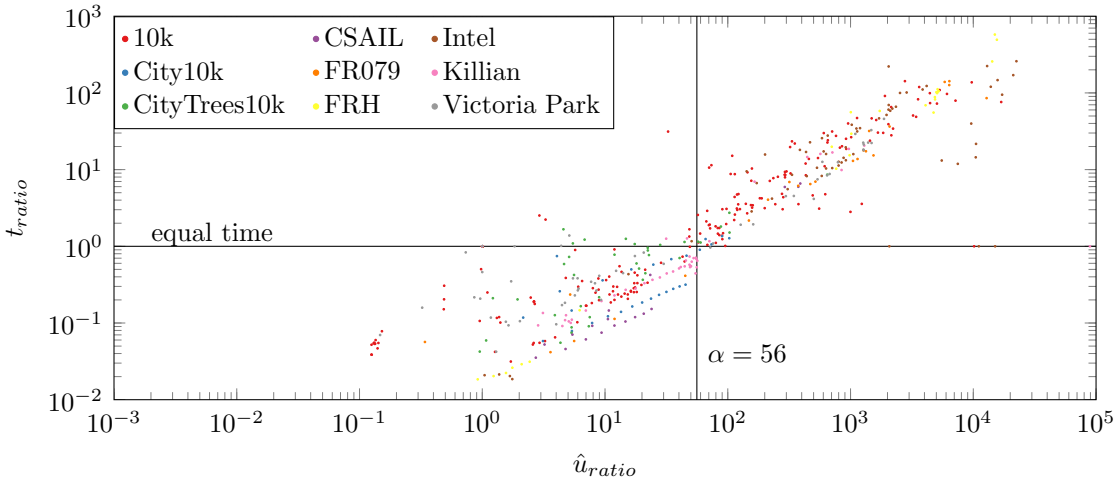


Figure 5.1: t_{ratio} as a function of $\hat{u}_{ratio}$ for 9 SLAM datasets using cost function (5.5) and density triggered reordering. The line displayed at $t_{ratio} = 1$ illustrates where the execution time is equal for both methods. Threshold selection based on minimising relative time in Quadrant 2 and 4 is found at $\alpha = 56$.

column permutations and triggers a full reordering when the density of $\mathcal{L} \geq 0.02$. When this is the case, the new Cholesky factor is calculated both by regular Cholesky decomposition and recovered using Factor Recovery. The ratios of (5.6) are then calculated and plotted in Figure 5.1 for each dataset. The lowest t_{ratio} displayed is 0.02, which translates to the Factor Recovery method being faster by a factor of 50 in the best case analysed, illustrating the need for Factor Recovery to be used. The samples are divided horizontally by $t_{ratio} = 1$ where points below have a lower runtime with Factor Recovery while those above execute faster with Cholesky. In order to select between both methods, a threshold value α on $\hat{u}_{ratio}$ must be selected and divides the points vertically, where points on the left use Factor Recovery and those on the right use Cholesky. The four quadrants obtained are explained in Table 5.2 and illustrated in Figure 5.1. The threshold α is selected such that the relative amount of time spent in Q2 (Factor Recovery selected when Cholesky is faster) and Q4 (Cholesky selected when Factor Recovery is faster) is minimised. Due to the small number of points, the discrete nature of the data, and the desire to avoid local

minima, this is done by calculating

$$\sum_{i \in Q2} (t_{ratio}(i) - 1) + \sum_{i \in Q4} (1 - t_{ratio}(i)) \tag{5.7}$$

for all possible intervals in the dataset and calculating α as the midpoint between the two values of $\hat{u}_{ratio}$ delimiting the interval where the minimum is located. Using Algorithm 4.3 is found to be advantageous if

$$\frac{\hat{u}_{Recovery}}{\hat{u}_{Cholesky}} < 56 \tag{5.8}$$

The threshold obtained is illustrated in Figure 5.1 while Table 5.2 shows the number of cases in each quadrant when the threshold of (5.8) is selected. These demonstrate that, for cost ratios below the threshold expressed in (5.8), the execution time of the Factor Recovery Algorithm is, on average, lower than Cholesky. There are a few cases, however,

Table 5.2: Operating Regions

Region	Fastest Method	Method Used	Number of Cases
Q1 ($t_{ratio} > 1$, $\hat{u}_{ratio} > \alpha$)	Cholesky	Cholesky	239
Q2 ($t_{ratio} > 1$, $\hat{u}_{ratio} < \alpha$)	Cholesky	Factor Recovery	18
Q3 ($t_{ratio} < 1$, $\hat{u}_{ratio} < \alpha$)	Factor Recovery	Factor Recovery	385
Q4 ($t_{ratio} < 1$, $\hat{u}_{ratio} > \alpha$)	Factor Recovery	Cholesky	7

where Factor Recovery is selected when its execution time is up to 12 times longer. This undesirable selection stems from cost functions (5.4) and (5.5) which are, for performance purposes, approximations of the actual cost of the corresponding algorithms. In some cases, this can lead to an underestimation (or overestimation) of the computational cost, producing a horizontal shift in the graph, which causes points in Figure 5.1 to be in a different quadrant. This could be solved by using more accurate cost functions but the accuracy of each function has to be carefully weighted against its complexity, which affects the overhead of the Hybrid Cholesky method.

5.2.2 Factorisation Time

In this section, the performance of Hybrid Cholesky with the values found in the previous section is compared with traditional Cholesky decomposition in a SLAM context.

To obtain the experimental data, a version of SLAM++ using Hybrid Cholesky with cost function (5.5) and threshold (5.8) is executed for 9 SLAM datasets and 3 reordering triggers (See Section 5.1 for details). Note that preliminary results using a density triggered reordering are also briefly described by the author in [103]. The total time, factorisation time and overhead time (cost function evaluation) are logged.

In Table 5.4, the factorisation time of traditional and Hybrid Cholesky, including the cost of computing (5.5) are compared for each dataset with the three different reordering triggers. For convenience, the cases where time is saved by using the Hybrid Cholesky method (negative differential values) are identified in bold font. The results are illustrated in Figure 5.2a. It can be seen that performance gains vary significantly across datasets and across reordering triggers for a given dataset. In the case of density triggered reordering, the performance gains range from -480.3% for *City10k* to 75.5% for *FRH*. An explanation for these large variations is found in Table 5.3, where it can be seen that, for density triggered reordering, Hybrid Cholesky is used only a small number of times (from 0 to 12 times depending on the dataset). Given the cost function approximation errors discussed when specifying the cost function (Section 5.2.1), large variations can be expected from such a small sample size. In the case of Batch-10 and Batch-5, the sample size is significantly increased which reduces the effect of the approximation error. For this reason, these will be used for the remainder of this section.

The *Victoria Park* and *cityTrees10K* datasets have inferior performance compared to other datasets for the Batch-10 and Batch-5 cases for an average of $-86.7\,\%$ for *Victoria Park* and $-37.8\,\%$ for *cityTrees10K*. This represents an increase of 11.3 s and 17.7 s respectively for the factorisation time using Batch-5 or 5.7 s and 5.4 s respectively using

Table 5.3: Number of Total and Factor Recovery (FR) Reorderings for 3 different triggers.

Dataset	Size	Density Reordering		Batch-10		Batch-5	
		Total	FR	Total	FR	Total	FR
10k	64311	34	1	1167	53	2045	53
City10k	20687	12	2	1104	28	2049	28
CityTrees10k	14442	13	0	1144	12	2138	12
CSAIL	1172	11	8	213	98	305	100
FR079	1217	8	8	183	105	273	116
FRH	2820	12	12	261	173	378	201
Intel	1835	18	8	231	96	309	100
Killian	3995	11	8	277	98	454	101
Victoria Park	10608	13	0	863	13	1557	13

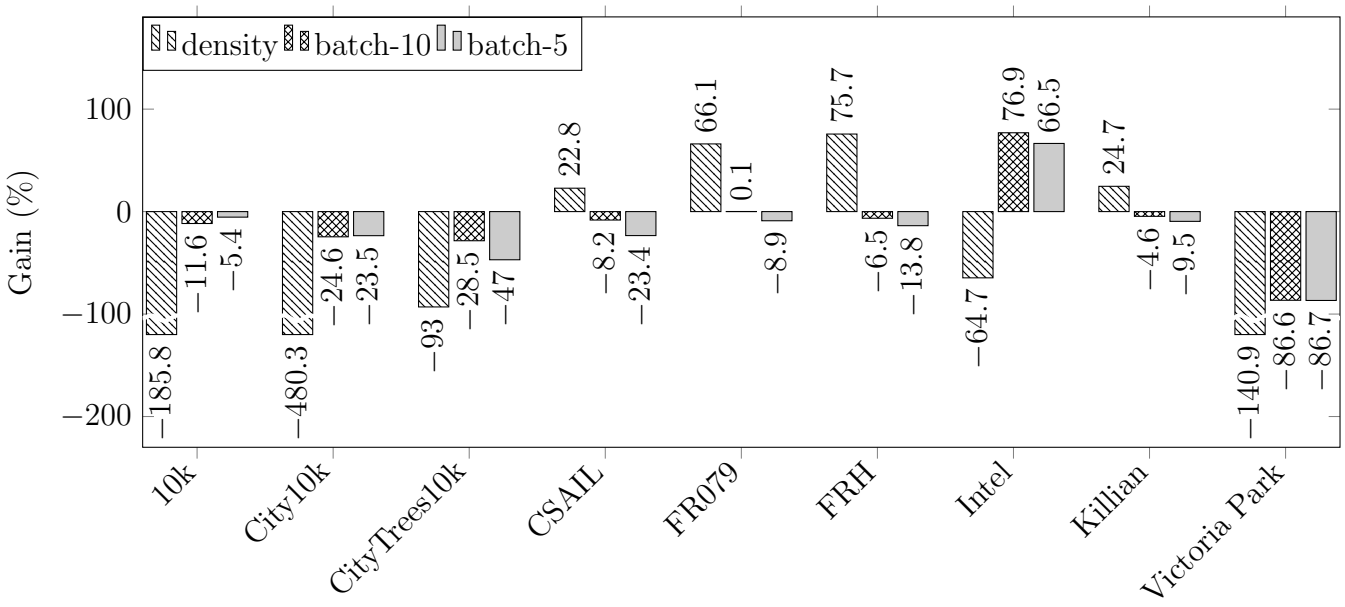
Batch-10. Table 5.3 shows that in these cases, Hybrid Cholesky is used in 0.6 % to 1.6 % of the reordering operations. Thus for the remaining reordering operations, (5.5) is evaluated but no Hybrid Cholesky performance gain is obtained, yielding a net loss for most steps and an inferior overall performance. The best results are obtained on the *Intel* dataset with an average performance gain of 71.7 % meaning that the savings of Hybrid Cholesky outweighed the cost of evaluating (5.5). This dataset also uses Hybrid Cholesky often, with an average use of 37 %.

The *CSAIL*, *FR079*, *FRH* and *Killian* datasets also use Hybrid Cholesky often (39 %, 50 %, 60 % and 29 % respectively) but have a negative performance gain. This indicates either that Hybrid Cholesky is more efficient only by a narrow margin or that frequent false positives lead to Hybrid Cholesky being selected in cases where it takes more time than the traditional method.

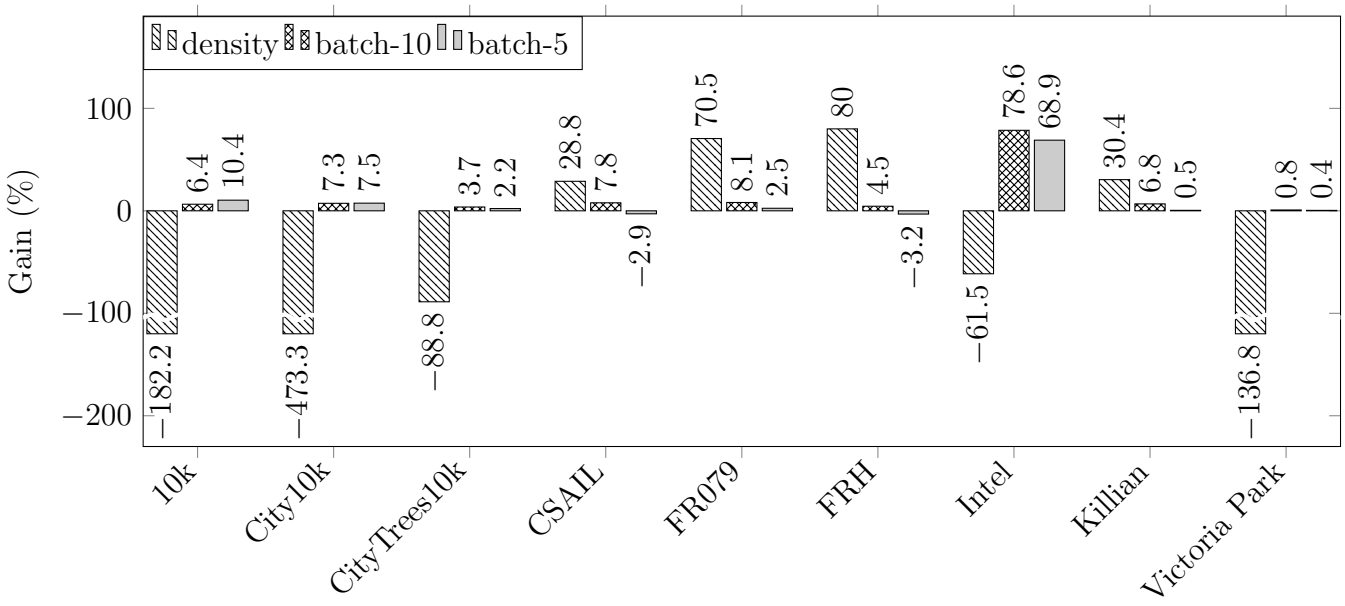
Looking at Table 5.5 and associated Figure 5.2b, which contains the factorisation time excluding the calculation of (5.5), it can be seen that the performance gain averaged between the Batch-5 and Batch-10 reordering triggers is positive for all datasets with an overall average of 11.68 %. Most notably, for the *Victoria Park* dataset, the average performance gain for batch triggered reordering is 0.6 %, up from −86.7 %. It can thus

be concluded that the calculation of (5.5) accounts for a significant portion of the Hybrid Cholesky execution time.

It can also be observed from Figure 5.2a that indoor datasets tend to have superior performance compared to outdoor datasets, with the real outdoor dataset *Victoria Park* having a lower performance than the artificial outdoor datasets. Amongst the artificial datasets, landmark based *CityTrees10K* has a lower performance than the other two (*10K* and *City10K*) and ranks closer to *Victoria Park*, which is also landmark based. In Figure 5.2b, where the overhead cost is omitted, there is no longer a noticeable demarcation between the indoor and outdoor datasets. In fact, the *10K* and *City10K* datasets rank second and third behind *Intel* in terms of performance. The current combination of cost function and threshold thus appear more expensive to compute for outdoor datasets, and even more so for outdoor datasets with low landmark density.



(a) Including Overhead of calculating (5.5)



(b) Excluding Overhead of calculating (5.5)

Figure 5.2: The factorisation time performance gain between the Hybrid Cholesky with cost function (5.5) and regular Cholesky for full reordering on each dataset.

Table 5.4: Comparison of the factorisation runtime (in seconds) of Hybrid and Traditional Cholesky using a single threshold for all datasets. The cost function (5.5) is used and included in the timing.

	standard			batch-10			batch-5		
Dataset	Hybrid	Regular	±	Hybrid	Regular	±	Hybrid	Regular	±
10k	0.08562	0.02996	0.05566	81.2982	72.8292	8.4690	156.4875	148.4624	8.0350
City10k	0.00398	0.00069	0.00330	64.8352	52.0261	12.8091	126.9698	102.8092	24.1606
CityTrees10k	0.00330	0.00171	0.00159	24.5467	19.1091	5.4376	55.2757	37.5998	17.6759
CSAIL	0.00082	0.00107	**−0.00024**	0.1070	0.0988	0.0081	0.2192	0.1777	0.0416
FR079	0.00016	0.00047	**−0.00031**	0.0972	0.0973	**−0.0001**	0.2012	0.1847	0.0165
FRH	0.00033	0.00137	**−0.00104**	0.2831	0.2658	0.0173	0.5617	0.4938	0.0679
Intel	0.01061	0.00645	0.00416	0.2272	0.9848	**−0.7576**	0.3736	1.1156	**−0.7419**
Killian	0.00085	0.00113	**−0.00028**	0.4950	0.4731	0.0219	1.0246	0.9354	0.0892
Victoria Park	0.00525	0.00218	0.00307	12.3216	6.6049	5.7168	24.3817	13.0578	11.3239

Table 5.5: Comparison of the Factorisation runtime (in seconds) of Hybrid and Traditional Cholesky using a single threshold for all datasets. The cost function (5.5) is used but excluded from the timing.

	standard			batch-10			batch-5		
Dataset	Hybrid	Regular	±	Hybrid	Regular	±	Hybrid	Regular	±
10k	0.08455	0.02996	0.05458	68.1430	72.8292	**−4.6862**	133.0325	148.4625	**−15.4300**
City10k	0.00393	0.00069	0.00325	48.2130	52.0261	**−3.8131**	95.1157	102.8092	**−7.6934**
CityTrees10k	0.00323	0.00171	0.00152	18.3970	19.1091	**−0.7121**	36.7672	37.5998	**−0.8327**
CSAIL	0.00076	0.00107	**−0.00031**	0.0911	0.0988	**−0.0077**	0.1829	0.1777	0.0052
FR079	0.00014	0.00047	**−0.00033**	0.0893	0.0973	**−0.0079**	0.1801	0.1847	**−0.0046**
FRH	0.00024	0.00137	**−0.00110**	0.2539	0.2658	**−0.0119**	0.5095	0.4938	0.0157
Intel	0.01041	0.00644	0.00396	0.2111	0.9848	**−0.7737**	0.3465	1.1156	**−0.7691**
Killian	0.00079	0.00113	**−0.00034**	0.4411	0.4731	**−0.0320**	0.9309	0.9354	**−0.0045**
Victoria Park	0.00519	0.00218	0.00298	6.5510	6.6049	**−0.0539**	13.0001	13.0578	**−0.0577**

5.2.3 Total Time

In this Section, the total execution time of SLAM++ is considered. This is used to identify if the Hybrid Cholesky factorisation's performance (Section 5.2.2) would benefit practical implementations in the field. Results where the calculation of the cost function (5.5) is excluded are also presented to understand the cost function's effect on the total execution time.

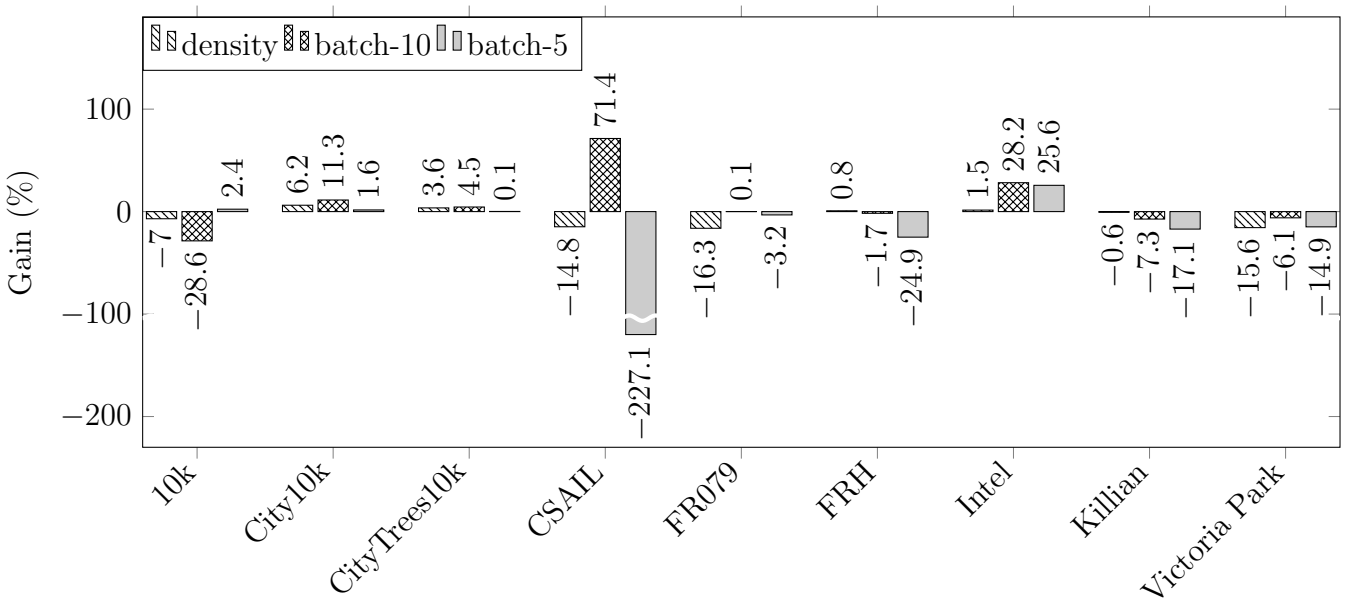
In Table 5.6 and associated Figure 5.3a it can be seen that three of the nine datasets show a performance increase due to Hybrid Cholesky. For the cases where the recomputation is triggered by density, the performance gain (positive or negative) is smaller in magnitude than when considering the factorisation time only (Figure 5.2a) because the small number of operations performed (Table 5.3), has a small impact on the total execution time. In the Batch-10 and Batch-5 cases, the reordering is performed more often and will play a greater role in the total time. It can also be seen that the *CSAIL* dataset shows some inconsistencies in the form of high variation between the Batch-10 and Batch-5 performance gains (71.4 % and −227.1 %). This is unexpected and sets *CSAIL* apart as an outlier amongst the other datasets.

The remaining results of Figure 5.3a show the highest performance gain is achieved by the *Intel* dataset at 26.9 %, which is consistent with the results obtained with the factorisation time in Section 5.2.2. The *City10k* and *CityTrees10k* also have a positive performance gain with an average of 2.3 % and 6.5 % respectively. Due to the negative performance gain obtained for these datasets when considering the factorisation performance, it can be hypothesised that the total performance gain is due to a difference in the orderings selected. Overall, the average performance gain of the Batch-5 and Batch-10 ordering triggers for all datasets (with the exception of *CSAIL*) is −1.9 %.

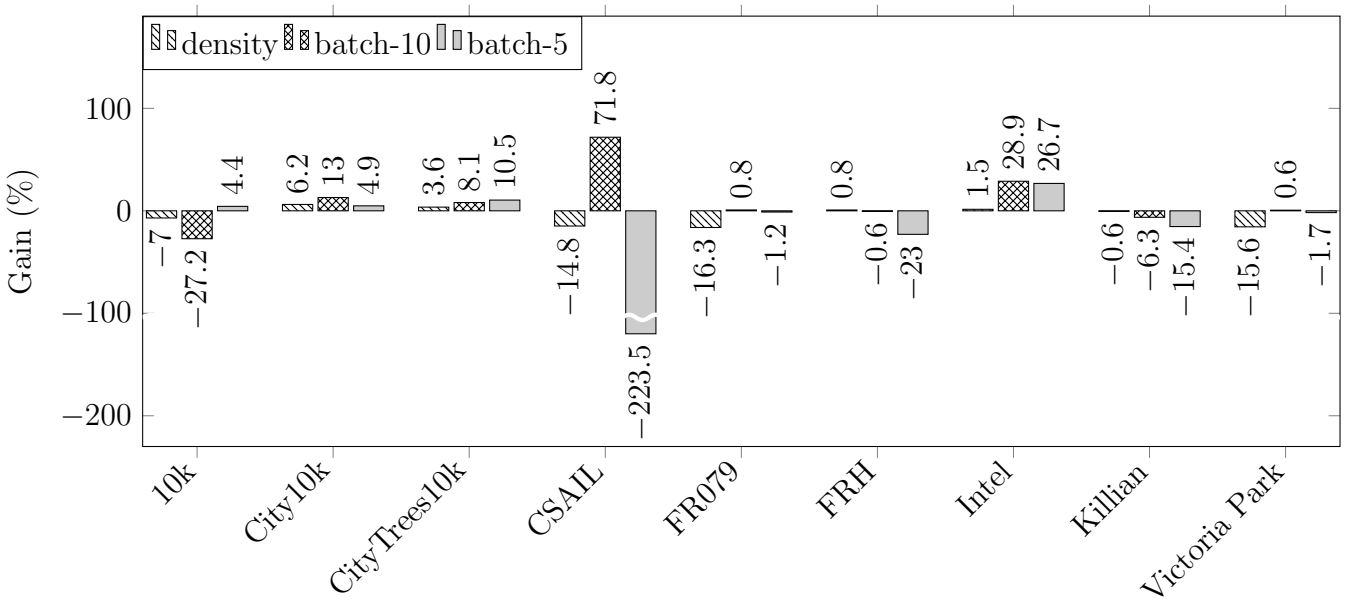
When excluding the computation time of the cost functions as seen in Figure 5.3b, the experiment shows similar results, with *Intel*, *City10k* and *CityTrees10k* achieving perfor-

mances gains of 27.8 %, 9 % and 9.3 % respectively. This translates into savings of 0.7 s, 115.6 s and 7.7 s respectively for Batch-10 or 0.6 s, 15.9 s and 0.2 s for the Batch-5 case. The other datasets still have negative performance gains.

It can be concluded that some datasets benefit from using the parameters of Section 5.2.1 and the Hybrid Cholesky algorithm but the performance gain is not generalised to all datasets. Furthermore, factorisation performance gains observed in Figure 5.2b do not necessarily translate to an increased in performance of the SLAM++ program. The user may consider using a Hybrid Cholesky base SLAM implementation with the cost function and threshold identified in Section 5.2.1 in certain specific situations. For example, in grid based outdoor environments such as *City10K* and *CityTrees10K* or indoor environments similar to the *Intel* dataset. However, careful considerations must be given to the fact that similar datasets have negative performance gains.



(a) Including Overhead of calculating (5.5)



(b) Excluding Overhead of calculating (5.5)

Figure 5.3: The total performance gain between Hybrid Cholesky with cost function (5.5) and regular Cholesky for full reordering on each dataset.

Table 5.6: Comparison of the Total runtime (in seconds) of Hybrid and Traditional Cholesky using a single threshold for all datasets. The cost function (5.5) is used and included in the timing.

	standard			batch-10			batch-5		
Dataset	Hybrid	Regular	±	Hybrid	Regular	±	Hybrid	Regular	±
10k	932.016	871.198	60.8184	1269.424	987.306	282.1186	1149.755	1177.555	**−27.8004**
City10k	1109.585	1183.161	**−73.5762**	905.665	1021.281	**−115.6161**	965.836	981.726	**−15.8899**
CityTrees10k	190.120	197.238	**−7.1174**	163.152	170.809	**−7.6575**	178.943	179.189	**−0.2461**
CSAIL	0.929	0.809	0.1199	1.007	3.516	**−2.5084**	3.272	1.000	2.2720
FR079	1.014	0.872	0.1419	1.004	1.005	**−0.0006**	1.069	1.035	0.0335
FRH	3.319	3.346	**−0.0261**	2.681	2.637	0.0444	3.280	2.626	0.6548
Intel	2.030	2.061	**−0.0303**	1.752	2.440	**−0.6878**	1.834	2.467	**−0.6326**
Killian	6.236	6.198	0.0375	5.931	5.528	0.4029	6.402	5.468	0.9341
Victoria Park	102.453	88.598	13.8553	91.544	86.292	5.2520	99.441	86.555	12.8859

Table 5.7: Comparison of the Total runtime (in seconds) of Hybrid and Traditional Cholesky using a single threshold for all datasets. The cost function (5.5) is used but excluded from the timing.

	standard			batch-10			batch-5		
Dataset	Hybrid	Regular	±	Hybrid	Regular	±	Hybrid	Regular	±
10k	932.015	871.198	60.8173	1256.269	987.306	268.9634	1126.300	1177.555	**−51.2554**
City10k	1109.585	1183.161	**−73.5763**	889.043	1021.281	**−132.2383**	933.982	981.726	**−47.7439**
CityTrees10k	190.120	197.238	**−7.1175**	157.002	170.809	**−13.8071**	160.434	179.189	**−18.7546**
CSAIL	0.929	0.809	0.1198	0.991	3.516	**−2.5242**	3.236	1.000	2.2356
FR079	1.014	0.872	0.1419	0.997	1.005	**−0.0084**	1.048	1.035	0.0124
FRH	3.319	3.346	**−0.0261**	2.652	2.637	0.0153	3.228	2.626	0.6026
Intel	2.030	2.061	**−0.0305**	1.736	2.440	**−0.7039**	1.807	2.467	**−0.6598**
Killian	6.235	6.198	0.0375	5.877	5.528	0.3489	6.308	5.468	0.8404
Victoria Park	102.453	88.598	13.8552	85.773	86.292	**−0.5187**	88.059	86.555	1.5043

5.3 Single Threshold - Improved Cost Function

In this section, a second cost function and threshold are presented to address the high computational time of the cost function presented in Section 5.2. Section 5.3.1 presents the selection of the improved parameters and cost function required by the Hybrid Cholesky algorithm. Section 5.3.2 and Section 5.3.3 discuss the factorisation time and total execution time of an Hybrid Cholesky enabled SLAM++ implementation using the new cost function.

5.3.1 Threshold Selection

In Section 5.2, it was found that the evaluation time of the cost function used by Hybrid Cholesky to choose the factorisation method can have an impact on performance, in some cases negating performance gains. To mitigate this cost, a more efficient cost function for Factor Recovery is proposed as follows.

$$\hat{u}_{Recovery} = \frac{n}{2} \sum_{i=1}^{n} |\hat{p}(i) - i| \tag{5.9}$$

where n is the size of $\mathcal{C}$ and $\hat{p}(i) = \bar{p}(p(i))$ is the differential ordering. Using (5.9), the cost associated with moving each column depends solely on the destination column whereas in (5.5), the cost took into account the effect of the columns displaced prior to the current one. This cost function offers the benefit of being computable directly in n operations without requiring any temporary copy of data. Using this cost function alters the values of t_{ratio} and u_{ratio}. The new plot is shown in Figure 5.4. Using this plot, a threshold is calculated. It results in

$$\frac{\hat{u}_{Recovery}}{\hat{u}_{Cholesky}} < 8 \tag{5.10}$$

The number of cases in each region of operation is illustrated in Table 5.8. Although the cost function (5.9) executes faster than (5.5), the compromise in accuracy can be seen by comparing the vertical spread of the scatter point in Figure 5.1 and Figure 5.4. This

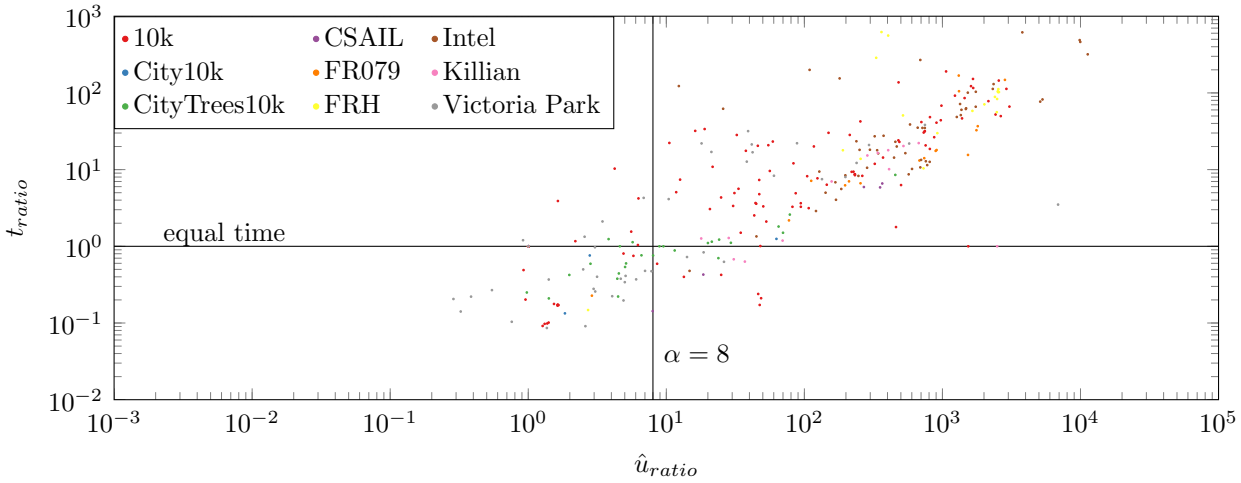


Figure 5.4: t_{ratio} as a function of $\hat{u}_{ratio}$ for 9 SLAM datasets using cost function (5.9) and density triggered reordering. The line displayed at $t_{ratio} = 1$ illustrates where the execution time is equal for both methods. The new threshold based on minimising relative time in Quadrant 2 and 4 is obtained at $\alpha = 8$.

Table 5.8: Operating Regions

Region	Fastest Method	Method Used	Number of Cases
Q1 ($t_{ratio} > 1$, $\hat{u}_{ratio} > \alpha$)	Cholesky	Cholesky	197
Q2 ($t_{ratio} > 1$, $\hat{u}_{ratio} < \alpha$)	Cholesky	Factor Recovery	55
Q3 ($t_{ratio} < 1$, $\hat{u}_{ratio} < \alpha$)	Factor Recovery	Factor Recovery	375
Q4 ($t_{ratio} < 1$, $\hat{u}_{ratio} > \alpha$)	Factor Recovery	Cholesky	22

results in an increased number of points in quadrant Q2 and Q4, illustrated in Table 5.8 compared to Table 5.2.

5.3.2 Factorisation Time

The factorisation time of the Hybrid Cholesky method with the new cost function is compared with the traditional approach. The same experiment as for the original cost function (Section 5.2.2) is repeated, yielding the results of Table 5.9 which are summarised in Figure 5.5.

The density reordering case suffers from the same issue identified when calculating the factorisation time (Section 5.2.2) regarding the low number of reordering with regards to the size of the dataset (Table 5.3). For Batch-5 and Batch-10, the average performance improvement is 12.21 %. All datasets have a positive performance gain with the exception of the Batch-5 reordering trigger for the *CSAIL* and *FRH* datasets, which have a performance gain of $-0.7\,\%$ and $-4.4\,\%$ respectively.

The performance gain of 12.21 % is higher than the 11.68 % that was obtained in Section 5.2.2 when the overhead was excluded. This shows that the cost function (5.9) is not only more computationally efficient than (5.5) but also provides a more accurate estimation of the computational cost of Hybrid Cholesky.

The results of Figure 5.5, are similar to those shown in Figure 5.2b for the previous threshold. With the exception of *Intel* dataset, *10k* and *city10k* have the highest performance gains, while the landmark based *CityTrees10k* and *Victoria Park* rank middle and last, respectively. As the execution time of the cost function is small, it can be eliminated as a possible cause of inferior performance. The comparatively poor performance of some datasets (most notably landmark based datasets) may be caused by a bias in the cost function or threshold selection, causing an overestimation of the work to be done or providing an erroneous cut-off. These would, in turn, result in Hybrid Cholesky consistently using the Factor Recovery algorithm less often than it should. Another possible explanation is that the cost function may be inexact and underestimate the cost at certain times. This

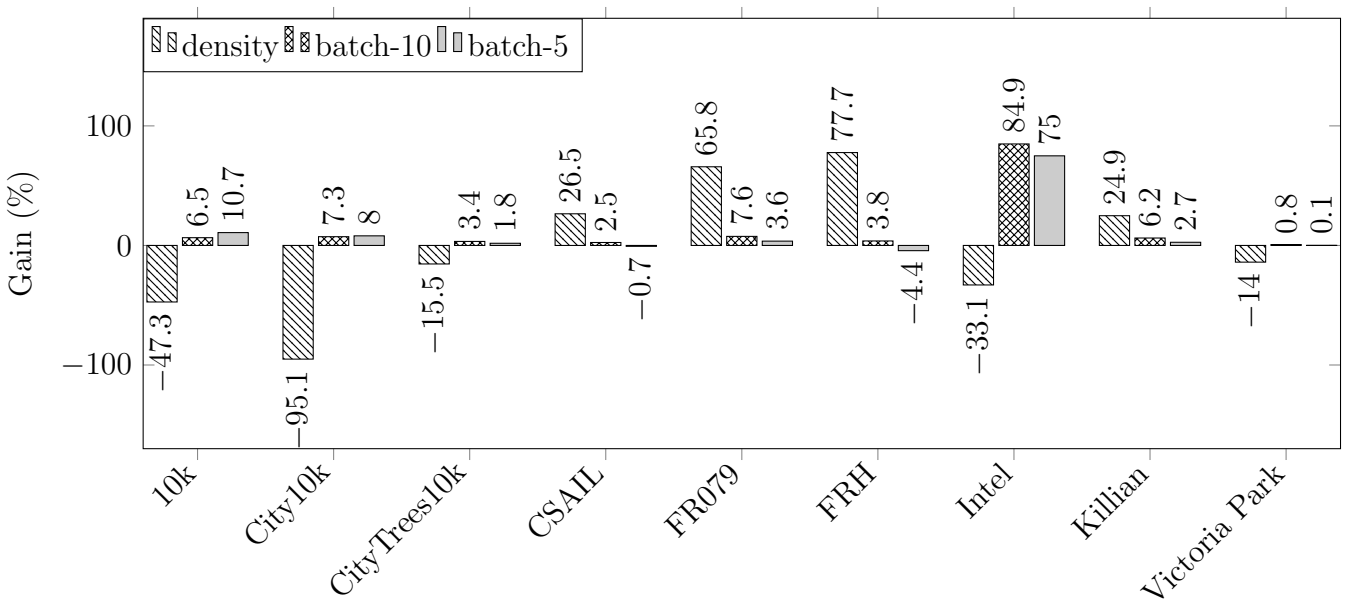


Figure 5.5: The factorisation time performance gain between the Hybrid Cholesky with cost function (5.9) and regular Cholesky for full reordering on each dataset.

would see Factor Recovery be used in disadvantageous situations, counterbalancing the gains made during previous uses of Factor Recovery.

Table 5.9: Comparison of the Factorisation runtime (in seconds) of Hybrid and Traditional Cholesky for single threshold using the cost function (5.9).

	standard			batch-10			batch-5		
Dataset	Hybrid	Regular	±	Hybrid	Regular	±	Hybrid	Regular	±
10k	0.04413	0.02996	0.01417	68.1517	72.8726	**−4.7209**	132.5808	148.5300	**−15.9491**
City10k	0.00130	0.00067	0.00063	48.2693	52.0857	**−3.8164**	94.6007	102.8668	**−8.2660**
CityTrees10k	0.00196	0.00170	0.00026	18.4711	19.1134	**−0.6424**	36.8897	37.5746	**−0.6849**
CSAIL	0.00078	0.00107	**−0.00028**	0.0919	0.0943	**−0.0023**	0.1785	0.1772	0.0013
FR079	0.00016	0.00047	**−0.00031**	0.0904	0.0978	**−0.0074**	0.1801	0.1868	**−0.0068**
FRH	0.00031	0.00137	**−0.00106**	0.2556	0.2656	**−0.0100**	0.5151	0.4935	0.0216
Intel	0.00854	0.00642	0.00212	0.1483	0.9824	**−0.8340**	0.2785	1.1122	**−0.8337**
Killian	0.00084	0.00112	**−0.00028**	0.4435	0.4729	**−0.0294**	0.8953	0.9197	**−0.0244**
Victoria Park	0.00250	0.00219	0.00031	6.6284	6.6787	**−0.0503**	13.0471	13.0546	**−0.0075**

Table 5.10: Comparison of the Total runtime (in seconds) of Hybrid and Traditional Cholesky based SLAM++ implementation for single threshold using the cost function (5.9).

	Density			Batch-10			Batch-5		
Dataset	Hybrid	Regular	±	Hybrid	Regular	±	Hybrid	Regular	±
10k	948.07	869.52	70.3007	1257.91	985.87	272.0364	1127.65	1033.18	94.4649
City10k	1115.92	1182.88	**−66.9621**	889.15	1022.67	**−133.5078**	920.98	982.32	**−61.3422**
CityTrees10k	189.20	196.23	**−7.0283**	157.27	170.73	**−13.4587**	160.99	179.26	**−18.2706**
CSAIL	0.93	0.81	0.1212	0.99	0.98	0.0068	1.03	0.99	0.0309
FR079	1.01	0.87	0.1445	1.00	1.01	0.0119	1.04	1.07	**−0.0284**
FRH	3.34	3.34	0.0027	2.65	2.62	0.0168	2.96	2.62	0.3446
Intel	2.03	2.09	**−0.0556**	1.66	2.44	**−0.7746**	1.74	2.47	**−0.7349**
Killian	6.32	6.27	0.0445	5.95	5.3	0.4180	6.35	5.43	0.9196
Victoria Park	102.60	88.95	13.6491	87.47	86.91	**−0.4813**	88.74	86.13	2.6005

5.3.3 Total Time

In this Section, the total execution time of SLAM++ is considered. This is used to identify if the Hybrid Cholesky factorisation performances discussed in Section 5.3.2 would benefit practical implementations in the field.

It can be seen in Table 5.10 and associated Figure 5.6 that the Hybrid Cholesky implementation with the new threshold, although demonstrated to be faster than traditional Cholesky for all but one dataset in Section 5.3.2, only yields a positive gain for four of the nine datasets. When compared to the total execution time of the original threshold illustrated in Figure 5.3b, it can be seen that results have improved. The average of Batch-5 and Batch-10 for the data sets with positive performance gain (*City10k*, *CityTrees10k* and *Intel*) have increased to 9.65 %, 9.05 % and 30.75 % from 6.45 %, 2.3 % and 26.9 % respectively. The overall average performance gain for Batch-5 and Batch-10 of all datasets has passed from −1.9 % to 1.9 %. Furthermore, the *CSAIL* dataset, although at a negative performance gain, is no longer an outlier.

This leads us to conclude that the Hybrid Cholesky method with the current threshold and cost function is more likely to outperform the traditional Cholesky method on autonomous vehicles, being on average 1.9 % faster if Batch-5 or Batch-10 is used. The performance gain is significant when the terrain has certain advantageous characteristics encountered in the datasets described above, in which case a speed-up of 9.05 % to 30.75 % is obtained.

Comparing the dataset characteristics in Table 5.1 with the results, some hypotheses can be made regarding outdoor datasets. The data suggests that outdoor datasets with above average loop closing density that are either landmark-based with low landmark density or pose-based with higher revisit-rate have a superior performance with the proposed Hybrid Cholesky method and selected cost function/threshold. Such statement is not possible in the case of indoor datasets as similar environments (*Killian* and *Intel*) have sig-

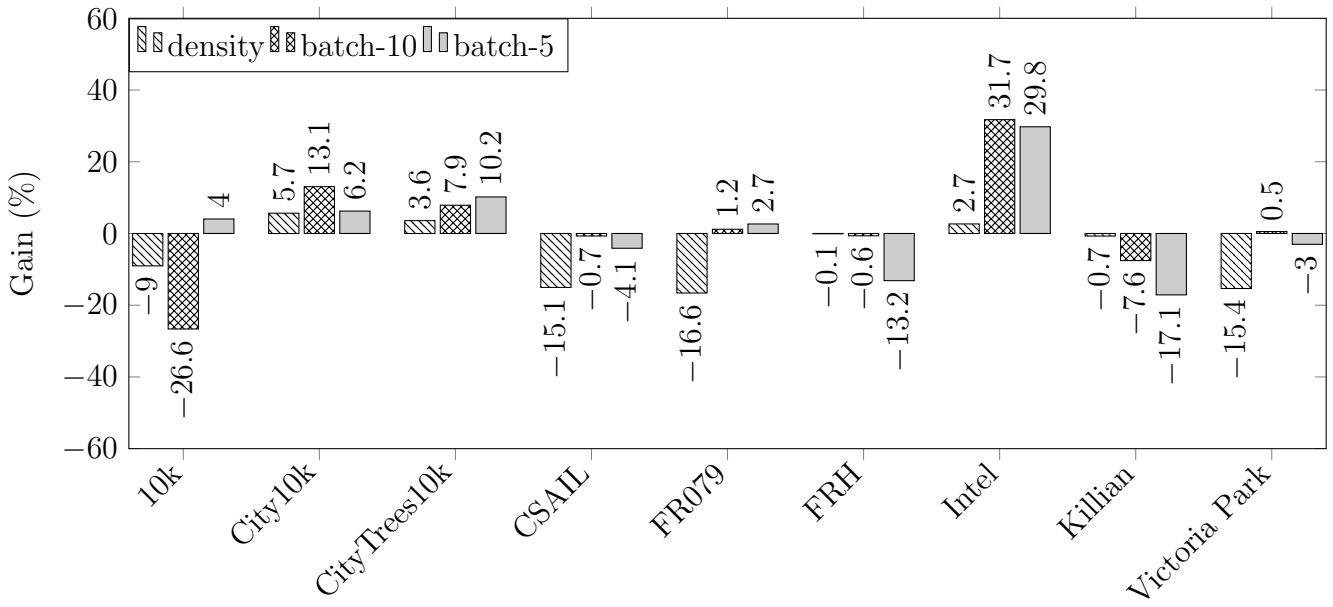


Figure 5.6: The total time performance gain between the Hybrid Cholesky with cost function (5.9) and regular Cholesky for full reordering on each dataset.

nificantly different performance. Objective identification of what constitutes advantageous terrain is left as future work.

5.4 Optimized Threshold - Improved Cost Function

The effect of dataset-specific thresholds on performance is analysed to explore the threshold's dependence on the problem structure. This is done to further improve the performance gains obtained the improved cost function (Section 5.3). Section 5.4.1 presents the selection of the new parameters required by the Hybrid Cholesky algorithm. Sections 5.4.2 and 5.4.3 discuss the factorisation time and total execution time of an Hybrid Cholesky enabled SLAM++ implementation using the new thresholds.

5.4.1 Threshold Optimisation

In Section 5.2.1 and Section 5.3.1 the dependence of the threshold value on the structure of the matrix $\mathcal{C}$ is mitigated by considering only SLAM structured problems when calculating the threshold. This however does not eliminate the issue altogether. Inherent differences between the datasets also influence the cost function ratio (5.6), with some datasets showing a bias for one side or the other when an approximation of the cost function is used. This can be illustrated by calculating a threshold for each dataset, as shown in the first column of Table 5.11, and comparing it to the threshold obtained in (5.10). Although some datasets, such as *CityTrees10k* and *Victoria Park*, are close to the single threshold value, others like *CSAIL* are significantly different. Ideally, the cost function would be based on characteristics of the underlying matrix structure that correlate with the computational cost of Hybrid Cholesky such that a single threshold can be used regardless of the environment the vehicle is exploring.

To explore the performance advantages an ideal cost function would have, a gradient descent optimisation is performed on each dataset to obtain the threshold yielding the smallest execution time. Due to the large execution time for some of the datasets and noisy results, SLAM++ is run five times for each optimisation step. The first run is discarded as it was found to often be an outlier. The results from the remaining four runs

Table 5.11: Calculated and Optimized Thresholds

	Threshold	
Dataset	Calculated	Optimized
10k	1	1.57
City10k	32	56.98
CityTrees10k	15	13.70
CSAIL	144	208.82
FR079	40	56.04
FRH	96	4.99
Intel	20	41.01
Killian	53	25.97
Victoria Park	9	8.44

are averaged to reduce the impact of noise and the result is used as the function output for optimisation purposes. The thresholds obtained by minimising the total execution time are shown in the second column of Table 5.11. It can be seen that in some cases, there is a significant discrepancy between the threshold calculated for a given dataset and the threshold obtained experimentally through minimisation. This reflects on the accuracy of the cost function selected in Section 5.3.1 and on the link between factorisation time and total execution time. If a sufficiently accurate cost function were used, the "Calculated" and "Optimised" columns of Table 5.11 would both contain values close to the single threshold value.

5.4.2 Factorisation Time

In this Section, the experiment of Section 5.3.2 is repeated using the optimal threshold as listed in Table 5.11. The execution times obtained are presented in Table 5.12 and displayed in Figure 5.7. The results obtained for *CSAIL* are $-770\,\%$, $-388\,\%$ and $-69\,\%$ for Density, Batch-10 and Batch-5 triggered reordering respectively. Such results are too far beyond the expected value and indicative of a convergence issue regarding the optimisation of the threshold for this particular dataset.

Table 5.12: Comparison of the factorisation runtime (in seconds) of Hybrid and Traditional Cholesky for optimized threshold

	standard			batch-10			batch-5		
Dataset	Hybrid	Regular	±	Hybrid	Regular	±	Hybrid	Regular	±
10k	0.03639	0.03004	0.00635	67.86377	72.85032	**−4.98655**	132.75327	148.43964	**−15.68637**
City10k	0.00130	0.00067	0.00062	48.28423	52.04020	**−3.75597**	94.69245	102.72156	**−8.02911**
CityTrees10k	0.00178	0.00170	0.00008	18.45856	19.11902	**−0.66046**	36.85913	37.75022	**−0.89109**
CSAIL	0.00958	0.00110	0.00848	0.46000	0.09430	0.36570	0.29919	0.17685	0.12234
FR079	0.00016	0.00050	**−0.00034**	0.09241	0.09743	**−0.00503**	0.18128	0.18521	**−0.00393**
FRH	0.00031	0.00138	**−0.00107**	0.25540	0.26634	**−0.01094**	0.51765	0.49499	0.02266
Intel	0.01317	0.00643	0.00673	0.21125	0.98273	**−0.77148**	0.35442	1.11324	**−0.75882**
Killian	0.00082	0.00112	**−0.00030**	0.44438	0.47306	**−0.02867**	0.90128	0.92444	**−0.02316**
Victoria Park	0.00249	0.00219	0.00030	6.55583	6.62658	**−0.07075**	12.95096	13.11936	**−0.16840**

Table 5.13: Comparison of the Total runtime (in seconds) of Hybrid and Traditional Cholesky based SLAM++ implementation for optimized threshold

	Density			Batch-10			Batch-5		
Dataset	Hybrid	Regular	±	Hybrid	Regular	±	Hybrid	Regular	±
10k	939.60	869.29	70.3007	1257.91	985.87	272.0364	1127.65	1177.53	**−49.8879**
City10k	1116.48	1182.51	**−66.0271**	888.76	1021.64	**−132.8735**	924.19	982.49	**−58.2993**
CityTrees10k	187.62	197.06	**−9.4416**	158.30	170.23	**−11.9289**	161.00	178.93	**−17.9269**
CSAIL	1.07	0.81	0.2650	1.38	0.97	0.4103	1.32	1.00	0.3299
FR079	1.06	0.91	0.1518	1.00	1.00	0.0024	1.05	1.04	0.0141
FRH	3.32	3.35	**−0.0242**	2.64	2.63	0.0031	2.96	2.64	0.3199
Intel	2.04	2.08	**−0.0346**	1.74	2.44	**−0.7049**	1.80	2.46	**−0.6615**
Killian	6.30	6.18	0.1162	5.91	5.52	0.3874	6.35	5.43	0.9127
Victoria Park	102.72	88.54	14.1727	85.70	86.91	**−1.2124**	87.37	86.50	0.8640

All other datasets, with the exception of Batch-5 for *FRH*, have a positive performance gain. The total average performance gain is 17.6 % for the Batch-5 and Batch-10 cases. This is much better than the original gain of 11.68 % when using the original threshold without the overhead (Section 5.2) or even the performance gain of 12.2 % when using the improved cost function (Section 5.3). This illustrates the need for a cost function and threshold that are more accurate predictors of the computational cost of Hybrid Cholesky.

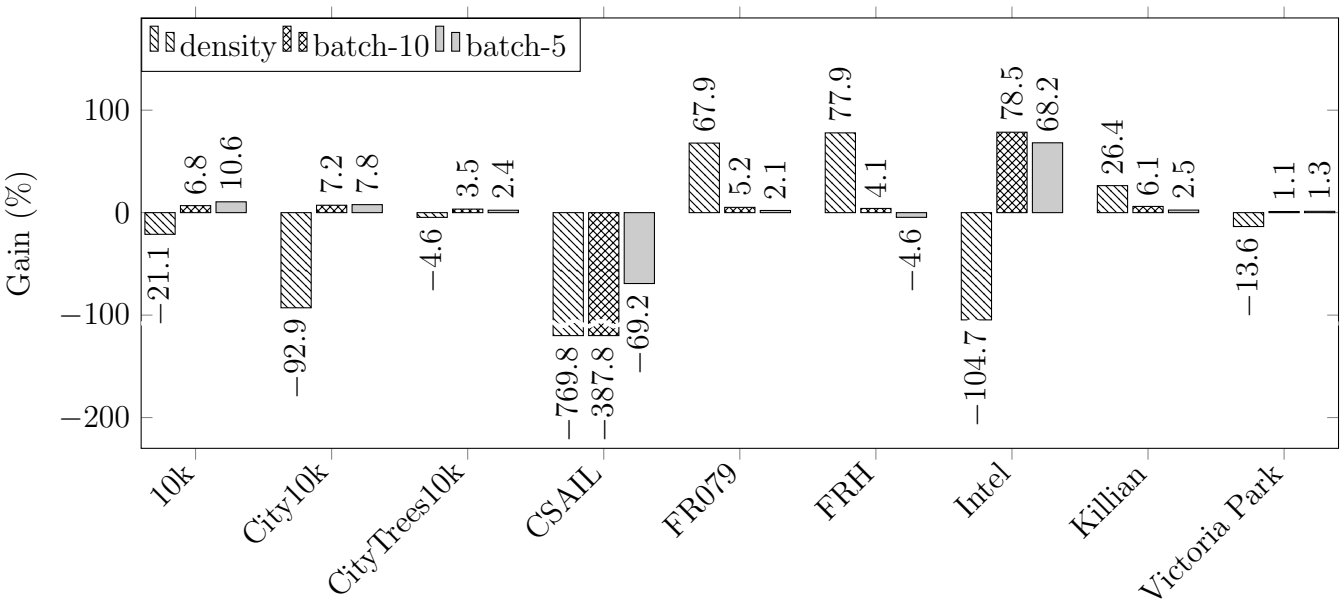


Figure 5.7: The factorisation time performance gain between the Hybrid Cholesky with cost function (5.9) and regular Cholesky. Dataset-specific threshold are used.

5.4.3 Total Time

In this section, the Hybrid Cholesky based SLAM++ implementation using the improved cost function defined in Section 5.3.1 and the optimal thresholds listed in Table 5.11 is considered. Its total execution time is compared to that of the SLAM++ implementation based on the traditional Cholesky factorisation. The results are displayed in Table 5.13 and illustrated in Figure 5.8.

The results for the *CSAIL* dataset were included in the graph for completeness, but they are not valid due to problems selecting a threshold for the Hybrid Cholesky algorithm (see Section 5.4.2). For the *City10k*, *CityTrees10k* and *Intel* datasets, performance gains of 9.45 %, 8.5 % and 27.85 % are obtained, representing average time savings of 95.6 s, 14.9 s and 0.68 s respectively. The performance gains for these datasets have decreased slightly from the results obtained with the improved cost function in Section 5.3.3 (9.65 %, 9.05 % and 30.75 % respectively). They are, however, above the performances obtained with the original cost function in Section 5.2.3 (6.45 %, 2.3 % and 26.9 %, respectively). The overall performance average (excluding *CSAIL*) of 1.9 % is the same as with the improved cost function. Based on these observations, it can be concluded that the use of optimised thresholds has a normalizing effect on the datasets' performance gain. It is expected that refining the optimal thresholds found would further increase the performance over those obtained with the improved cost function.

Although the optimised thresholds described in Section 5.4.2 have shown a substantial increase over the improved threshold in terms of factorisation time, this does not translate to an overall improvement in the total computation time. It is thus not recommended to use terrain-specific threshold with Hybrid Cholesky in arbitrary environments. Unless knowledge of the operational area point to a performance gain using optimized threshold, the simpler improved threshold should be used.

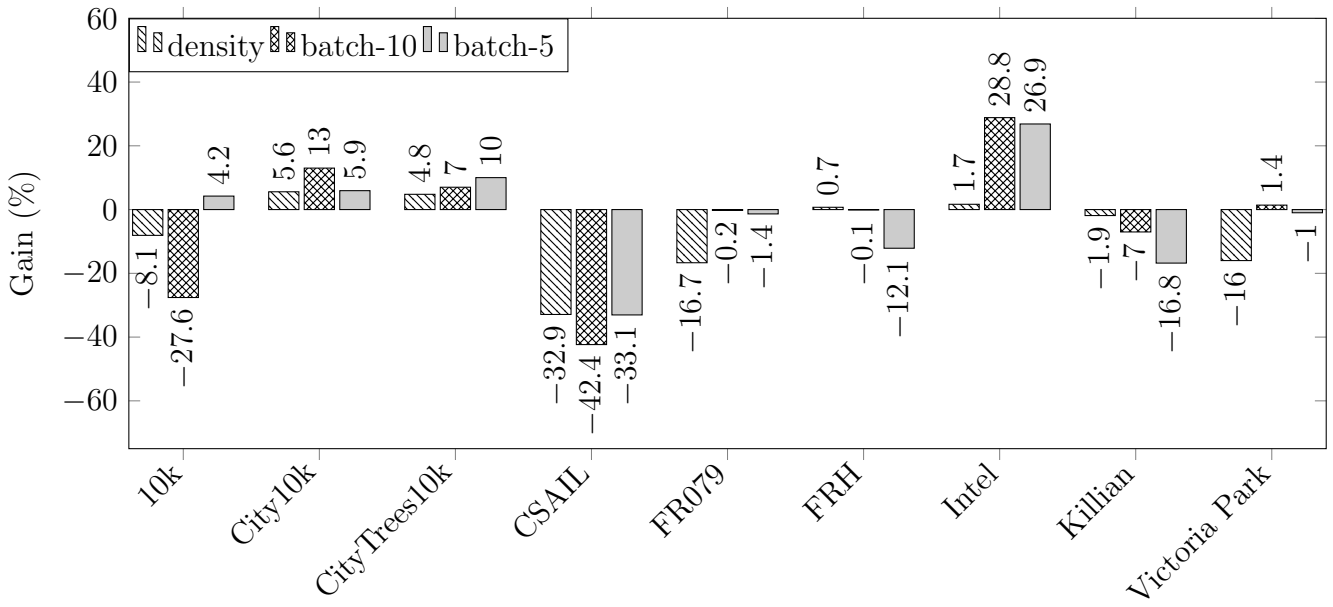


Figure 5.8: The total time performance gain between the Hybrid Cholesky with cost function (5.9) and regular Cholesky. Dataset-specific threshold are used.

5.5 Summary

In this chapter, a SLAM++ implementation using Hybrid Cholesky was compared against the traditional SLAM++ application in terms of factorisation time and execution time. The details of the experimental setup and datasets used was presented in Section 5.1.

Section 5.2 introduced an intuitive cost function and demonstrated that Hybrid Cholesky provides a boost in Factorisation speed (up to 71.7 %, 11.7 % on average) but this does not translate into general performance gains in terms of total execution time (up to 27.8 %, −1.9 % on average). It was demonstrated that the cost of computing the overhead has a significant effect on the factorisation time.

Section 5.3 proposed an improved cost function and threshold to minimise the overhead computation time. This yielded improvements in the factorisation performance (up to 80 %, 12.2 % on average) and a slight performance gain for the total computation time (up to 30.8 %, 1.9 % on average)

Section 5.4 proposed to use individual thresholds for each dataset in order to further increase performance. This did not result in any significant improvement on the improved cost function with a single threshold. The wide range of values obtained for the dataset thresholds cast doubt on the suitability of the cost function. Thresholds obtained by execution time minimisation, differing from the single threshold value as well as their calculated counterpart, further illustrated the work remaining to be done on this topic.

To conclude, the Hybrid Cholesky technique has shown to have the potential of significantly reducing the execution time of some SLAM solution algorithms. However, the gains achieved vary significantly from one dataset to the other. The importance of a carefully selected cost function and threshold to choose between Factor Recovery and Cholesky decomposition accurately, reliably and efficiently was demonstrated.

Chapter 6

Conclusion

The objective of this thesis is to explore a new and more general way to obtain a batch solution in SAM that is more computationally efficient, yet mathematically equivalent to the current state of the art. This is done by proposing three algorithms, culminating in the Hybrid Cholesky decomposition, which allows to choose between regular Cholesky and the Factor Recovery algorithm to ensure that the most efficient method is always selected.

6.1 Summary of Contributions

The three main contributions of this thesis are summarised as follows:

- **Factor Recovery Algorithm :** Solving a linear system using matrix factorisation is an important part of SAM and Cholesky factorisation is one of the most popular for such applications. The Factor recovery algorithm is a method to recover the Cholesky factor of a reordered symmetric positive definite matrix from its pre-reordering Cholesky factor. The new algorithm (Algorithm 4.3) treats the change of ordering as a series of column permutations and it was demonstrated that the computational cost of such permutations depends only on the number of non-zeros in the columns involved. In cases

where the change between orderings is localised, Factor Recovery is a viable alternative to recomputing the Cholesky factor, which is computationally expensive. The method presented here is, to the author's best knowledge, the first incremental Cholesky factor recovery algorithm. This has practical implications in the field of SLAM, where it can be used to replace batch reordering operations. The static nature of most nodes in SLAM graphs usually allows for small changes from one ordering to the next. However, it was found that this is not always the case and when significant changes occur, the Factor Recovery algorithm is less efficient than the batch method.

- **Threshold Selection :** To characterise the conditions in which Factor Recovery is more efficient than Cholesky re-computation, two cost functions ((5.5) and (5.9)) were proposed to evaluate the cost of the Factor Recovery operation. Experimental data on the factorisation speed between Cholesky and Factor Recovery was obtained using widely available datasets. For each method, a threshold ((5.8) and (5.10)) is found to identify the point of equal efficiency. The possibility of having an optimized threshold (Table 5.11) for each dataset was also analysed.

- **The Hybrid Cholesky Algorithm :** The Hybrid Cholesky algorithm (Algorithm 4.4) combines the two previous contributions, resulting in an algorithm that can be used in existing state of the art Cholesky based incremental SAM solvers with minimal software changes. By allowing reordering to occur at any time and only resort to batch operations if required, the quest for a truly incremental algorithm is taken one step further. Experimental results of SLAM++ with and without the Hybrid Cholesky algorithms show that the factorisation time can be reduced by 18 %, which translates to an average reduction of the total execution time by 1.9 %. The best case, obtained for the *Intel* dataset, shows a reduction of 78 % in factorisation time and 31 % in total execution time. while for the popular Victoria Park dataset, the execution time of batch steps are

reduced by an average of 7.1 %. For the worst case, *10K*, the execution time increases by 15.3 %.

6.2 Future Research

Although the contribution of this thesis significantly improves how the current state of the art handles column reordering, more work is required before a truly incremental SAM algorithm is obtained. In this section, improvements to the proposed algorithms are discussed and various areas of future research considered as out of the scope of this thesis are proposed.

- **Improved Threshold Selection:**

  In Section 5.2.1, the cost of recalculating the Cholesky factor or recovering it from the factor of the previous step is approximated and a threshold value for the ratio of these two quantities is obtained experimentally. This is used in Hybrid Cholesky to select the most efficient algorithm. Although the results obtained are generally satisfactory, this selection method has some limitations. One issue, as illustrated in Table 5.2, is the number of false positives. This is even more present in Table 5.8 of Section 5.3.1 where a more computationally efficient threshold is proposed, at the expense of precision.

  Section 5.4.1 illustrated in Table 5.11 that the thresholds vary significantly from one dataset to the next as well as from the calculated and experimental threshold for the same dataset when calculated for each datasets.

  This can be addressed by adjusting the exactitude of the cost function. However, more precise functions are typically harder to compute. More research is needed to formally identify measurable graph properties linked to the computational complexity and select cost functions that appropriately balance accuracy and speed.

Another area of improvement lies in the use of a single constant in (5.8). Using a function of multiple parameters instead of a fixed ratio as in (5.8) could help compensate for the approximation error and allow for a more general solution without requiring the evaluation of exact cost functions. To do so, the parameters affecting the error in the cost functions would need to be identified and their effects modelled.

The two avenues of improvement to increase the accuracy come with an increased overhead cost of Hybrid Cholesky and introduce tuning issues to obtain a proper balance of performance and speed. Future research should investigate if the information that has to be computed for both Factor Recovery and Cholesky decomposition (such as the elimination tree) as well as quantities that can be incrementally calculated (such as Cholesky factor and adjacency matrix density) can be leveraged to select the most efficient algorithm while minimising unnecessary computations. Desired characteristics of a cost function and threshold are: based on incrementally obtainable metrics, efficiently calculated, exact, and precise.

- **Incremental Re-orderings:**

  The Hybrid Cholesky algorithm presented in this thesis yields some significant time savings on reordering steps but the savings do not translate to significant reductions in the total execution time. This is illustrated by the low number of reordering operations done by SLAM++ (Table 5.1) and is due to current methods being designed to avoid these traditionally expensive operations. The Hybrid Cholesky method's performance for recovering Cholesky Factor after reordering offers the foundation for methods that use variable reordering more liberally and could be combined with online graph clustering algorithms such as Fennel [104] or xDGP [105] for a truly incremental reordering phase.

- **Compatible Orderings:**

The algorithm used for reordering when obtaining experimental results, COLAMD, does not take into consideration the current ordering. This can cause two consecutive orderings to vary greatly in terms of the number of manipulations required on columns to go from one to the other (KTD).

In this thesis, an attempt was made to use elimination tree rotations to choose an ordering that has the same number of fill-ins but is closer in terms KTD to the previous one. Although the concept was promising, mixed results were obtained. More investigation is needed to obtain an algorithm able to successfully pre-process the elimination tree such that it is in the most desirable configuration for Cholesky factor recovery.

- **Factor Recovery on the Elimination Tree:**

  In Section 4.4.1 and 4.4.2, the factor recovery operation is developed in terms of adjacent columns of $\mathbf{L}$. Reformulating and extending this algorithm to work on the elimination tree would allow better integration with graph based algorithms for ordering calculation and threshold selection. Furthermore, extending Proposition 4.4.1 to work with the path $P(j)$ instead of parent/child relationships would allow for more efficient sorting algorithms to be used instead of BubbleSort.

- **Integrated Reordering and Recovery:**

  In all cases surveyed as well as in the proposed algorithm, the calculation of an advantageous ordering and solving SLAM are two distinct problems. However, they are deeply connected and could benefit from being solved using the same algorithm. A promising area of research is the combination of the ordering calculation and factorisation steps. To to so, the METIS ordering algorithm could be made incremental, taking into consideration the previous order and moving columns only when required. Combined with the improvement suggested regarding Factor Recovery on the elimination tree, this would allow for a truly incremental and unified SAM algorithm.

References

[1] N. Fairfield, G. Kantor, and D. Wettergreen, "Real-time slam with octree evidence grids for exploration in underwater tunnels," *Journal of Field Robotics*, vol. 24, no. 1-2, pp. 03–21, 2007.

[2] J. Prado and L. Marques, "Energy efficient area coverage for an autonomous demining robot," in *ROBOT2013: First Iberian Robotics Conference*, Springer, 2014, pp. 459–471.

[3] S. Dogru and L. Marques, "A physics-based power model for skid-steered wheeled mobile robots," *IEEE Transactions on Robotics*, vol. 34, no. 2, pp. 421–433, 2018.

[4] J. Larson, B. Okorn, T. Pastore, D. Hooper, and J. Edwards, "Counter tunnel exploration, mapping, and localization with an unmanned ground vehicle," in *SPIE Defense+ Security*, International Society for Optics and Photonics, 2014, 90840Q–90840Q.

[5] J. Wilson, "A new era for airships," *Aerospace America*, vol. 42, no. 5, pp. 27–31, 2004.

[6] M. Evered, P. Burling, and M. Trotter, "An investigation of predator response in robotic herding of sheep," *International Proceedings of Chemical, Biological and Environmental Engineering*, vol. 63, pp. 49–54, 2014.

[7] S. Marden and M. Whitty, "Gps-free localisation and navigation of an unmanned ground vehicle for yield forecasting in a vineyard," in *Recent Advances in Agricul-*

tural Robotics, International workshop collocated with the 13th International Conference on Intelligent Autonomous Systems (IAS-13), 2014.

[8] C. S. Lee, S. Nagappa, N. Palomeras, D. E. Clark, and J. Salvi, "Slam with sc-phd filters: An underwater vehicle application," *IEEE Robotics & Automation Magazine*, vol. 21, no. 2, pp. 38–45, 2014.

[9] P. Ridao, M. Carreras, D. Ribas, and R. Garcia, "Visual inspection of hydroelectric dams using an autonomous underwater vehicle," *Journal of Field Robotics*, vol. 27, no. 6, pp. 759–778, 2010.

[10] J. A. Adams, J. L. Cooper, M. A. Goodrich, C. Humphrey, M. Quigley, B. G. Buss, and B. S. Morse, "Camera-equipped mini UAVs for wilderness search support: Task analysis and lessons from field trials," *Journal of Field Robotics*, vol. 25, no. 1-2, 2007.

[11] S. M. Adams and C. J. Friedland, "A survey of unmanned aerial vehicle (UAV) usage for imagery collection in disaster research and management," in *9th International Workshop on Remote Sensing for Disaster Response*, 2011.

[12] F. Colas, S. Mahesh, F. Pomerleau, M. Liu, and R. Siegwart, "3d path planning and execution for search and rescue ground robots," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2013, pp. 722–727.

[13] R. Sheh, A. Jacoff, A.-M. Virts, T. Kimura, J. Pellenz, S. Schwertfeger, and J. Suthakorn, "Advancing the state of urban search and rescue robotics through the robocuprescue robot league competition," in *Field and service robotics*, Springer, 2014, pp. 127–142.

[14] Z. A. Mundher and J. Zhong, "A real-time fall detection system in elderly care using mobile robot and kinect sensor," *International Journal of Materials, Mechanics and Manufacturing*, vol. 2, no. 2, pp. 133–138, 2014.

[15] W. Churchill and P. Newman, "Experience-based navigation for long-term localisation," *The International Journal of Robotics Research*, vol. 32, no. 14, pp. 1645–1661, 2013.

[16] M. Cummins and P. Newman, "Appearance-only slam at large scale with fab-map 2.0," *The International Journal of Robotics Research*, vol. 30, no. 9, pp. 1100–1123, 2011.

[17] A. Rottmann, M. Sippel, T. Zitterell, W. Burgard, L. Reindl, and C. Scholl, "Towards an experimental autonomous blimp platform," in *Proc. of the European Conf. on Mobile Robots (ECMR)*, 2007.

[18] D. B. Barber, J. D. Redding, T. W. McLain, R. W. Beard, and C. N. Taylor, "Vision-based target geo-location using a fixed-wing miniature air vehicle," *Journal of Intelligent and Robotic Systems*, vol. 47, no. 4, pp. 361–382, 2006.

[19] B. Kim, M. Kaess, L. Fletcher, J. Leonard, A. Bachrach, N. Roy, and S. Teller, "Multiple relative pose graphs for robust cooperative mapping," in *IEEE International Conference on Robotics and Automation*, 2010, pp. 3185–3192.

[20] L. Polok and I. Viorela, *Slam++*, 2015. [Online]. Available: http://sourceforge.net/projects/slam-plus-plus/.

[21] J. Neira and J. D. Tardós, "Data association in stochastic mapping using the joint compatibility test," *IEEE Transactions on Robotics and Automation*, vol. 17, no. 6, pp. 890–897, 2001.

[22] K. Murphy, "Bayesian map learning in dynamic environments," *Advances in Neural Information Processing Systems*, vol. 12, pp. 1015–1021, 1999.

[23] F. Dellaert and M. Kaess, "Square root SAM: Simultaneous localization and mapping via square root information smoothing," *The International Journal of Robotics Research*, vol. 25, no. 12, pp. 1181–1203, 2006.

[24] T. A. Davis and W. W. Hager, "Modifying a sparse Cholesky factorization," *SIAM Journal on Matrix Analysis and Applications*, vol. 20, no. 3, pp. 606–627, 1999.

[25] J. W. Liu, "The role of elimination trees in sparse factorization," *SIAM Journal on Matrix Analysis and Applications*, vol. 11, no. 1, pp. 134–172, 1990.

[26] A. H. Sherman, "On the efficient solution of sparse systems of linear and nonlinear equations," PhD thesis, Yale, 1975.

[27] A. George and J. W. Liu, "An optimal agorithm for symbolic factorization of symmetric matrices," *SIAM Journal on Computing*, vol. 9, no. 3, pp. 583–593, 1980.

[28] T. A. Davis, *Direct methods for sparse linear systems*. Siam, 2006, vol. 2.

[29] Y. Chen, T. A. Davis, W. W. Hager, and S. Rajamanickam, "Algorithm 887: CHOLMOD, supernodal sparse Cholesky factorization and update/downdate," *ACM Transactions on Mathematical Software*, vol. 35, no. 3, p. 22, 2008.

[30] J. D. Hogg, J. K. Reid, and J. A. Scott, "Design of a multicore sparse Cholesky factorization using DAGs," *SIAM Journal on Scientific Computing*, vol. 32, no. 6, pp. 3627–3649, 2010.

[31] D. Zou and Y. Dou, "Implementation of parallel sparse Cholesky factorization on GPU," in *International Conference on Computer Science and Network Technology*, 2012, pp. 2228–2232.

[32] S. C. Rennich, D. Stosic, and T. A. Davis, "Accelerating sparse Cholesky factorization on GPUs," in *Proceedings of the Fourth Workshop on Irregular Applications: Architectures and Algorithms*, 2014, pp. 9–16.

[33] J. K. Reid and J. A. Scott, "An out-of-core sparse Cholesky solver," *ACM Transactions on Mathematical Software*, vol. 36, no. 2, p. 9, 2009.

[34] T. A. Davis and W. W. Hager, "Row modifications of a sparse Cholesky factorization," *SIAM Journal on Matrix Analysis and Applications*, vol. 26, no. 3, pp. 621–639, 2005.

[35] M. Yannakakis, "Computing the mininimum fill-in is NP-complete," *SIAM Journal on Algebraic and Discrete Methods*, vol. 2, no. 1, 1981.

[36] P. Agarwal and E. Olson, "Variable reordering strategies for SLAM," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2012, pp. 3844–3850.

[37] W. F. Tinney and J. Walker, "Direct solutions of sparse network equations by optimally ordered triangular factorization," *Proceedings of the IEEE*, vol. 55, no. 11, pp. 1801–1809, Nov. 1967, ISSN: 0018-9219.

[38] D. J. Rose, "A graph-theoretic study of the numerical solution of sparse positive definite systems of linear equations," *Graph theory and computing*, vol. 183, p. 217, 1972.

[39] P. R. Amestoy, T. A. Davis, and I. S. Duff, "An approximate minimum degree ordering algorithm," *SIAM Journal on Matrix Analysis and Applications*, vol. 17, no. 4, pp. 886–905, 1996.

[40] T. A. Davis, J. R. Gilbert, S. I. Larimore, and E. G. Ng, "Algorithm 836: COLAMD, a column approximate minimum degree ordering algorithm," *ACM Transactions on Mathematical Software*, vol. 30, no. 3, pp. 377–380, 2004.

[41] A. George, "Nested dissection of a regular finite element mesh," *SIAM Journal on Numerical Analysis*, vol. 10, no. 2, pp. 345–363, 1973.

[42] R. J. Lipton, D. J. Rose, and R. E. Tarjan, "Generalized nested dissection," *SIAM journal on numerical analysis*, vol. 16, no. 2, pp. 346–358, 1979.

[43] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM Journal on scientific Computing*, vol. 20, no. 1, pp. 359–392, 1998.

[44] ——, "A parallel algorithm for multilevel graph partitioning and sparse matrix ordering," *Journal of Parallel and Distributed Computing*, vol. 48, no. 1, pp. 71–95, 1998.

[45] D. LaSalle and G. Karypis, "Multi-threaded graph partitioning," in *IEEE International Symposium on Parallel & Distributed Processing*, 2013, pp. 225–236.

[46] G. Karypis and V. Kumar, "Metis: Unstructured graph partitioning and sparse matrix ordering system, version 2.0, 1995," *Department of Computer Science, University of Minnesota*, 1995.

[47] R. Smith, M. Self, and P. Cheeseman, "Estimating uncertain spatial relationships in robotics," in *Autonomous robot vehicles*, 1990, pp. 167–193.

[48] M. G. Dissanayake, P. Newman, S. Clark, H. F. Durrant-Whyte, and M. Csorba, "A solution to the simultaneous localization and map building (slam) problem," *IEEE Transactions on Robotics and Automation*, vol. 17, no. 3, pp. 229–241, 2001.

[49] G. Dissanayake, H. Durrant-Whyte, and T. Bailey, "A computationally efficient solution to the simultaneous localisation and map building (slam) problem," in *IEEE International Conference on Robotics and Automation*, vol. 2, 2000, pp. 1009–1014.

[50] E. Hygounenc, I.-K. Jung, P. Soueres, and S. Lacroix, "The autonomous blimp project of laas-cnrs: Achievements in flight control and terrain mapping," *The International Journal of Robotics Research*, vol. 23, no. 4-5, pp. 473–511, 2004.

[51] L. G. Mirisola and J. Dias, "Tracking from a moving camera with attitude estimates," in *Israeli Conference on Robotics*, 2008.

[52] S. Ahrens, D. Levine, G. Andrews, and J. P. How, "Vision-based guidance and control of a hovering vehicle in unknown, gps-denied environments," in *IEEE International Conference on Robotics and Automation*, 2009, pp. 2643–2648.

[53] W. Bath and J. Paxman, "Uav localisation & control through computer vision," in *Proceedings of the Australasian Conference on Robotics and Automation*, 2004.

[54] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse, "Monoslam: Real-time single camera slam," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 6, pp. 1052–1067, 2007.

[55] D. Migliore, R. Rigamonti, D. Marzorati, M. Matteucci, and D. G. Sorrenti, "Use a single camera for simultaneous localization and mapping with mobile object tracking in dynamic environments," in *ICRA Workshop on Safe navigation in open and dynamic environments: Application to autonomous vehicles*, 2009.

[56] J. Kim and S. Sukkarieh, "Real-time implementation of airborne inertial-slam," *Robotics and Autonomous Systems*, vol. 55, no. 1, pp. 62–71, 2007.

[57] J. J. Leonard and H. F. Durrant-Whyte, "Simultaneous map building and localization for an autonomous mobile robot," in *IEEE International Workshop on Intelligent Robots and Systems*, 1991, pp. 1442–1447.

[58] S. Huang and G. Dissanayake, "Convergence and consistency analysis for extended kalman filter based slam," *IEEE Transactions on Robotics*, vol. 23, no. 5, pp. 1036–1049, 2007.

[59] S. J. Julier and J. K. Uhlmann, "A general method for approximating nonlinear transformations of probability distributions," Robotics Research Group, Department of Engineering Science, University of Oxford, Tech. Rep., 1996.

[60] N. Sunderhauf, S. Lange, and P. Protzel, "Using the unscented kalman filter in mono-slam with inverse depth parametrization for autonomous airship control," in *IEEE International Workshop on Safety, Security and Rescue Robotics*, 2007, pp. 1–6.

[61] R. Van Der Merwe and E. A. Wan, "The square-root unscented kalman filter for state and parameter-estimation," in *Acoustics, Speech, and Signal Processing, 2001. Proceedings.(ICASSP'01). 2001 IEEE International Conference on*, IEEE, vol. 6, 2001, pp. 3461–3464.

[62] S. Holmes, G. Klein, and D. W. Murray, "A square root unscented kalman filter for visual monoslam," in *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, IEEE, 2008, pp. 3710–3716.

[63] G. P. Huang, A. I. Mourikis, and S. I. Roumeliotis, "A quadratic-complexity observability-constrained unscented kalman filter for slam," *IEEE Transactions on Robotics*, vol. 29, no. 5, pp. 1226–1243, 2013.

[64] S. Tang, C. Yeong, E. Su, Y Subramaniam, and P. Chin, "Range and bearing-based simultaneous localization and mapping of unmanned ground vehicle using unscented kalman filter," *Journal of Telecommunication, Electronic and Computer Engineering (JTEC)*, vol. 8, no. 11, pp. 119–123, 2016.

[65] K Shala and Y Gao, "Comparative analysis of localisation and mapping techniques for planetary rovers," in *Proceedings of the International Symposium on Artificial Intelligence, Robotics and Automation in Space*, 2010.

[66] S. Thrun, Y. Liu, D. Koller, A. Y. Ng, Z. Ghahramani, and H. Durrant-Whyte, "Simultaneous localization and mapping with sparse extended information filters," *The International Journal of Robotics Research*, vol. 23, no. 7-8, pp. 693–716, 2004.

[67] M. R. Walter, R. M. Eustice, and J. J. Leonard, "Exactly sparse extended information filters for feature-based slam," *The International Journal of Robotics Research*, vol. 26, no. 4, pp. 335–359, 2007.

[68] V. Ila, J. M. Porta, and J. Andrade-Cetto, "Information-based compact pose slam," *IEEE Transactions on Robotics*, vol. 26, no. 1, pp. 78–93, 2010.

[69] R. M. Eustice, H. Singh, J. J. Leonard, and M. R. Walter, "Visually mapping the rms titanic: Conservative covariance estimates for slam information filters," *The international journal of robotics research*, vol. 25, no. 12, pp. 1223–1242, 2006.

[70] E. Asadi and M. Bozorg, "A decentralized architecture for simultaneous localization and mapping," *IEEE/ASME Transactions on Mechatronics*, vol. 14, no. 1, pp. 64–71, 2009.

[71] A. Doucet, N. De Freitas, K. Murphy, and S. Russell, "Rao-blackwellised particle filtering for dynamic bayesian networks," in *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*, 2000, pp. 176–183.

[72] A. Doucet and A. M. Johansen, "A tutorial on particle filtering and smoothing: Fifteen years later," *Handbook of nonlinear filtering*, vol. 12, no. 656-704, p. 3, 2009.

[73] M. Montemerlo, "Fastslam: A factored solution to the simultaneous localization and mapping problem with unknown data association," PhD thesis, University of Washington, 2003.

[74] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, "Fastslam 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges," in *International Joint Conference on Artificial Intelligence*, vol. 18, 2003, pp. 1151–1156.

[75] C. Kim, H. Kim, and W. K. Chung, "Exactly rao-blackwellized unscented particle filters for slam," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, IEEE, 2011, pp. 3589–3594.

[76] X. Zhu, F. Deng, Y. Ou, L. Liu, and E. Wang, "Vision-based semantic unscented fastslam for indoor service robot," *Mathematical Problems in Engineering*, vol. 2015, 2015.

[77] D. Hahnel, W. Burgard, D. Fox, and S. Thrun, "An efficient fastslam algorithm for generating maps of large-scale cyclic environments from raw laser range measurements," in *IEEE International Conference on Intelligent Robots and Systems*, vol. 1, 2003, pp. 206–211.

[78] C. Schröter, H.-J. Böhme, and H.-M. Gross, "Memory-efficient gridmaps in rao-blackwellized particle filters for slam using sonar range sensors.," in *European Conference on Mobile Robots*, 2007.

[79] C. Schroeter and H.-M. Gross, "A sensor-independent approach to rbpf slam-map match slam applied to visual mapping," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2008, pp. 2078–2083.

[80] R. Sim, P. Elinas, M. Griffin, and J. J. Little, "Vision-based slam using the rao-blackwellised particle filter," in *International Joint Conference on Artificial Intelligence - Workshop on Reasoning with Uncertainty in Robotics*, vol. 14, 2005, pp. 9–16.

[81] K. M. Wurm, C. Stachniss, and G. Grisetti, "Bridging the gap between feature and grid-based slam," *Robotics and Autonomous Systems*, vol. 58, no. 2, pp. 140–148, 2010.

[82] T. Duckett, "A genetic algorithm for simultaneous localization and mapping," in *IEEE International Conference on Robotics and Automation*, vol. 1, 2003, pp. 434–439.

[83] M. Begum, G. K. Mann, and R. G. Gosine, "An evolutionary slam algorithm for mobile robots," in *IEEE/RJS Conference on Intelligent Robots and Systems*, 2006, pp. 4066–4071.

[84] L. Moreno, S. Garrido, D. Blanco, and M. L. Muñoz, "Differential evolution solution to the slam problem," *Robotics and Autonomous Systems*, vol. 57, no. 4, pp. 441–450, 2009.

[85] D. J. Feng, S. Wijesoma, and A. P. Shacklock, "Genetic algorithmic filter approach to mobile robot simultaneous localization and mapping," in *Control, Automation, Robotics and Vision, 2006. ICARCV'06. 9th International Conference on*, IEEE, 2006, pp. 1–6.

[86] C.-H. Wu, Y.-H. Chen, Y.-Y. Lee, and C.-H. Tsai, "A fast genetic slam approach for mobile robots," in *IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*, 2013, pp. 563–568.

[87] M. Kaess, A. Ranganathan, and F. Dellaert, "iSAM: Incremental smoothing and mapping," *IEEE Transactions on Robotics*, vol. 24, no. 6, pp. 1365–1378, 2008.

[88] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. J. Leonard, and F. Dellaert, "Isam2: Incremental smoothing and mapping using the bayes tree," *The International Journal of Robotics Research*, vol. 31, no. 2, pp. 216–235, 2012.

[89] G. Huang, R. Truax, M. Kaess, and J. J. Leonard, "Unscented iSAM: A consistent incremental solution to cooperative localization and target tracking," in *IEEE European Conference on Mobile Robots*, 2013, pp. 248–254.

[90] L. Polok, M. Solony, V. Ila, P. Smrz, and P. Zemcik, "Incremental Cholesky factorization for least squares problems in robotics," in *Intelligent Autonomous Vehicles*, vol. 8, 2013, pp. 172–178.

[91] H. Strasdat, J. Montiel, and A. J. Davison, "Real-time monocular slam: Why filter?" In *IEEE International Conference on Robotics and Automation*, 2010, pp. 2657–2664.

[92] F. Moosmann and C. Stiller, "Velodyne slam," in *IEEE Intelligent Vehicles Symposium*, 2011, pp. 393–398.

[93] J. McDonald, M. Kaess, C. Cadena, J. Neira, and J. J. Leonard, "6-dof multi-session visual slam using anchor nodes," in *European Conference on Mobile Robotics*, vol. 13, 2011.

[94] T. A. Davis and W. W. Hager, "Multiple-rank modifications of a sparse Cholesky factorization," *SIAM Journal on Matrix Analysis and Applications*, vol. 22, no. 4, pp. 997–1013, 2001.

[95] L. Polok, M. Solony, V. Ila, P. Smrz, and P. Zemcik, "Efficient implementation for block matrix operations for nonlinear least squares problems in robotic applications," in *IEEE International Conference on Robotics and Automation*, 2013, pp. 2263–2269.

[96] R. Küummerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, "G 2 o: A general framework for graph optimization," in *IEEE International Conference on Robotics and Automation*, 2011, pp. 3607–3613.

[97] V. Ila, L. Polok, M. Solony, P. Smrz, and P. Zemcik, "Fast covariance recovery in incremental nonlinear least square solvers," in *IEEE International Conference on Robotics and Automation*, 2015, pp. 4636–4643.

[98] P. E. Gill, G. H. Golub, W. Murray, and M. A. Saunders, "Methods for modifying matrix factorizations," *Mathematics of Computation*, vol. 28, no. 126, pp. 505–535, 1974.

[99] G. Grisetti, C. Stachniss, S. Grzonka, and W. Burgard, "A tree parameterization for efficiently computing maximum likelihood maps using gradient descent.," in *Robotics: Science and Systems*, vol. 3, 2007, p. 9.

[100] R. Kümmerle, B. Steder, C. Dornhege, M. Ruhnke, G. Grisetti, C. Stachniss, and A. Kleiner, *Slam benchmarking*, 2015. [Online]. Available: http://kaspar.informatik.uni-freiburg.de/~slamEvaluation/index.php.

[101] M. Bosse, P. Newman, J. Leonard, and S. Teller, "Simultaneous localization and map building in large-scale cyclic environments using the atlas framework," *The International Journal of Robotics Research*, vol. 23, no. 12, pp. 1113–1139, 2004.

[102] T. M. Chan and M. Pătraşcu, "Counting inversions, offline orthogonal range counting, and related problems," in *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, Society for Industrial and Applied Mathematics, 2010, pp. 161–173.

[103] S. Touchette, W. Gueaieb, and E. Lanteigne, "Efficient cholesky factor recovery for column reordering in simultaneous localisation and mapping," *Journal of Intelligent & Robotic Systems*, vol. 84, no. 1, pp. 859–875, Dec. 2016, ISSN: 1573-0409. DOI: 10.1007/s10846-016-0367-7. [Online]. Available: https://doi.org/10.1007/s10846-016-0367-7.

[104] C. Tsourakakis, C. Gkantsidis, B. Radunovic, and M. Vojnovic, "FENNEL: Streaming graph partitioning for massive scale graphs," in *Proceedings of the 7th ACM international conference on Web search and data mining*, ACM, 2014, pp. 333–342.

[105] L. Vaquero, F. Cuadrado, D. Logothetis, and C. Martella, "XDGP: A dynamic graph processing system with adaptive partitioning," *ACM Computing Research Repository*, 2013.

[106] J. E. Guivant and E. M. Nebot, "Optimization of the simultaneous localization and map-building algorithm for real-time implementation," *IEEE Transactions on Robotics and Automation*, vol. 17, no. 3, pp. 242–257, 2001.

[107] S. B. Williams, "Efficient solutions to autonomous mapping and navigation problems," PhD thesis, The University of Sydney, 2001.

[108] S. B. Williams, G. Dissanayake, and H. Durrant-Whyte, "An efficient approach to the simultaneous localisation and mapping problem," in *IEEE International Conference on Robotics and Automation*, vol. 1, 2002, pp. 406–411.

[109] C. Cadena and J. Neira, "Slam in o (logn) with the combined kalman-information filter," *Robotics and Autonomous Systems*, vol. 58, no. 11, pp. 1207–1219, 2010.

[110] T. Bailey, "Mobile robot localisation and mapping in extensive outdoor environments," PhD thesis, University of Sydney, 2002.

[111] C. Estrada, J. Neira, and J. D. Tardós, "Hierarchical slam: Real-time accurate mapping of large environments," *IEEE Transactions on Robotics*, vol. 21, no. 4, pp. 588–596, 2005.

[112] K. Ni, D. Steedly, and F. Dellaert, "Tectonic SAM: Exact, out-of-core, submap-based SLAM," in *IEEE International Conference on Robotics and Automation*, 2007, pp. 1678–1685.

[113] K. Ni and F. Dellaert, "Multi-level submap based SLAM using nested dissection," in *IEEE International Conference on Intelligent Robots and Systems*, 2010, pp. 2558–2565.

[114] J. M. Montiel, J. Civera, and A. J. Davison, "Unified inverse depth parametrization for monocular slam," Robotics: Science and Systems, 2006.

[115] K. Choi, J. Park, Y.-H. Kim, and H.-K. Lee, "Monocular slam with undelayed initialization for an indoor robot," *Robotics and Autonomous Systems*, vol. 60, no. 6, pp. 841–851, 2012.

[116] S. Saeedi, M. Trentini, M. Seto, and H. Li, "Multiple-robot simultaneous localization and mapping: A review," *Journal of Field Robotics*, vol. 33, no. 1, pp. 3–46, 2016.

[117] M. Bryson and S. Sukkarieh, "Architectures for cooperative airborne simultaneous localisation and mapping," *Journal of Intelligent and Robotic Systems*, vol. 55, no. 4-5, pp. 267–297, 2009.

[118] A. Gil, Ó. Reinoso, M. Ballesta, and M. Juliá, "Multi-robot visual slam using a rao-blackwellized particle filter," *Robotics and Autonomous Systems*, vol. 58, no. 1, pp. 68–80, 2010.

[119] D. Benedettelli, A. Garulli, and A. Giannitrapani, "Cooperative slam using m-space representation of linear features," *Robotics and Autonomous Systems*, 2012.

[120] A. Cunningham, M. Paluri, and F. Dellaert, "DDF-SAM: Fully distributed SLAM using constrained factor graphs," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2010, pp. 3025–3030.

[121] A. Cunningham, K. M. Wurm, W. Burgard, and F. Dellaert, "Fully distributed scalable smoothing and mapping with robust multi-robot data association," in *IEEE International Conference on Robotics and Automation*, 2012, pp. 1093–1100.

[122] A. Cunningham, V. Indelman, and F. Dellaert, "DDF-SAM 2.0: Consistent distributed smoothing and mapping," in *IEEE International Conference on Robotics and Automation*, 2013, pp. 5220–5227.

[123] E. Guerra, R. Munguia, Y. Bolea, and A. Grau, "New validation algorithm for data association in slam," *ISA transactions*, vol. 52, no. 5, pp. 662–671, 2013.

[124] H. Andreasson, T. Duckett, and A. Lilienthal, "Mini-slam: Minimalistic visual slam in large-scale environments based on a new interpretation of image similarity," in *Robotics and Automation, 2007 IEEE International Conference on*, IEEE, 2007, pp. 4096–4101.

[125] G. Silveira, E. Malis, and P. Rives, "An efficient direct approach to visual slam," *IEEE Transactions on Robotics*, vol. 24, no. 5, pp. 969–979, 2008.

[126] J. Engel, T. Schöps, and D. Cremers, "Lsd-slam: Large-scale direct monocular slam," in *European Conference on Computer Vision*, Springer, 2014, pp. 834–849.

[127] K. P. Murphy, "Dynamic bayesian networks," PhD thesis, UC Berkeley, 2002.

[128] C. Kerl, J. Sturm, and D. Cremers, "Robust odometry estimation for rgb-d cameras," in *IEEE International Conference on Robotics and Automation*, 2013, pp. 3748–3754.

[129] A. J. Glover, W. P. Maddern, M. J. Milford, and G. F. Wyeth, "Fab-map+ ratslam: Appearance-based slam for multiple times of day," in *IEEE International Conference on Robotics and Automation*, 2010, pp. 3507–3512.

APPENDICES

Appendix A

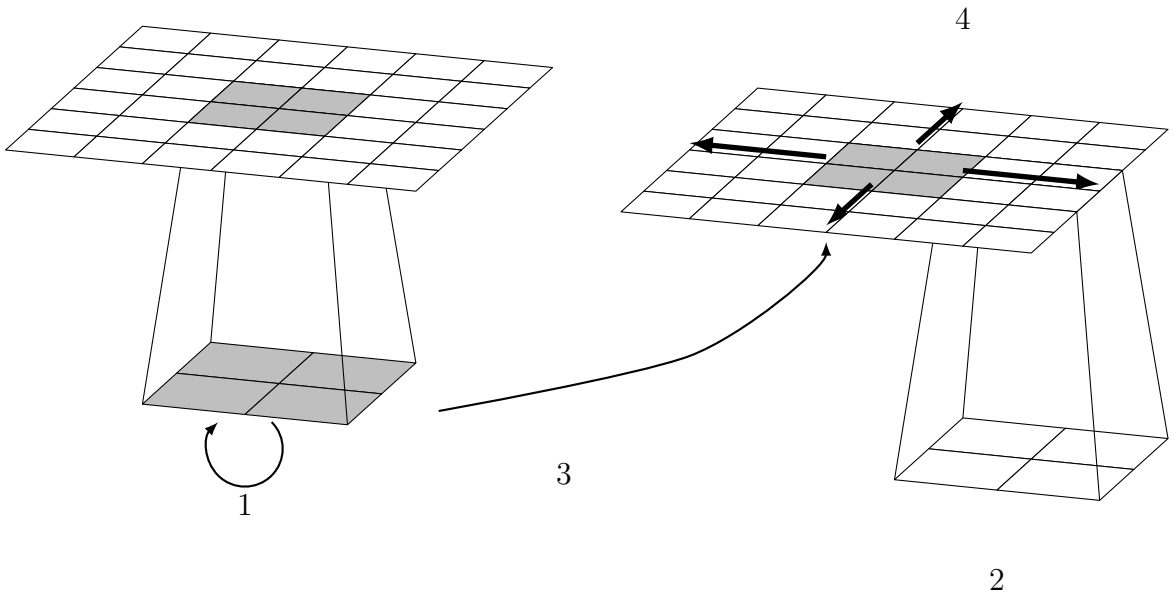
SLAM Algorithms - Extensions

As SLAM algorithms are tested in increasingly large environments and conditions, methods have been proposed to allow them to cope with more exacting requirements. In this appendix, the most important problems encountered by the basic SLAM algorithms discussed in Section 3.1 are presented. Proposed solutions and their advantages are discussed.

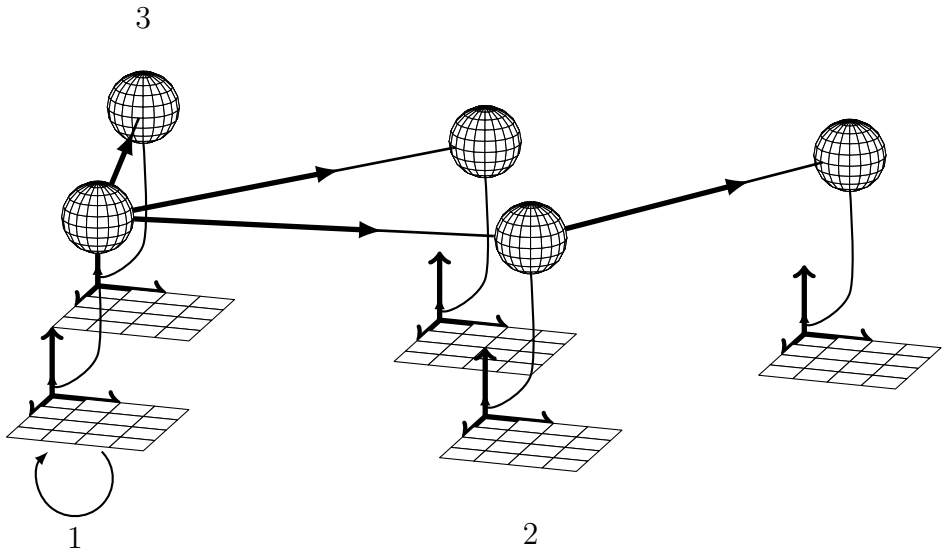
A.1 Dealing with Large Maps: Submap Methods

Many have proposed solutions to further mitigate the computational complexity of the EKF in large environments by the use of submaps. Such approaches consist of dividing large areas into smaller problems in the scope of which the algorithms of Section 2.1 are computationally tractable. A second algorithm manages update propagations to other submaps or to the global map. The methods to accomplish this can be divided in two categories based on whether they keep a global map (Figure A.1a) or represent it as linked submaps (Figure A.1b).

Following the global map approach, Guivant *et al.* [106] present a postponement method using a block matrix decomposition of the EKF equations called Compressed EKF (CEKF). This method allows to separate a small number n_l of landmarks from the other $n - n_l$



(a) Global map submap implementation. SLAM is done on the local map (1) until a new submap is loaded (2). The previous local map is integrated in the global map (3) and the complete map is updated (4).



(b) Local submap implementation. SLAM is done on the local map (1) until a new submap is loaded (2). The effect of the changes to the previous submap is propagated to other submaps (3).

Figure A.1: Submap implementations based on global map (A.1a) work on small sections of the global map and updates are propagated only when needed and amortised over longer periods. Implementations based on local submaps (A.1b) consist of a graph of relations between submaps, which is also updated only when needed.

landmarks of the map. The n_l landmarks constitute the local map, on which the SLAM problem is solved using the EKF. Integrating new data to the local map has a cost of $O(n_l^2)$. When a new local map is required or when the vehicle leaves the area, the full covariance matrix is updated at the Full SLAM cost of $O(n_l(n-n_l)^2)$. This can be done in the background and amortised over the time the vehicle spends in the area. The overall complexity of the algorithm is thus $O(n_l^2 + \frac{n_l(n-n_l)^2}{P})$ where the price of a full update is amortised over the number of steps P the vehicle stays in the local map. The authors show, using simulations, that their method gives the same numerical results as EKF based SLAM after the global update is performed. They also present a suboptimal method that discards some landmarks to further reduce computation time. In the suboptimal case, the total number of states is divided in n_p and n_d landmarks. Every landmark's states are updated but the covariances are only updated for the n_p landmarks, while the others retain their previous covariance values. This results in conservative estimates for n_d landmarks. Further improvements can be done by using local reference frames for submaps. The benefit of this method is somewhat negated if the vehicle changes area often and also requires the area to be larger than the sensor range to avoid detecting landmarks from other submaps. Although the computational improvement is significant, the availability of a global map is delayed and the cost is still proportional to the number of landmarks.

Williams [107] and Williams *et al.* [108] propose the Constrained Local Submap Filter (CLSF) which, in the same fashion as the CEKF, uses a block matrix decomposition of the EKF and local reference frames assigned to sub-maps. The difference lies in the fact that submaps are not geographically defined and are created whenever it is deemed appropriate, such as when the current submap has too many landmarks. When a global update is required, the local submap is transformed to the global reference frame and a data association strategy is used to find common landmarks between the local and global map. These matches are then used to update the global map at a cost of $O(n_c n^2)$ where n_c is the number of common landmarks and n is the total number of landmarks. This method has

a total complexity similar to the CEKF but the full update is dependent on the number of landmarks that are common to both local and global maps rather than the number of local landmarks. This yields a complexity of $O(n_l^2 + \frac{n_c n^2}{P})$ where n_l is the number of landmark in the local map. This method is not as sensitive to fast moving vehicles as the CEKF but still requires the map to be larger than the sensor range. It also depends heavily on correct data association between local and global maps during the full update. A very desirable by-product of using a delayed update method such as the CEKF and CLSF is that data association errors that are caught before the global update can be resolved. Corruption of the global map is thus avoided. Cadena *et al.* [109] propose a Combined Filter solution (CF-SLAM) which implements a sub-map based algorithm where individual submaps are solved using EKFs and merged to the global map using an EIF. The resulting algorithm has an amortised cost of $O(n \log n)$ in the best case.

Using the connected submap approach, Bailey [110] presents the Network Coupled Feature Maps (NCFM) which uses a graph representation of the world where the edges hold the coupling transformation and covariance between independent submaps represented as nodes. The vehicle states are local to the submap currently used and transferred to another via the coupling transformation when moving to the next area. The algorithm presented has a storage complexity of $O(kn_l^2)$ and an update computation complexity of $O(n_l^2)$ where k is the number of submaps and n_l is the number of landmarks per submap. The vehicle pose with regards to any reference frame can be computed in $O(k)$. The sequence of local submap transformations to use for pose calculations is obtained by breadth first search starting from the initial position, which leads to a suboptimal solution. A method searching for loop closures is also presented, but is treated as an additional step and not included in the computational cost. Estrada *et al.* [111] present the hierarchical SLAM algorithm. This method is similar to NCFM as the lower level consists of multiple statistically independent submaps that are updated with traditional EKF techniques. The coupling transformation between maps are, however, included in the individual submaps' filter. At the global

level, the individual submaps are represented as graph nodes with links to neighbouring submaps defined by the coupling information stored. The coupling states and covariance are also stored in a global filter which maintains cross-correlation and propagate changes to other couplings. The loop closing is also handled as a special case and overlapping maps where the loop was detected are fused. Closing loops improves the accuracy of all couplings in the global map, but this does not propagated back to the local maps. As for all submap methods, observations near the frontier of submaps ignore the correlation between landmarks belonging to different submaps to avoid correlating the two submaps.

Ni *et al.* [112] propose a two level submap extension to $\sqrt{\text{SAM}}$ called Tectonic Smoothing and Mapping (T-SAM). In their algorithm, the set of all variables (vehicle poses or landmarks) is geographically divided into submaps in such a way that some measurements (inter-measurements) link variables of two submaps while others (intra-measurements) are local. Each submaps is assigned a local reference frame to preserve the intra-measurements' linearisation points and solved independently using $\sqrt{\text{SAM}}$. The variables at the intersection of two submaps (boundary variables) and all local reference frames are then optimised using the inter-measurements. This step has the effect of aligning the submaps with regards to each other. The authors report 50 % computational savings on linearisation with regards to $\sqrt{\text{SAM}}$ by re-linearising only inter-measurements. The authors have tested their algorithm on the Victoria Park dataset and demonstrate that results are obtained 83 % faster with T-SAM than with $\sqrt{\text{SAM}}$. This optimisation comes at the cost of accuracy since re-linearisation is not applied to all variables. This algorithm is further improved by Ni *et al.* [113] who use the nested dissection algorithm to obtain a tree structure of multi-level submaps. In T-SAM2 , the Full SLAM trajectory is divided in a way that minimises the number of inter-measurements between submaps, speeding up the submap adjustment computations. Furthermore, the multi-level map increases scalability. The authors report, using the Victoria Park dataset, that T-SAM2 yields computational time savings of 95 % when compared with $\sqrt{\text{SAM}}$. This comes as a possible cost in precision and can only be

executed off-line since the map dissection requires knowledge of the complete trajectory and map. Therefore, T-SAM2 is of limited use for online implementations.

A.2 Dealing with non-linearities - Parametrisation

In SLAM implementations using a camera as a sensor, termed visual SLAM, the EKF can exhibit convergence problems. These are, in part, due to the non-linearities of the projection equations used to bring the known state information (x_w, y_w, z_w) of each landmark from the world coordinate system to (u, v) image coordinates, where the depth (z axis in the camera frame) is at the denominator. The matrix formulation of these equations is

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \mathbf{K}\mathbf{T}_w^i \begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix} = \begin{bmatrix} \frac{w_p}{w_i}(f\frac{x_c}{z_c} + c_x) \\ -\frac{h_p}{h_i}(f\frac{y_c}{z_c} + c_y) \\ 1 \end{bmatrix} \tag{A.1}$$

where $\mathbf{T}_w^i$ is the transformation matrix from the world to the image reference frame, $\mathbf{K}$ is the matrix of camera parameters, (x_c, y_c, z_c) are the position of an object in the camera coordinate system, f is the focal distance of the lens, (c_x, c_y) is the center of the sensor, (w_p, h_p) is the dimensions of the picture and (w_i, h_i) are the dimensions of the sensor. It can be seen in (A.1) that z_c is at the denominator, resulting in a non-linear equation prone to linearisation error. To remove this non-linearity from the equations, Montiel *et al.* [114] propose a modification to the classical parametrisation used in SLAM. They parametrise a landmark's position by specifying the position at which the landmark was first viewed (s_x, s_y and s_z), an observation direction (ϕ, θ) and the inverse distance from the viewing

position along the specified direction (ρ). The resulting equation is

$$P(s_x, s_y, s_z, \theta, \phi, \rho) = \begin{bmatrix} s_x \\ s_y \\ s_z \end{bmatrix} + \frac{1}{\rho} m(\theta, \phi) \tag{A.2}$$

where $m(\theta, \phi)$ converts the angles to a unit direction vector. The viewing position and orientation are immediately available and the inverse distance is initialised as a Gaussian random variable between 0 and 1, accounting for distances from 1 m to infinity. The result requires no change to the EKF formulation, allow for undelayed initialisation of landmarks (initialisation after the first observation), reduce linearisation error and can be used to characterise landmarks at infinity. This comes at the cost of three extra parameters for each landmark (if 3D position is used) making the complexity grow four times faster. Despite the increased complexity, the ability to initialise features on the first observation has made this parametrisation popular with SLAM implementations using a monocular camera such as MonoSLAM [54].

A different parametrisation is proposed by Choi *et al.* [115], which uses the landmark position in the 3D camera frame with an assumed high covariance depth instead of the pixel coordinates, also bypassing non-linearities in the projection equations. This is done by converting the 2D measurement to the 3D camera frame with unit depth using the calibration matrix ($\mathbf{K}$) and rotation matrix between the image and camera frames ($\mathbf{R}_c^i$) resulting in the direction vector $\mathbf{c}_t$ as shown in (A.3).

$$\begin{bmatrix} \mathbf{c}_t \end{bmatrix} = (\mathbf{K}\mathbf{R}_c^i)^{-1} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \tag{A.3}$$

Choosing depth d from previous observations, the observation in the camera frame can be calculated by

$$\mathbf{z} = d\frac{c_t}{||c_t||} \tag{A.4}$$

Which is compared to the landmarks state converted to the camera frame for the update step of the filter. The sensor noise covariance of the depth is initialised to a sufficiently high value to avoid affecting the current depth estimate. The noise covariance of the other two axes depend on the depth d, tracking error ϵ_{tr} and calibration error ϵ_c as well as the focal length in pixel units and is calculated by

$$\sigma^2 = \sqrt{\epsilon_{tr}^2 + \epsilon_c^2}\frac{d}{f} \tag{A.5}$$

This method maintains the same number of parameters as the classical method, hereby incurring no growth penalty. The method also performs better than inverse depth in cases where forward movement is concerned and equally well in other cases. The downside of this method is the lack of support for the observation of features too far for their position to be measured by the available camera and reasonable vehicle displacement. For such features, such as stars, only a direction measurement is possible.

A.3 Dealing with Multiple Vehicles - Distributed Mapping

Distributed mapping is of prime importance in SLAM as many tasks require or are accomplished faster by teams of multiple mobile vehicles. Many authors have proposed adaptations of previously discussed algorithms to distributed environment, sharing map between vehicles in a consistent fashion. A thorough survey of multi-vehicle SLAM can be found in [116].

Bryson *et al.* [117] propose an algorithm that allows multiple vehicles to cooperate. It used the EIF to merge maps created by independent vehicles in both centralised and decentralised algorithms. The authors demonstrate that the decentralised algorithm's bandwidth savings come at the cost of convergence speed (5 to 6 times longer) but does not compromise the end result's accuracy. Gil *et al.* [118] extend the FastSLAM algorithm to allow for multiple vehicles. More specifically, they focus on associating and successfully merging features observed by different robots. Their experimentation with artificial data show that teams of robots can produce maps with lower RMS error than single robots. Their method requires, however, that the processing be centralised or duplicated locally on each vehicle. Benedettelli *et al.* [119] use a completely different approach and present and EKF based solution tailored for M-space environment representations, which allows for partial feature initialisation and efficient map fusion. In their implementation, the vehicles operate independently but share map information when they meet. Their results on experimental data show that using two robots, the same area can be mapped in half the time without affecting the map's accuracy. Furthermore, the map produced is more detailed in terms of the number of features present.

There has been many adaptations of the iSAM smoothing method to multiple vehicles. Kim *et al.* [19] use a central graph representing all individual vehicle trajectories and maps. This is done by introducing two concepts: encounter factors, which link observations of a landmark in another vehicle's map or of the other vehicle itself and anchor nodes, which represent the relation between a global frame and a given vehicle trajectory's reference frame. The algorithm is tested on simulation and experimental data with success but assumes there is no limitation on connectivity and bandwidth, thus neglecting the cost of copying large maps between vehicles.

Cunningham *et al.* [120] introduce the Distributed Data Fusion Smoothing and Mapping (DDF-SAM) algorithm where each vehicle has a local graph (robot poses and landmarks) and a neighbourhood map. The neighbourhood map is generated at each vehicle

by marginalising out all pose information from the local graph. The result is transmitted to other vehicles. Each vehicle accumulate neighbourhood maps received from others and add it to their own neighbourhood map. To avoid including the same information multiple times in the SLAM algorithm, the neighbourhood map information is never merged back into the local graph. Bandwidth usage is kept lower than the anchor node method previously explained but, since the neighbourhood map is not merged, its information does not increase the accuracy of the trajectory estimation of individual vehicles. Data association is also assumed to be correct. Cunningham *et al.* [121] build uppon DDF-SAM to include multi-vehicle data association. Cunningham *et al.* [122] also propose DDF-SAM2, which introduces the notion of anti-factors to cancel unwanted information from the neighbourhood map, allowing it to be merged directly with the local graph. This enables the neighbourhood information to contribute to the trajectory estimation accuracy. The authors also present three methods of producing the neighbourhood map and show that their anti-factor method produces consistent estimates.

A.4 Dealing with Wrong Data Association

Data association errors occur when a detected feature is associated to a landmark it does not correspond to. This can have severe consequences on Bayes filter based SLAM algorithms (such as Kalman filters) because once added to the filter, the erroneous data can't be corrected. Since such data association errors can lead to divergence, special care must be taken to ensure that measurements and landmarks are identified and paired correctly. Most methods based on Full SLAM (2.18) are not affected by this as all the information is stored and can later be modified.

A.4.1 Robust Data Association

One way to solve problems associated with data association errors is by using methods that take more information, such as covariance or sets of observations, in consideration when associating measurements to landmarks. Bailey [110] presents the Combined Constrained Data Association (CCDA) which considers associations in groups rather than individually, allowing for more robust results. The first step consists of building a compatibility graph where each node is a possible measurement-landmark association and links connect compatible nodes. A second step searches the graph to find the largest cluster of compatible associations. The total cost of this algorithm is $O(m^2)$ where m is the number of measurements.

Guerra *et al.* [123] introduces the Highest Order Hypothesis Compatibility Test (HOHCT) as a method of data association for monocular SLAM. The proposed method reduces the appearance of false landmarks and has a complexity that rarely exceeds second order. This is done by first associating landmarks and then trying to obtain the largest subset of compatible matches starting with the hypothesis that all matches are correct and iteratively decreasing the number of matches assumed correct until a solution is found. The authors compare their technique to Joint Compatibility Branch and Bound [21] and report a 50% reduction in average computation time and a standard deviation an order of magnitude smaller.

A.4.2 Direct Methods

Another way to solve the data association problem is to bypass the landmark matching step entirely. Andreasson *et al.* [124] introduce the mini-SLAM algorithm, which uses odometry and omnidirectional visual images. Image similarities are used to identify the pose difference between images and its uncertainty, while multilevel relaxation is used to obtain a maximum likelihood solution. In experimental results, their algorithm maintains

a position estimate error below 3 m while odometry only goes above 200 m. Silveira *et al.* [125] have formulated the visual SLAM problem as an image registration task, thus removing the requirement of identifying and matching landmarks. They go further by considering not only position, but structure, illumination and rigidity parameters to build a high precision map containing more information. Engel *et al.* [126] propose LSD-SLAM, a direct method based on estimating the 3D transformation between images. Experimental results on popular datasets show that their method has a RMSE error an order of magnitude bellow mono-SLAM in certain cases.

A.4.3 Others

Although not designed specifically to solve the data association problem, most submap methods discussed in Section A.1 are more robust to such errors. The delay between the local and global map solutions is such that multiple measurements of each features can be integrated in the local map before it is merged with the global map. Unstable features will be discarded rather than merged, thus avoiding the introduction of errors in the global map.

A.5 Dealing with Dynamic Environments

Dealing with dynamic environments has always been a consideration for SLAM problems. In early works, Murphy [22, 127] accounts for changes in the environment in metric maps. Migliore *et al.* [55] propose a method combining monoSLAM and a bearing only tracker in parallel to increase the robustness and maintain an estimation of feature trajectories. Working with RGB-D cameras, Kerl *et al.* [128] present a robust camera motion estimator that exclude moving objects in the scene from motion calculations. Most other algorithms

can also be adapted to dynamic environments by storing higher order states for the landmarks or targets such as in U-iSAM [89].

Dynamic environments in SLAM is not limited to obstacles in motion but also extends to weather, seasons and time of day. Glover *et al.* [129] introduce an algorithm combining FAB-MAP's lighting invariant data association with RatSLAM to produce an appearance based method capable of performing localisation and mapping in outdoor environments at different times of day. Churchill *et al.* [15] propose an experience based implementation of SLAM that allows the vehicle to successfully recognise a slowly varying environment. This is done by keeping experiences of the environment when the same area is revisited but no features are recognised. The author demonstrates that this accounts for changes such as time of day and weather. From data recorded under various conditions, they show that after the 30^{th} visit, the vehicle fails to recognise the environment less than 10% of the time, compared to an average of about 30% during the first 15 visits.