# Sparsity and Computation Reduction for High-Rate Visual-Inertial Odometry

by

Kristoffer M. Frey

S.B., Massachusetts Institute of Technology, 2015

Submitted to the Department of Aeronautics and Astronautics in partial fulfillment of the requirements for the degree of Master of Science in Aeronautics and Astronautics

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2017

Author ............... Signature redacted ........
Department of Aeronautics and Astronautics
August 24, 2017

Certified by.... Signature redacted ..................
Jonathan P. How
R. C. Maclaurin Professor of Aeronautics and Astronautics, MIT
Thesis Supervisor

Certified by Signature redacted ......................
Theodore J. Steiner
Senior Member of the Technical Staff, Draper
Thesis Supervisor

Accepted by ............ Signature redacted .........
Hamsa Balakrishnan
Associate Professor of Aeronautics and Astronautics, MIT
Chair, Graduate Program Committee

# Sparsity and Computation Reduction for High-Rate Visual-Inertial Odometry

by

Kristoffer M. Frey

Submitted to the Department of Aeronautics and Astronautics
on August 24, 2017, in partial fulfillment of the
requirements for the degree of
Master of Science in Aeronautics and Astronautics

## Abstract

The navigation problem for mobile robots operating in unknown environments can be posed as a subset of Simultaneous Localization and Mapping (SLAM). For computationally-constrained systems, maintaining and promoting system sparsity is key to achieving the high-rate solutions required for agile trajectory tracking. This thesis focuses on the computation involved in the elimination step of optimization, showing it to be a function of the corresponding graph structure. This observation directly motivates the search for measurement selection techniques to promote sparse structure and reduce computation. While many sophisticated selection techniques exist in the literature, relatively little attention has been paid to the simple yet ubiquitous heuristic of decimation. This thesis shows that decimation produces graphs with an inherently sparse, partitioned super-structure. Furthermore, it is shown analytically for single-landmark graphs that the even spacing of observations characteristic of decimation is near optimal in a weighted number of spanning trees sense. Recent results in the SLAM community suggest that maximizing this connectivity metric corresponds to good information-theoretic performance. Simulation results confirm that decimation-style strategies perform as well or better than sophisticated policies which require significant computation to execute. Given that decimation consumes negligible computation to evaluate, its performance demonstrated here makes decimation a formidable measurement selection strategy for high-rate, realtime SLAM solutions. Finally, the SAMWISE visual-inertial estimator is described, and thorough experimental results demonstrate its robustness in a variety of scenarios, particularly to the challenges prescribed by the DARPA Fast Lightweight Autonomy program.

Thesis Supervisor: Jonathan P. How
Title: R. C. Maclaurin Professor of Aeronautics and Astronautics, MIT

Thesis Supervisor: Theodore J. Steiner
Title: Senior Member of the Technical Staff, Draper

## Acknowledgments

None of this work would have been possible without the indefatigable guidance of my advisor, Professor How, and Draper mentor, Ted Steiner. Professor How kept me asking the right questions, and whenever one line of investigation dried up, helped ground my thinking properly in the search for the next. Similarly, Ted has always made time for me, scheduled or not, and has offered immeasurable guidance through all aspects of graduate school.

I also cannot thank Kasra Khosoussi enough for his insights and tireless contribution to some of the analytic results of Chapter 3. Beyond that, his comments helped clarify much of the motivation and story behind this thesis in recent weeks.

I am immensely grateful to Draper for supporting my work and providing something of a parallel education. Particular thanks should be given to Rob Truax for all the code reviews and guidance he gave as I joined the SAMWISE effort, and for all the jokes he inserted along the way. In the same vein, being a part of the ACL (Building 41 in particular) has been one of the most enjoyable aspects of showing up to work every day.

Being a member of the Draper-MIT team on the DARPA FLA program has made research both purposeful and fun, both because of the work I have been able to be a part of and because of the people I have had the pleasure of working with. After all those trips to central Florida, few establishments have a place in my heart quite like the Taqueria Merlo or the patio of the Residence Inn.

Lastly, I would be remiss not to thank Michael Grenier and the rest of the Muddy Charles staff for providing a bit of balance in my life. While not representing a substantial portion of my time, being a part of the Muddy has been a substantial part of my MIT experience.

## Assignment

## Acknowledgment of Support

This thesis was supported by the Defense Advanced Research Projects Agency (DARPA) under the Fast Lightweight Autonomy program. The contents, views, and/or findings expressed are those of the author and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

The navigation problem is ubiquitous in mobile robotics [7, 8, 13, 21, 27, 29, 46, 59]. In this thesis, *navigation* refers to the determination of the robot's position (more generally position and orientation, or *pose*) at every instant in time with respect to some fixed local or global frame. Here, navigation will not refer to trajectory *planning*, which concerns where to go next, but rather *localization* of the robot at the current time or in the past.

Having a solution to the navigation problem is a prerequisite to trajectory planning and control, and it is difficult to imagine a real-world mission of a mobile robot that does not require some form of navigation. For example, an autonomous vehicle which needs to maneuver to a goal position needs to know its own position at every instant in order to make sure it is heading in the right direction. A robot searching for a specific target or exploring a large area needs to reason about where it has been in order to ensure it searches efficiently and can communicate the target location or describe the environment afterward.

The solution to the navigation problem can be as simple as relying on the GPS satellite constellation via many off-the-shelf solutions, or using some other domain-specific infrastructure such as motion-capture. However, these dependencies on existing infrastructures or *known* maps restrict the applicability of such systems to environments in which such infrastructure exists. In many environments of interest to robotic systems, such as underwater, indoors, and the surfaces of other planets, GPS is completely unavailable. Additionally, systems such as GPS can be crippled or jammed by hostile actors, so solutions which can operate independently or degrade "gracefully" under loss of GPS signal are highly desirable in military or other critical applications.

Many self-contained, or *onboard*, navigation solutions have been proposed over the past half-century. Inertial navigation methods and well-known algorithms such as the Kalman Filter [34] and its derivatives [3,30,39,46] have been used in pose estimation for decades. In 1969, the Apollo navigation system [27] pioneered by the MIT Instrumentation Lab demonstrated the capability of fusing inertial measurements with external references (star sightings) to successfully navigate the lunar capsule to the moon and back.

Inertial methods rely primarily on sensors such as gyroscopes and accelerometers, which often are bundled as Inertial Measurement Units (IMUs). They are completely self-contained, relatively inexpensive, and low-SWaP (Size, Weight, and Power). Additionally, they can provide a high-rate ($> 1$ KHz) of data, enabling high-closed-loop control. In combination with monocular vision, IMUs provide valuable observability of metric scale as well as the gravity vector [43]. IMUs directly measure angular velocities and linear accelerations, and estimates of orientation and position are produced by *integrating* these signals. Besides the fact that this means the initial pose is unobservable, the inevitable presence of noise in the sensor signal is compounded in integration. Without correction, the resulting estimates will "drift" over time. As a further complexity, consumer-grade IMUs can demonstrate non-negligible biases which vary slowly over time. Without additional sensing, these biases are unobservable, and can significantly degrade output quality. In many cases, in particular for small robots with inexpensive inertial sensors, the results of naive integration (i.e. "dead-reckoning") can drift so much as to become operationally useless in as little as a few seconds.

The drift issue can be mitigated or eliminated by enforcing consistency with additional, *extrinsic* observations of the robot's pose relative to landmarks or some global fixed frame. GPS satellites or other known references can be well-suited for this task, but are often unavailable or unreliable as mentioned previously.

For these reasons, it is desirable for navigation solutions to leverage ambient information in *unknown*, unstructured environments. In the context of navigation, unknown environments are characterized by the lack of known reference landmarks. Thus, navigation solutions in unknown environments often make use of repeated sightings of recognizable *opportunistic* landmarks which exist naturally in the environment. Some examples of opportunistic landmarks detected in visual data in both outdoor and indoor environments are shown in Figure 1-1.

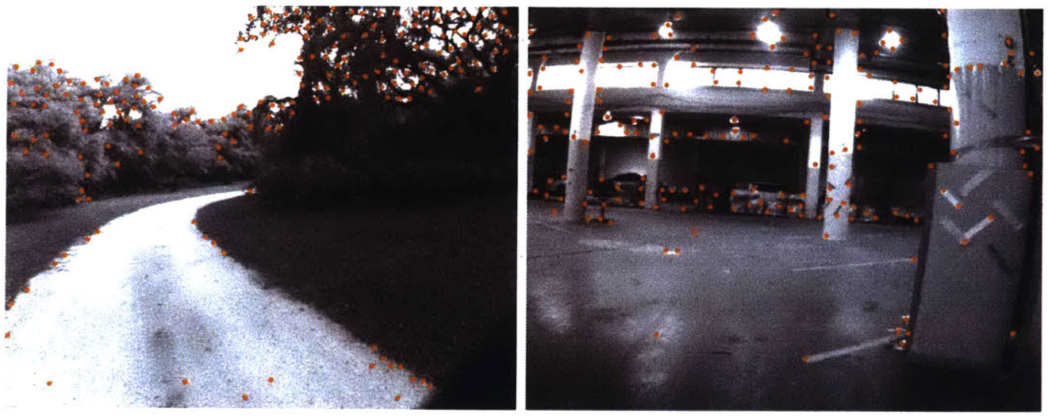


Figure 1-1: Visual landmark detections (orange) produced by the Shi-Tomasi detector [56], which selects pixel regions of high intensity gradient. Individual landmarks can be tracked from frame to frame in a video stream, producing a *feature track*. By leveraging a projective camera model (Section 1.2.1), each track provides a set of noisy geometric constraints which constrain camera motion over time.

Because these landmarks do not have known position *a priori*, and are observed via noisy sensors, their positions must be simultaneously estimated alongside the robot's pose. This is often formulated as a Simultaneous Localization and Mapping (SLAM) problem, as both the robot's state and the set of latent landmarks it observes (the map) are estimated simultaneously. For this reason, even though *mapping*, or building a representation of the environment, is not a direct goal of the navigation task, in SLAM approaches it is an integral, simultaneous process.

While SLAM can be formulated generally, independent of the particular sensor modalities in use, the choice of sensors has significant impact on the specific challenges of the problem. In GPS-denied environments, the most common extrinsic sensors are body-mounted laser scanners (LIDAR) or cameras. LIDARs are active sensors, giving high accuracy and wide field-of-view (FOV) at least in two dimensions, but at the cost of increased SWaP. Additionally, their scan patterns are often limited to only 2D planar slices, limiting their reliability for robots not confined to the plane. Cameras, by contrast, are passive sensors amenable to much lower SWaP budgets, and indeed are already present on many consumer cell phones, quadrotors, and automobiles. They naturally operate in full 3D, and their measurement range can extend to the full line-of-sight. However, cameras do induce some challenges of their own, primarily the non-linearity of their measurements (see Section 1.2) and their limited field-of-view. Nonetheless, the use of such vision information has been the subject of much of the last decade-and-a-half of research in this area [7,37,39,43,46,60], and indeed this will be the focus of much of this thesis.

### A realtime vision-aided navigation solution

A central contribution of this thesis is a presentation of the SAMWISE vision-aided navigation system (originally published in [59]). Designed for computation-constrained, fast-moving autonomous systems, SAMWISE provides high-rate state estimates to facilitate both closed-loop control and long-term planning. Though SAMWISE places a large emphasis on the combination of inertial and vision-based sensing, it can also accommodate a suite of other sensors, such as laser altimeters, barometers, and GPS.

In Chapter 4, the architecture of SAMWISE is discussed in detail. SAMWISE makes several key innovations over standard state estimation libraries that facilitate robust, low-latency navigation and planning for agile vehicles. Developed in part for the DARPA Fast Lightweight Autonomy (FLA) program [49], SAMWISE is designed specifically to enable high-speed, agile autonomous flight on the Draper-MIT quadrotor platform shown in Figure 1-2. Experimental results taken from a recent FLA program milestone are provided, demonstrating robustness in a variety of conditions. Additionally, ground-truth comparison is performed on the open-source EuRoC MAV dataset [5].

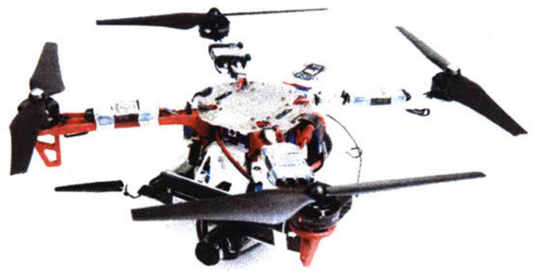


Figure 1-2: The latest iteration of the Draper-MIT autonomous quadrotor platform developed for the DARPA FLA program.

### Measurement selection for computation reduction

Computation management is a vital, open challenge in SLAM today. Realtime, computationally-constrained systems require high-rate data fusion to produce accurate, robust state-estimation and navigation solutions. As is discussed at great length in this thesis, maintaining sparsity in the underlying optimization is crucial for efficient performance. As the measurements incorporated into the estimate ultimately determine both sparsity and estimation performance, intelligent measurement selection has the potential to maximize accuracy while minimizing computation.

Indeed, simple measurement selection heuristics such as decimation already abound in practice. For many low-SWaP robots, the sensors they carry can easily generate much more data than can be processed by the computational resources available. This is especially true in camera-equipped systems, as each frame represents significant pre-processing, even before being incorporated into a SLAM solver. In order to achieve realtime performance, many of these measurements have to be discarded or approximated. Intuitively, this incurs some loss of accuracy when compared to the solution which could be derived from the full set of data.

At the heart of realtime SLAM is a fundamental tradeoff between accuracy and computation. It is well-known (and intuitive) that incorporation of more information will generally lead to better estimation performance. However, this often comes at the cost of increased computation, potentially both in pre-processing and in SLAM optimization. The exact relationship between the number and nature of measurements and the ultimate computation is difficult to quantify and depends on the SLAM method used. However, for iterative smoothing methods (defined in Section 1.3.2) common in recent literature, incorporated measurements have a direct impact on the *sparsity*, and therefore computational complexity, of the underlying system. Though vitally important, the precise relationship between sparsity and the resulting computational complexity is left somewhat vague in the SLAM literature. For reasons discussed in Chapter 2, this relationship is difficult to quantify fully. In this thesis, the *elimination complexity* is introduced as a direct (although incomplete) measure of the computational complexity of a particular problem.

Compared to other modalities, the video stream provided by a camera is data-rich, and can easily provide hundreds of potential measurements per frame, at a high data rate. The obvious heuristic, and what is often done in practice, is *decimation*, where only every $n$-th measurement or image frame is accepted. More sophisticated approaches have been proposed to this end, but they generally require extensive computation that often scales poorly with the size of the problem, making them prohibitive for realtime use. Additionally, as will be seen, these approaches often implicitly assume that reducing the number of measurements corresponds directly to reduced computation. As will be seen, this assumption is correct in some ways, but too simplistic in others, often leading to underwhelming computation savings. In contrast, decimation-style policies are shown to produce an inherently-sparse super-structure which fundamentally bounds elimination complexity. Numerical experiments verify that decimated graphs demonstrate significant computation savings, even when discarding relatively few edges. Furthermore, decimation is shown to have near-optimal connectivity characteristics in simple graphs. The surprising effectiveness of decimation is the subject of Chapter 3.

The remainder of this chapter will formulate the visual-inertial navigation problem as a graphical SLAM problem, and introduce measurement selection as an optimization over this graph. Chapter 2 provides a detailed discussion of computation in graph optimization, and introduces elimination cost as a representation of the computational complexity of a particular graph. Chapter 3 rigorously analyzes the commonly-implemented decimation heuristic in landmark SLAM, and demonstrates that it is in fact quite effective, providing comparable or better performance than more sophisticated approaches. Finally, Chapter 4 introduces the SAMWISE vision-aided navigation system implemented as part of the DARPA FLA program, and provides some recent performance results in challenging, real-world trials.

## 1.1 Related Work

### 1.1.1 Visual-inertial navigation

In the past decade, significant progress has been made in the direction of robust, efficient visual-inertial navigation. Several filtering approaches based on the EKF have been proposed, specifically tailored to the challenges presented by visual measurements. The Multi-State Constrained Kalman Filter (MSCKF) [46] avoids representing landmarks within the state vector by instead estimating the recent history of states simultaneously. All observations of a particular landmark are incorporated at once, as a single measurement, with a linearization point based on the latest estimate (which is assumed more accurate than any earlier-available estimates). This dramatically improves accuracy and reduces both computation and memory requirements. [39] show that the use of multiple linearization points for pose variables over time cause an observability mismatch in VIN systems, and ultimately leads to inconsistency. By simply locking the linearization points of variables in the MSCKF, they demonstrate improved consistency and accuracy. In a similar vein, [26] pose an explicit observability-constrained formulation of the standard EKF. However, as will be discussed in Section 1.3.1, filtering methods suffer from the limitation that they cannot re-linearize past measurements as the estimate is updated.

Unlike filtering methods, *bundle adjustment* or *smoothing* methods explicitly estimate multiple robot or camera poses simultaneously as a nonlinear least squares minimization. One of the earliest bundle adjustment methods demonstrated for use in computational constrained environments is Parallel Tracking and Mapping (PTAM) [37]. PTAM successfully tracks camera motion and a sparse set of high-gradient features in small workspaces, using parallel (but separate) mapping and tracking (localization) optimizations. The sliding-window filter of Sibley et al. [57] employs a smoothing framework over the recent history of poses to estimate planetary surface geometry for a landing craft. A similar technique is used by Chiu et al. [8] which combines a short-term sliding window graph leveraging the incremental iSAM2 solver [32] with a long-term map used for global loop closures.

Rather than relying on the outputs of a costly feature extraction step, *direct* approaches leverage a photo-metric error model to pose an optimization on the raw intensity values of the image. These allow for the construction of denser maps, as depth estimates can be formed for each pixel in the image, rather than for only a sparse set of keypoints. Semi-Direct Visual Odometry (SVO) by Forster et al. [21] use frame-to-frame alignment for fast tracking, and initializes landmarks in $\mathbb{R}^3$ only for pixels with well-estimated depths. LSD-SLAM [18] optimizes a pose graph with associated depth maps to build large-scale, dense maps and exploit global loop closures. Multi-Level Mapping [22] by Greene et al. improves upon LSD-SLAM by gracefully selecting high-texture image regions to focus computation on, improving the density of the output maps. Direct Sparse Odometry [17] incorporates a sophisticated camera model and access to the current exposure setting to account for abrupt lighting changes and high-order camera affects. While the dense maps constructed by these methods alongside the localization solution can be very useful for obstacle detection and avoidance, the navigation outputs are not suitable for high-rate closed-loop control. Additionally, these methods require significant multi-core CPU resources or even GPU acceleration, making them prohibitive for many computationally-constrained systems.

SAMWISE, the subject of Chapter 4, was originally published in [59]. It is designed for high-rate state estimation for agile vehicles requiring closed-loop stabilization and high-performance trajectory tracking. Designed for small vehicles which cannot afford sufficient baseline for a stereo camera setup, it works with a single camera and IMU.

### 1.1.2 Measurement selection for computation reduction

Measurement selection has a long history in robotics and large-scale estimation. In SLAM, measurement selection generally falls into two broad categories: keyframing methods, and per-measurement methods.

Keyframing methods [29, 58, 61] select which robot states (usually poses) are most valuable to represent within the SLAM graph. In vision-based systems, restricting feature detection and tracking to a sparse set of keyframes can save significant preprocessing. Stalbaum et al. [58] provide a suite of heuristics capturing the number of previous and new landmarks detected in a given frame, as well as number of recent keyframes, which can be used to identify new keyframes. Wang et al. [61] derive a similar heuristic based on the Kullback-Leibler divergence to select poses based on the "impact" of the candidate frame's set of landmark observations. As shown by Ila et al. [29], each pose node in the graph represents a linearizing approximation that ultimately contributes to inconsistency. By reducing the number of discrete pose states, this inconsistency can be reduced. To this end, they limit "redundant" pose vertices in the graph by enforcing a probabilistic threshold over distance traveled.

Similar to [29], Forster et al. [20] recognize that the inclusion of every intermediate pose node (i.e. at IMU rate) leads to inconsistency and increased computation. By developing a nonlinear method of *pre-integrating*, or bundling, consecutive IMU measurements directly on the SE(3) manifold, they define computationally efficient IMU factors integrating many sequential IMU readings. This allows the robot's trajectory to be represented accurately with a much sparser set of discrete nodes.

In contrast to keyframing methods, per-measurement methods select measurements individually to accept or remove. Carlone and Karaman [7] attempt to replicate the concepts of anticipation and attention for visual systems by processing only image sub-regions and subsets of all available landmarks which are most likely to remain in frame given the planned trajectory. Ila et al. [29] reject potential loop closures which do not reduce the uncertainty over the latest state by a given threshold. By clever bookkeeping, the evaluation of uncertainty reduction is kept inexpensive for relatively simple graphs containing only odometry and loop closure constraints. For more general graphs, in particular for landmark-SLAM, this calculation is much more expensive.

From a graphical perspective, many methods have interpreted measurement selection directly as an optimization over graph structure. Using either information-theoretic [6,28,44] or graph-theoretic optimization [35,36], many sophisticated methods aim to prune edges while minimizing Kullback-Leibler divergence (KLD) or maximizing graph connectivity. Inspired by the relationship between maximizing $t$-connectivity and minimizing uncertainty volume [35], Khosoussi et al. [36] propose greedy and semi-definite-relaxation algorithms which select the $k$ best edges to prune (or keep). In [6,28,44], a given set of original measurements are removed and replaced with a new set of linear [6] or nonlinear [44] "virtual" measurements in a Chow-Liu Tree [6,9] or $\mathcal{L}_1$-sparsified [28] configuration. However, these methods are computationally expensive, with some requiring iterative optimization. Furthermore, the information-theoretic optimization assumes a linearization point, which may be arbitrarily bad.

While perhaps applicable for long-term SLAM problems within a bounded geometric area (and therefore lots of loop closures), such computationally-intensive methods are less suitable for realtime use on high-rate, computationally-constrained systems. Additionally, as will be discussed in Chapter 2, the computational complexity of a graph is not solely dependent on edge count. Thus the actual computational savings of approaches such as [36] can be underwhelming, even after aggressive pruning.

An often-implemented heuristic is decimation. Decimation can be applied on a keyframing level, in which only every $r$-th image frame or LIDAR scan is processed and represented with a corresponding pose state in the graph. At the per-measurement level, only every $r$-th observation from a given landmark might be added to the graph. Decimation is applied broadly in practice [17,18,21,22,59], particularly in vision-based systems as a method of downsampling the raw video stream. As will be discussed later in this thesis, decimation has several nice properties, producing efficiently-optimizable graphs and promoting structures with near-optimal connectivity properties.

## 1.2 Visual-inertial navigation

Visual-inertial navigation (VIN) is an increasingly popular approach to the navigation problem for mobile robots. The modalities of inertial and visual sensing are complementary, and can be readily accommodated on inexpensive, low-SWaP (Size, Weight, and Power) systems. Inertial measurements provide high-rate ($>$ 100Hz), full-rank constraints between sequential poses, and grant observability of metric scale [43]. Each vision measurement can be considered a low-rank observation of a particular landmark from a particular camera pose. Indirectly, the set of observations for a given landmark can be considered a constraint between all of the corresponding poses. Thus, landmarks which are observed over many frames can significantly reduce the estimation drift which would result from pure inertial integration.

Cameras provide data at a reliable (though generally lower) rate and leverage "ambient" information that already exists in the environment rather than relying on external infrastructure or known beacons. As passive sensors, cameras consume far less energy than active sensors like LIDARS, and can make observations up to the full line-of-sight. Cameras can be significantly less expensive than LIDARs, and the recent industry demand for cameras in consumer devices such as smart phones has only increased the availability of high-quality, low-SWaP devices.

However, visual measurements present several unique challenges. As will be seen in Section 1.2.1, the projective transformation of the implicit observation model is nonlinear. The Jacobian representing the first-order approximation is highly sensitive to the linearization point, and the landmark depth in particular. The degree of non-linearity in vision measurements can make traditional SLAM approaches that assume linear or simple one-time linearization fail to achieve acceptable accuracy, and even diverge entirely. This sensitivity is compounded by the fact that any single monocular observation grants no observability into depth (rather it noisily measures the *ray* from the focal center to the landmark). Additionally, the camera model is dependent on a variety of calibration parameters and distortion effects, which vary due to manufacturing irregularities. While calibration can be done off-line and is generally considered stable, it can be prohibitively time-consuming for large fleets of camera-equipped robots, and imperfect calibration can cause performance degradation if not accounted for. Third, rapid rotations can make feature tracking difficult and only allow for landmarks to remain in view for a few frames, limiting their utility in constraining robot motion over longer timescales. Fourth, as cameras are passive sensors, they rely on the ambient lighting of the environment. Harsh lighting changes can induce additional challenges, for example when entering a building from outdoors.

### 1.2.1 Monocular measurement model

In most vision-based approaches, landmarks are characterized by points $\boldsymbol{l}^{(w)} \in \mathbb{R}^3$ in the fixed world frame. Let the camera pose at time $i$ be represented by $\mathbf{x}_i^{(w)} \triangleq (\mathbf{R}_i^{(w)}, \mathbf{t}_i^{(w)}) \in \mathrm{SE}(3)$, defined by rotation matrix $\mathbf{R}_i^{(w)}$ and translation $\mathbf{t}_i^{(w)}$. When viewed from the camera at pose $\mathbf{x}_i$, the landmark $\boldsymbol{l}^{(w)}$ is first transformed to the camera frame and then undergoes a projective transformation [24]

$$\boldsymbol{l}^{(\mathbf{x}_i)} \triangleq \mathbf{R}_i^{(w)}(\boldsymbol{l}^{(w)} - \mathbf{t}_i^{(w)}) \tag{1.1}$$

$$\mathbf{h}(\boldsymbol{l}^{(\mathbf{x}_i)}) \triangleq \begin{bmatrix} \frac{f_x x}{z} + c_x & \frac{f_y y}{z} + c_y \end{bmatrix}^T \tag{1.2}$$

with $\boldsymbol{l}^{(\mathbf{x}_i)} \triangleq \begin{bmatrix} x & y & z \end{bmatrix}^T \in \mathbb{R}^3$, camera focal lengths $f_x, f_y \in \mathbb{R}$, and principal point $(c_x, c_y) \in \mathbb{R}^2$.

It is often assumed that the dominant source of noise is an additive uncertainty $\boldsymbol{v} \sim N(\mathbf{0}, \sigma^2 I_2) \in \mathbb{R}^2$ which can intuitively be ascribed to uncertainty in the feature detection step [39,45,46]. Note that the uncertainty parameter $\sigma$ is specified in units of pixels. Combining (1.1) and (1.2), the measurement $\mathbf{z} \in \mathbb{R}^2$ is assumed to be generated according to

$$\mathbf{z} = \mathbf{h}(\boldsymbol{l}^{(\mathbf{x}_i)}) + \boldsymbol{v} \tag{1.3}$$

The vision model (1.3) has the Jacobian

$$\mathbf{H}(\boldsymbol{l}^{(\mathbf{x}_i)}) = \begin{bmatrix} \frac{f_x}{z} & 0 & -\frac{f_x x}{z^2} \\ 0 & \frac{f_y}{z} & -\frac{f_y y}{z^2} \end{bmatrix} \tag{1.4}$$

The Jacobian $\mathbf{H}$ represents the first-order linearization and is inversely sensitive to the landmark depth $z$. Indeed, Montiel et al. [45] argue for using an inverse-depth parameterization to initialize landmark estimates. For systems which attempt to initialize landmarks immediately, from one or two observations, this can make a big difference. However, the true posterior over the landmark does approach a Gaussian in $\mathbb{R}^3$ with increasing observations and camera translation. Therefore, for more flexible systems which can wait until sufficient observations have been acquired, such as the SAMWISE system in Chapter 4, a standard $\mathbb{R}^3$ position parameterization is used.

## 1.3 The smoothing formulation

### 1.3.1 Limitations of filtering in nonlinear systems

Traditional filtering approaches derive fundamentally from the Kalman Filter [34], and are characterized by an explicit representation of uncertainty in either the $n \times n$ covariance matrix $\mathbf{\Sigma}$ or information matrix $\mathbf{\Lambda} \triangleq \mathbf{\Sigma}^{-1}$. This representation of Gaussian uncertainty acts as a compact representation of the complete (linearized) measurement history. In turn, this allows for constant-time algorithmic complexity, which is desirable for realtime applications. For linear, Gaussian systems, this Gaussian representation is lossless, and the Kalman Filter is an optimal unbiased estimator [34].

However, nonlinearity arises in many real-world systems, and many nonlinear extensions of the Kalman Filter have been proposed. Unfortunately, optimality and even consistency cannot be guaranteed for these methods in general. Proper handling of the nonlinearity arising from system dynamics or sensor observation models is key to achieving good accuracy and robustness. While there are several approaches to handling nonlinearity, such as simple first-order linearization (as in the Extended Kalman Filter, or EKF) or the use of the unscented transform ( [30]), the inevitable linearization error is irrecoverably "baked-in" to the uncertainty representation at every step. Over time, this error builds up, leading to inconsistency and possibly even complete divergence.

As observed in the previous section, visual observations are nonlinear, and in practice very sensitive to the chosen linearization point. A modified filtering approach, the Multi-State Constrained Kalman Filter (MSCKF) proposed by [46] mitigates this problem by waiting until the full observation history of a landmark has been collected before incorporating them all at once into the estimate. This allows a more accurate linearization point to be chosen, leading to improved consistency and accuracy.

Even in the case of the MSCKF, variables and measurements are only linearized once and cannot be re-linearized once incorporated and as the current estimate is corrected.

The Iterated EKF [3] allows for repeated iteration of measurement updates, where the current measurement can be relinearized until convergence at each step. However, this iterative relinearization is limited to the current measurements.

An additional drawback of naive filtering approaches is the storage space and complexity associated with large covariance matrices. Although individual measurements may only correlate a small number of variables, the resulting covariance matrix is generally dense. For dense matrices, storage requirements grow with $\mathcal{O}(n^2)$, and computation with $\mathcal{O}(n^3)$. For systems hoping to estimate many landmarks simultaneously, this can quickly become prohibitive.

For all these reasons, the filtering formulation is not well-suited to highly nonlinear, large (in terms of number of estimated variables) estimation problems like VIN.

### 1.3.2 Smoothing as nonlinear least squares

In contrast to filtering, *smoothing* approaches remove the moratorium on explicitly maintaining previous measurements. Instead of condensing the measurement history into a linear-Gaussian uncertainty matrix, the smoothing formulation explicitly represents inference as a nonlinear least squares (NLLS) optimization over the measurements $Z$. This naturally lends itself to (and generally requires) the estimation of *multiple* instantaneous states representing the robot's trajectory over time, rather than simply the current state. Thus, smoothing can use current information to update the estimate of past states.

$$\underset{X}{\operatorname{argmin}} \frac{1}{2} \sum_{i=0}^{m} (\mathbf{h}_i(X_i) - \mathbf{z}_i)^T \mathbf{\Sigma}_i^{-1} (\mathbf{h}_i(X_i) - \mathbf{z}_i) \tag{1.5}$$

Because the full (nonlinear) measurements are maintained, each corresponding to a term in the summation in (1.5), they can be relinearized at each iteration of the solver, as the estimate is updated. While this optimization does not require a probabilistic interpretation, under linearization and an assumption of additive Gaussian noise it corresponds to Maximum-Likelihood (ML) inference over a high-dimensional, multivariate Gaussian distribution. The full covariance matrix for a particular estimate is thus well-defined, and can be recovered for a given linearization point. Nonetheless, it is not required for ML inference.

This problem is well-studied, with both linear algebra and graphical interpretations [13, 52]. Because the uncertainty is never condensed into a dense covariance matrix, optimization can naturally take advantage of underlying sparsity in the information matrix to solve much more efficiently. This sparsity is key in making the smoothing formulation feasible for realtime systems.

### 1.3.3 Smoothing and SLAM

Simultaneous Localization and Mapping (SLAM) is a ubiquitous problem throughout robotics that has been studied for decades. As discussed previously, visual navigation on a mobile robot using landmarks which are *unknown a priori* is itself a SLAM problem (as inferring the set of landmark positions is at least a byproduct, if not a stated goal). SLAM (and therefore VIN) is naturally represented as an NLLS smoothing problem.

### 1.3.4 Representation as graphical inference

It is often convenient to represent SLAM as a graphical inference problem. Graph theory has a rich history, as does Maximum-Likelihood (ML) estimation theory. There are several different variants of graphical models for inference, particularly *undirected* models and *factor graphs.*

Undirected models represent estimation variables as nodes in a graph, connected by edges representing probabilistic dependencies implied by measurements. In undirected models, cliques represent fully-correlated sets of variables. The *Markov property* of undirected models implies that any variable $x$ is independent of all other variables given its neighbors $N(x)$.

A factor graph, on the other hand, is made up of variable nodes representing the values being estimated $X$, and factor nodes $\mathcal{F}$, which are generated from the independent, possibly non-linear measurements $Z$ constraining them. Factor nodes represent measurements or priors, and can relate one, two, or more variable nodes (unlike a single edge in an undirected model).

A simple example of an undirected model and equivalent factor graph are shown in Figure 1-3.

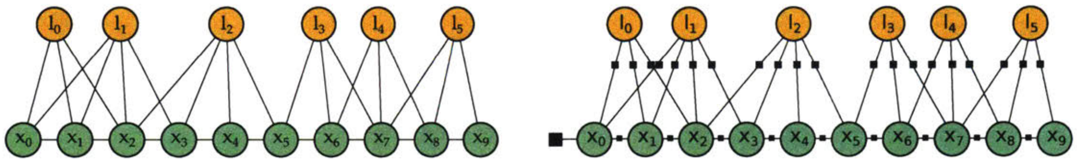


Figure 1-3: Undirected model (left) and factor graph (right) representations of the same SLAM problem. The robot states at discrete time steps are represented as vertices $\mathbf{x}_i$. Landmark positions are represented as vertices $\mathbf{l}_i$. In the undirected model, variables related by a common measurement are adjacent. On the other hand, factor graphs represent measurements explicitly via factor nodes (black squares). Note that unary factors, such as the prior on variable $\mathbf{x}_0$, are represented explicitly in this model.

Factors can be considered $n$-ary edges rather than a class of vertices, as in either case they connect the $n$ variables which they depend on. The factor nodes represent potentials over subsets of the variable set, and their product is proportional to the value of the joint distribution. Indeed, the factors $\mathcal{F} \triangleq \{\Phi_1, \Phi_2, \ldots, \Phi_m\}$ represent the factorization of the joint distribution

$$\mathrm{P}(X|Z) = \prod_{i=1}^{N} \Phi_i(X_i) \tag{1.6}$$

Factor graphs are more expressive, but undirected models can be more compact. For measurements involving exactly two variables (such as odometry relating consecutive poses, or observations of a particular landmark from a particular pose), there is a one-to-one correspondence between the factors in the factor graph and edges in the undirected model. Thus, both representations are used throughout this thesis.

The chief advantage of graphical models, and factor graphs in particular, is the way they naturally represent the Markovian independence structure between variables. An important assumption is that each measurement (factor) $\Phi_i$ is purely dependent its neighbors $X_i \in X$ and *independent* noise. Thus, non-adjacent variables (here "non-adjacent" refers to variables that do not share a common factor) are conditionally independent given some separating set. This fact follows directly from the factorization shown in (1.6).

## 1.4 Solving the smoothing problem

Give the joint factorization (1.6), Maximum Likelihood (ML) estimation takes a min-sum form

$$\underset{X}{\operatorname{argmax}}\ \mathrm{P}(X|Z) = \underset{X}{\operatorname{argmax}}\ \log \mathrm{P}(X|Z) = \underset{X}{\operatorname{argmin}}\ -\sum_{i=0}^{m} \log \Phi_i(X_i) \qquad (1.7)$$

For arbitrary potential functions $\Phi_i$, representing and optimizing the full high-dimensional joint density is intractable. However, in SLAM, the potentials are generally assumed to be multivariate Gaussian about some nonlinear observation model: $\mathcal{N}(\mathbf{h}_i(X_i), \mathbf{\Sigma}_i)$, where the symmetric matrix $\mathbf{\Sigma}_i$ represents the noise covariance. In this case, the optimization in (1.7) reduces to a NLLS optimization of the form (1.5).

Non-iterative solvers for (1.7) and certificates for global optimality have been proposed in special cases. For example, in pose-graph SLAM where all variables are elements of SE(3) and all edges are full-rank relative transforms between them, Rosen et al. [53] developed a method doing just that. They prove that for realistic noise regimes, their method can recover the *global* optimum, and often do so faster than iterative methods. Unfortunately, this method does not immediately extend to general measurement models, and in particular the low-rank observations acquired via monocular vision.

For general landmark-SLAM, iterative methods such as Gauss-Newton are used [8,13, 57]. Under Gauss-Newton, the joint distribution is iteratively re-linearized to produce a series of approximating *linear* least-squares problems of the form

$$\mathbf{A}^T\mathbf{A}\mathbf{x} = -\mathbf{A}^T\mathbf{b} \tag{1.8}$$

In general, multiple Gauss-Newton iterations are required before converging to a (possibly only local) minimum. Each iteration corresponds to solving a linear system of the form (1.8). For dense system matrices, each of these linear solves has $\mathcal{O}(d^3)$ complexity, which can be prohibitive.

### 1.4.1 The smoothing problem is sparse

Fortunately, Dellaert and Kaess [13] pointed out that the SLAM problem has a sparse block structure which can be exploited to solve (1.8) much more efficiently. Sparsity refers to the prevalence of zeros in the system Jacobian $\mathbf{A}$, and the corresponding lack of edges in the undirected graph relative to a complete graph. In the factor graph representation, it means that measurement factors each only relate a very few variables (nodes). From a statistical perspective, sparsity refers to the fact that the joint distribution $\mathrm{p}(X)$ can be factorized into many simpler potentials $\Phi_i(X_i)$, as in (1.6).

As will be discussed in detail in Chapter 2, this sparsity arises naturally in SLAM because of physical and logical sensor limitations. Furthermore, the computational complexity of solving (1.8) is much reduced when this sparsity is present. Thus, understanding the relationship between sparsity and computation is vital to identifying and promoting computationally-efficient graph structures.

### 1.4.2 Incremental solvers

For mobile robots, realtime SLAM can be considered to represent only *incremental* updates to this optimization (in the form of new factors $\mathcal{F}_{\text{new}}$ and variables $\mathcal{X}_{\text{new}}$). This realization is leveraged in iSAM [33] to perform incremental solves of the underlying linear system, greatly reducing redundant computation when the linearization point does not change much. The second-generation solver, iSAM2 [32], represents the inference problem as a Bayes Tree [31]. The Bayes Tree represents the dependency-structure of the inference problem, and allows for re-linearization, re-ordering, and back-substitution to be performed only in regions of the graph which are significantly affected by new information. This allows iSAM2 to avoid re-solving the whole system or operating over the entire graph at each step, facilitating efficient real-time optimization.

iSAM2 currently represents the state-of-the-art in incremental algorithms for general Gauss-Newton graph optimization. The SAMWISE estimation library (the subject of Chapter 4) leverages the GTSAM [12] implementation of iSAM2 for back-end optimization of a high-rate VIN front-end.

Note that iSAM and iSAM2 are not truly incremental. If the linearization point changes significantly, the entire graph may have to be re-linearized and re-eliminated in what essentially reduces to a batch solve. Thus, the graph size must be bounded in order to bound worst-case and even mean computation.

Furthermore, both iSAM and iSAM2 are still heavily reliant on sparsity. The structure of the Bayes Tree is determined by the structure of the corresponding factor graph, and the extent to which re-linearization can be performed "locally" depends heavily on this structure. More significantly, the elimination process used by iSAM2 is fundamentally equivalent to that of batch solutions to (1.8) (albeit with intelligent recycling of computation), and thus shares the same dependence on sparsity.

## 1.5 Measurement selection as a means of computation reduction

Sparsity naturally arises in SLAM problems, due to both assumed observation models and to real sensor limitations. For example, data from wheel odometry is generally formulated as a sensor measurement which can only relate *consecutive* poses. Similarly, ranging signals may only relate pose vertices to landmark vertices. Physical sensor limitations play a role as well, as at any given moment, the robot may only be able to observe landmarks which are nearby, or within a limited field of view. Ultimately, these mechanisms lead to sparsity in the realized graph SLAM graph.

For many computationally-constrained systems, the naturally-arising sparsity may still not be sufficient. Modern IMUs are capable of very high data rates (greater than 1000 Hz), and naive application can lead to gross over-sampling of the trajectory. Thus, IMU bundling, or *pre-integration*, approaches such as [20] have been used to reduce "redundant" or intermediate optimization variables. Cameras too are capable of high frame rates, and at each frame can observe tens or hundreds of landmarks. Inclusion of every available observation can result in a relatively dense graph, with many high-degree poses and landmarks. For long-term SLAM problems, characterized by loopy trajectories in a restricted geographic area, the inclusion of loop-closure constraints can also undermine sparsity. Though often very informative, loop closures can potentially be made between any pair of pose vertices. Without any restriction, loop closures can ultimately destroy graph sparsity. Thus, measurement selection has long been motivated [6,28,29,36,61] as a means of promoting sparsity and managing computation.

In high-rate realtime systems, measurement selection strategies must also be computationally efficient to execute. Many of the sophisticated methods presented in the literature (see Section 1.1.2), while effective in offline systems, require iterative optimization or computation that scales with the size of the graph. This makes them impractical for use in computationally-restricted systems that must run at a high rate.

Much existing work [28, 36] focuses simply on reducing the number of edges (measurements) in the graph. In contrast, this thesis aims to more holistically exploit the relationship between structure and computation. In Chapter 2, this relationship is explored in detail, and a metric of graph complexity is introduced.

As an example of a realtime-implementable strategy, the often-implemented decimation heuristic in analyzed in Chapter 3, and is shown to promote a particularly sparse graph super-structure. In combination with good connectivity properties, this makes decimation a very effective technique that often outperforms more sophisticated methods.

### 1.5.1 Graph connectivity metrics

As measurements in SLAM correspond to graph edges, measurement selection is (at least in part) an optimization over *graphs*. Therefore, a thorough understanding of how graph structure impacts inference should prove valuable.

It has long been understood that the connectivity of a graph is strongly related to estimate robustness. For linear SLAM graphs, analytic results exist [35] connecting particular graph connectivity metrics to specific characteristics of the ML solution. While it is difficult to rigorously extend these results to nonlinear problems such as VIN, empirical evaluation suggests that these connections still approximately hold.

#### Average node degree

*Average degree* is one of the simplest measures of graph connectivity. For a fixed number of vertices (and under the restriction that all edges (measurements) correspond to exactly two variables), average degree corresponds one-to-one with the number of edges. As was observed empirically by Olson and Kaess [47] and proven by Khosoussi et al. [35], average node degree relates approximately to the expected "over-fitting" of the corresponding ML estimate. More precisely, if the value of the negative log-likelihood function $f(X)$ evaluated at the *true* parameter setting $X_0$ is $f_0 \triangleq f(X_0)$, and the value at the ML setting $X^\star$ is $f^\star$, several observations can be made:

1. $f^\star \leq f_0$ by the definition of the minimum.
2. $\mathbb{E}\left[\frac{f^\star}{f_0}\right] \approx 1 - \frac{2}{\lambda}$, where $\lambda$ is the average node degree.
3. Thus, as $\lambda$ *increases*, $\mathbb{E}\left[\frac{f^\star}{f_0}\right] \to 1$.

This makes sense intuitively, as an increasing $\lambda$ corresponds to the use of increasing number of Gaussian measurements. The more measurements are incorporated, the less likely it becomes that the ML configuration will be "far" from the true setting of parameters which generated the measurements.

### (Weighted) number of spanning trees

The number of spanning trees in the graph (also referred to as $t$-connectivity or $t(G)$) has long been studied as a robustness metric in network theory. A *spanning tree* $T(\mathcal{V}, \mathcal{E}_T) \in \mathcal{T}(G)$ of graph $G(\mathcal{V}, \mathcal{E})$ corresponds to a selection of edges $\mathcal{E}_T \subset \mathcal{E}$ such that $T$ is connected and a tree. As $t(T) = 1$ for any spanning tree, and is non-decreasing with the incorporation of additional edges, $t(G)$ provides a natural measure of connectivity [35]. For a complete graph $G_c$, $t(G_c) = n^{n-2}$ [1].

This definition can be generalized with the incorporation of edge weights $w_{ij}$. The weight $\mathbb{V}(T)$ associated with spanning tree then corresponds to the product of the edges which define it, and the *weighted* number of spanning trees $t_w(G)$ is defined

$$\mathbb{V}(T) \triangleq \prod_{(i,j)\in\mathcal{E}_T} w_{ij} \tag{1.9}$$

$$t_w(G) \triangleq \sum_{T\in\mathcal{T}(G)} \mathbb{V}(T) \tag{1.10}$$

Note that unlike the similar-sounding concept of weighted spanning trees, the weighted number of spanning trees $t_w$ involves a *product* over edge weights rather than the sum.

This means that if all $w_{ij} = 1$ for all edges $(i,j) \in \mathcal{E}$, then $\mathbb{V}(T) = 1$ for all spanning trees, and $t_w(G) = t(G)$.

More recently, Khosoussi et al. [35] proved that, at least in linear SLAM problems, the uncertainty volume $\det(\boldsymbol{\Sigma}) = \det(\boldsymbol{\Lambda}^{-1})$ corresponds directly to the weighted number of spanning trees, where the edge weights correspond to the measurement precision $w_{ij} = \frac{1}{\sigma_{ij}^2}$. This result links graph structure directly to inference quality, and demonstrates that maximizing $t$-connectivity specifically corresponds to better estimation performance.

While this result was shown rigorously only for a limited class of SLAM problems (linear measurements, spherical noise covariances), numerical results suggest that the connection extends to nonlinear systems as well. As measurement selection fundamentally corresponds to edge selection in the SLAM graph, $t$-connectivity provides a useful metric in analyzing policies. We can expect policies which promote $t$-connectivity will produce better estimation quality than those which do not.

### 1.5.2 Kullback-Leibler divergence

The Kullback-Leibler divergence (KLD, also $\mathrm{D_{KL}}$) can be loosely understood as a measure of distance between two probability distributions. Often, for a given *full* distribution $\mathrm{p}(X)$, we would like to find the best *approximating* distribution $\mathrm{q}(X)$ that has some desired structure. This is often defined as minimizing $\mathrm{D_{KL}}\big(\mathrm{p}(X) \mid\mid \mathrm{q}(X)\big)$, corresponding to finding the "nearest" approximating distribution, where the KLD is defined

$$\mathrm{D_{KL}}\big(\mathrm{p}(X) \mid\mid \mathrm{q}(X)\big) \triangleq \int \mathrm{p}(X) \log \frac{\mathrm{p}(X)}{\mathrm{q}(X)} \,\mathrm{d}X \tag{1.11}$$

It is important to note that unlike Euclidean distances, KLD is in general *not* symmetric

$$\mathrm{D_{KL}}\big(\mathrm{p}(X) \mid\mid \mathrm{q}(X)\big) \neq \mathrm{D_{KL}}\big(\mathrm{q}(X) \mid\mid \mathrm{p}(X)\big) \tag{1.12}$$

Usually, the reference distribution $\mathrm{p}(X)$ is taken to be the first argument, and the approximating distribution q$(X)$ as the second.

In general, evaluating the KLD over arbitrary continuous-valued distributions is too expensive for realtime evaluation on computationally-constrained systems. However, by making the standard assumption of Gaussianity about some linearization point (usually taken to be the current estimate), the KLD adopts a convenient form depending only on the first and second moments of the corresponding distributions

$$\mathrm{D_{KL}}\big(\mathcal{N}(\boldsymbol{\mu}_p, \boldsymbol{\Lambda}_p^{-1}) \mid\mid \mathcal{N}(\boldsymbol{\mu}_q, \boldsymbol{\Lambda}_q^{-1})\big) = \frac{1}{2}\Big[\mathrm{trace}(\boldsymbol{\Lambda}_q\boldsymbol{\Lambda}_p^{-1}) - d + \log\det(\boldsymbol{\Lambda}_q\boldsymbol{\Lambda}_p^{-1}) + ||\boldsymbol{\mu}_p - \boldsymbol{\mu}_q||^2_{\boldsymbol{\Lambda}_q}\Big] \tag{1.13}$$

For large graphs, KLD can be expensive to evaluate in realtime. Additionally, it is not clear how specific measurement selection choices (and therefore different choices of $q$) affect KLD without explicit calculation and comparison. However, KLD is valuable as a comprehensive metric to empirically evaluate particular measurement selection schemes. Compared to simpler metrics such as Root-Mean-Squared-Error (RMSE) which only compare the ML configurations (i.e. the distribution means), KLD also takes into account the distribution uncertainty. This makes it a more complete measure of how well the full distribution is approximated.

## 1.6 Thesis contributions

This central focus of this thesis is in computation management and reduction for realtime SLAM problems via efficient measurement selection.

In Chapter 2, the relationship between graph structure and requisite computation is explored. A complexity measure $\mathcal{C}$ adopted from the sparse linear algebra community [25, 52] is introduced specifically for the SLAM smoothing problem. $\mathcal{C}(G, \mathcal{P})$ approximates the operation count of the elimination phase for a given ordering $\mathcal{P}$, which, following [31] is shown to be equivalent to a sparse, block-wise factorization of the linearized system matrix (1.8). $\mathcal{C}$ is verified experimentally to predict the elimination computation required for simulated SLAM problems, and is also shown to correlate with the update time of incremental solvers such as iSAM2 [32]. This measure and the rational behind it are introduced to facilitate rigorous analysis of sparse graph super-structures used in Chapter 3. More broadly, they are presented as a method of defining the complexity of graph SLAM in a standard way, which is lacking in the SLAM literature.

Second, a rigorous analysis of decimation-style heuristics is explored in Chapter 3. Decimation is a very simple policy for measurement selection, but it produces a distinct sparsity pattern and super-structure in the resulting graph. It is proven by construction that elimination orderings exist for decimated graphs that result in bounded complexity at each step of elimination and therefore bounded total $\mathcal{C}$. Simulated results demonstrate that in practice, decimated graphs often outperform these bounds by significant margins, due to the additional level of sparsity which arises naturally but unpredictably due to sensor limitations. In parallel, it is shown that the "even" spacing of observations produced by decimation is near $t_w$-optimal (see Section 1.5.1) for a class of single-landmark graphs. This suggests that decimated graphs are well-connected, and therefore maintain desirable estimation qualities.

In light of these insights, the improved `dec++` heuristic policy is introduced to address specific shortcomings of decimation in incremental estimation. Empirical results from simulated data are provided to confirm these analytic findings empirically. Furthermore, they demonstrate that despite its simplicity, the `dec++` heuristic performs as well or better than much more sophisticated (and computationally expensive) pruning strategies.

Finally, the SAMWISE visual navigation system is presented in Chapter 4. A thorough description of the system is provided, as well as benchmarking evaluation on the open EuRoC MAV dataset [5]. The `dec++` heuristic is implemented in SAMWISE and shown to reduce computation significantly with acceptable accuracy degradation on this dataset. Furthermore, results from recent stress-tests and a DARPA FLA program milestone are shown, demonstrating robustness in a variety of challenging real-world conditions.

# Chapter 2

# Structure and Computation in SLAM Graphs

As described in Section 1.4, the smoothing problem can be represented as inference over a graph. Though fully-nonlinear solvers have been proposed [53], these make specific assumptions about the types of variables and measurement constraints involved. Real-world problems, especially VIN, often involve heterogeneous combinations of sensors and measurement factors, including IMU factors [20], projective camera measurements (described in Section 1.2), relative pose constraints [6,44], and others. In addition to robot poses and landmark positions, various sensor biases and camera calibrations may also be estimated. For these more general factor graphs, iterative Gauss-Newton [4] approaches remain the standard [13,32,57].

At each iteration of Gauss-Newton, the nonlinear factors are linearized, producing a linear system which must be solved. Though this system may have large dimension (often hundreds or thousands of poses and landmarks), its inherent sparsity allows it to be solved efficiently [13]. As described in Section 1.4.1, sparsity refers to the fact that nodes in the graph are only adjacent to a small number of other nodes. A tree is the most sparse connected graph, and a complete graph is the least.

It should be noted, however, that it is not simply the number of edges in the graph, but also their arrangement (or structure), which determines computation. As will be a main point of this chapter, the relationship between computation and graph structure is non-trivial, yet vitally important when evaluating measurement selection strategies for computation reduction.

Incremental solvers such as iSAM [33] and iSAM2 [32] aim to perform minimal updates to the prior solution as the graph is incrementally augmented with new measurements and variables. These methods have been shown to dramatically reduce computation in many systems, making SLAM more accessible to computationally-restricted platforms. Just like batch solvers, however, incremental methods are highly dependent on the sparsity of the given graph.

In this view of SLAM, measurement selection for computation reduction can be seen as optimization over the edges in the graph. Measurements correspond to factors in a factor graph, and to edges in undirected models (see Section 1.3.4). Thus, selecting measurements in a way that promotes sparse structure, or conversely removing measurements which hinder sparsity, can be a powerful method of computation management. In order to evaluate measurement selection strategies for their impact on computation, the precise relationship between graph structure and computation must be understood.

In this chapter, this relationship is explored and to some degree quantified. The graph *elimination complexity* is introduced as a measure of the intrinsic complexity of a particular graph. Derived from the fundamental complexity of factorizing sparse linear systems, elimination complexity $\mathcal{C}$ represents an approximate operation count of the elimination phase of optimization. Later, in Chapter 3, this metric is used to lend insight into efficient graph structures and demonstrate analytically the inherent sparsity which results from simple decimation-style pruning.

## 2.1 Computation in graph optimization

One of the outstanding challenges in real-world application of the SLAM framework is computation management. Physical computers have finite memory and processing resources, and in low-SWaP systems are especially limited. To compound this challenge, fast-moving, unstable systems often require high-rate and low-latency estimate updates in order to achieve closed-loop stability and adequate control tracking.

Constant-time update computation is a fundamentally desirable aspect of any real-time navigation algorithm. Conventional filtering-based approaches based on the Kalman Filter [34] have been used since the Apollo moon missions [27] for time-critical aerospace applications. Proponents have long championed the constant-time characteristics of such approaches. Indeed, the sliding-window formulation of more general smoothing approaches [8,57] is intended to maintain a semblance of bounded computation. This serves to bound the size of the graph, and therefore the complexity of solving the linearized system is naively bounded by $\mathcal{O}(d^3)$, where $d$ refers to the overall scalar dimension of the system. Nonetheless, it is the iterative re-linearization of the Gauss-Newton solver which ultimately defies bounding. Arbitrarily limiting the number of re-linearization steps can provide a trivial ultimate bound, although at the cost of limiting one of the chief advantages of the smoothing formulation.

Of course, the existence of a bound itself is not sufficient for realtime performance. In VIN, robust and accurate solutions require the simultaneous estimation of many poses and landmarks. The large number of variables active in many real-world problems make such worst-case analysis somewhat unhelpful, as the worst-case computation is generally prohibitive, but fortunately also rare. Additionally, approaches such as the decoupled strapdown propagation discussed in Chapter 4 somewhat relax the requirement for fast worst-case updates, as propagation from the IMU is generally sufficient to stabilize the vehicle through even the longest observed update steps.

It has long been recognized by the linear algebra community [10,42,50] that sparsity in the matrices defining a linear system can be leveraged to significantly outperform the naive $\mathcal{O}(d^3)$ bound. This sparsity is manifested as a large number of zeros in the system matrix $\mathbf{A}^T\mathbf{A}$, and in the corresponding graphical system in the fact that most vertices are adjacent to only a small neighborhood of the total graph. Dellaert et al. [13] argued that this sparsity was ubiquitous in many SLAM problems, and that performance could be further improved by leveraging it more explicitly. Indeed, the state-of-the-art iSAM [33] and iSAM2 [32] algorithms continued that trend by developing near-incremental (i.e. near-constant-time) methods of updating the (non)linear system. As the SLAM system is augmented with new variables and measurements, sparsity facilitates a sense of locality, and changes to a particular region of the graph have diminishing effect in "distant" regions.

### 2.1.1 Macro- and micro-sparsity

In many SLAM systems, to a large extent sparsity is a natural result of the types of physical sensors available. Odometry sensors such as wheel encoders or IMUs give a noisy measurement of the relative transform in robot pose between two consecutive (discrete) time instants, i.e. $x_i$ and $x_{i+1}$. Similarly, a common formulation of visual measurements relates the camera pose $x_i$ at a particular instant to a particular landmark $l_j$. Thus, the types of sensors, and the definitions of the corresponding observation models, directly constrain which vertices in the SLAM graph may be connected by an edge. Thus, given a particular choice of sensor configuration and corresponding formulation of measurement factors, the sparsity pattern of the resulting system matrix can be loosely upper-bounded. This imposed super-structure can be thought of as *macro*-sparsity.

However, the actual sparsity realized during any particular run will in general be hard to predict, as most landmarks are observed opportunistically. Due to physical limitations like sensor field of view, sensor range, and occlusions, most landmarks will only be observed from a relatively small number of poses. Additionally, some sensors or measurements may only be available in certain environments or at unpredictable times, such as a differential GPS receiver that only has signal outdoors, or a laser altimeter which only works on certain surfaces. Though unpredictable *a priori*, this *micro*-sparsity is often quite significant.

Macro-sparsity can be predicted *a priori* for a given system, as it depends on known factors such as sensor suite. It provides a worst-case sparsity bound, which in turn maps to a worst-case optimization complexity. Micro-sparsity, on the other hand, refers to the "extra" level of sparsity realized in practice, often due to sensor limitations. Often, micro-sparsity is crucial for realtime performance, as the worst-case bound defined by macro-sparsity can still be prohibitive.

### 2.1.2 Quantifying computation

Fundamentally, the nonlinear, iterative nature of most smoothing SLAM solvers (see Section 1.3.2) makes fully quantifying computation time difficult. Specifically, it is difficult to predict how many Gauss-Newton iterations will be necessary before convergence, or even if convergence will ever occur. Convergence and convergence rate depend on many factors, including the specific measurement functions involved, the measured data itself, and the initialization point.

In contrast, the per-iteration computation simply corresponds to solving a positive-definite linear system [13] (repeated from Section 1.4)

$$\mathbf{A}^T\mathbf{A}\mathbf{x} = -\mathbf{A}^T\mathbf{b} \tag{2.1}$$

where the $d \times d$ matrix $\mathbf{A}^T\mathbf{A}$ is assumed positive definite.

Linear systems of the form (2.1) are well-studied, and the computational complexity at this level is much more amenable to analysis. As the total computation of the nonlinear optimization is essentially the sum of a series of these linear solves, computational savings at the linear level corresponds to multi-fold savings in total. Additionally, the computation involved at the linear system level is a direct function of sparsity, and therefore of graph structure. Thus, understanding the relationship at this level provides a clear link from structure to total computation.

### 2.1.3 The elimination step

The two fundamental steps involved in solving the linear system of the form in (2.1) are *elimination* and *back-substitution.* Elimination is equivalent to factorization of the system $\mathbf{A}^T\mathbf{A} = \mathbf{R}^T\mathbf{R}$ into the upper-triangular square matrix $\mathbf{R}$ [13, 50]. As will be seen, the complexity of elimination is highly dependent on graph structure, and in the worst (fully dense) case is $\mathcal{O}(d^3)$. Elimination represents the bulk of computation in many SLAM algorithms, including incremental methods like iSAM2 [32].

Back-substitution uses the factor $\mathbf{R}$ to determine the maximum-likelihood assignments for variables $\mathbf{x}$, often via an intermediate vector $\mathbf{y}$.

$$\begin{aligned} \mathbf{A}^T\mathbf{A}\mathbf{x} = \mathbf{R}^T(\mathbf{R}\mathbf{x}) &= \mathbf{A}^T\mathbf{b} \\ \implies \qquad \mathbf{R}^T\mathbf{y} &= \mathbf{A}^T\mathbf{b} \qquad (2.2) \\ \mathbf{R}\mathbf{x} &= \mathbf{y} \qquad (2.3) \end{aligned}$$

Because $\mathbf{R}$ is triangular, back-substitution to solve (2.2) and (2.3) for $\mathbf{x}$ is relatively inexpensive compared to the elimination step which produced $\mathbf{R}$. Back-substitution with a sparse $\mathbf{R}$ involves $\theta(\mathbf{R})$ operations [11], where $\theta(\mathbf{R}) \leq d^2$ is defined as the number of non-zeros in $\mathbf{R}$.

Because elimination carries a worst-case $\mathcal{O}(d^3)$ complexity, it often represents the majority of computation in practice [32]. For this reason, this thesis focuses on *elimination complexity* as a measure of the inherent complexity represented by a graph.

## 2.2 Graph elimination as sparse factorization

The system (2.1) is often solved by QR or Cholesky factorization of $\mathbf{A}$ or $\mathbf{A}^T\mathbf{A}$, respectively [25,31,38,42,50,52]. For dense matrices, this entails $\mathcal{O}(d^3)$ operations [38]. In the case that these matrices are sparse (as in the smoothing problem), factorization can be done much more efficiently [11,13].

As noted by [31,42,52] and others, the complexity of sparse QR elimination of this system follows the pattern of node elimination on a graph. In node elimination, variable nodes are eliminated one-by-one from the graph, corresponding to the marginalization of the corresponding variable from the joint distribution over the remaining variables. When a node is eliminated, it is removed from the graph, and edges are induced such that all its remaining neighbors form a fully-connected clique. These new edges which did not exist in the original graph constitute *fill*, and represent intermediate dependencies between variables induced by a particular elimination ordering. In the final upper-triangular $\mathbf{R}$ factor, these fill edges correspond to nonzero "filled-in" entries that were zero in the original system matrix $\mathbf{A}^T\mathbf{A}$. The process of node elimination is illustrated in Figure 2-1.

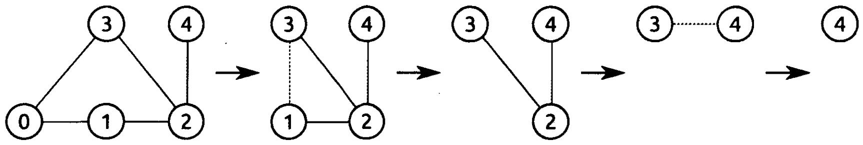


Figure 2-1: The node elimination algorithm executed on a simple graph $G^{(0)}$. Nodes are eliminated in the order (0, 1, 2, 3, 4), producing a series of elimination graphs $G^{(i)}$. Induced edges are shown with dotted lines.

It should be noted that for certain graph structures and elimination orders, fill-in can be catastrophic, destroying sparsity. For example, as shown by Duff [16], sparse factorization of random matrices with initially very few non-zeros (corresponding to very few edges in the graph) almost always results in near $d^3$ computation, as fill-in quickly densifies the graph.

As is well-understood from the sparse linear algebra literature [10, 13, 62], fill is dependent on the chosen variable ordering $\mathcal{P}$. Though the solution itself is unaffected by ordering, different orderings can result in widely differing fill at each step of the optimization. Determining the optimal (i.e. minimum complexity) ordering is NP-complete [62]. In practice, efficient heuristics such as Column-Approximate Min-Degree (`COLAMD`) [10] are widely used.

Each step of node elimination corresponds to computing one step of the corresponding sparse QR or Cholesky factorization [13,25,52], and scales with the size of the neighbors of the eliminated node. From this perspective, solving the full system (2.1) is equivalent to solving a series of sub-problems. Thanks to sparsity, these sub-problems are generally small in size, and by exploiting it to deconstruct (2.1) in this way, significant computational savings can be achieved [25,38]. The complexity of factorizing the full sparse system is then simply the sum of the complexities of the individual dense sub-problems.

As early as 1972, Rose [52] showed by a simple counting of operations that computing the $\mathbf{R}^T\mathbf{R}$ decomposition of a sparse $n \times n$ matrix can be performed in

$$\frac{1}{2}\sum_{i=1}^{n-1} d(i,\mathcal{P})(d(i,\mathcal{P})+3) \sim \sum_{i=1}^{n-1} d(i,\mathcal{P})^2 \tag{2.4}$$

multiplications, where $d(i,\mathcal{P})$ refers to the degree of the $i$-th eliminated node in the elimination graph $G^{(i)}$ produced by ordering $\mathcal{P}$. Indeed, the asymptotic form of (2.4) is equivalent to the Cholesky FLOP count used by [41]. Note that for a fully dense matrix (corresponding to a fully-connected graph), $d_i \sim n$ and factorization approaches the $n^3$ complexity for dense matrices.

From a purely linear algebra perspective, factorization of the system (2.1) occurs one row or column at a time. The corresponding graph $G^{(0)}$ includes $n$ nodes, matching the scalar dimension of the system. However, as described by Dellaert and Kaess [13], when referring to SLAM systems, (2.1) has additional block structure. In SLAM, the variables of interest are often multi-dimensional quantities such as positions and rotations, and measurements generally are defined on the level of these "macro-variables".

In this case, it is the *block* sparsity pattern of (2.1) which is represented in the factor graph.

By applying ordering heuristics such as `COLAMD` [10] on the block structure directly, [13] showed improved performance and less fill. Following this fact, modern SLAM solvers such iSAM2 [32] apply elimination directly on the "macro-variables" of the factor graph. This motivates the definition of a version of (2.4) which accounts for the block structure of SLAM.

**Definition 1.** *The* elimination complexity $\mathcal{C}(G, \mathcal{P})$ *of a factor graph $G$ with variables $\mathcal{X}$ and ordering $\mathcal{P}$ is defined*

$$\mathcal{C}(G, \mathcal{P}) \triangleq \sum_{i=1}^{|\mathcal{X}|} d_f(i) d_s(i, \mathcal{P})^2$$

*where $d_f(i, \mathcal{P})$ and $d_s(i, \mathcal{P})$ are the total scalar dimension of the $i$-th frontal variable $\mathbf{x}_f$ and its corresponding separator set $\mathbf{x}_s$, respectively.*

Note that under scalar elimination, which corresponds to frontal variables of singular dimension $d_f(i) = 1$, Definition 1 reduces to the asymptotic form of (2.4).

**Lemma 1.** *For a fixed elimination ordering $\mathcal{P}$ and graph $G$, let $G^+$ be constructed by adding an edge to $G$. Then, $\mathcal{C}(G, \mathcal{P}) \leq \mathcal{C}(G^+, \mathcal{P}$.*

The proof of Lemma 1 is shown in Appendix A. Lemma 1 confirms the intuition that for a fixed ordering, adding an edge to the graph cannot decrease elimination complexity. Equivalently, removing an edge cannot *increase* complexity.

A justification for elimination complexity $\mathcal{C}(G, \mathcal{P})$ as a representation of the computation performed by block-wise, sparse factorization is provided in the following section.

### 2.2.1 Block-wise sparse factorization

Following the block structure of linear systems characteristic of SLAM problems [13], block-wise sparse factorization proceeds block-by-block. From a graphical perspective, this corresponds to node elimination on the factor graph, where "macro-variables" $\mathbf{x}_i \in \mathcal{X}$ are eliminated one at a time. As is typical in SLAM, $\mathbf{x}$ may represent a robot pose, a landmark position, a sensor calibration, etc.

At the $i$-th step of elimination, the variable being eliminated is referred to as the *frontal* variable $\mathbf{x}_f$, of dimension $d_f$. Those variables adjacent to $\mathbf{x}_f$ (i.e. those which share a common factor) in the elimination graph $G^{(i)}$ at step $i$ are referred to as the *separator* variables $\mathbf{x}_s$.

Thanks to the conditional independence properties implied by the factor graph (1.6), no other variables in $\mathcal{X}$ are involved at this step

$$\mathrm{p}(\mathbf{x}_f|\mathcal{X}) = \mathrm{p}(\mathbf{x}_f|\mathbf{x}_s) \propto \Phi(\mathbf{x}_f, \mathbf{x}_s) \tag{2.5}$$

where the *joint potential* $\Phi(\mathbf{x}_f, \mathbf{x}_s)$ over the active set $\mathbf{x}_f \cup \mathbf{x}_s$

$$\Phi(\mathbf{x}_f, \mathbf{x}_s) \propto \exp\big\{ -\frac{1}{2}\|\mathbf{A}_f\mathbf{x}_f + \mathbf{A}_s\mathbf{x}_s - \mathbf{b}\|^2\big\} \tag{2.6}$$

is formed by collecting the measurement Jacobians $\mathbf{A}_f$ and $\mathbf{A}_s$ over all measurements with respect to $\mathbf{x}_f$ and $\mathbf{x}_s$, respectively. Assume the total dimension of the active measurements is $m$, then $\mathbf{A}_f$ is $m \times d_f$ and $\mathbf{A}_s$ is $m \times d_s$. Here $\mathbf{b}$ refers to the $m$-dimensional right-hand side vector (from the measurement residuals).

In order to eliminate $\mathbf{x}_f$, the new marginal factor $\Phi'(\mathbf{x}_s)$ must be computed

$$\begin{aligned}
\Phi'(\mathbf{x}_s) &\propto \exp\big\{ -\frac{1}{2}\|\mathbf{A}'\mathbf{x}_s + \mathbf{b}'\|^2\big\} && (2.7)\\
&\propto \exp\big\{ -\frac{1}{2}(\mathbf{x}_s^T\mathbf{A}'^T\mathbf{A}'\mathbf{x}_s - \mathbf{b}^T\mathbf{A}'\mathbf{s})\big\} && (2.8)\\
\mathbf{A}' &\triangleq \mathbf{A}_s - \mathbf{A}_f(\mathbf{A}_f^T\mathbf{A}_f)^{-1}\mathbf{A}_f^T\mathbf{A}_s && (2.9)\\
\mathbf{b}' &\triangleq \mathbf{b} - \mathbf{A}_f(\mathbf{A}_f^T\mathbf{A}_f)^{-1}\mathbf{A}_f^T\mathbf{b}
\end{aligned}$$

$\Phi'(\mathbf{s})$ represents the summarized information over the separators $\mathbf{x}_s$ which was represented by the frontal set $\mathbf{x}_f$ [31]. This can be thought of as *marginalization* of $\mathbf{x}_f$, and (2.9) represents the Schur complement. As $\Phi'(\mathbf{x}_s)$ represents a fully-correlated marginal over $\mathbf{x}_s$, it causes the nodes making up $\mathbf{x}_s$ to become fully-connected, and therefore possibly creating fill.

Maximizing the conditional probability $\mathrm{p}(\mathbf{x}_f|\mathbf{x}_s) \propto \Phi(\mathbf{x}_f, \mathbf{x}_s)$ for a given $\mathbf{x}_s$ leads to

$$\mathbf{A}_f^T\mathbf{A}_f\mathbf{x}_f = \mathbf{A}_f^T\mathbf{b} - \mathbf{A}_f^T\mathbf{A}_s\mathbf{x}_s \tag{2.10}$$

representing the reduced system over $\mathbf{x}_f$. (2.10) is solved during during the back-substitution step, computing the ML estimate of $\mathbf{x}_f$ given the ML estimate of $\mathbf{x}_s$.

In summary, each step of elimination produces a small sub-problem over $\mathbf{x}_f$ and its neighbors in the elimination graph $\mathbf{x}_s$. Solving this sub-problem involves computing the marginal factor $\Phi'(\mathbf{x}_s)$ via (2.8) or (2.9). Additionally, it involves factorizing the reduced system (2.10) into a convenient form to be solved efficiently during the back-substitution phase.

As will be seen, this can be accomplished using an appropriate *dense* QR or Cholesky factorization over a reduced system, each representing $\sim d_f d_s^2$ operations. In both cases, we define the augmented Jacobian matrix $\bar{\mathbf{A}} \triangleq \begin{bmatrix}\mathbf{A}_f & \mathbf{A}_s & \mathbf{b}\end{bmatrix}$ of dimension $m \times (d_f + d_s + 1)$.

### Via dense QR factorization

$$\bar{\mathbf{A}} \triangleq \begin{bmatrix}\mathbf{A}_f & \mathbf{A}_s & \mathbf{b}\end{bmatrix} = \begin{bmatrix}\mathbf{Q}_1 & \bigstar\end{bmatrix} \begin{bmatrix}\mathbf{R} & \mathbf{M}_s & \mathbf{m} \\ \mathbf{0} & \bigstar & \bigstar\end{bmatrix} \tag{2.11}$$

The orthogonal matrix $\mathbf{Q} = \begin{bmatrix}\mathbf{Q}_1 & \bigstar\end{bmatrix}$ is formed column-by-column via Gram-Schmidt or Householder Reflections [51], as the right-hand-side matrix is computed row-by-row. Only the first $d_f$ steps of decomposition need to be performed. This partial QR requires $\sim d_f(d_f + d_s + 1)^2$ multiplications [51].

From the block matrix equality (2.11), it can be shown that the following equalities hold

$$\mathbf{A}_f = \mathbf{Q}_1\mathbf{R} \qquad \mathbf{A}_f^T\mathbf{A}_f = \mathbf{R}^T\mathbf{R} \tag{2.12}$$
$$\mathbf{Q}_1^T\mathbf{A}_s = \mathbf{M}_s \qquad \mathbf{Q}_1^T\mathbf{b} = \mathbf{m}$$

Combining (2.12) with (2.7) and (2.10), the necessary outputs can be directly (and inexpensively) computed

$$\begin{aligned}\mathbf{A}' &\triangleq \mathbf{A}_s - \mathbf{A}_f(\mathbf{A}_f^T\mathbf{A}_f)^{-1}\mathbf{A}_f^T\mathbf{A}_s \\ &= \mathbf{A}_s - (\mathbf{Q}_1\mathbf{R})(\mathbf{R}^T\mathbf{R})^{-1}(\mathbf{Q}_1\mathbf{R})^T\mathbf{A}_s \\ &= \mathbf{A}_s - \mathbf{Q}_1\mathbf{Q}_1^T\mathbf{A}_s \\ &= \mathbf{A}_s - \mathbf{Q}_1\mathbf{M}_s \end{aligned} \tag{2.13}$$

$$\begin{aligned}\mathbf{b}' &\triangleq \mathbf{b} - \mathbf{A}_f(\mathbf{A}_f^T\mathbf{A}_f)^{-1}\mathbf{A}_f^T\mathbf{b} \\ &= \mathbf{A}_s - \mathbf{Q}_1\mathbf{b} \end{aligned} \tag{2.14}$$

$$\begin{aligned}\mathbf{A}_f^T\mathbf{A}_f\mathbf{x}_f &= \mathbf{A}_f^T\mathbf{b} - \mathbf{A}_f^T\mathbf{A}_s\mathbf{x}_s \\ \mathbf{R}^T\mathbf{R}\mathbf{x}_f &= (\mathbf{Q}_1\mathbf{R})^T\mathbf{b} - (\mathbf{Q}_1\mathbf{R})^T\mathbf{A}_s\mathbf{x}_s \\ &= \mathbf{R}^T\mathbf{m} - \mathbf{R}^T\mathbf{M}_s\mathbf{x}_s \\ \implies \quad \mathbf{x}_f &= \mathbf{R}^{-1}(\mathbf{m} - \mathbf{M}_s\mathbf{x}_s) \end{aligned} \tag{2.15}$$

(2.13) and (2.14) are computed during elimination to determine the marginal factor $\Phi'(\mathbf{x}_s)$. (2.15) is computed during back-substitution. Because $\mathbf{R}$ is upper-triangular, and $\mathbf{Q}_1$, $\mathbf{M}_s$, and $\mathbf{m}$ are formed during the factorization step, these all can be computed inexpensively.

### Via dense Cholesky factorization

Performing Cholesky decomposition of the augmented information matrix to produce $\bar{\mathbf{A}}^T\bar{\mathbf{A}} = \bar{\mathbf{R}}^T\bar{\mathbf{R}}$, the upper-triangular $\bar{\mathbf{R}}$ is defined:

$$\bar{\mathbf{R}} \triangleq \begin{bmatrix} \mathbf{R} & \mathbf{M}_s & \mathbf{m} \\ \mathbf{0} & \bigstar & \bigstar \\ \mathbf{0} & \mathbf{0} & \bigstar \end{bmatrix} \tag{2.16}$$

Interpreting this as a block-matrix equality:

$$\bar{\mathbf{A}}^T\bar{\mathbf{A}} = \begin{bmatrix} \mathbf{A}_f^T\mathbf{A}_f^T & \mathbf{A}_f^T\mathbf{A}_s & \mathbf{A}_f^T\mathbf{b} \\ \bigstar & \bigstar & \bigstar \\ \bigstar & \bigstar & \bigstar \end{bmatrix} = \bar{\mathbf{R}}^T\bar{\mathbf{R}} = \begin{bmatrix} \mathbf{R}^T\mathbf{R} & \mathbf{R}^T\mathbf{M}_s & \mathbf{R}^T\mathbf{m} \\ \bigstar & \bigstar & \bigstar \\ \bigstar & \bigstar & \bigstar \end{bmatrix} \tag{2.17}$$

As only the first $d_f$ rows of the augmented Cholesky factor $\bar{\mathbf{R}}$ in (2.16) are needed, factorization again requires $\sim d_f(d_f + d_s + 1)^2$ operations [51].

(2.17) provides the necessary equalities in order to efficiently compute outputs required by (2.8) and (2.10).

$$\begin{aligned} \mathbf{A}'^T\mathbf{A}' &\triangleq \mathbf{A}_s^T\mathbf{A}_s - (\mathbf{A}_s^T\mathbf{A}_f)(\mathbf{A}_f^T\mathbf{A}_f)^{-1}(\mathbf{A}_f^T\mathbf{A}_s) \\ &= \mathbf{A}_s^T\mathbf{A}_s - \mathbf{M}_s^T\mathbf{R}(\mathbf{R}^T\mathbf{R})^{-1}\mathbf{R}^T\mathbf{M}_s \\ &= \mathbf{A}_s^T\mathbf{A}_s - \mathbf{M}_s^T\mathbf{M}_s \end{aligned} \tag{2.18}$$

$$\mathbf{A}'^T\mathbf{b} = \mathbf{A}_s^T\mathbf{b} - \mathbf{M}_s^T\mathbf{m} \tag{2.19}$$

$$\begin{aligned} \mathbf{A}_f^T\mathbf{A}_f\mathbf{x}_f &= \mathbf{A}_f^T\mathbf{b} - \mathbf{A}_f^T\mathbf{A}_s\mathbf{x}_s \\ \mathbf{R}^T\mathbf{R}\mathbf{x}_f &= \mathbf{R}^T\mathbf{m} - \mathbf{R}^T\mathbf{M}_s\mathbf{x}_s \\ \implies \quad \mathbf{x}_f &= \mathbf{R}^{-1}(\mathbf{m} - \mathbf{M}_s\mathbf{x}_s) \end{aligned} \tag{2.20}$$

### 2.2.2 Summary

As discussed in Section 2.2, sparse factorization of the full linear system (1.8) corresponds to elimination on a graph. Leveraging the block structure common in SLAM problems [13], state-of-the-art solvers like iSAM2 [32] perform elimination block-wise rather than one row or column at a time. Each step of this elimination process represents a small sub-problem over frontal variable $\mathbf{x}_f$ and separators $\mathbf{x}_s$.

As generally $d_s \gg d_f$, the complexity results of either QR- or Cholesky-based elimination can be simplified to $\mathcal{O}(d_f d_s^2)$. This lends theoretical justification to the use of elimination complexity $\mathcal{C}(G, \mathcal{P})$ (from Definition 1) as a measure of the computation involved in solving the system represented by graph $G$ with ordering $\mathcal{P}$. Experimental verification will be provided in Section 2.3.

It should be immediately clear that the dimension of the separator set $d_s$ has a large impact on computation. Even if the frontal variables have small dimension $d_f$, the complexity of eliminating $\mathcal{X}_f$ scales with the square of the dimension of the separators $\mathcal{X}_s$. As edges are induced (fill-in) during the process of elimination, naive elimination orderings or adverse graph structures can result in large separators $d_s$ and therefore significant computation, even if the degree of each node in the original graph is relatively low [16].

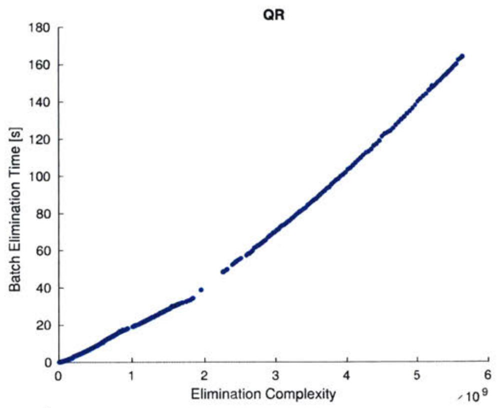
#### Elimination complexity and iSAM2

Characterizing the update-time computation of incremental solvers such as iSAM2 [32] is in general difficult. Designed to avoid re-eliminating the full graph at each update, the iSAM2 update re-linearizes and re-orders variables within the Bayes Tree structure [31] as needed, depending on the numerical values of the measurements and current estimate. Additionally, in order to avoid full re-elimination at each update, the elimination ordering represented in the Bayes Tree is semi-static, and depends on the update history of that particular iSAM2 problem. Though generally guided by a `COLAMD` ordering [10], the ordering implicitly "baked-in" to the Bayes Tree may not necessarily match $\mathcal{P}_{\texttt{COLAMD}}(G)$ at any time. Furthermore, iSAM2 applies a *multifrontal* [14] approach which operates on cliques in the Bayes Tree, rather than on block variables one at a time. By facilitating parallel, distributed computation, multifrontal methods can take advantage of multi-core processors if present. As each clique corresponds to a grouping of multiple variables, this results in a slightly different procedure and operation count than captured by $\mathcal{C}$ as presented in Section 2.2.1.

Despite all these factors, by representing the elimination complexity of the factor graph $G$, $\mathcal{C}$ can be shown numerically to correlate well with "worst-case" update time. This is demonstrated in the next section.

## 2.3 Experimental validation

For elimination complexity $\mathcal{C}(G, \mathcal{P})$ to be a useful measure of graph complexity, it should correlate linearly with computation time. Experimental timing results from simulated SLAM problem described in more detail in Section 3.4) were collected, and are shown in Figure 2-3. Over several runs with varying levels of measurement pruning, the elimination complexity using a heuristic `COLAMD` [10] ordering was computed at each step. The sparse blockwise elimination process described in Section 2.2.1 was performed using either dense QR or Cholesky factorization at each elimination step. Additionally, the iSAM2 update time was recorded.



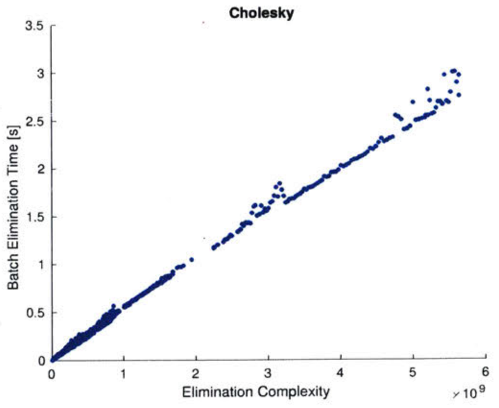


Figure 2-2: Simulation results demonstrate the linear relationship between elimination complexity and total computation time. The data used is a conglomeration of several simulation runs with varying parameter settings. (left) Using dense QR at each elimination step shows a near-linear relation with residual $R^2 \approx 0.984$. (right) Using dense Cholesky at each step shows a linear relation with residual $R^2 \approx 0.997$.

Figure 2-2 plots the batch elimination time with either dense QR or Cholesky factorization. As expected, the batch elimination time in both cases follows linearly in $\mathcal{C}(G, \mathcal{P}_{\texttt{COLAMD}})$. The $R^2$ statistics for the linear fits were approximately 0.984 and 0.997, respectively. While QR and Cholesky demonstrate similar asymptotic complexity, the actual operation counts have different constant coefficients. As can be easily seen in Figure 2-2, using Cholesky rather than QR for each sub-problem produces a significant computation time reduction.

The computation involved in the incremental updates of iSAM2 is much more difficult to predict. iSAM2 achieves significant computational savings by only propagating updates over the a local region of the graph, and employs fluid relinearization and reordering schemes to avoid batch Gauss-Newton iteration [32]. Predicting relinearization and update propagation requires access to the numerical values of the current estimate, putting them outside the scope of a graph metric like $\mathcal{C}$. Nonetheless, the *in practice* worst-case incremental update time of iSAM2 in Figure 2-3 also shows a positive (sub-linear) trend with elimination complexity.

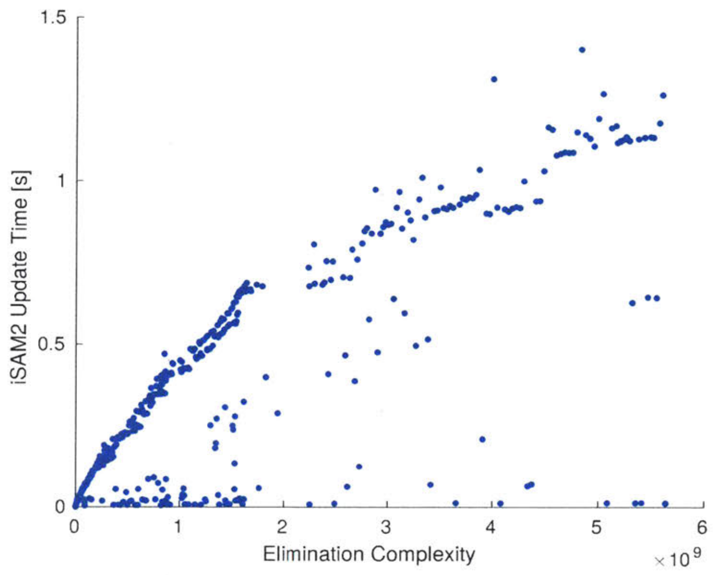


Figure 2-3: Simulation results comparing incremental iSAM2 update time with $\mathcal{C}(G, \mathcal{P}_{\texttt{COLAMD}})$. As an incremental solver, iSAM2 often produces relatively quick updates, and is in general difficult to predict. Nevertheless, the (in practice) worst-case performance still trends (sub-linearly) with elimination complexity.

## 2.4 Optimal elimination complexity

Use of $\mathcal{C}(G, \mathcal{P})$ is complicated by the fact that it depends heavily on elimination order $\mathcal{P}$ [25, 52].

**Definition 2.** *The* optimal *elimination complexity $\mathcal{C}^\star(G)$ is the minimum $\mathcal{C}(G, \mathcal{P})$ over all possible permutations $S(|\mathcal{X}|)$ of variables $\mathcal{X}$.*

$$\mathcal{C}^\star(G) \triangleq \min_{\mathcal{P} \in S(|\mathcal{X}|)} \mathcal{C}(G, \mathcal{P})$$

The *optimal* elimination ordering provides a measure of the intrinsic complexity of a graph $G$. Unfortunately, determining the optimal complexity $\mathcal{C}^\star$ for general graphs is NP-complete [41, 62]. However, from its definition, we can always upper-bound the minimum complexity $\mathcal{C}^\star(G) \leq \mathcal{C}(G, \mathcal{P})$ using any chosen ordering $\mathcal{P}$. This provides a useful method for analyzing the predictable macro-sparsity of particular SLAM architectures. The effectiveness of measurement selection strategies which impact macro-sparsity (such as decimation) can also be analyzed this way.

## 2.5 Summary

Efficient solutions to the smoothing SLAM problem rely heavily on the underlying sparsity of the system. Gauss-Newton methods used to solve general nonlinear problems involve repeated solution of linearized systems of large dimension. Because these linearized systems share the same sparsity as the full nonlinear problem, they can be solved efficiently. Elimination over the graph is the most expensive phase of this procedure, and is equivalent to sparse QR or Cholesky factorization of the corresponding system matrix. By assessing the complexity of this factorization, the elimination complexity $\mathcal{C}(G, \mathcal{P})$ quantifies the computation required to eliminate a particular graph $G$ using a particular ordering $\mathcal{P}$.

One key observation is that elimination complexity, and therefore overall computation, is not simply a function of edge or factor count. Instead, computation in SLAM is much more a function of of graph structure. Practically speaking, this means that depending on their positions in the graph (i.e. the variable nodes they connect), two otherwise similar measurements can have dramatically different impacts on computation. For measurement selection, and computation management in general, this has significant implications.

Many existing measurement selection methods in the SLAM literature focus on removing a fixed [36] or $\mathcal{L}_1$-determined [28] number of edges from the graph, irrespective of their arrangement. While it is true that removing edges cannot increase elimination complexity $\mathcal{C}$, pruning done without regard to structure can produce an underwhelming reduction in complexity. In contrast, measurement selection strategies which promote sparse structure directly can achieve more significant savings. As will be seen in Chapter 3, decimation-style strategies do exactly that, and therefore can be quite effective.

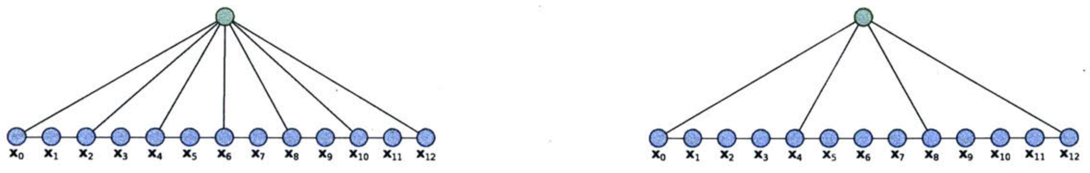
By understanding and bounding the optimal elimination complexity $\mathcal{C}^\star(G)$ for particular graph super-structures, the macro-sparsity of graph architectures and measurement selection strategies can be assessed. Ultimately, this can aid in the design and analysis of inherently sparse graph structures and measurement selection strategies, making SLAM accessible to increasingly computationally-constrained systems.

# Chapter 3

# A Rigorous Look at Decimation

Chapter 2 developed the elimination complexity metric, providing a connection between graph structure and the computational complexity associated with optimization. Because measurements correspond to factors/edges in this graph, intelligent measurement selection can then be used to minimize elimination complexity and therefore reduce computation.

As one of the simplest selection policies, *decimation* can be defined as the policy of taking every $r$-th measurement from a particular data stream. From the perspective of landmark SLAM, a decimation policy might accept only every $r$-th landmark observation. Some simple examples of decimated graphs are shown in Figure 3-1.


x0 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12
x0 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12

(a) Decimation rate of 2 (b) Decimation rate of 4

Figure 3-1: Results of a simple decimation rule on the SLAM graph with a single landmark (green). Assuming that observations are available from any of the robot poses (blue), decimation rules select only every $r$-th observation possible. Pose nodes are labeled in a time-ordered fashion.

In this chapter, fundamental insights into the sparsity and connectivity of decimated graphs are provided. First, it is shown in Section 3.1 that decimation, if applied consistently, produces a naturally sparse super-structure that necessarily bounds fill-in at each step of elimination. In practice, this results in very sparse graphs that can maintain many more factors at the same elimination complexity as graphs without this super-structure. Second, for simple single-landmark graphs, the even spacing between observations characteristic of decimation is proven in Section 3.2 to be near optimal in a weighted tree-connectivity sense. As the weighted number of spanning trees has been shown to be related to the uncertainty volume of the joint distribution represented by the graph [36], this suggests decimation is also effective from an estimation perspective.

Despite decimation's simplicity, this combination of properties make it an effective primitive for graph sparsification. Section 3.3.2 introduces `dec++`, a decimation-inspired policy tailored towards incremental SLAM. Empirical evidence from a simple visual-inertial SLAM simulation demonstrates that `dec++` performs quite well, matching or even outperforming more sophisticated strategies.

### 3.0.1 Decimation as a measurement selection policy

Decimation can be a very general concept, and therefore it is important to define it specifically in the context of the goals of this chapter.

In a SLAM context, decimation often arises as a keyframing method. In a keyframing application, decimation can be used to downsample the incoming video stream by only accepting every $r$-th image frame. Like [61], [29], and [58], keyframing decimation can provide a rule for determining when to insert a new pose variable in the graph.

Alternatively, in landmark-SLAM, decimation can be used to select individual measurements, on a per-*track* basis. A track is the sequence of observations associated with a particular landmark. In general, per-track decimation allows for cases in which the decimation patterns of different landmarks may be *offset* from each other by a constant in the range $\{0, 1, 2, \ldots, r-1\}$. Though less common in the literature, it is this interpretation of decimation that will be the main focus of this chapter.

Furthermore, it will be assumed that the choice of pose nodes included in the graph is fixed. The fact that all pose nodes, including "intermediate" nodes which may not be associated with any incorporated observations, are still represented in the optimization is important for several reasons. First, in applications such as the local-frame-planning described in Section 4.1.7, these intermediate poses may be of direct interest. Second, fixing the set of pose nodes makes analysis and comparison between strategies more straightforward, as the set of estimation variables is kept consistent between methods.

If the set of pose nodes in the graph is taken to be fixed, decimation can take two different forms: aligned and non-aligned. The *aligned* case is most similar to a keyframing policy, in which all landmarks share the same decimation offset. In the more general *non-aligned* case, this alignment is not enforced, and the decimation offset of a particular landmark is determined by the pose index of the first included observation. These topologies are shown side-by-side in Figure 3-2.

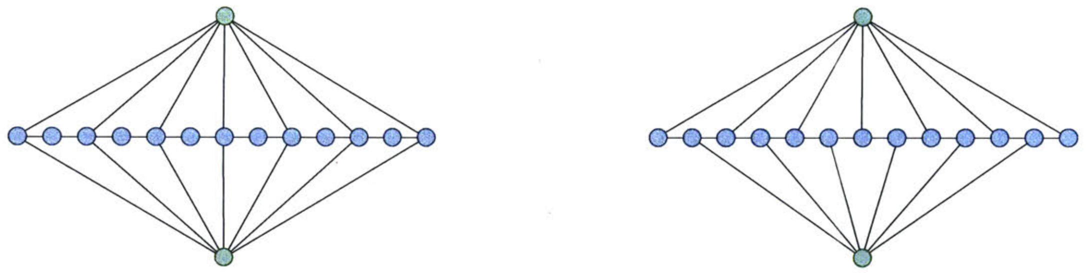


Figure 3-2: Simple side-by-side example of aligned (left) vs. non-aligned (right) decimation for the two-landmark case. In both cases, a decimation rate of $r = 2$ is applied. In the aligned case, observations from the two landmarks (green) can be considered to be decimated per-pose, while in the non-aligned case, decimation is applied per-track. Poses in the aligned topology can be labeled as either a *keypose* or an *intermediate* pose according to the presence or absence of associated landmark observations.

In general, tracks can be decimated fully independently from one another, with different offsets and potentially even different decimation rates. However, for the purposes of this thesis, a consistent, global decimation rate $r$ for all tracks will be assumed.

One last technicality must be cleared up. For the purposes of the discussion here, the decimation rate $r$ is taken with respect to the corresponding pose variable *indices.* For example, imagine a landmark is observed from a series of consecutive poses $\{3, 4, 5\}$, leaves the sensor horizon and is not observed by poses $\{6, 7, 8\}$, and then returns to be observed by pose $\{9\}$. This corresponds to a track with associated pose indices $\{3, 4, 5, 9\}$. The style of decimation discussed here with $r = 2$ would keep the observations from poses $\{3, 5, 9\}$, *not* only $\{3, 5\}$ as would be produced if only the first and third *observations* were accepted. The resulting offset associated with this track would be 1, as determined by the index of the first associated pose.

One emphasis of this chapter is to discuss the differences between these two topologies in terms of elimination complexity $\mathcal{C}$ and estimation quality. As can perhaps be expected by their more restricted structure, aligned graphs are shown to be again much sparser than their non-aligned counterparts, and therefore much cheaper to optimize. However, this comes at the cost of lower connectivity and therefore reduced estimation quality over the full set of variables (including intermediate poses).

In the analysis and experiments to follow in this chapter, the following additional assumptions are made.

1. As is conventional in landmark SLAM, landmarks are considered to be distributed (*a priori*) independently of one another, and therefore there are no landmark-landmark edges.

2. As is the case in odometry-focused problems, it assumed there are no explicit pose-pose loop closures, and therefore, pose node $i$ is only adjacent to pose nodes $i - 1$ and $i + 1$.

3. No other variable types are represented in the graph.

The last two assumptions can be relaxed with some restrictions without affecting the following results, as will be discussed in Section 3.5.3. Nevertheless, they are assumed here for simplicity and clarity.

## 3.1 Sparsity of decimated graphs

As discussed in Chapter 2, sparsity refers to the desirable structure and properties of a graph that make it amenable to efficient optimization. Trees are the sparsest connected graphs, and complete graphs are the least sparse. Additionally, from Lemma 1 we know that removing an edge from the graph will maintain or decrease elimination complexity. Nonetheless, elimination complexity is not purely a function of edge count, and graphs with equivalent numbers of vertices and edges can have dramatically different complexities.

### 3.1.1 Empirical motivation

The claim that structure rather than edge count determines computation can easily verified experimentally. On a simulated visual SLAM problem (described in more detail in Section 3.4), the computation savings resulting from decimation were evaluated in a batch fashion. Given the full SLAM graph with all original observations, pruning was then performed under aligned, non-aligned, and random strategies in order to quantify the reduction in *de facto* optimization complexity, $\mathcal{C}_{\texttt{COLAMD}}$. A subset of the resulting pruned graphs are shown in Figure 3-3, and computational results are shown in Table 3.1.

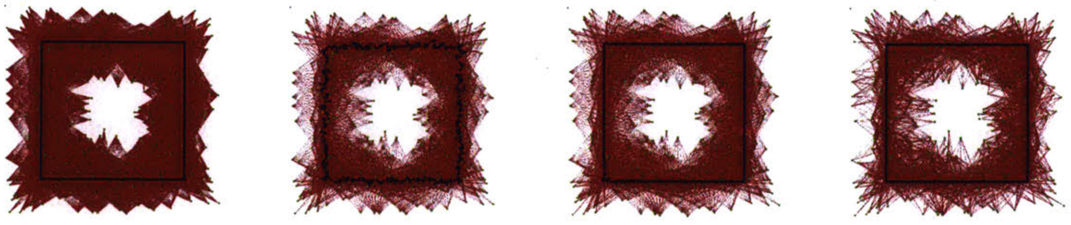


Figure 3-3: Undirected graph of simulated VIN SLAM problem with a robot maneuvering along a square trajectory. Pose nodes are shown in blue, with landmarks in green. Red edges represent visual observations. From left to right: The full graph with all possible observations; under aligned decimation; under non-aligned decimation; under random decimation. All pruned graphs here are the result of a pruning rate $r = 6$. Note that estimation quality varies significantly across different strategies – this will be discussed in Section 3.1.4.

Table 3.1: Simulation results demonstrating that decimation strategies produce significantly sparser graph structure for the same number of factors (edges). $\mathcal{C}_{\texttt{COLAMD}}$ corresponds to the *de facto* elimination complexity of the resulting graph, and elimination time is the computation time of performing the batch elimination operation as described in Section 2.2.1. Both are reported as percentages of the corresponding values for the full graph, and thus give a clear metric of computation reduction. Note that elimination time generally matches $\mathcal{C}_{\texttt{COLAMD}}$. The details of the full graph are shown in the top row, and relative quantities (with respect to the full graph) are shown for the particular pruning methods. Note that random pruning **rand** removes a similar number of factors but fails to reduce computation proportionally.

| **Method** | **# Factors** | $\mathcal{C}_{\texttt{COLAMD}}$ | **Elim Time** |
|---|---|---|---|
| `full` | 355503 | $3.24e^9$ | 3.4 [s] |
| `adec2` | 18030 | 14.0 % | 14.5 % |
| `ndec2` | 18101 | 46.5 % | 45.6 % |
| `rand2` | 18002 | 75.6 % | 83.6 % |
| `adec4` | 9300 | 2.2 % | 2.7 % |
| `ndec4` | 9417 | 22.0 % | 24.4 % |
| `rand4` | 9251 | 43.3 % | 54.8 % |
| `adec6` | 6372 | 0.8 % | 1.2 % |
| `ndec6` | 6505 | 12.2 % | 12.7 % |
| `rand6` | 6334 | 40.4 % | 50.6 % |

The details of the full graph are shown in the row labeled `full`, and results under the various pruning strategies are reported with either absolute or relative values for easy comparison. **adec** and **ndec** refer to aligned and non-aligned decimation topologies, respectively. As a baseline, **rand** refers to random pruning, as edges are selected uniformly to be discarded. This strategy was parameterized to prune a similar number of measurements as the decimation strategies. Each strategy was evaluated over a series of $r$ values in $\{2, 4, 6\}$.

For any particular value of $r$, all three strategies maintain a similar number of factors (edges), but produce graphs of dramatically different computational complexities. As can be seen from this experiment, both decimation strategies produce graphs of significantly lower complexity than random pruning. It will be shown shortly that this is precisely due to the characteristic structure of decimated graphs.

### 3.1.2 The sparse super-structure of decimated graphs

In this section, it will be shown that decimated graphs have a global super-structure that makes elimination relatively inexpensive. By analyzing this super-structure, we can naturally bound the worst-case optimal elimination complexity.

For conciseness in the following statements, it is assumed that all variables in the graph (both landmark and pose variables) have the same dimension $d = 1$. Generalizing the analysis to account for varying variable dimensions is straightforward.

#### Elimination complexity in the non-aligned case

In the non-aligned case, decimation produces a specific super-structure illustrated in Figure 3-4. Taking Figure 3-4a to be a typical SLAM graph, non-aligned decimation produces the graph shown in Figure 3-4b. This decimated graph can be re-drawn as shown in Figure 3-4c, demonstrating a partitioned structure which is generalized in Figure 3-4d.

In order to properly analyze this structure, some notation must be defined. Assume the pose nodes are numbered sequentially in time as shown. Each pose node $i$ can then be assigned to exactly one set $\Pi_k$, where $k = i \mod r$ is assigned based on the decimation rate $r$. We can similarly assign each landmark node to exactly one set $\mathcal{L}_k$, based on the label of its first incorporated observation. We will refer to this label $k \in \{0, 1, 2, \ldots, r-1\}$ as the decimation offset.

Letting $n_\pi$ refer to the number of poses and $n_l$ the number of landmarks, the vertices of the decimated graph are partitioned into $r$ subgraphs. After decimation, each

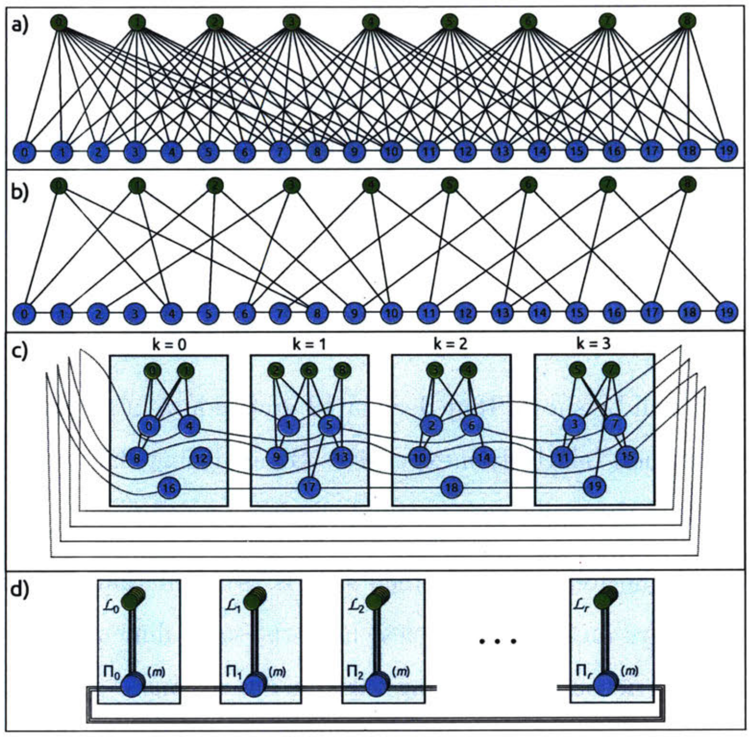


Figure 3-4: (a) Example SLAM graph with 9 landmarks (green) and $n_\pi = 20$ poses (blue). Due to sensor limitations and the nature of the trajectory, each landmark will in general only be observed from a subset of all poses. (b) The same graph after a non-aligned decimation with $r = 4$ is applied. The first available observation of each landmark is kept, thus determining the decimation offset of that track. (c) The graph (b) can be re-drawn, showing that decimation has essentially partitioned the graph into $m \triangleq \frac{n_\pi}{r}$ subgraphs. These subgraphs have limited inter-connections (the odometry edges). (d) A generalized illustration of the partitioning structure produced by non-aligned decimation.

landmark node is adjacent *only* to poses of one particular set $\Pi_k$. This partitioned structure is illustrated in Figure 3-4d.

Thus, the vertices of the decimated graph are partitioned into disjoint subsets $(\Pi_k, \mathcal{L}_k)$ parameterized by decimation offset $k$. As can be seen, the neighborhood of landmark $j \in \mathcal{L}_k$ is necessarily a subset of $\Pi_k$. Most importantly, these subgraphs have limited inter-connectedness, as the nature of the odometry edges restricts members of set $\Pi_k$ to only be adjacent to $\Pi_{k-1}$ and $\Pi_{k+1}$. Note that if $r$ divides $n_\Pi$ evenly, $|\Pi_k| = m \triangleq \frac{n_\pi}{r}$ for all $k$.

**Proposition 1.** *The optimal elimination complexity of a landmark-pose SLAM graph subject to non-aligned decimation is upper bounded*

$$\mathcal{C}^\star \leq (n_l + 9rm)m^2$$

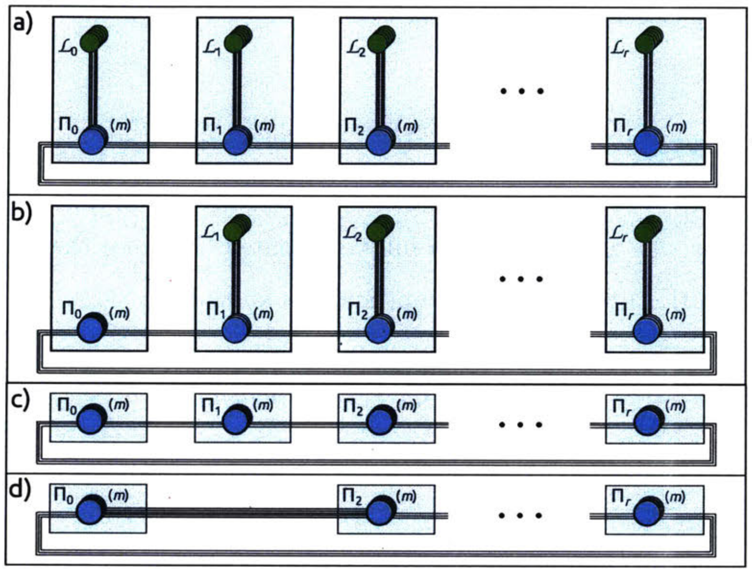


Figure 3-5: Proposed elimination process in the non-aligned case. (a) General non-aligned decimated graph. (b) After elimination of landmarks $\mathcal{L}_0$, edges are induced which in the worst-case cause $\Pi_0$ to become a complete subgraph. (c) After landmark elimination for the remaining $\mathcal{L}_k$. (d) All nodes in $\Pi_1$ are eliminated, inducing edges between $\Pi_0$ and $\Pi_2$. The elimination of the remaining $\Pi_k$ proceeds similarly from here.

*Proof.* We can upper-bound the optimal elimination complexity (i.e. the min complexity over all orderings) by evaluating the elimination complexity for any particular ordering. The procedure is shown in Figure 3-5.

Start by eliminating all landmarks, incurring a complexity of

$$C_l \leq \sum_{k=0}^{r-1} \sum_{i=1}^{|\mathcal{L}_k|} |\Pi_k|^2 = \sum_{k=0}^{r-1} |\mathcal{L}_k| m^2 = n_l m^2$$

Because the neighborhood of each landmark $j \in \mathcal{L}_k$ is limited to $\Pi_k$, this cannot induce edges outside of $\Pi_k$. In the worst case, edges have been induced such that the variables within each $\Pi_k$ subgraph form fully-connected cliques of size $m$.

We can proceed from $k = 0$ to $k = r - 1$, eliminating *all* pose variables within $\Pi_k$ at each step. At the point of eliminating each pose variable $i \in \Pi_k$, the neighborhood of $i$ is a subset of $\Pi_{k-1} \cup \Pi_k \cup \Pi_{k+1}$, which has cardinality $3m$. In total, pose node elimination is loosely upper-bounded by

$$C_\pi \leq \sum_{k=0}^{r-1} \sum_{i=1}^{m} (3m)^2 = rm(3m)^2 = 9rm^3$$

After these steps, the graph has been fully eliminated. Because the chosen ordering cannot (by definition) be better than optimal, we know that

$$C^\star \leq C_l + C_\pi \leq (n_l + 9rm)m^2$$

□

## Elimination complexity in the aligned case

A similar analysis can be performed in the aligned case. Compared to the non-aligned case, the aligned case has an even more restrictive structure, as a consistent decimation offset is used for all landmarks. Pose nodes in an aligned decimation graph either belong to the set of $m$ keyposes $\Pi_0$ or the set of intermediate poses $\bar{\Pi}$. The intermediate pose nodes are not adjacent to any landmarks, and form simple odometry chains between keyposes as illustrated in Figure 3-6.

**Proposition 2.** *The optimal elimination complexity of a landmark-pose SLAM graph subject to aligned decimation is upper bounded*

$$C^\star \leq 4(r-1)m + (n_l + m)m^2 \sim (n_l + m)m^2$$

*Proof.* Again, we derive an upper-bound for the elimination complexity over the op-

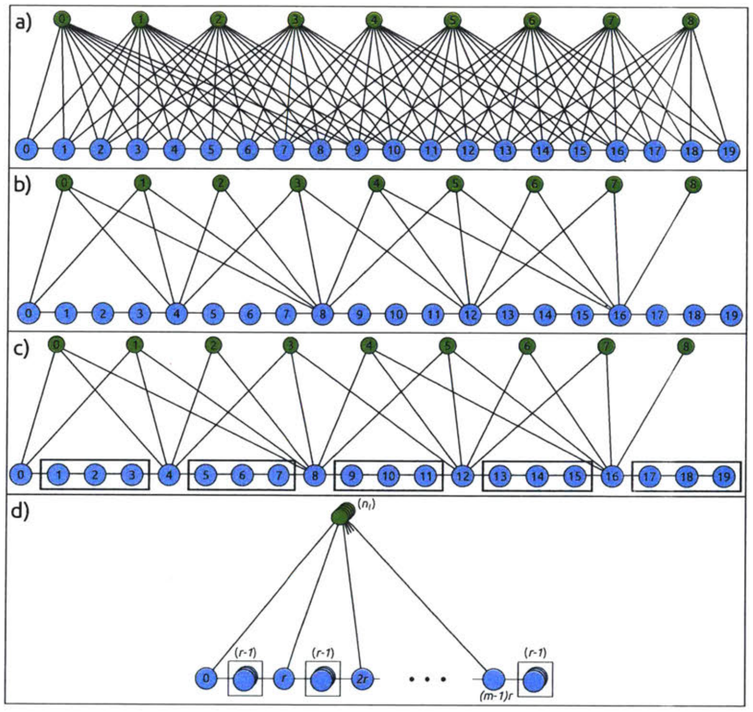


Figure 3-6: (a) Example SLAM graph with 9 landmarks (green) and $n_\pi = 20$ poses (blue). (b) The same graph after an aligned decimation with $r = 4$ is applied (the offset is 0 in this case). (c) Intermediate poses form a single odometry path between consecutive keyposes. (d) A generalized illustration of graph (c).

timal order by evaluating the complexity for a particular order. The procedure is shown in Figure 3-7.

Because intermediate poses form single chains, they can be eliminated inexpensively. Each intermediate pose $i \in \bar{\Pi}$ has at most two neighbors (the prior and following pose nodes $i-1$ and $i+1$), so this step has complexity

$$C_{\bar{\Pi}} = \sum_{i=1}^{|\bar{\Pi}|} 2^2 = 4(r-1)m$$

Next, elimination over the $n_l$ landmarks has complexity of at most

$$C_l \leq \sum_{j=1}^{n_l} m^2 = n_l m^2$$

This induces edges between the remaining poses $\Pi_0$, at worst leaving them a complete subgraph. Finally, eliminating the $m$ keyposes comes at a worst-case (fully-connected) cost of

$$C_{\Pi_0} \leq \sum_{i=1}^{m} m^2 = m^3$$

Thus, the total elimination cost with this ordering is upper bounded by

$$\mathcal{C}^{\star} \leq \bar{\mathcal{C}} \triangleq C_{\bar{\Pi}} + C_l + C_{\Pi_0} = 4(r-1)m + (n_l + m)m^2$$

$\square$

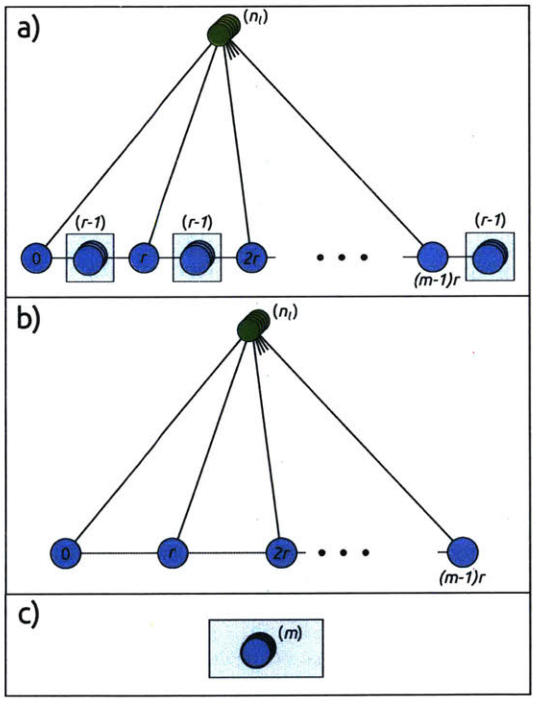


Figure 3-7: Proposed elimination process in the aligned case. (a) General aligned decimated graph. (b) We can start by sequentially eliminating intermediate pose chains, resulting in single induced edges connecting consecutive keyposes. (c) Elimination of all landmarks results in a (at worst) clique over the $m$ keyposes $\Pi_0$. From here, the keyposes can be sequentially eliminated.

In the non-aligned case, the key property defining sparsity was the partitioning of the graph into $r$ subgraphs. In the aligned case, the key characteristic is the selection of only $m$ keypose nodes from which landmarks are observed. Obviously, the aligned superstructure is a more restrictive special case of the non-aligned superstructure.

### 3.1.3 The utility of these bounds

The bounds provided in Propositions 1 and 2 are quite loose, as the worst-case connectivity assumed at each step of the derivations is conservative. Additionally, these bounds assume that all landmarks are observable from all robot poses (i.e. the case of no micro-sparsity), although sensor limitations dictate this almost never occurs in practice. Thus, these bounds speak only to the macro-sparsity (see Section 2.1.1) induced by decimation. The optimal ordering, and the heuristic ordering chosen in practice, will almost certainly do much better given the micro-sparsity of the particular graph in question.

Nonetheless, the superstructures demonstrated in Figures 3-4 and 3-6 still have significance. They imply that, given a reasonable variable ordering, fill-in is contained within a limited region of the graph at each step of elimination. Put another way, graphs with this partitioning property are guaranteed to have orderings that produce bounded levels of fill.

In the case of pruning techniques without this structure, such as `rand`, a similarly "good" elimination ordering may not exist. Using similar arguments to those described here, the corresponding worst-case complexity bound for more general graphs is $\mathcal{O}((n_l + n_\pi)n_\pi^2)$, resulting from first eliminating the landmarks and then the poses. Without a similar partitioning property, more generally-pruned graphs can be arbitrarily complex up to this general bound.

Compared to this bound, non-aligned and aligned decimation produce bounds which are asymptotically better by a factor of almost $r^2$. As was seen experimentally in Table 3.1, the difference in super-structure between decimated and randomly-pruned graphs can correspond to significant differences in elimination complexity in practice. This explains why decimated graphs can afford many more edges while still being less computationally expensive than graphs produced via other pruning strategies.

### 3.1.4 Aligned decimation is the most sparse

These bounds suggest that graphs produced by an aligned decimation scheme should be much more efficient than even non-aligned decimation schemes, largely due to the ability to inexpensively "pre-eliminate" intermediate pose nodes. This idea is verified empirically in Table 3.1, as the aligned graph with a near-identical number of factors (edges) results in significantly lower elimination cost. As, in many cases, intermediate poses need not be represented in the first place, this result can be applied to all keyframing methods (not just decimation). Compared to the full graph, Proposition 2 implies that representing only one $r$-th of all input frames as keyframes corresponds to a complexity reduction that scales with a factor between $r^2$ and $r^3$ (depending on whether $n_l$ or $n_\pi$ is dominant). Thus, keyframing is a very powerful method of computation reduction, although with limitations as discussed in Section 3.3.1.

Of course, sparsity is not the only metric of interest. As will be seen experimentally in Section 3.4, non-aligned decimation produces a more connected, better-constrained estimation problem, and often significantly better accuracy.

## 3.2 $t_w$-optimality in single-landmark graphs

Of course, computational efficiency is not the only consideration when considering a measurement selection policy. It is also desirable that the distribution represented by the pruned graph be a good approximation of the distribution represented by the full set of data. While several information-theoretic metrics can be used, including the Kullback-Leibler Divergence (see Section 1.5.2), such metrics generally depend on the particular measurement values. For the purposes of understanding the estimation performance of decimation as a general strategy, graph-theoretic analysis is much more applicable.

As shown by Khosoussi et al. [35] and introduced in Section 1.5.1, the weighted number of spanning trees $t_w(G)$ is related to uncertainty volume $\det(\Sigma)$ in SLAM problems. In this perspective, edges (measurements) are weighted by their precision $w_{i,j} = \frac{1}{\sigma^2}$.

Put another way, pruning strategies which maintain a large $t_w$ should in turn minimize the inevitable uncertainty increase over the full set of variables. In this section, it is shown that in single-landmark graphs, the even spacing of observations characteristic of decimation is in fact near $t_w$-optimal. This suggests that decimation in general may have good information-theoretic performance, which is later demonstrated experimentally in Section 3.4.

**Lemma 2.** *Useful properties of $t_w(G)$ for connected, weighted graph $G$ with positive weights:*

- *$t_w(G) > 0$ for any connected graph.*
- *$t_w(G)$ is non-decreasing as edges are added to $G$.*
- *For any edge $(i,j)$ in $G$, $t_w(G) = t_{w,\{\times(i,j)\}}(G) + t_{w,\{\backslash(i,j)\}}(G)$. Here, $t_{w,\{\times(i,j)\}}(G)$ is defined as the weighted number of spanning trees which* necessarily *include $(i,j)$. $t_{w,\{\backslash(i,j)\}}(G)$ is defined as the weighted number of spanning trees which do not include edge $(i,j)$.*

**Lemma 3.** "Dangling chains": *For any graph $G^-$, consider a graph $G$ formed by attaching a chain of edge length $n > 1$ at one end to exactly one vertex in $G^-2$, such that $G$ and $G^-$ have the same number of cycles. Let $v \triangleq \prod_{i=1}^{n} w_i$ be defined as the product of the edge weights of the added chain. Then*

$$t_w(G) = v t_w(G^-)$$

For brevity, proofs of intermediate results have been left to Appendix A.

We will begin by examining $t_w$-connectivity in a simplified, single-landmark class of SLAM graphs. Imagine that a robot follows a trajectory along $n+1$ poses, along which it records odometry measurements and observes a single landmark, represented by node $l$. Assume that the discrete pose variables $\{x_0, x_1, \ldots, x_n\}$ are evenly spaced in time. Additionally, assume the robot is constrained to take at most $m+1$ observations of the landmark, and each has to be from one of the $n+1$ discrete poses. Two such scenarios are illustrated in Figure 3-8.

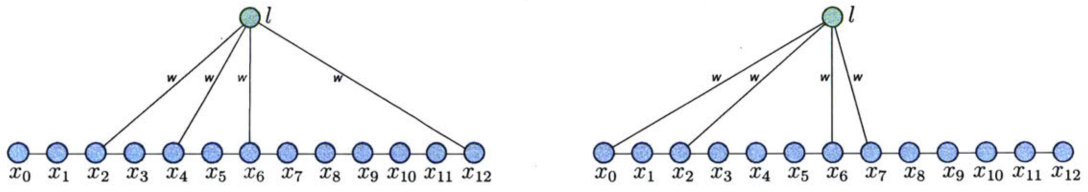

l
w
w
w
w
x0 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12
l
w
w
w
w
x0 x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11 x12

Figure 3-8: Two example single-landmark graphs from the set $\bar{\mathcal{G}}_{m,n}$ with $m = 3$ and $n = 12$. Note that the observation edges are each assigned weight $w$, and odometry edges have unity weight. Each of these graphs corresponds to a particular choice of $m$ observation edges for a robot driving along $n+1$ poses.

Weights can be assigned to the edges in the corresponding graph according to the precision of the corresponding measurements. Thanks to the assumption that the state variables are distributed uniformly in time, and because odometry sensors such as IMUs are generally modeled as continuous-time systems with additive white-noise, the discrete-time noise along each odometry edge is also uniform (see Appendix B). Thus, we can weight all odometry edges equally with $w_x$.

Additionally, we will make the simplifying assumption that all observations are also equally precise, and assign weights $w_o$. In general, particularly in the case of visual observations, this is not true, and even the first-order noise propagation is sensitive to scene geometry (see Section 1.2.1). Nonetheless, the approximation is necessary to facilitate the following analysis, and ultimately offers some illuminating insight. Because a global scaling applied to all edge weights scales $t_w$ equivalently, for notational simplicity let us assume all weights are scaled such that odometry edges have unity weight and observation edges are weighted by $w \triangleq \frac{w_\mathbf{o}}{w_\mathbf{x}}$. Note that all original weights, and therefore the simplified weight $w$, are assumed to be strictly positive.

The natural question is of optimality: with $n+1$ observation "opportunities" and a "budget" for $m+1$ edges, how do we select observations to maximize $t_w$? First off, it is known from Lemma 2 that $t_w$ is non-decreasing with additional edges. Thus, there will always exist an optimal graph with *exactly* $m$ observation edges, thus without loss of generality it will be assumed that exactly $m$ observations are taken.

Define $\bar{\mathcal{G}}_{m,n}$ as the family of single-landmark graphs with $n+1$ discrete poses and $m$ observations. Figure 3-8 shows two examples. For a fixed $m$ and $n$, members of $\bar{\mathcal{G}}_{m,n}$ are differentiated only by the selection of observation edges connecting pose nodes $x$ to the landmark node $l$. Selecting observations corresponds to selecting observation edges, and therefore to selecting members of $\bar{\mathcal{G}}_{m,n}$.

The problem of maximizing $t_w$ for single-landmark graphs can be expressed as the following optimization:

**Problem 1.**

$$\underset{G\in\bar{\mathcal{G}}_{m,n}}{\operatorname{argmax}}\ t_w(G)$$

Thanks to the following lemma (proved in Appendix A), a maximizer of Problem 1 exists, and must belong to a more restricted class of graphs.

**Lemma 4.** *A maximizer of Problem 1 must exist, and must include the first and last possible observation edges. That is, it must include the observation edges from pose* $x_0$ *and* $x_n$.

This family will be referred to as $\mathcal{G}_{m,n} \subset \bar{\mathcal{G}}_{m,n}$. Two examples of this sub-family are shown in Figure 3-9.

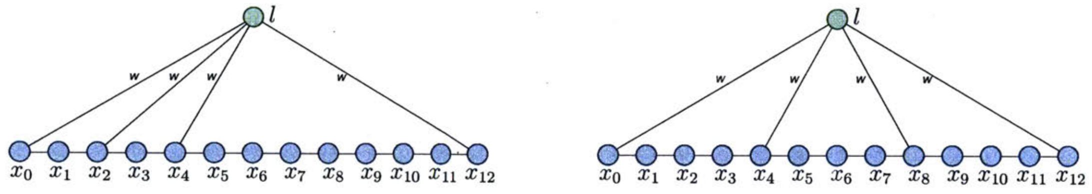

l
w
w
w
w
$x_0$ $x_1$ $x_2$ $x_3$ $x_4$ $x_5$ $x_6$ $x_7$ $x_8$ $x_9$ $x_{10}$ $x_{11}$ $x_{12}$
l
w
w
w
w
$x_0$ $x_1$ $x_2$ $x_3$ $x_4$ $x_5$ $x_6$ $x_7$ $x_8$ $x_9$ $x_{10}$ $x_{11}$ $x_{12}$

Figure 3-9: Two example single-landmark graphs from the set $\mathcal{G}_{m,n} \subset \bar{\mathcal{G}}_{m,n}$ with $m=3$ and $n=12$. By Lemma 4, any maximizer of Problem 1 must be of this form.

As seen in Figure 3-10, members $G_m(\mathbf{k}) \in \mathcal{G}_{m,n}$ can be parameterized by the spacing *between* observations. Each of the $m$ "triangular" cycles is defined by a single chain of odometry edges of length $k_i \in \mathbb{Z}_{++}$. The sum of these chain lengths, of course, must equal $\sum_{i=1}^{m} k_i = n$.

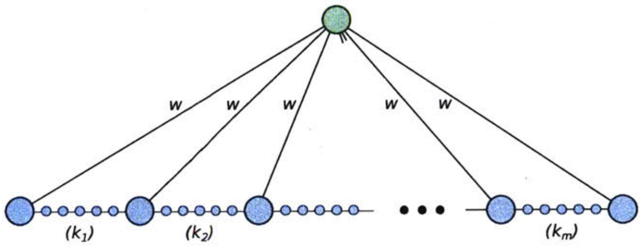


Figure 3-10: A generalized example of a member of $\mathcal{G}_{m,n}$, parameterized by the observation *spacing* $\mathbf{k}$. $k_i$ corresponds to the number of odometry edges between the pose nodes of consecutive observation edges. Note that $\sum_{i=1}^{m} k_i = n$.

Thus, Problem 1 can be equivalently interpreted as an optimization over only $\mathcal{G}_{m,n} \subset \bar{\mathcal{G}}_{m,n}$, and given a more convenient form:

**Problem 2.** *Define objective* $f_m(\mathbf{k}, w) \triangleq t_w(G_m(\mathbf{k}))$, *which maps a spacing vector* $\mathbf{k}$ *and weight* $w$ *to the weighted number of spanning trees of the corresponding graph.*

$$\begin{aligned} \underset{\mathbf{k}\in\mathbb{Z}_{++}^m}{\text{argmax}} \quad & f_m(\mathbf{k}, w) \\ \text{subject to} \quad & \sum_{i=1}^{m} k_i = n \end{aligned}$$

As will be seen shortly, selecting a "uniform" spacing of observations provides a *near-optimal* solution to Problem 2. This corresponds to choosing $k_i = \frac{n}{m} \triangleq r \quad \forall k_i$, or taking every $r$-th possible observation. This corresponds precisely to the pattern produced by decimation with a rate of $r$.

Of course, the existence of an integer $r$ requires that $n$ is a multiple of $m$. As Problem 2 is technically an integer optimization of objective $f_m(\mathbf{k}, w)$ over $\mathbf{k}$, this will be assumed from here on out.

With that, we can introduce (and then prove) the following claim:

**Claim 1.** *Assuming that* $m$ *evenly divides* $n$, *a uniform spacing of observation edges* *such that* $k_i = \frac{n}{m} \triangleq r \quad \forall k_i$ *provides a near-optimal solution to Problem 2. Defining* $\mathbf{r} \triangleq \begin{bmatrix} r & r & \dots r \end{bmatrix}^T$*:*

$$\log \frac{f_m(\mathbf{r}, w)}{f_m^\star(w)} \geq m \log \frac{rw+1}{rw+2}$$

To prove Claim 1, we derive upper and lower bounds for $f_m(\mathbf{k})$ inductively in Proposition 3, and then substitute the solution $\mathbf{r}$.

**Proposition 3.** *The objective* $f_m$ *is upper- and lower-bounded:*

$$w \prod_{i=1}^{m} (k_i w + 1) \leq f_m(\mathbf{k}, w) \leq w \prod_{i=1}^{m} (k_i w + 2)$$

*Proof.* For a given $\mathbf{k}$, induction is performed on $m$, starting from $m = 1$.

The base case of $m = 1$ is easily established. An illustration is shown in Figure 3-11. Because this graph is a simple loop, the set of spanning trees can be easily enumerated. Trivially, $f_1(\mathbf{k}, w) = w(k_1 w + 2)$.

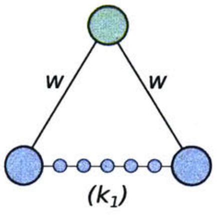


Figure 3-11: Base case of Proposition 3, for which $m = 1$. This graph forms a single cycle. The spanning trees can be trivially enumerated, and the weighted sum is $f_1(\mathbf{k}, w) = w(k_1 w + 2)$.

Next, the inductive step must be established, for $m > 1$. The graph $G_m$ can be constructed recursively by appending an additional segment of size $k_m$ on the *right* side of graph $G_{m-1}(\mathbf{k})$, as shown in Figure 3-12. Note that here, $\mathbf{k}$ has dimension $m - 1$.

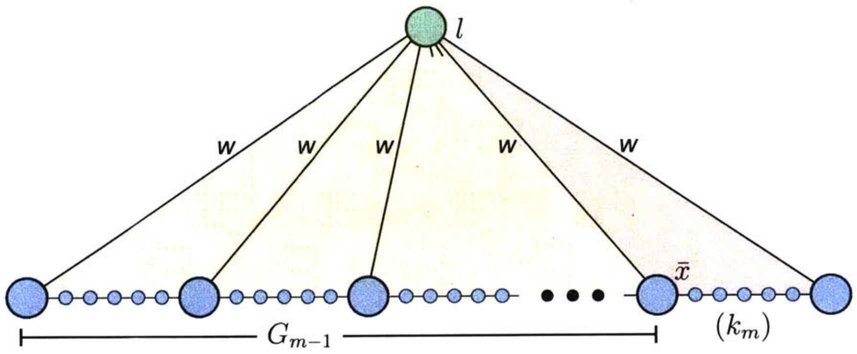


Figure 3-12: Inductive step of Proposition 3, for which $m > 1$. The graph $G_m$ is constructed from $G_{m-1}$ by appending a new path of length $k_m + 1$ between $\bar{x}$ and $l$, forming a new cycle.

Put another way, induction corresponds to adding a new path of $k_m$ odometry edges between pose node $\bar{x}$ and landmark $l$. This new path is shown on the right side of Figure 3-12, highlighted in red, and will be referred to as the "right-most path". By construction, the right-most path is comprised of $k_m$ odometry edges and one observation edge, which has weight $w$. By enumerating spanning trees with respect to this path, a recursive expression for $f_m$ can be found.

Every spanning tree of $G_m$ includes either the complete right-most path, or is missing exactly one of its $k_m + 1$ edges. Note that if two or more edges are removed from this path, the graph becomes disconnected and therefore contains no spanning trees. Thus, we can decompose the set of spanning trees into $\mathcal{T}(G_m) = \mathcal{T}_{\text{broken}} \cup \mathcal{T}_{\text{complete}}$, where the explicit dependence on $G_m$ has been dropped for brevity. $\mathcal{T}_{\text{broken}}$ refers to the set of spanning trees which are missing exactly one edge from the right-most path, and $\mathcal{T}_{\text{complete}}$ refers to those spanning trees for which the path is complete.

We can similarly decompose $t_w(G_m)$ into a sum of two terms

$$t_w(G_m) = \underbrace{\sum_{T \in \mathcal{T}_{\text{broken}}} \mathbb{V}(T)}_{t_{w,\text{broken}}(G_m)} + \underbrace{\sum_{T \in \mathcal{T}_{\text{complete}}} \mathbb{V}(T)}_{t_{w,\text{complete}}(G_m)} \tag{3.1}$$

We will start by finding a recursive equality for $t_{w,\text{broken}}(G_m)$. Note that there are

$k_m + 1$ ways to break the right-most path. If one of the $k_m$ odometry edges is broken, the observation edge with weight $w$ must be maintained. The other option is that the observation edge is broken, in which case all odometry edges must be maintained. In either case, the result is one or two dangling chains attached to graph $G_{m-1}$. These possibilities are illustrated in Figure 3-13. By applying Lemma 3, $t_{w,\text{broken}}(G_m)$ has the form

$$t_{w,\text{broken}}(G_m) = wk_m f_{m-1}(\mathbf{k}) + f_{m-1}(\mathbf{k}) = (wk_m + 1) f_{m-1}(\mathbf{k}) \tag{3.2}$$

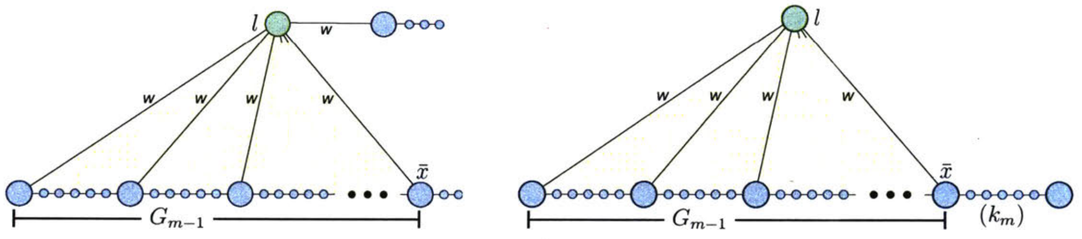


Figure 3-13: Two example graphs with broken right-most paths with respect to $G_m$. (left) One of $k_m$ cases where the spanning tree is missing an odometry edge, and maintains the new observation edge with weight $w$. (right) The graph missing the new observation edge, leaving a single chain of odometry poses. In either case, these graphs demonstrate dangling chains and Lemma 3 applies.

Next, we find a simple recursive inequality for $\mathcal{T}_{\text{complete}}$. For spanning trees necessarily containing the complete right-most loop, this forms a fixed, direct path of weight $w$ between $\bar{x}$ and landmark $l$, as shown in Figure 3-14. In these trees, edge $(\bar{x}, l)$ cannot be included as it would form a cycle with the right-most path. However, fixing the right-most path is functionally equivalent to fixing edge $(\bar{x}, l)$, in the sense that

$$t_{w,\text{complete}}(G_m) = t_{w,\{\times(\bar{x},l)\}}(G_{m-1}) \tag{3.3}$$

$$t_{w,\{\times(\bar{x},l)\}}(G_{m-1}) \triangleq \sum_{T \in \mathcal{T}_{\times(\bar{x},l)}(G_{m-1})} \mathbb{V}(T) \tag{3.4}$$

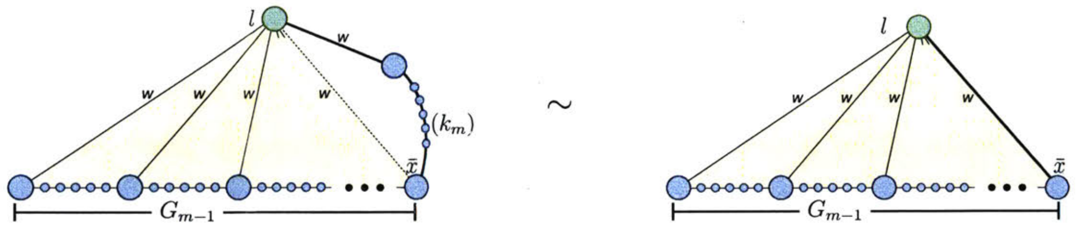


Figure 3-14: (left) Spanning trees within $\mathcal{T}_{\text{complete}}$ necessarily include the full right-most path, shown in bold. This in turn implies that they cannot include dashed edge $(\bar{x}, l)$, as that would form a cycle. (right) In terms of weighted number of spanning trees, the fixed right-most path can be condensed into a single edge of weight $w$, and $t_{w,\text{complete}}(G_m) = t_{w,\{\times(\bar{x},l)\}}(G_{m-1})$.

Leveraging Lemma 2, the weighted number of spanning trees with a fixed, complete right-most path is

$$0 < t_{w,\text{complete}}(G_m) = t_{w,\{\times(\bar{x},l)\}}(G_{m-1}) = t_w(G_{m-1}) - t_{w,\{\setminus(\bar{x},l)\}}(G_{m-1}) < t_w(G_{m-1}) \tag{3.5}$$

where the strict inequality arises from the fact that removing $(\bar{x}, l)$ does not disconnect $G_{m-1}$ and therefore $t_{w,\{\setminus(\bar{x},l)\}}(G_{m-1}) > 0$.

Armed with recursive relations (3.2) and (3.5), substituting into the decomposition (3.1) it can be seen that

$$f_m \triangleq t_w(G_m) = t_{w,\text{broken}}(G_m) + t_{w,\text{complete}}(G_m) \tag{3.6}$$

$$= (wk_m + 1) f_{m-1}(\mathbf{k}, w) + t_{w,\{\times(\bar{x},l)\}}(G_{m-1}) \tag{3.7}$$

and therefore

$$(wk_m + 2) f_{m-1}(\mathbf{k}, w) \geq f_m \geq (wk_m + 1) f_{m-1}(\mathbf{k}, w) \tag{3.8}$$

Combining the base case $f_1 = w(wk_1 + 2)$ and the recursive inequalities (3.8) in a straightforward application of induction, Proposition 3 is clear. □

Once the lower and upper bounds provided by Proposition 3 have been established, we seek to upper-bound the optimal solution to Problem 2, $f^\star(w)$. It should be noted that the constrained maximization of the upper-bound $\Phi(\mathbf{k}) \triangleq \prod_{i=1}^{m}(k_i w + 2)$ is equivalent to maximizing the volume of an $m$-dimensional hyper-cube subject to a total side length constraint. This maximum is achieved with $k_i = r \quad \forall k_i$, and therefore $\Phi^\star = (rw+2)^m$.

Thus

$$f_m(\mathbf{k}, w) \leq f_m^\star(w) \leq \Phi^\star = (rw+2)^m \tag{3.9}$$

By (3.9) and the lower-bound from Proposition 3, it can be seen that

$$\log \frac{f_m(\mathbf{r}, w)}{f_m^\star(w)} \geq \log \frac{f_m(\mathbf{r}, w)}{(rw+2)^m} \geq \log \frac{(rw+1)^m}{(rw+2)^m} = m \log \frac{rw+1}{rw+2} \tag{3.10}$$

This proves Claim 1. □

## 3.2.1 Discussion

The proof of Claim 1 demonstrates that a "uniform" spacing of landmark observations is near-optimal in a $t_w$ sense for fixed number of observations $m+1$. Decimation, by definition, maintains every $r$-th landmark observation, achieving precisely this uniform pattern. For a given number of poses, Claim 1 is equivalent to the claim that decimation makes a near-$t_w$-optimal selection of $m = \frac{n}{r}$ observations, at least for single-landmark systems.

From a keyframing perspective, assuming a consistent raw data rate, aligned decimation by definition evenly spaces keyposes in time. As shown in Appendix B, odometry processes such as IMUs are often modeled with additive Gaussian noise and produce odometry factors whose noise parameters scale with time. Thus, aligned decimation will produce graphs with uniformly-precise (uniformly-weighted) odometry links similar to Figure 3-10, and thus are near-$t_w$-optimal by Claim 1.

In non-aligned decimation, landmarks are not necessarily all connected to the same poses. Unfortunately, Claim 1 does not necessarily generalize to this multi-landmark case. However, each single-landmark subgraph is still "well-connected," and experimental results in Section 3.4 will show that non-aligned decimation performs well from an information volume (the inverse of uncertainty volume) perspective.

Determining the $t_w$-optimal strategy analytically in a more general multi-landmark case is non-trivial. As the precision of nonlinear observations such as vision depend heavily on scene geometry, the observation weights $w_j$ will not all be equal. Additionally, due to sensor limitations, all landmarks often cannot be observed from all points in the trajectory. Thus, the truly $t_w$-optimal choice of landmark edges cannot be determined *a priori* without specific knowledge of the available observations. Greedy and semi-definite-relaxed algorithms proposed by [36] exist, but are computationally impractical for high-rate realtime systems.

## 3.3 Decimation in implementation

In light of the above observations about the favorable properties of decimated graphs, several decimation-style strategies for efficient measurement selection can be proposed. There are two main categories, corresponding roughly to "per-frame" and "per-landmark" styles. Note that here "frame" refers to an image frame, a laser scan, or other sensor reading that contributes a set of simultaneous landmark observations.

In landmark SLAM, of which VIN is a particular case, the odometry factors connecting consecutive poses are generally not pruned. These measurements are often full rank, meaning that the set of odometry factors themselves (along with a gauge fix) are sufficient to fully constrain the ML estimate of pose nodes in SE(2) or SE(3). Thus, they lend a sense numerical stability to the problem. Additionally, as odometry factors usually only connect pose nodes $\mathbf{x}_i$ and $\mathbf{x}_{i+1}$, they form a single chain and thus are naturally sparse. For these reasons, measurement selection and pruning often deals exclusively with landmark observation factors.

### 3.3.1 Keyframe selection

As was originally mentioned in Section 3.0.1, keyframe selection refers to the problem of determining which discrete poses (representing the continuous-time robot trajectory) to include explicitly in the SLAM optimization [58]. From the graphical perspective, these discrete poses correspond to the set of pose nodes in the SLAM graph. As noted by [20, 29], reducing the number of discrete pose nodes along the robot's trajectory can promote consistency. Additionally, the pre-processing associated with each image frame or LIDAR scan can be non-negligible. Thus, processing only a fraction of incoming measurements represents significant computational savings.

An important caveat of keyframing is that only measurements associated with key-poses can be included in the optimization. At any time, currently-tracked landmarks can leave the field-of-view or sensor detection range. As new landmarks cannot be initialized until the next keyframe, this can lead to points in time during which insufficient landmarks are tracked. This is especially an issue for monocular vision-based systems, which require several observations before reliably triangulating a landmark [45]. For this reason, heuristic selection techniques like [58, 61] explicitly aim to maintain sufficient landmarks.

While decimation can be (and is often) used as a keyframing method, its use must be tempered by the need to ensure a sufficient number of landmarks at all parts of the trajectory. Thus, there often exists a maximal extent to which decimation-style keyframing can be used safely, beyond which more flexible per-measurement selection strategies can be leveraged. For example, a 60 Hz image stream from a camera may might be decimated to 20 Hz, with pose nodes added to the graph at this rate. Thus, new landmarks could be detected and initialized at up to 20 Hz. For visual systems with narrow FOV cameras undergoing aggressive rotation, this may be the minimum rate to ensure a sufficient number of landmarks are well-estimated at all times. Beyond this point, any further measurement selection must be performed on a more flexible, per-measurement basis.

Additionally, the estimated trajectory (rather than simply the latest estimate) may be of direct interest. For example, as discussed in Section 4.1.7, the estimate of past states can be used for local-frame trajectory tracking. As only those pose estimates which are represented in the graph can be accessed, this requires a sufficiently high pose incorporation rate.

### 3.3.2 Per-landmark decimation

Given a fixed set of pose and landmark variables, the remaining question is which subset of all available measurements best approximates the full distribution. In this case, the measurement selection process is not restricted to selecting or discarding all measurements associated with a particular pose as a group, but rather each measurement or factor can be evaluated independently. Thus, strategies inspired by the patterns of non-aligned decimation are applicable.

As discussed in Section 3.1, decimation naturally partitions graphs into subgraphs with restricted inter-connectivity. This naturally bounds fill-in and thus significantly reduces elimination complexity compared to un-partitioned graphs. By the same token however, the extent to which subgraphs are partitioned or disconnected from each other is the extent to which the estimation problems are decoupled. Put another way, this reduced connectivity leads to a less constrained estimation problem, and therefore reduced estimation performance. This tradeoff is verified numerically in Section 3.4.

In practice, it should be remembered from Section 2.1.1 that realized micro-sparsity is still significant – that is, realized performance is dependent on both graph superstructure and the actual measurements acquired during operation. This means that sparsity in practice will *gracefully* degrade as the assumptions of strict partitioning are violated. Thus, the partitioning structure presented in Figure 3-4 can be interpreted more as a general guideline rather than a strict rule. As estimation performance intuitively should improve with increased coupling between partitions, the inclusion of "violations" should improve estimation without prohibitively impacting computation. Fortunately, as the following simulation results would suggest, this tradeoff is actually quite favorable. It requires relatively few violations of partitioning to achieve estimation performance on par with much more generally connected structures, while still achieving near-partitioned computational performance.

Two methods are presented here, one which is a straightforward implementation of non-aligned decimation and one which is slightly modified to address specific aspects of incremental SLAM.

`ndec` refers to straightforward non-aligned decimation. Based on the index of the first pose node from which a landmark is observed, only observations corresponding to every $r$-th pose node are maintained. In this scheme, the first observation of a particular landmark is always included, and it determines the decimation "offset" of all future observations associated with the landmark. The partitioning structure of Figure 3-4 is strictly upheld.

### Decimation++

In incremental SLAM, several weaknesses of `ndec` become apparent. First, at initialization, all landmarks observed from the robot's starting pose $x_0$ will be initialized with a decimation offset of 0. Unless new landmarks are introduced immediately, it is unlikely that the next $r-1$ pose nodes will be constrained by many landmark observations. Over time, more landmarks are initialized, and the distribution of landmark decimation offsets becomes more uniform. However, this initial transient pattern is undesirable and can lead to increased error in the early part of the trajectory. Second, the latest pose node is often of most direct interest for closed-loop control. Under simple `ndec` however, the observations associated with latest pose $x_n$ are decimated just like the rest. Instead, if pruning is simply delayed until *after* the update, $x_n$ will be left temporarily un-pruned, with all possible observation constraints. In this way, accuracy of $x_n$ can be increased at little computational cost. At each solve, the observations associated with pose $x_n$ that are due to be decimated represent a small set of violations to the partitioning structure of pure `ndec`. These violations add a small level of coupling between partitions, which in practice significantly improves estimation quality at cost of a slight increase in computation.

In light of these observations, an improved strategy, `dec++`, is proposed. This strategy has two slight modifications addressing specific issues in incremental SLAM.

1. For landmarks observed from initial pose $x_0$, early observations are dropped by a simple scheme in order to distribute the landmark offsets uniformly. Assuming the landmarks are labeled sequentially according to their initialization order, dropping the observations of landmark $j$ before pose $k = j \mod r$ will accomplish this.
2. Pruning is only done up to the second-to-latest pose $x_{n-1}$. All observations associated with pose $x_n$ are kept in the graph, and decimation is only performed on these observations *after* pose $x_{n+1}$ is added.

The performance of these per-landmark methods will be evaluated on synthetic data in the following section.

## 3.4 Simulation Results

A suite of simulation experiments were performed to verify the analytic results discussed above in a full 3D, nonlinear VIN setting. In the simulation, a robot drives a square trajectory, observing nearby landmarks according via a monocular visual sensor, using a pinhole camera model. Poses are represented as elements of SE(3), and landmarks as Cartesian points in $\mathbb{R}^3$.

Consecutive poses are linked via noisy odometry measurements, and noisy visual observations link poses to landmarks. At each step of the simulation, a new pose node is added to the graph, and newly-triangulated landmarks are added to the graph. Because of the under-rank nature of monocular measurements, landmarks are not initialized (i.e. added to the graph) until they have been observed a minimum number of times.

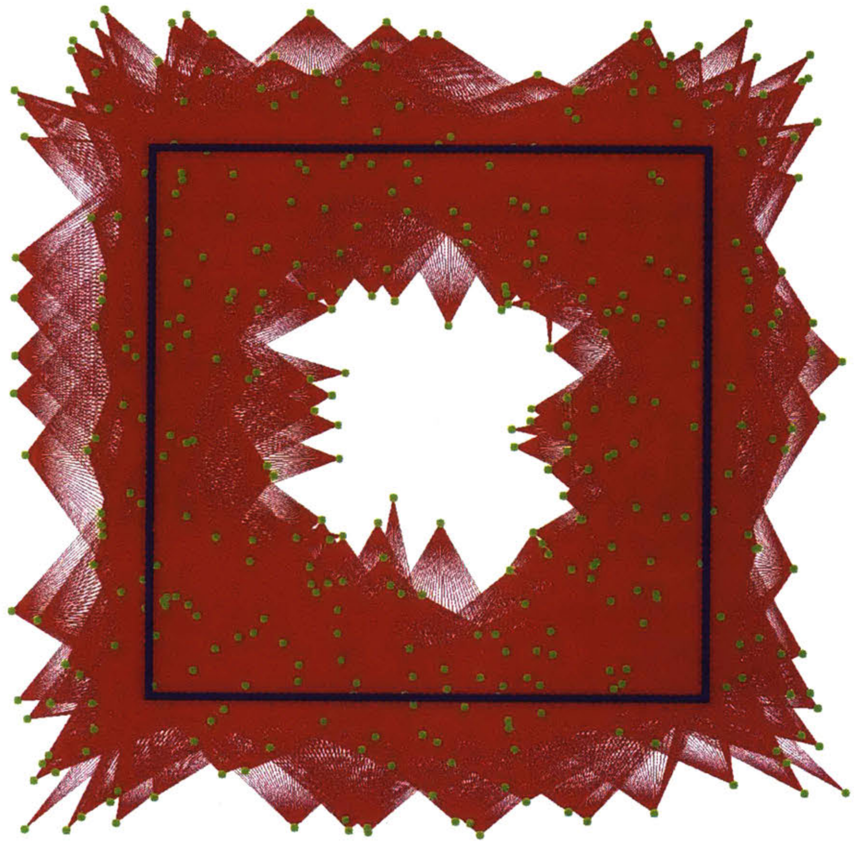


Figure 3-15: Top-down view of the simulation trajectory and full set of measurements. The robot trajectory is shown as the series of blue pose nodes, and landmarks are shown in green. The robot moves in a counter-clockwise direction, starting from the lower-left corner. Red edges indicate the full set of monocular vision observations.

The current estimate is updated at each step of the simulation via the incremental iSAM2 algorithm [32] as implemented in GTSAM [12]. The batch elimination complexity is evaluated and timed at each step as well, to provide a more direct measure of graph sparsity unaffected by the somewhat obfuscating optimization of the incremental solver. As will be seen, the computation time of the incremental solver generally tracks the trends of the full batch computation. All timing experiments are performed on the same desktop machine with an Intel i7 processor running at a nominal 4.0 GHz.

The goal of these experiments is to compare the performance of decimation schemes versus a more sophisticated pruning approach. In all cases, only visual observations (pose-landmark edges) are considered for pruning. The comparison algorithm used is the $t$-optimizing greedy strategy of [36], which at each step greedily removes the $n$ visual observations from *anywhere* in the graph which reduce $t$-connectivity the least. The number of observations removed $n$ is based on a budget of $k$ observations per landmark, and is tuned to approximately maintain a similar number of edges in the graph as the compared decimation approaches. It should be noted that this approach is computationally expensive to implement, and the computation involved in actually executing this pruning procedure is not accounted for in the following results. Rather, the intent here is to demonstrate that, even if such a strategy were computationally "free", it still would not outperform decimation for these problems.

Several pruning strategies are evaluated here.

- `adec`: Aligned decimation.
- `ndec`: Non-aligned decimation.
- `dec++`: Decimation-based strategy introduced in Section 3.3.2.
- `rand`: Random pruning.
- `tgreedy`: Greedy $t$-optimizing approach of [36] [1].

All strategies are tuned to maintain approximately $\frac{1}{r}$ of the full set of observations in the graph. For example, `ndec4` and `rand4` will produce graphs of different structures, but with approximately one-quarter of the observation factors used in the original graph.

Each pruning strategy produces a corresponding approximating graph and distribution over the set of variables. The Kullback-Leibler divergence (KLD) between the

[1]Because only observation edges are considered for pruning, and because all simulated vision measurements are corrupted by an equivalent Gaussian noise, the *unweighted* variant of [36] is used. However, because the vision model (see Section 1.2.1) shows that measurement Jacobian depends on scene geometry, one could imagine a more sophisticated, weighted variant being used, although it is not necessarily clear how weights would be generated.

approximating Gaussian and the full distribution (using all measurements) is used to evaluate estimation quality. Also, the log-determinant of the information matrix $\mathbf{\Lambda} \triangleq \mathbf{\Sigma}^{-1}$ over the linearized joint distribution is plotted. As the information matrix is the inverse of the covariance matrix, higher values here correspond to lower uncertainty. As the $t$-greedy approach is designed to explicitly maximize the information log-determinant, it does the best here for a given number of edges. However, in terms of KLD, decimation can produce graphs which much more efficiently (in terms of computational complexity per edge) approximate the full distribution.

### 3.4.1 Batch graph pruning

In order to validate the impact of decimation structure on estimation and computation, a set of batch experiments were performed. The full graph and set of observations is illustrated in Figure 3-15.

From this full graph, a suite of pruning strategies were executed and evaluated, with various decimation rates. After pruning was performed, the graph was re-optimized, and the divergence and information volume were computed. Because of the large size of the problem, `tgreedy` was too expensive to run in batch.

As can be seen from Figure 3-16, all pruning strategies removed a similar number of factors at each value of pruning rate $r$. As expected, the `rand` graph produced graphs of consistently higher *de facto* elimination complexity $\mathcal{C}_{\texttt{COLAMD}}$ than the decimation-style strategies. Furthermore, the complexity reduction achieved by the random strategy plateaus quickly, and increasing the pruning rate does little to further reduce complexity. In contrast, the decimation-based strategies produce graphs of much lower complexity.

In terms of divergence, the decimation-based `dec++` proposed in Section 3.3.2 does only comparably to or better than `rand`, while producing graphs of much lower complexity. From an information volume perspective, `dec++` and `rand` perform almost identically (higher is better). As expected, `adec` produces extremely inexpensive (low

$\mathcal{C}$) graphs, but at the cost of significantly increased KLD and reduced information volume with respect to the full set of variables.

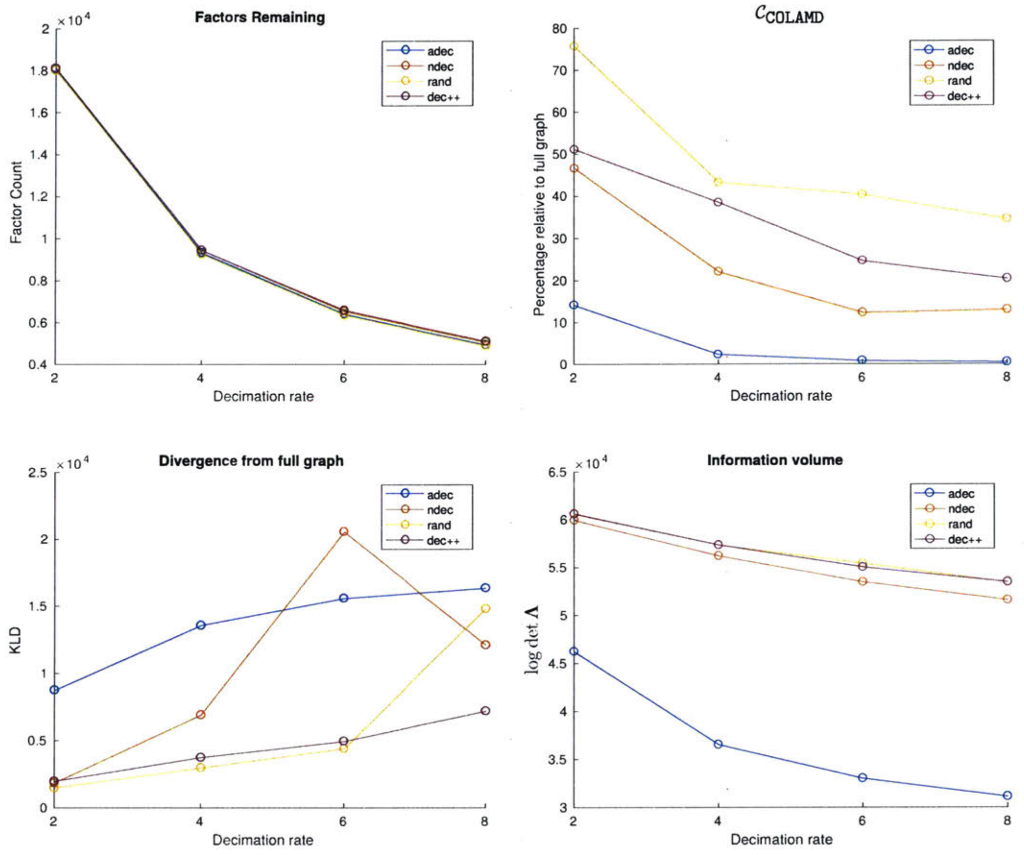


Figure 3-16: Simulation batch pruning results under various strategies and equivalent decimation rates. (top left) Factor counts are near equal for each strategy for a given decimation rate. (top right) Elimination complexity is dramatically reduced for **adec** and **ndec** relative to the unpartitioned graph produced by **rand**. (bottom left) **dec++** achieves a similar KLD relative to the full graph to **rand**, at reduced complexity. **ndec** and **adec** show significantly higher divergence. (bottom right) The log information volume (log inverse of uncertainty volume) is highest with **rand** and **dec++**, as these graphs are most connected. (Higher is better).

### 3.4.2 Incremental SLAM

The robustness of decimation strategies was also evaluated in an incremental SLAM setting. Running the simulation data one iteration at a time, pruning was implemented at each time step. After pruning, an iSAM2 update was executed. The divergence between the approximating graph and the full graph at each step was computed, along with several other metrics.

It should be noted that at each iteration, the `tgreedy` strategy was free to prune observations from anywhere in the graph, while `rand` was restricted to only new observations. Again, it should be stressed that for a given pruning rate $r$, all strategies were tuned to remove a similar number of factors at each step as straightforward decimation. Results under $r = 4$ and $r = 6$ are shown in Figures 3-17 and 3-18, respectively.

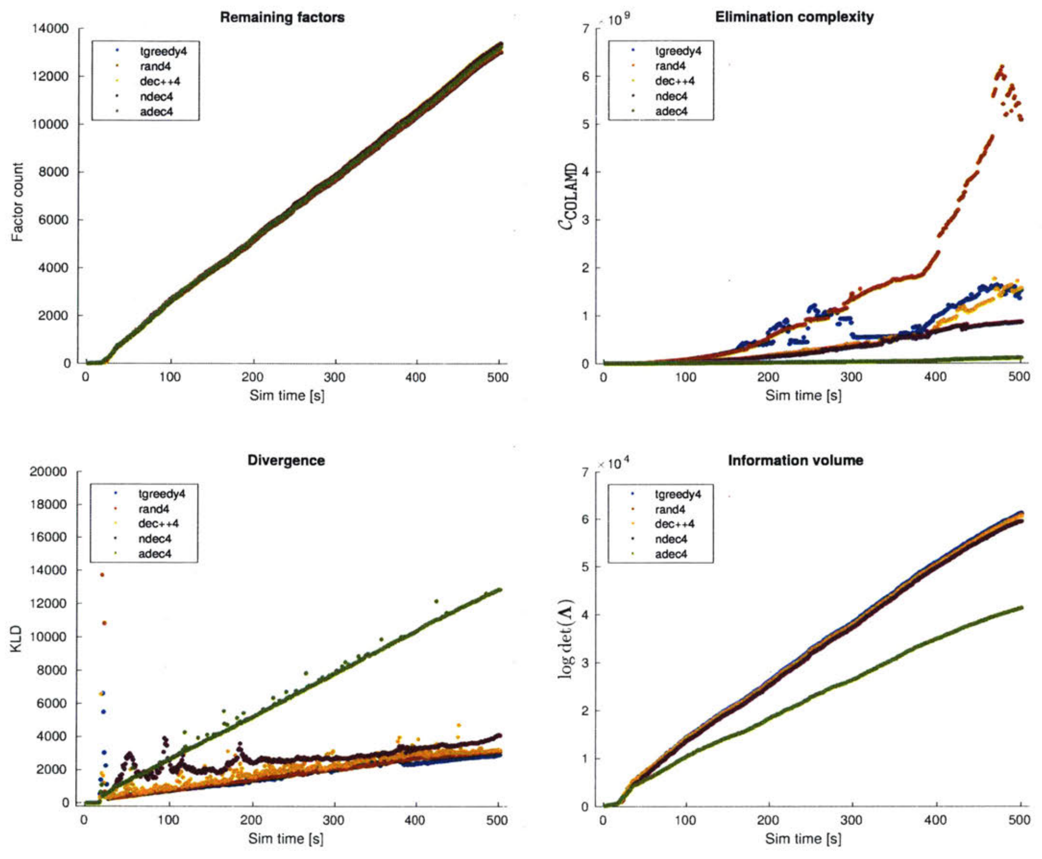


Figure 3-17: Simulation incremental pruning results under various strategies for $r = 4$. (top left) Factor counts are near equal for each strategy. (top right) Elimination complexity is smallest under `adec`. `ndec` and `dec++` produce similar complexity until near the end of the trajectory. `rand` and `tgreedy` are both much higher, but in this case `tgreedy` happens to produce a relatively sparse structure near the end of the trajectory. (bottom left) `dec++` achieves a similar KLD to `rand` and `tgreedy`, at reduced complexity. `ndec` and `adec` show significantly higher divergence, with `ndec` showing large fluctuation as the solution switches between nearby local minima. (bottom right) The log information volume (log inverse of uncertainty volume) is similar for all strategies besides `adec`. (Higher is better).

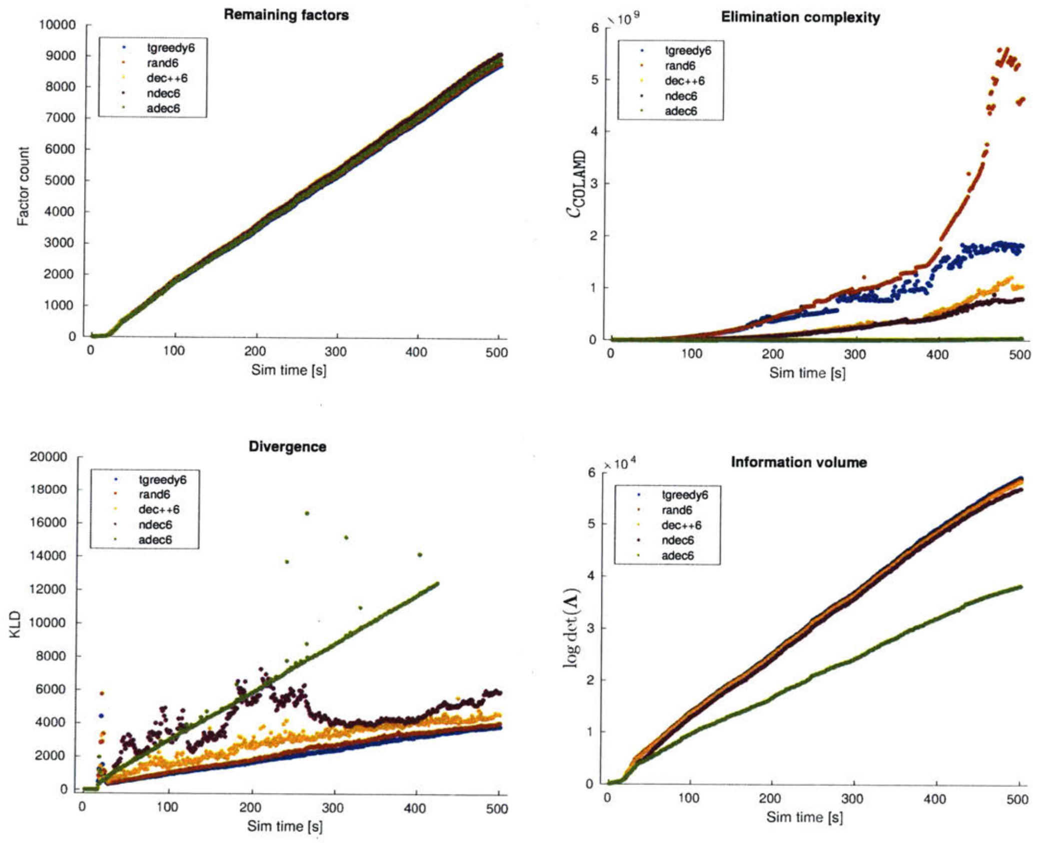


Figure 3-18: Simulation incremental pruning results under various strategies for $r = 6$, demonstrating similar trends as Figure 3-17. (top left) Again, factor counts are near equal for each strategy. (top right) As before, elimination complexity is lowest under **adec**, and similar between **ndec** and **dec++**. (bottom left) **dec++** demonstrates similar divergence to **rand** and **tgreedy** at much reduced complexity. (bottom right) Again, all strategies besides **adec** produce a similar log information volume. (Higher is better).

In all cases, the KLD measured between the estimate produced using the full data and that using the pruned data generally increases at each step of the simulation. For a given pruning rate $r$, the **rand** and **tgreedy** strategies both show slow, linear divergence growth over time. The others, particularly **ndec**, have a tendency to jump between different local minima, causing some fluctuation over time. The subtle modifications made in dec++ suffice to stabilize these fluctuations and generally reduce divergence, making it comparable with **rand** and **tgreedy**. **adec** shows the worst estimation performance with respect to the full set of variables. For the $r = 6$ case, as seen in Figure 3-18, the adec6 simulation diverges significantly around iteration 430, and does not recover.

The log information volume for each strategy is also tracked. Defined as the determinant of the information matrix, the information volume is the inverse of the uncertainty volume. Thus, increasing information volume corresponds to increasing estimate confidence. As can be seen, dec++ improves upon ndec to achieve comparable performance with rand and tgreedy.

Recall that tgreedy is generally expensive to implement for large graphs, and is therefore impractical for realtime systems. In general, tgreedy is expected to produce structure with $C$ noticeably larger than that of decimated graphs, as the algorithm applies no concept of sparse structure. However, Figure 3-17 shows that for this SLAM instance in particular, tgreedy at points produces complexity comparable to ndec and dec++. This may simply be an artifact of chance, as Figure 3-18 shows the expected larger complexity of tgreedy at a higher pruning rate.

In contrast to tgreedy, the random pruning strategy, rand, is certainly implementable in realtime, and is observed to produce good estimation performance. However, its inherent randomness destroys the particular macro-sparsity achieved by decimation, resulting in more computationally complex graph structures.

A more direct comparison is shown in Figure 3-19. In these plots, dec++2 manages to maintain more factors at lower complexity, and achieves lower divergence and higher information volume. This suggests that the inherent sparsity produced by decimation-based strategies allow them to prune *less* to achieve the same computation savings, and in the process maintain a better approximation of the original distribution.

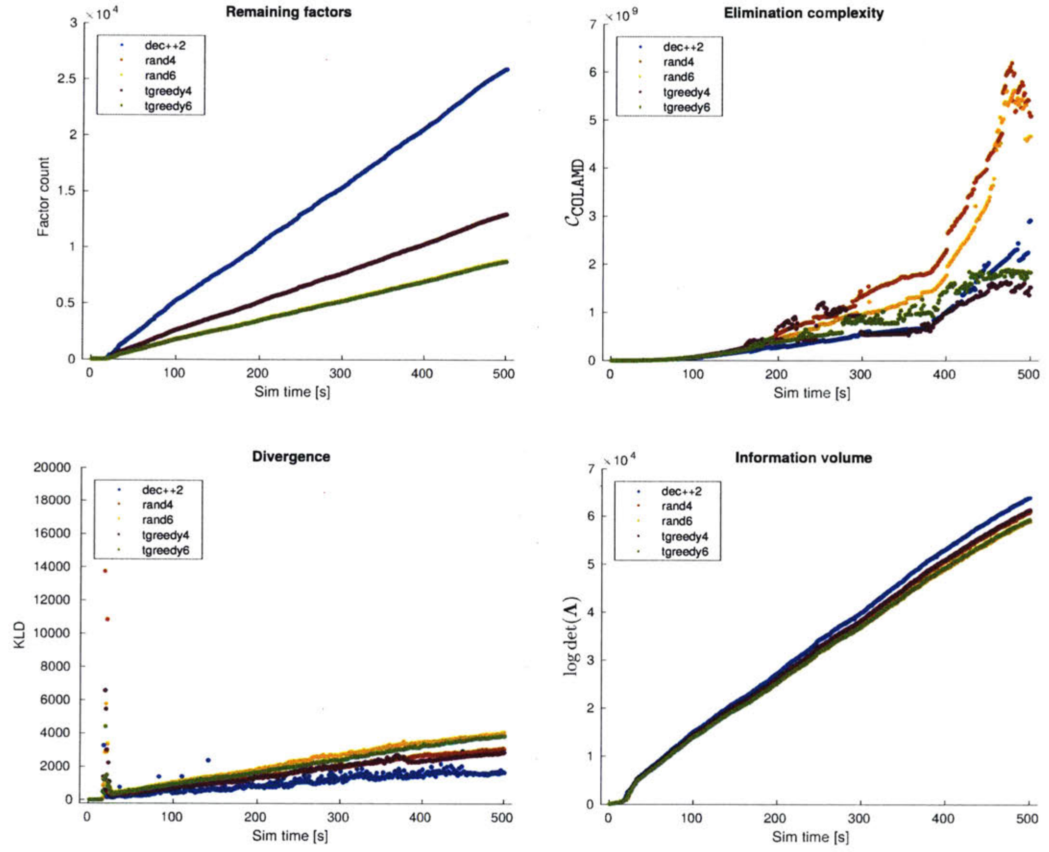


Figure 3-19: Direct comparison of incremental results across $r$ values. (top left) Unsurprisingly `dec++2` maintains double or triple the number of observation factors as the other strategies. (top right) Despite maintaining more factors, `dec++2` demonstrates a comparable or lower complexity than the competitors. (bottom row) `dec++2` performs best in both divergence and information-volume metrics.

## 3.5 Discussion

Despite their simplicity, decimation-style pruning strategies are highly effective. Decimation produces a characteristic graph structure that is inherently sparse, with bounded elimination complexity. In practice, this allows decimated graphs to maintain many more edges at lower computational cost than approaches without this structure.

Additionally, the uniform placement of pose-landmark edges incurred by decimation has good $t$-connectivity characteristics, corresponding to low uncertainty in the Gaussian-approximated distribution. In the single-landmark case, this structure is in fact near $t_w$-optimal. In general problems, this means decimation strategies can perform comparably with sophisticated $t$-maximizing strategies for similar graph complexity, while being significantly simpler and less expensive to implement.

The decimation-style strategy `dec++` was introduced to improve on a pure application of non-aligned decimation. Using simulation data, this policy was shown to achieve comparable or better estimation while maintaining comparable or better sparsity. Given the simplicity of its implementation, this makes it a formidable pruning option for realtime systems.

Because they are able to maintain more edges for similar computational cost, decimated graphs can achieve a higher average degree. This is known from [47] and [35] to correspond with better expected ML accuracy. This could partially explain the strong KLD approximation performance.

Additionally, it was observed that *random* selection of new edges produced estimation performance on par with the `tgreedy` approach proposed by [36]. However, random pruning lacks any of the partitioning superstructure of decimation, and so produces graphs that are much more expensive to optimize.

### 3.5.1 Auxiliary advantages of decimation

The decimation procedure is negligible computationally, and it can be known *a priori* whether a measurement will be kept or discarded. From an incremental, realtime approach, this has several advantages. In the case of aligned decimation, pre-processing can be reduced or skipped completely for intermediate poses, as it is known that no produced measurements will be kept. For visual systems, this is particularly useful as feature detection and tracking can be expensive operations. Unlike simple decimation, heuristic methods proposed by [58, 61] still require feature detection and tracking to be performed. Additionally, if estimates for intermediate pose are not needed for output, they never need to be added to graph in the first place. Besides simply reducing the number of variables in the optimization, this can also have accuracy benefits. As argued by Ila et al. [29], each intermediate pose variable represents a linearizing approximation (and a first-order noise propagation), so “over-sampling” the trajectory can lead to estimator overconfidence. Thus, if intermediate poses are not included, linearization is applied less frequently (in terms of the number of trajectory samples per time).

If intermediate frames are simply not represented in the aligned-decimated graph (for example in Figure 3-2), the resulting reduced graph can be considered a non-decimated graph with fewer pose nodes. This is essentially the application of decimation to keyposing. In this case, one might define metrics like KLD or information volume over only the variables in this reduced graph, a subtle yet important modification. Though beyond the scope of this thesis, the effectiveness of decimation for keyposing could be the subject of experiments similar to those in the previous section, but using these modified metrics.

For incremental solvers, it is also desirable if measurements can be removed “up front” rather than being pruned in a delayed sense. In iSAM2, removal of measurement factors from arbitrary locations in the graph degrades the incremental assumptions of the algorithm, and may result in extra computation. If measurements are instead discarded immediately, as is the case with decimation, or shortly after being added, as in the case of `dec++`, non-local modification of the iSAM2 Bayes Tree [32] can be avoided.

As was also seen, aligned decimation produces much sparser graphs than non-aligned decimation. However, non-aligned decimation provides a better approximation. Additionally, non-aligned strategies are more flexible, as new landmarks can be detected and initialized at a higher frequency. This can be important for fast-moving robots that undergo fast rotations or have limited field-of-view. As landmarks may be observable for only a short time, it is important that new landmarks can be detected and added to graph at a high enough frequency to replace those that disappear.

Put in terms of VIN systems, a combined strategy would apply both per-frame and per-landmark decimation. For example, imagine the camera generates image frames at 60 Hz. The frames could be decimated by 6, meaning poses are only represented in the graph at 10 Hz. Then, to achieve further computational savings, a per-landmark (non-aligned) decimation rate of 2 could be applied, so each landmark only generates observations at an effective rate of 5 Hz.

This graph reduction can allow the estimator to run on systems with very limited computation, or can allow for more landmarks to be estimated simultaneously. In Chapter 4, the `dec++` policy is implemented in SAMWISE and shown to effectively reduce computation while maintaining good estimation characteristics.

### 3.5.2 Can decimation be beaten?

In this thesis, decimation is analyzed from a graph-theoretic perspective, where all observations are assumed to be of equal precision. Of course, for most real systems including VIN, the noise characteristics of an observation can depend upon scene geometry, sensor parameters, the robot's trajectory, and countless other factors [45]. These factors are domain-specific, and taking them into account would somewhat limit the generality of the analysis. With the additional information available at run-time, a smarter strategy could no doubt be imagined, perhaps garnering significant performance improvements.

From one perspective, decimation can be considered an *a priori* measurement selection technique that prescribes a graph super-structure regardless of the realized measurements or trajectory. The resulting macro-sparsity is most valuable in "worst-case" graphs in which all landmarks are observed from all poses in the trajectory. However, in any particular SLAM instance, sensor limitations severely restrict the number of landmarks feasible from any given pose, providing another level of sparsity. This micro-sparsity is unpredictable, but can be as, or more, significant than the macro-sparsity achievable by decimation. Thus, *runtime* measurement selection policies that take into account realized graph structure at each step can potentially achieve much better complexity reduction in practice.

Nonetheless, decimation shines most in its simplicity. If the ultimate goal of measurement selection is computation reduction, any worthwhile strategy must net computational savings. This sets a high bar for any strategy that aims to meaningfully improve upon decimation.

### 3.5.3 Handling loop closures

Loop closures, a conspicuous feature of most long-term SLAM solutions, were explicitly excluded from much of the analysis in this chapter. Because they represent a significant generalization of graph structure, their presence makes analysis more complex.

In terms of graph sparsity, the presence of arbitrary loop closure constraints can annihilate the partitioning super-structure that allows efficient optimization. Fortunately, with some slight restriction, loop closures can be accommodated without ill effect.

For example, in the case of an aligned-decimated graph, loop closures from keypose to keypose will not affect the optimization upper-bound. If intermediate frames are not represented in the first place, this is accomplished naturally. In the case of non-aligned decimation, if loop closures only connect nodes within the same $\Pi_k$ cluster, the analysis will not be affected. In many systems, pose nodes are generated several times per second, so finding a "same-offset" node to connect a loop closure should not be that difficult.

## 3.6 Summary

Decimation is a very simple, intuitive approach to measurement selection used commonly in practice. To the best of the author's knowledge, its performance has thus far not been analyzed in a rigorous analytic sense.

Under such analysis, decimation proves to have highly desirable characteristics for computationally-constrained SLAM. Both aligned and non-aligned topologies produce super-structures which are relatively inexpensive to eliminate. This means that compared to more general graphs, decimated graphs can afford to incorporate many more measurements (i.e. edges) for the same computational complexity. Additionally, the uniform spacing of measurements promoted by decimation tends to promote a high weighted number of spanning trees, and is in fact near $t_w$-optimal in single landmark systems. As the weighted number of spanning trees has been related to reduced uncertainty volume, this suggests good information-theoretic performance. Ultimately, the combination of these two factors make decimated graphs highly efficient, high quality approximations of the original (full) graph.

The modified decimation-style approach presented in Section 3.3.2 was shown to be comparable with, and often outperform, the more sophisticated (and computationally infeasible) graph pruning approach of [35]. As `dec++` is trivial to implement and evaluate, it can be an effective computation reduction strategy for realtime landmark-SLAM systems, particularly in VIN.

Decimation was compared side-by-side with a more sophisticated approach [36] that was much more expensive to evaluate at each step. In these simulation experiments, decimation produced comparable or better approximation performance for the same graph complexity. Considering that decimation-style policies consume negligible computation and convey the auxiliary benefits discussed in Section 3.5.1, the results presented in this chapter demonstrate a clear suitability for high-rate, computation-constrained realtime systems.

# Chapter 4

# The SAMWISE Navigation System

Smoothing And Mapping With Integrated State Estimation (SAMWISE) is a multi-sensor state-estimation library for mobile robots developed by Draper. Primarily developed for the DARPA Fast Lightweight Autonomy (FLA) program [49], it is designed to enable agile, closed-loop flight for small UAVs in GPS-denied environments with a limited computational budget. An earlier version of SAMWISE was published in [59], but since then the system has matured considerably, and new results are presented here.

Though with a heavy emphasis on monocular visual-inertial sensing modalities, SAMWISE can incorporate (and benefit from) many other types of sensors, such as laser altimeters, 2D LIDAR scan-matching, and GPS if available. Under the hood, SAMWISE leverages the GTSAM [12] implementation of the efficient incremental iSAM2 [32] solver.

While sharing many similarities with other indirect visual-inertial navigation (VIN) approaches, SAMWISE includes several critical design innovations to meet the needs of low-latency, high-accuracy navigation for aerial vehicles. Because aerial vehicles like quadrotors are inherently unstable, they require high-rate, accurate state estimates to achieve stability and good reference tracking performance. However, attaining maximal accuracy requires iterative, non-linear solvers which can take up to a second to converge. Though iSAM2 updates are generally fast, completing in under 100ms, they can occasionally take much longer. In order to provide guaranteed state updates at IMU rate (> 100Hz) and still achieve maximal estimation accuracy, SAMWISE performs fast, IMU-based state prediction *decoupled* from the iterative smoothing optimization. This allows all active poses and landmarks to be optimized simultaneously at each solve step, facilitating high accuracy, robust estimation, while still allowing high-rate control.

In the GPS-denied context, SAMWISE is primarily an *odometry* system. Unlike general SLAM approaches, it does not facilitate long-term loop closures or aim to build a *globally*-consistent map. This is primarily because reliable place-recognition is still a challenging, open problem, and a single incorrect loop-closure constraint can severely impact estimate quality, potentially even causing divergence. As SAMWISE is designed to produce state estimates for low-level control, such a failure could lead to a loss of stability and catastrophic failure. Additionally, facilitating global loop closures is generally costly, requiring some representation of previously-visited locations to be maintained in the system. If place-recognition is available, the application of loop-closures can handled at a higher level of the system, without over-complicating the low-level estimator or exposing it to faulty recognitions. For all these reasons, SAMWISE makes no pretense of supporting global loop closures, and will demonstrate (small) estimation drift over time, even in loopy trajectories.

### 4.0.1 Design requirements

Several key requirements must be met for a robust, GPS-denied navigation system on a fast-moving quadrotor:

- Combine data from a number of possible sensor classes, notably IMU and monocular vision.
- Be robust to non-synchronized sensor streams with variable communication delays.

- High rate ($\geq$ 100 Hz), low latency ($\leq$ 1 ms) attitude and position output for closed-loop stability and control.
- Low-drift estimation (i.e. $\leq 2\%$ of distance traveled).
- Remain within a limited computation budget amenable to fully-onboard processing on SWaP-limited platforms.
- Operate robustly under widely-varying lighting conditions and environments, indoors and outdoors.
- Robustly handle outlier visual measurements, such as glare and moving (dynamic) objects in the scene.
- Large estimate corrections must be addressed to facilitate smooth trajectory tracking.

These challenges will be addressed in the following section. Results from several challenging DARPA FLA trials are presented, demonstrating the robustness and effectiveness of SAMWISE in closed-loop. SAMWISE is also evaluated on the EuRoC MAV [5] dataset and compared against ground truth.

## 4.1 Approach

The overall system architecture of SAMWISE is illustrated in Figure 4-1. As an indirect visual method, SAMWISE operates on a set of sparse feature detections extracted from each image frame in the video stream. Spurious tracks and other "outliers" are detected and discarded by triangulation and comparison to the current smoothed estimate of the landmarks and trajectory, as described in Section 4.1.6.

Measurements from the IMU and other sensors, such as a barometer or laser altimeter, form parallel sensor streams which are debuffered and converted in factors. This process is described in Section 4.1.4, and is flexible to measurements which are slightly delayed, dropped, or are otherwise asynchronous.

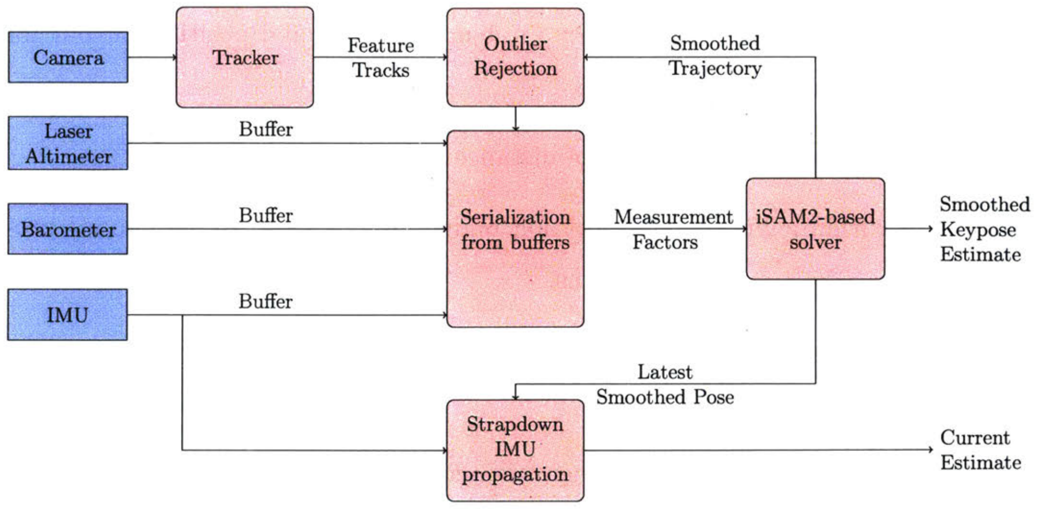


Figure 4-1: High-level SAMWISE system diagram.

In order to provide high-rate, low-latency estimates of the current state, as decoupled IMU-based strapdown propagator runs in a separate thread. After each incremental update performed by the smoothing optimization, this module is updated with the latest smoothed pose estimate, and IMU data from the timestamp of that pose onwards is seamlessly re-propagated. More details on this module can be found in Section 4.1.3.

SAMWISE estimates the (recent) state trajectory of the robot, along with the positions of visual landmarks, camera calibrations, IMU biases, and other supporting variables. To facilitate representation in a graph structure, it approximates the continuous-time trajectory $x(t)$ with a series of discrete variables $\{x_{i-n}, x_{i-n+1}, \ldots, x_i\}$. Each state variable $x_i \in \mathrm{SE}(3) \times \mathbb{R}^3$ represents a robot pose and velocity, and describe the vehicle state at the instant $t_i$.

All estimated variables are represented as the nodes of a factor graph, and nonlinear measurements as the factor edges connecting them. Unlike a traditional filtering-based approach, measurements are fully retained in memory until they are removed or marginalized. This allows them to be re-linearized as needed, granting higher accuracy. It also makes it easy to insert and remove measurements to/from the graph as desired, granting greater flexibility.

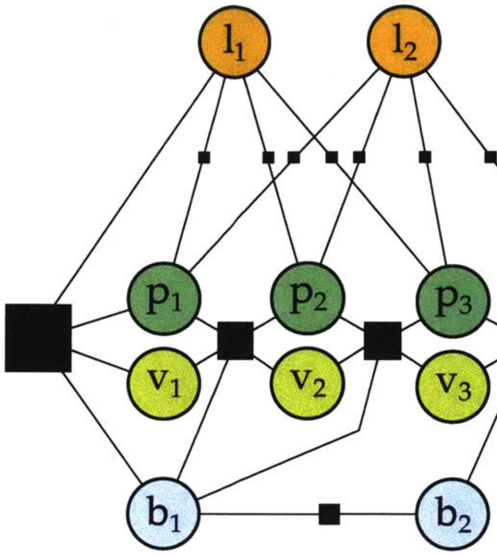


Figure 4-2: Architecture of SAMWISE factor graph. Each state variable is a pairing of a pose $p_i$ and velocity $v_i$. Time-varying IMU biases are represented by periodic $b_j$ nodes. Consecutive states are connected via IMU factors, which also depend on biases. Landmarks $l_k$ are connected to poses from which they are observed via observation factors, which may also depend on estimates of calibration parameters (not shown). The dense prior induced via sliding-window marginalization is shown as the large factor adjacent to $p_1$, $v_1$, $b_1$, and $l_1$.

### 4.1.1 The IMU as a motion predictor

Body-fixed IMUs are ubiquitous sensors in mobile systems, and for good reason. They can be very inexpensive and low SWaP, and exist in most consumer mobile phones. IMUs generally operate at much higher rates than many other sensors, up to hundreds of Hertz. They also provide a full-rank 6-DoF measurement, sufficient to predict the affine transform between two rigid-body poses in SE(3) separated in time. Additionally, in environments with a known, non-zero gravity vector, such as the surface of the earth, the IMU grants direct observability of the pitch and roll orientation of the body-frame. This makes the IMU an ideal candidate for a motion predictor in a state estimation system such as SAMWISE.

The IMU directly measures rotation rates, and linear acceleration, both affected by additive, time-varying bias terms. These bias terms, as well as inertial-frame linear velocities, must be estimated as well. Because the accelerometer grants observability of the gravity vector, which is fixed in the navigation frame, the vehicle's pitch and roll are directly observable. However, like most odometry systems, absolute position and heading are unobservable, even after the incorporation of SLAM-style landmarks. This *gauge freedom* is addressed in SAMWISE by applying a strong prior to the initial pose, positioning it at the origin with zero heading.

Following the methodology and implementation of Forster et al. [20], many consecutive IMU measurements can be "bundled" or *pre-integrated* to concisely express the predicted relative transform between pose variables $x_i$ and $x_{i+1}$ separated by an arbitrary time difference $\mathrm{d}t$. This pre-integration is performed directly in SE(3), and is formulated to allow dynamic re-estimation of the IMU bias terms without costly re-integration. These IMU bundles comprise odometry factors directly connecting consecutive state variables $x_i$ and $x_{i+1}$, as well as the current bias node $b_j$. This is desirable in that it eliminates the need to introduce a state variable $x_i$ for each IMU reading, making the factor graph much more concise, and the resulting optimization faster. Additionally, it avoids unnecessary intermediate application of first-order noise propagation (i.e. linearization) to the dynamics, promoting consistent noise estimation.

### 4.1.2 Monocular vision

The use of bi- or trinocular camera rigs with overlapping fields-of-view for vision-aided navigation is quite common [5,23,55,57], as it confers several benefits in terms of observability and robustness. Because the same landmark can be observed simultaneously by multiple cameras, the landmark can be triangulated immediately, and the posterior distribution over the landmark position approaches Gaussianity faster [45]. Additionally, the baseline between cameras is fixed and known. Compared to a series of monocular observations separated by time (and an *unknown* baseline which is estimated simultaneously), the known baseline between paired stereo observations grants direct observability of metric scale and better estimation accuracy. Furthermore, monocular triangulation fails when the landmark lies near the optical center and motion is along the optical axis, and stereo rigs do not [57].

Nonetheless, the value of stereo or trinocular measurements is heavily dependent on the physical separation of the cameras. For very small platforms, providing significant baseline can be prohibitive. Thus, despite the inherent challenges, monocular estimation provides the most universal solution.

While it would be straightforward to implement stereo matching within SAMWISE, it currently is designed to leverage only monocular observations. With that in mind, all experiments discussed in this chapter are performed using a single camera, even if others are available as in the case of the EuRoC MAV dataset [5].

### 4.1.3 Decoupled IMU propagation

Because SAMWISE is designed to facilitate closed-loop control and trajectory tracking, it must be able to output updated state estimates at a rate high enough to provide adequate control bandwidth. Additionally, these estimates must have low *latency*, defined as the time between when the update is published (produced) and when it was valid (when the corresponding sensor data was measured). However, even state-of-the-art smoothers such as iSAM2 cannot compute updates fast enough, and do not necessarily hold to constant compute time. For this reason, SAMWISE runs a separate IMU propagation thread which is decoupled from the graph optimization.

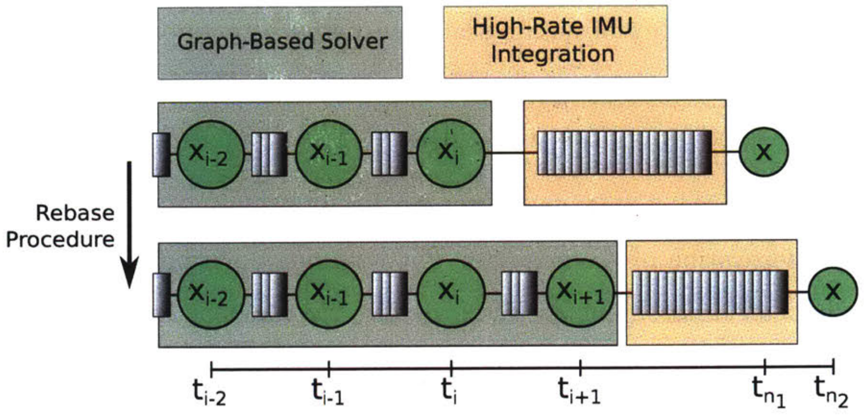


Figure 4-3: In order to facilitate high-rate state estimation independent of the iterative solve thread, SAMWISE runs a decoupled IMU propagator which allows access to the latest pose estimate $\mathbf{x}$ at any time. New, high-rate IMU measurements are integrated sequentially starting from the latest smoothed pose $\mathbf{x}_i$. After the graph-based optimizer completes a new solve, the propagator is *rebased* onto the new latest smoothed pose $\mathbf{x}_{i+1}$. In the process of rebasing, IMU measurements after $t_i$ must be re-integrated. This process takes some amount of time, indicated here by the additional IMU measurements incorporated from $t_{n_1}$ to $t_{n_2}$ during the rebase. In implementation, the process of rebasing is handled carefully to ensure that the current estimate $\mathbf{x}$ can be accessed at any time, even while a rebase is occurring.

When an IMU measurement is received, it is added to the IMU queue to be eventually incorporated into the factor graph and to a separate IMU propagator module. Within the IMU propagator, the measurement is stored within a circular buffer describing the IMU history since the last *base* state $\mathbf{x}_i$ (at time $t_i$, with bias estimate $\mathbf{b}_i$). When a new set of measurements is incorporated and solved in the solve thread, the propagator is *rebased* onto the latest state $\mathbf{x}_{i+1}$ at time $t_{i+1}$ and bias estimate $\mathbf{b}_{i+1}$. During the rebase, measurements older than $t_{i+1}$ are discarded from the propagator buffer, and the state history from $t_{i+1}$ onwards is recomputed. As this process takes non-negligible time, IMU measurements continue to accumulate in the buffer. This process is illustrated in Figure 4-3.

At any time (including while a rebase is occurring), the IMU propagator can be queried in constant-time for a state prediction for any time in the range $[t_i, t_{n_1}]$ (before the rebase), or $[t_{i+1}, t_{n_2}]$ (after the rebase). The output of these queries is a pose prediction that combines the latest smoothed estimate with the latest IMU data, interpreted via the current bias estimate. Thus, as long as the graph-based solver achieves a sufficiently-high update rate, the propagation period will not grow too long. The quality of the IMU and the dynamics of the vehicle will determine for how long measurements can be propagated without accumulating dangerous levels of error. In practice, using a mid-range ADIS16448 IMU (also used in the EuRoC MAV dataset [5]), the FLA quadrotor performs well and maintains stability even with occasional several-second solves.

### 4.1.4 Measurement buffering

In order to build the optimization factor graph illustrated in Figure 4-2, pose and velocity nodes must be created to represent the robot's state at the corresponding measurement times. Said another way, a non-IMU measurement taken at time $t_j$ must be connected to the pose node $x_j$. If this node has not already been created, it must be added to the graph, and connected to the previous node $x_{j-1}$ via an odometry factor. As described in Section 4.1.1, the pre-integrated IMU factors of [20] are used to represent this odometry, and are formed by pre-integrating the IMU readings in the interval $[t_{j-1}, t_j]$.

Clearly, the above process require pose nodes creation, and therefore measurement processing, to be performed in synchronized order. Nevertheless, it is difficult to time synchronize the many sensors that may be leveraged in a navigation system. Sensors operate at different frequencies and may even use different clocks. Additionally, communication buses like USB may be shared among multiple sensors and other devices, causing unpredictable communication delays. For these reasons, SAMWISE implements a flexible measurement buffering scheme which allows for the incorporation of delayed and non-synchronized sensor streams. It assumes only that the measurements from each individual sensor are ordered properly, and allows for the small communication backups that may occur on, for example, a busy USB bus.

Each sensor is associated with an ordered stream of time-stamped measurements. When new measurements are added to SAMWISE, they are added to a corresponding internal buffering queue. Before processing any measurements, SAMWISE waits until at least one measurement is present in each enabled queue. This ensures that measurements, which may be received out-of-order in relation to measurements from different streams, are still processed in the correct overall order.

The above scheme can easily handle delayed measurements. For example, vision measurements require extensive pre-processing and screening before they can be added to the graph. By reserving a "ghost" state $x_f$ corresponding to the time $t_f$ a frame is captured, the corresponding vision measurements can be added to the graph later, when they are done processing. This allows normal processing to continue uninterrupted in spite of delays, and delayed information can be used as it becomes available.

### 4.1.5 Sliding-window marginalization

In order to limit graph growth over time, a time-parameterized sliding-window is maintained. Poses which are older than a fixed duration are marked inactive, as are landmark variables which are adjacent to only inactive poses. Time-varying bias, calibration, and other auxiliary variables become inactive in a similar way. Inactive variables and their corresponding measurements are marginalized into a prior during the solve step. This keeps computation within reasonable limits and allows for long-duration flights. As SAMWISE is not designed to accommodate long-term loop closures, inactive states are simply discarded.

When variables are marginalized out, the corresponding (adjacent) measurement factors are removed from the graph, and replaced by a dense linear marginal connecting all adjacent remaining variables. Statistically, this is equivalent to the dense, correlated marginals described as Generic Linear Constraints in [6], and the sliding-window approach of [57]. However, we can leverage the iSAM2 Bayes Tree to more efficiently compute this marginal factor, in a way identical to [8]. By pre-ordering the variables we wish to marginalize such as to eliminate them first, they form a set of subtrees in the Bayes Tree. The cached elimination result of each of these subtrees is the corresponding linear marginal, and the subtree (made of variables to be marginalized) can be simply removed from the tree.

Before a variable can be safely marginalized, its corresponding estimate must be assumed to have converged to be near its true value. Once a variable and its related measurements are marginalized out, the corresponding information can no longer be re-linearized, and any linearization error becomes baked into the corresponding marginal factor. As was pointed out by [15], the linearization points of the remaining variables *adjacent* to the marginalized set must *also* have their linearization points locked in order to prevent inconsistency. Thus, it is important that variables be kept active (and not marginalized) until they, and their neighboring variables, have had time to converge reasonably well.

In practice, we have found a window length of only 3 seconds to provide a sufficient compromise between computation time and accuracy for our application. By that time, the vast majority of landmark estimates have converged to stable values, and further re-linearization is not required.

### 4.1.6 Feature track handling

As an indirect visual method, SAMWISE first extracts keypoints, or *features*, from the image and then incorporates these observations as noisy, geometric constraints on the vehicle pose over time. Features correspond to high-gradient regions of the image, and are tracked across multiple successive frames producing a feature *track* of all associated observations. Each of these observations corresponds to a pixel coordinate in the image, $u \in \Omega^2$.

It is assumed that each track corresponds to a well-defined, static landmark point in $\mathbb{R}^3$. Using a projective pinhole-camera model (see Section 1.2.1, each feature observation describes a probabilistic geometric constraint between a vehicle pose and the landmark position. In SAMWISE, each landmark is represented in the graph with its own variable node, and each image frame corresponds to a pose node. Thus, a track defines a set of these constraints relating a particular landmark to the set of poses from which it was observed.

Feature tracking is performed with a relatively unsophisticated but computationally inexpensive Lucas Kanade tracker [40]. While being significantly less computationally expensive compared to more sophisticated descriptors such as ORB [54] or SURF [2], it is more prone to unpredictable, incorrect matches. Besides outright matching failures, features with more ambiguous or weaker gradient in one or more directions can cause noisier tracking. Because features are matched between consecutive frames rather than by comparing to the first observation, *feature drift* can also occur, as small differences in the tracked feature compound over time. Additionally, in dynamic environments, the static landmark assumption may be violated, and the corresponding tracks will be inconsistent with ego-motion alone. If included in the optimization, these outliers can be very damaging and cause catastrophic divergence.

In SAMWISE, outlier detection and mitigation is handled in two ways. In order to avoid incorporating bad tracks, observations along a track are pre-triangulated to produce a preliminary landmark estimate, using the currently-available trajectory estimate. If this triangulation fails to produce a reasonable landmark estimate, this suggests a corrupted track. On the other hand, if a reasonable landmark estimate is produced, a reprojection test is applied to each observation individually. If the reprojection error falls below a fixed threshold, the observation is considered an inlier. Assuming sufficient inliers are found, the landmark and its inlier measurements are introduced into the estimator. However, if too many consecutive observations fail this test, the track is assumed to have been corrupted.

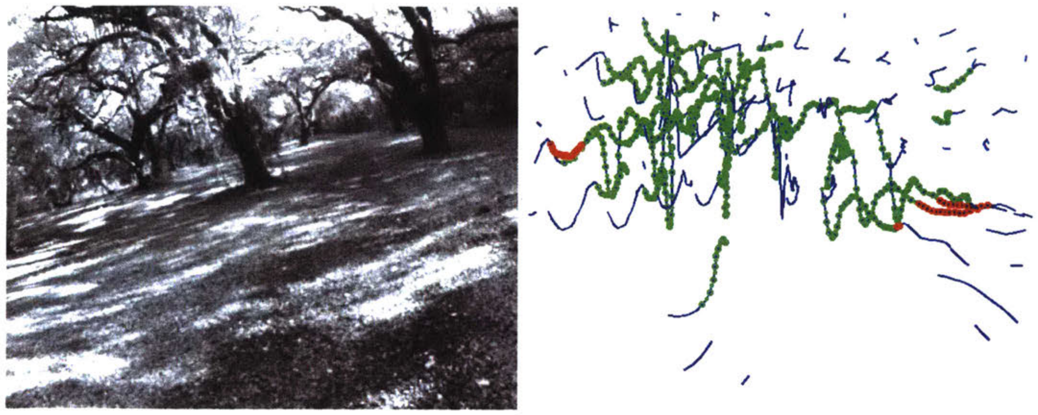


Figure 4-4: SAMWISE feature tracking during outdoor evaluation. (Left) Input monochrome image. (Right) SAMWISE feature tracks. Each KLT track is shown with a blue trace, indicating the feature position in the recent history of frames. When a track reaches a minimum length, the landmark position is triangulated. Each detection is subject to a reprojection test (using the current trajectory estimate), and if the error falls outside of a fixed window, it is marked as an outlier (red). Detections whose error falls within a narrower window are marked as green, and added to the graph as measurement factors.

A key distinction of this method is that each track is essentially compared against the current estimate, rather than against the other tracks via a RANSAC-based [19] consensus determination. Assuming the current trajectory estimate is close to the true trajectory, this method is effective and efficient. Additionally, large dynamic objects in the scene may produce a self-consistent set of observations which are still "bad" observations, in the sense that the do not correspond to static landmarks. This case is correctly rejected by this scheme, but may not be by a pure consensus approach. Furthermore, by evaluating observations individually, the set of good measurements from a track can still be used even if the detector at some point "jumps" to a different feature or has a bad match at any point.

In order to further mitigate outliers which may slip through, visual observations are modeled via a Huber cost function rather than a conventional squared error. The Huber error grows only *linearly* for large error, so inconsistent observations do not exert undue influence over the full nonlinear cost function.

Because observations are processed and added to the graph in a delayed fashion, landmarks are initialized with several observations at once. This reduces the need for complex landmark parameterizations such as inverse depth, advocated by Montiel et al. [45] and used extensively in filtering-based approaches. While more accurate under Gaussianity assumptions when only one or two measurements are available, such parameterizations converge to a Gaussian distribution over $\mathbb{R}^3$ quickly once more observations become available. In practice, they also seem more sensitive to linearization point, requiring more frequent re-linearization. When implemented, we did not notice any increased accuracy, but did see increased computation. For that reason, for SAMWISE a straightforward $\mathbb{R}^3$ landmark position representation is used.

### 4.1.7 Keypose estimation and output

Like all discrete-time estimators, the SAMWISE estimate is prone to discontinuous corrections as new sensor data is incorporated. In a closed-loop system where the estimate is used for aggressive control, such discontinuities can lead to twitchy, unstable control actions. Such jumps must be mitigated for system reliability and good performance.

As a smoother, SAMWISE provides explicit access to updated estimates of both the current *and* prior states. Often, when such discontinuities occur in the inertial-frame estimate of the current state, the *relative* transform between the current pose and recent previous poses remains relatively smooth. This is illustrated in Figure 4-5.

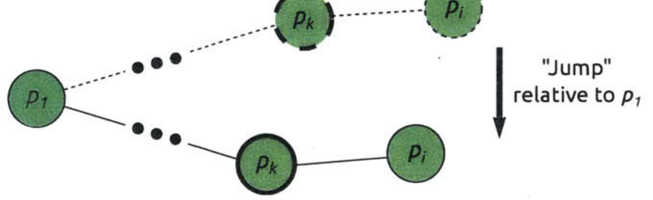


Figure 4-5: Example of an estimate correction causing a "jump" relative to the base pose $p_1$. In this case, both current pose $p_i$ and recent keypose $p_k$ are corrected together, and the *relative* transform is largely unaffected.

Thus, if trajectory tracking is done in the *local* frame defined by a recent *keypose*, most of these discontinuities are smoothed out. In many robotic systems, and the FLA system in particular, trajectory re-planning and generation is done frequently to account for new obstacle data. As obstacle and environment geometry detected by body-mounted sensors are naturally described in a local frame, the corresponding trajectory is naturally defined in a local frame as well. Thus, the tracking error used for control can be defined in the local frame in which the trajectory was planned, rather than in the fixed navigation frame. When an estimate correction occurs, the tracking error in this local frame will be unaffected, and control will remain smooth.

To enable such downstream behavior, SAMWISE publishes a set of recent keyposes after each solve alongside the updated current estimate. These keyposes generally correspond to the pose at the time of each image frame, but can be specified in any way, and have a specified lifetime. SAMWISE guarantees that that these keyposes will be updated and published for the duration of this lifetime. As long as this lifetime exceeds the replanning period of the downstream planner and controller, trajectories can be smoothly tracked in the local frame in which they were generated.

## 4.2 Evaluation on the EuRoC MAV Dataset

In order to better quantify estimator performance, SAMWISE was evaluated on a subset of the EuRoC MAV dataset [5] provided by ETH Zurich. This data was taken on a Micro-Aerial Vehicle (MAV) with a similar configuration to our FLA vehicles, combining an ADIS16448 IMU at 200Hz with forward-facing 20Hz stereo cameras. As SAMWISE currently only supports monocular vision (to support the Draper-MIT single-camera vehicle), only one of the two provided image streams was used.

The particular datasets used were taken from indoor flight in a motion-capture space and come with provided ground truth. Unlike the typical FLA challenges, these datasets do not cover long distances, but instead stress high angular rates in relatively confined environments. The large and frequent rotations meant landmarks are not observed for long before leaving the field of view. To compound this difficulty, the dataset only provides image data at 20 Hz (although the Aptina MT9V034 cameras [48] used were capable of providing a full 60 Hz). Because monocular estimation requires multiple sequential observations to reliably triangulate a landmark, this is especially challenging for monocular-only systems like SAMWISE.

As the provided camera calibration is quite high-fidelity, SAMWISE's online calibration functionality was disabled for these evaluations. Some iteration in determining reasonable initial estimates of the time-varying accelerometer biases was made. As these biases are largely a product of slight manufacturing variations and misalignment in IMU mounting, they are relatively stable for a particular vehicle over time, and the same values were used for both evaluations done here. In order to increase SAMWISE's ability to re-linearize landmark positions given the challenges imposed by this dataset, the sliding window length was increased to 8.0 seconds. This also had the benefit of better demonstrating the role of measurement selection, as demonstrated by the set of experiments in Section 4.2.2. The set of relevant parameter settings is shown in Table 4.1. Otherwise, no specific parameter optimization was done.

Table 4.1: SAMWISE parameter values used for the EuRoC dataset.

| **Parameter** | **Value** |
|---|---|
| Accel noise density | 1.6968e-4 [rad / s $\times$ s$^{\frac{1}{2}}$] |
| Gyro noise density | 2.000e-3 [m / s$^2$ $\times$ s$^{\frac{1}{2}}$] |
| Init accel bias | (-0.011, 0.133, 0.080) [m / s$^2$] |
| Init gyro bias | (0.0, 0.0, 0.0) [rad / s] |
| Feature noise sigma | 1.0 [px] |
| Sliding window | 8.0 [s] |
| Max features per frame | 70 |

The SAMWISE position estimate and VICON-provided ground truth were aligned by fixing the initial pose estimate to that reported by VICON. Without fixed references, the efficacy of SAMWISE was defined by the odometric drift which then accumulated over time.

In order to validate the improved `dec++` decimation-based policy introduced in Section

3.3, dec++ was implemented in SAMWISE. A suite of decimation rate settings are evaluated and compared in Section 4.2.2. For brevity, attitude and velocity results are omitted.

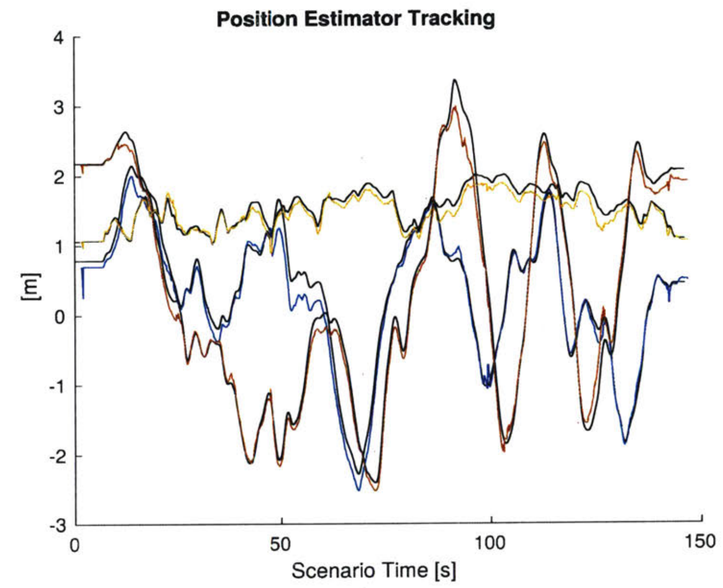
### 4.2.1 Results

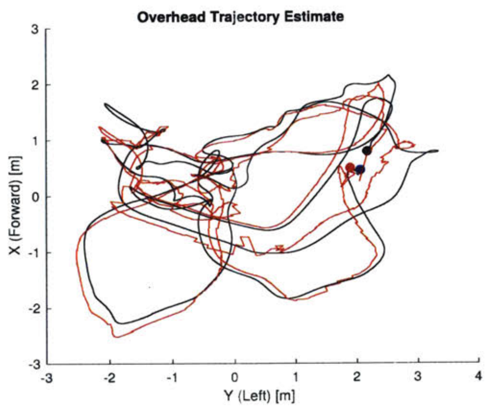
Despite the challenging qualities of this dataset, SAMWISE performs well. On the V1_01 "Easy" dataset, shown in Figure 4-6, SAMWISE achieved an overall position RMSE of 0.2609 m. In this dataset, the image tracker performed well and was able to track many landmarks for long durations, generating sufficient individual observations to overwhelm the solver. As will be shown in more detail in the next section, the result in Figure 4-6 was obtained using the dec++ policy with a rate of $r = 2$. To put this number in perspective, the cumulative distance traveled by the vehicle (estimated from the ground-truth VICON data) for the duration of the flight was 60.1 m.

SAMWISE was also evaluated in the on the aptly-named V1_03 "Difficult" dataset, which included much harsher motions. The tracker struggled to track well through harsh motions on the relatively low-rate 20 Hz video stream, and fewer observations were generated than for the V1_01 dataset. For this reason, no decimation was performed for the trial shown in Figure 4-7. As expected, slightly higher position RMSE of 0.5561 m was incurred. For comparison, a total of 75.1 m were traveled in total.

In these runs, the iSAM2-based solver only achieved rates of between 1-5 Hz. Nonetheless, because of the decoupled IMU propagation described in Section 4.1.3, SAMWISE still produced high-quality state estimates at the full IMU rate of 200 Hz. The results shown here are the realtime, high-rate outputs which would be available to low-level planner and controller in a closed-loop application.

As expected, in both runs the computation time of the iSAM2 update tends to follow the number of active factors.





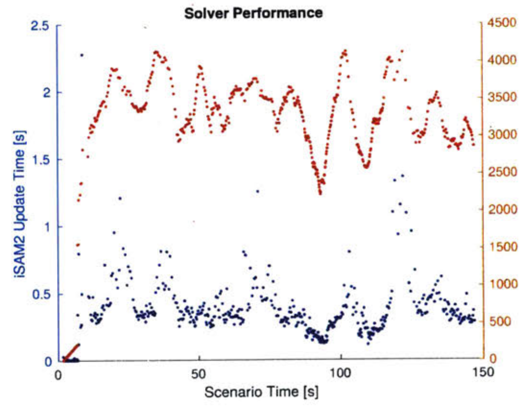
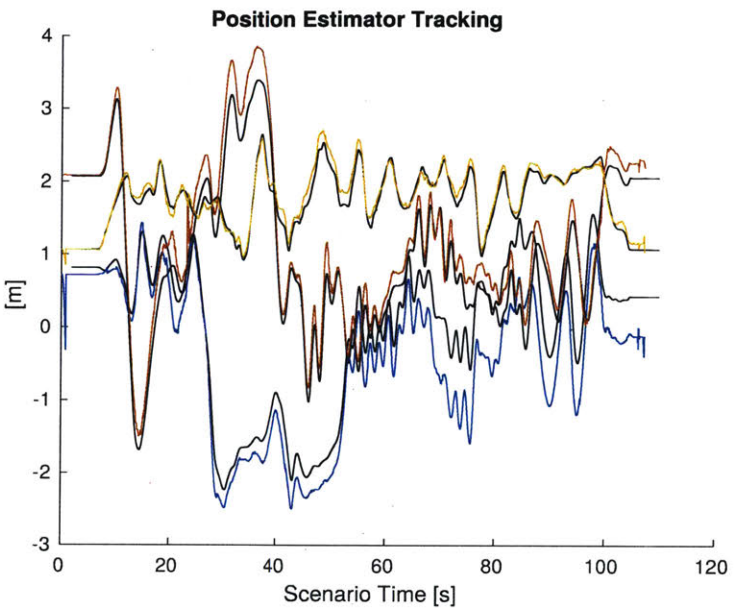


Figure 4-6: Real-time SAMWISE performance on the "Easy" dataset, using the `dec++` policy with $r = 2$. (top) Tracking performance in each position coordinate (truth in black). SAMWISE demonstrated good performance, achieving a position RMSE of 0.2609 m compared to VICON truth over the 60.1 m trajectory. (lower left) Overhead view of trajectory estimate (red) compared to truth (black). (lower right) iSAM2 update time plotted with active factor count.



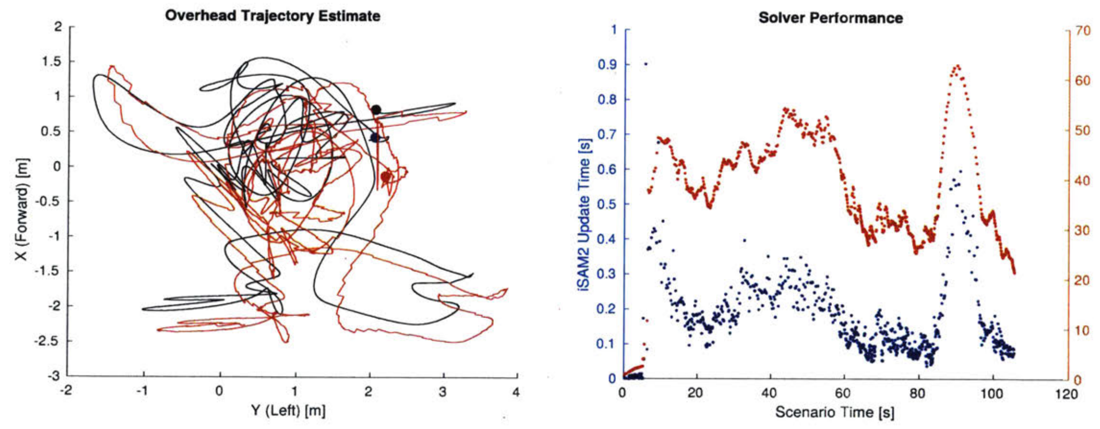


Figure 4-7: SAMWISE performance on the "Difficult" dataset (with no decimation). (top) Tracking performance in each position coordinate (truth in black). Despite the challenging maneuvers in this dataset, SAMWISE achieved an RMSE of 0.5561 m over a trajectory exceeding 75.0 m in length. (lower left) Overhead view of trajectory estimate (red) compared to truth (black). (lower right) iSAM2 update time plotted with active factor count.

### 4.2.2 Varying the decimation parameter

For the parameter set described in Table 4.1, the V1_01 dataset provided sufficient quantities of long landmark tracks to facilitate experimentation over varying levels of decimation. As can be seen in the rows corresponding to `full` in Table 4.2, the full data was sufficient to overwhelm the iSAM2 solver, resulting in long solve times, fewer total updates, and ultimately poor accuracy. Though the decoupled IMU propagation described in Section 4.1.3 means that state estimates are available at IMU rate irrespective of the update rate, beyond a certain point long update times are problematic for accuracy. Because new variables (poses and landmarks) added to the nonlinear optimization are initialized by the current estimate, the quality of these initializations depends in part on how much "raw" (un-smoothed) IMU propagation it is based on. Thus, long update times can lead to increasingly noisy initializations, which can can cause the the Gauss-Newton solver to converge to incorrect local minima. Ultimately this demonstrates a motivation for real-time measurement selection at the core of this thesis.

The remaining entries of Table 4.2 correspond to SAMWISE performance under various decimation levels. Because the quality of the estimate often degraded unpredictably during the landing maneuver of the last few seconds of the trajectory, and to make the statistics more representative of in-flight performance, only results within the time window of [4.0, 130.0] s were taken (out of a 146 second total scenario) are used. Additionally, SAMWISE is a multi-threaded library running image tracking, graph optimization, and IMU propagation in different threads. This makes performance on a particular run somewhat stochastic, and therefore two trials of each measurement policy are reported.

Table 4.2: Results of varying decimation rate on the EuRoC V1_01 dataset. In each case, two trials were run, and statistics were computed over a select time window [4.0, 130.0] s for which performance was most consistent. The number of factors incorporated without decimation shown in the row labeled `full` caused long update times and ultimately poor RMSE. The use of `dec++` decimation was effective in reducing update times, actually improving accuracy over the `full` case. As expected, starting from `dec++2`, increasing decimation rate results in increasing RMSE and decreasing estimator performance. Note that SAMWISE performance is somewhat stochastic, as can be seen in the RMSE differences in between trials of `dec++3` and `dec++5`.

| **Method** | **Trial** | **Avg. Factor Count** | **Avg. Update Time [s]** | **Updates Completed** | **RMSE [m]** |
|---|---|---|---|---|---|
| `full` | 1 | 4800 | 0.478 | 311 | 0.947 |
| | 2 | 4793 | 0.476 | 312 | 1.022 |
| `dec++2` | 1 | 2882 | 0.323 | 461 | **0.271** |
| | 2 | 2826 | 0.328 | 455 | **0.231** |
| `dec++3` | 1 | 2221 | 0.244 | 611 | 0.329 |
| | 2 | 2127 | 0.208 | 716 | 0.861* |
| `dec++4` | 1 | 1826 | 0.191 | 778 | 0.373 |
| | 2 | 1818 | 0.197 | 754 | 0.385 |
| `dec++5` | 1 | 1547 | **0.145** | 1020 | 1.250* |
| | 2 | 1489 | **0.164** | 903 | 0.555 |

## 4.3 Evaluation for FLA

The DARPA Fast Lightweight Autonomy (FLA) program is designed to drive algorithmic improvement for closed-loop, fully onboard autonomy for small, fast-flying quadrotor UAVs. Vehicles must complete missions spanning hundreds of meters of combined indoor and outdoor environments, with no access to GPS or a detailed prior map. Successful completion requires lightweight (i.e. computationally inexpensive) solutions in navigation, obstacle detection, and planning.

During the last three evaluations of Phase 1 of the program, SAMWISE provided a complete VIN state estimate and navigation solution for the Draper-MIT vehicle. The performance during the first of these, in November 2016, was discussed in [59]. Since then, the SAMWISE system has continued to improve and to be strenuously tested in challenging, real-world conditions. The most recent evaluation in May 2017 showed SAMWISE working very reliably. Estimate drift was consistently below 3%, and often below 1%, over trajectories spanning hundreds of meters.

### Sensor configuration

While the FLA program specified several aspects of the vehicle design, most notably the airframe and battery cell count, the sensor and computational payload design was left to the individual teams. The Draper-MIT entry to the competition evolved over time, with several main iterations. At the time of the May 2017 milestone, two main variants were in use, differing mainly in the onboard computer and the presence or absence of the 2-D scanning Hokuyo LIDAR. The differences were mainly driven by the needs of two different obstacle perception and mapping approaches, and the state estimation configuration used by SAMWISE was identical in either case.

The sensors used by SAMWISE were consistent between variants, and are listed here

- IMU: ADIS 16448
- Camera: PointGrey Flea3
- Downward-pointing laser altimeter: Garmin LidarLite
- Barometer: MEAS MS5611

Computationally, the system was restricted to an Intel NUC i7 or Intel Skullcanyon compact computer carried onboard the vehicle, with a dual- or quad-core Intel i7 CPU respectively. The more powerful Skullcanyon system was developed to accommodate sophisticated vision-processing algorithms used for other aspects of the mission. In either case, the peak SAMWISE consumption was approximately one (physical) core for feature tracking and optimization.

### 4.3.1 Draper-MIT warehouse

A long-duration stress test was performed to evaluate the estimator (and whole-system) robustness. The vehicle was made to fly to a series of arbitrary clicked waypoints entered by an untrained user in a moderately-cluttered indoor warehouse environment in Charlestown, Massachusetts. The mission lasted a little over three minutes, corresponding to the full duration of the flight battery, and included several obstacle avoidance maneuvers.

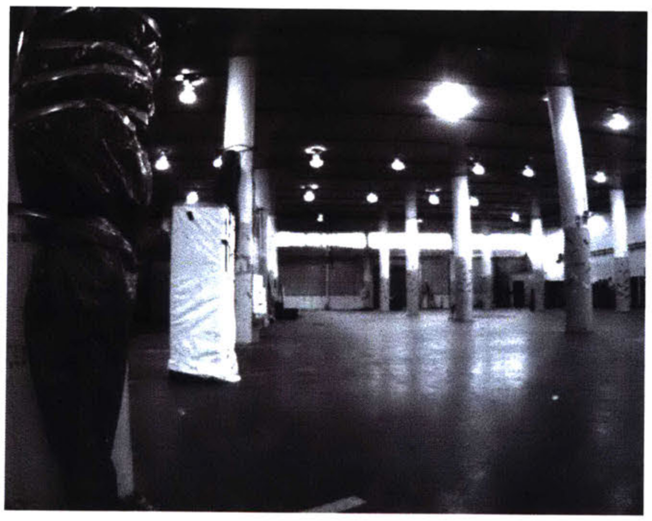


Figure 4-8: Sample image taken from the vehicle's forward-mounted camera at the Draper-MIT warehouse test site. Note that the camera is monochrome, as is the input image to SAMWISE.

No ground truth data was available, but the vehicle was ultimately commanded to land back at the takeoff location (the origin), and was observed to in fact land within 1m of that position. The landing logic is in fact quite loose, and from the estimate output it clearly landed knowing it was still about 1m short of the start location, as seen in 4-9. As the flight included over 250m of distance traveled, this corresponds to below 0.5% estimate drift.

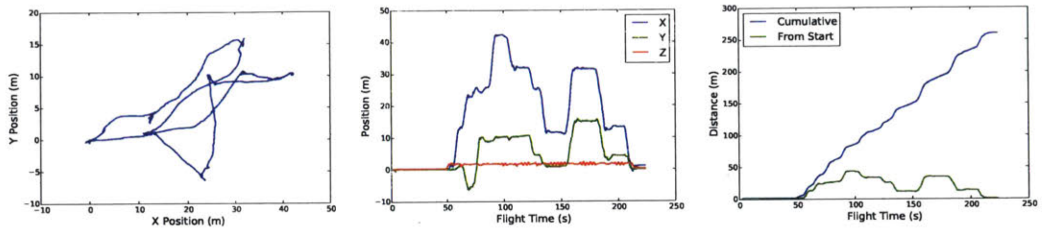


Figure 4-9: SAMWISE position estimation during warehouse stress test. Ground truth was unavailable, but the vehicle was observed to land within 1m of the true commanded location at the end of the flight. (Left) Overhead trajectory estimate showing the complex, user-defined maneuver. (Middle) Component-wise position estimates. As the vehicle was commanded to series of waypoints with some time in between, the roughly rectilinear traces are expected. (Right) Total distance traveled alongside distance from start. With over 250m traveled in total, an achievement of less than 1m final error corresponds to $< 0.5\%$ estimate drift.

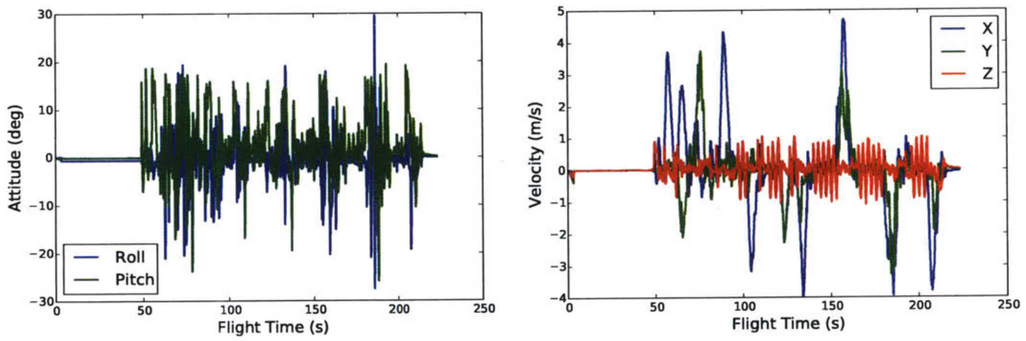


Figure 4-10: SAMWISE orientation (left) and velocity (right) estimates during warehouse stress test.

## 4.3.2 DARPA FLA results

The May 2017 DARPA FLA milestone stressed long outdoor traverses as well as indoor flight in a large warehouse. Flights took place during daylight hours, but lighting conditions varied significantly between early morning and afternoon. These lighting changes were exacerbated by the presence of forest canopy, indoor-outdoor transitions, and the need to fly directly into the sun at times. Nonetheless, SAMWISE enabled 67 flight attempts over the week with no state-estimation related failures.

As no ground truth is available (vehicles were not allowed to use or log GPS data), it is difficult to quantitatively evaluate the performance of SAMWISE for these flights. However, as all missions were essentially out-and-back, state estimation drift can be evaluated by how far away the vehicle landed from the start.

### 4.3.3 Forest flight

Part of the evaluation involved flight through a relatively dense forest, with a thick overhead canopy. Besides stressing the vehicle's autonomous obstacle avoidance system, the environment was challenging due to the varying lighting conditions involved. Additionally, as the obstacle avoidance system often introduced harsh maneuvers and rotations, SAMWISE did not have the luxury of many long feature tracks. Nonetheless, the estimator performed well throughout the day.

A selection of screenshots from the onboard Flea3 camera during one of the successful flights is shown in Figure 4-11. The maximum speed achieved during this flight was 8.8 m/s, which is significant considering the degree of clutter. The total flight covered over 250 m and lasted about 50 seconds, after which the system (as can be seen in the image sequence) landed within 2m of the start location. This corresponds to below 1% error. The realtime position estimate is shown in Figure 4-12.

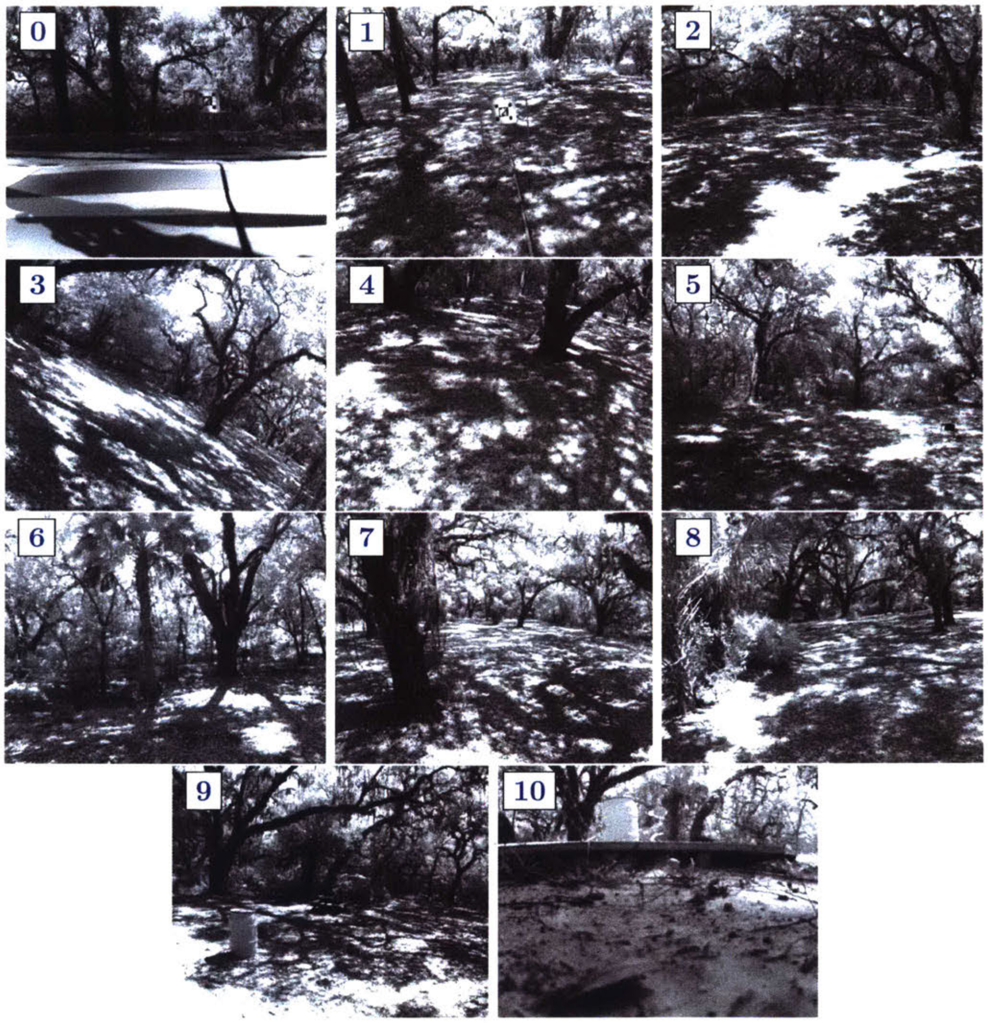


Figure 4-11: Sequence of images from the onboard Flea3 camera taken during an autonomous forest flight. Note the challenging lighting conditions imposed by the forest canopy and tree shadows. The first and last images are taken from the takeoff pad and the landing pad, respectively. Note that the last image clearly shows the takeoff pad immediately in front of the vehicle's final landing position, indicating that SAMWISE successfully navigated the system back to the start with little drift.

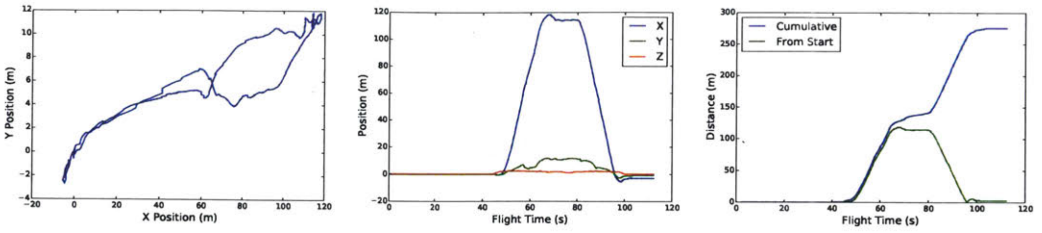


Figure 4-12: SAMWISE position estimation during forest flight. (Left) Overhead trajectory estimate. (Middle) Component-wise position estimates. (Right) Total distance traveled alongside distance from start.

### 4.3.4 Combined outdoor and indoor flight

One of the main challenges posed by the FLA program is the ability to traverse hundreds of meters through relatively open space outdoors and navigate through tighter indoor environments within the same mission. Besides preventing "fine-tuning" of system parameters to cater to a single environment, this is especially difficult for visual systems because of extreme lighting changes during indoor/outdoor transitions. Even if the camera exposure is automatically adjusted to accommodate these varying conditions, feature tracking be significantly impacted immediately after a transition occurs. This often results in the loss of all feature tracks during a harsh lighting change, which can significantly degrade estimator performance.

In these evaluations, the vehicle had to navigate to a large warehouse doorway, enter the building, and then navigate a short way inside to find the goal. Then, it had to turn around and find its way back to the start. While the outdoor portion of the flight was relatively uncluttered, except for a large stand of trees to the right, the warehouse environment was much more complex. Additionally, as can be seen from the images in Figure 4-13, the doorway presented a significant lighting transition, as the inside of the building was much darker than the outside. In combination with several extreme obstacle avoidance maneuvers, this caused feature tracking to be inconsistent and even interrupted entirely at several points during flight.

The flight shown in Figure 4-13 involved a total traverse of 650 m, and a maximum speed of 6.5 m/s. While SAMWISE would have performed well even at higher speeds, the maximum speed had to be restricted to ensure safe obstacle avoidance inside the cluttered warehouse. When the vehicle returned to the start location about 209 seconds into the flight, approximately 20 m of estimator drift had built up, causing the vehicle to overshoot the start. Because it was nearing the boundary of the flight area, the safety pilot took manual control a few seconds later as a precaution, and flew the vehicle back to the start location.

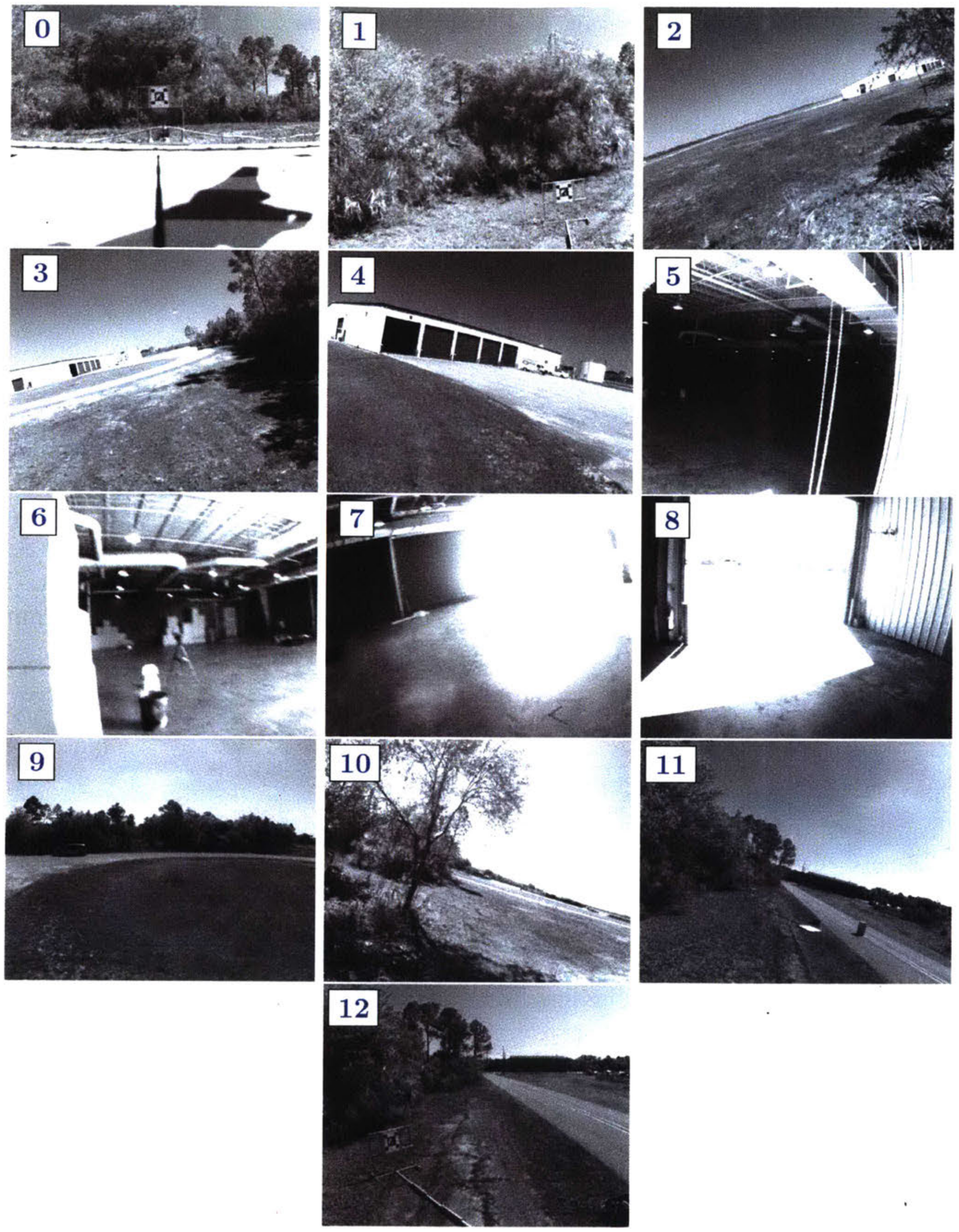


Figure 4-13: Sequence of images from the onboard camera taken during an autonomous flight including both outdoor and indoor elements. The vehicle traversed hundreds of meters, starting outdoors and navigating to a goal location inside a warehouse before flying back to the start. The lighting conditions during the indoor/outdoor transitions were particularly challenging, causing feature tracking to be interrupted several times. Due to accumulated drift, the vehicle overshot the start location. The moment that the vehicle passes over the start location is captured in the last image.

The position estimate can be seen in Figure 4-14. This final error corresponds to $\approx 3\%$ estimator drift over the greater than 700 m of total flight. The increase in error here over the results of the previous sections reflects the interruptions to feature tracking presented by the particularly challenging lighting conditions of this scenario.

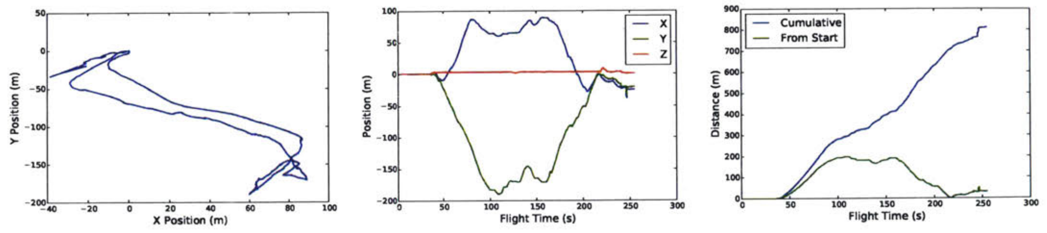


Figure 4-14: SAMWISE position estimation during outdoor/indoor flight. (Left) Overhead view of trajectory estimate. (Middle) Component-wise position estimates. (Right) Total distance traveled alongside distance from start. Navigated by SAMWISE, the vehicle finished about 20m from the start, after about 700m of flight. This corresponds to $\approx 3\%$ estimator drift. About 209 seconds into the flight, the manual safety pilot took over, navigating the vehicle safely back to the start location.

## 4.4 Summary

SAMWISE is a fully-functional, flight-tested state estimator and navigation solution for fast-moving robots. Designed for unstable aerial robots requiring high-rate closed-loop control, SAMWISE provides low-latency state estimates at IMU rate. Leveraging an asynchronous iSAM2-based solver, SAMWISE provides high accuracy at relatively low computation. By adopting a smoothing, factor-graph based representation, SAMWISE naturally accommodates delayed measurements and multiple sensor streams. As a smoother, SAMWISE also supports local-frame trajectory planning.

Computationally, SAMWISE combines the (on average) efficiency of iSAM2 with a decoupled IMU propagator to guarantee high-rate output. This allows for high-accuracy, nonlinear smoothing even on computationally-restricted vehicles.

SAMWISE was evaluated through a challenging series of tests during Phase 1 of the DARPA FLA program. Large-scale missions stressed the estimator's global consistency over hundreds of meters and its reliability with flights consistently lasting several minutes. Varied environmental and lighting conditions, along with high speeds and large rotations, stressed the visual pipeline. Nonetheless, SAMWISE performed reliably, with consistently low levels of drift ($0.5 - 3\%$, depending on the trajectory and environment).

While SAMWISE currently only accommodates monocular camera measurements due to the constraints of the FLA vehicle, adding support for stereo would be a natural extension and would offer increased observability and significantly reduced drift. Additionally, work is currently being done to accommodate short-term loop closures using descriptive features which can be recognized even after leaving the frame. This will significantly improve robustness to large rotations.

As it continues to mature, we believe SAMWISE will provide a robust, accurate, general-purpose vision-aided inertial navigation solution with applications to both ground and aerial systems.

# Chapter 5

# Conclusions

The navigation problem is ubiquitous throughout robotics and aerospace, in applications and environments as disparate as Augmented Reality [37] and the Apollo program [27]. Whether autonomous or manned, many systems need to be able to localize themselves. However, infrastructure-based solutions such as GPS are not available at all in certain environments, or may be unreliable due to hostile action. For this reason, there is a need for self-contained, onboard solutions.

For small robots or vehicles, onboard computational resources can be quite restricted. To compound this difficulty, agile, inherently unstable vehicles such as quadrotors require state estimate updates to be produced with low-latency and at a high rate to ensure stability and achieve maximal trajectory tracking performance. Thus, algorithms for navigation and estimation must be lightweight and efficient.

One promising approach to this problem is via a smoothing formulation of landmark-SLAM [13]. Rather than estimating only the latest robot state at each step, smoothing formulations simultaneously solve for multiple robot and landmark states simultaneously. By leveraging sparsity in the problem, the smoothing optimization can be performed relatively efficiently.

## 5.1 Computation in graph SLAM

Computation management is a vital, open challenge in SLAM today. Realtime, computationally-constrained systems require high-rate data fusion to produce accurate, robust state estimation and navigation solutions. The smoothing approach has been shown to be much more accurate, robust, and flexible than filtering methods, but relies heavily on system sparsity for efficiency. As measurements correspond directly to factors or edges in landmark-SLAM, the choice of incorporated measurements directly impacts sparsity. Thus, intelligent measurement selection has the capacity to significantly improve sparsity and reduce computation over naive SLAM.

Understanding the relationship between graph structure and computation is therefore vital. Chapter 2 introduced the optimal elimination complexity $\mathcal{C}^{\star}(G)$ as a reflection of the intrinsic computational complexity represented by an estimation graph. The elimination complexity $\mathcal{C}$ is derived from the approximate operation count of solving the corresponding sparse linear system (i.e. performing variable elimination), which is well-understood from a linear algebra perspective [41, 52]. One contribution of this thesis was to demonstrate empirically that elimination complexity also correlates with the update computation of incremental solvers like iSAM2. In Chapter 3, this complexity metric provided a basis for sparsity analysis of decimation-style measurement selection policies.

## 5.2 Decimation as a measurement selection policy

Though sparsity arises naturally in SLAM systems, it is not always sufficient for computationally-constrained systems. Modern sensors such as cameras can easily produce enough data and measurements to overwhelm even state-of-the-art incremental solvers. Fortunately, through intelligent measurement selection and pruning, it is possible to achieve significant computational savings with acceptable degradation in estimation performance.

For high-rate landmark-SLAM systems, particularly those involving vision, measurement selection policies can play a large role in computation reduction. However, many of the sophisticated methods proposed in the literature (see Section 1.1.2) are computationally expensive and impractical for real-time use in many high-rate systems. Additionally, many remove edges without regard for graph structure [28, 36], which can result in underwhelming computation reduction, even after aggressive edge pruning.

Decimation-style policies, on the other hand, are simple to implement and computationally negligible to evaluate. As argued in Chapter 3, decimation produces an inherently sparse super-structure, which allows decimated graphs to maintain many more measurements at lower computational cost. This corresponds to a higher average node degree, which was shown by [35, 47] to reduce over-fitting and improve accuracy. Furthermore, in Section 3.2 it was shown that the even spacing of observations characteristic of decimation was near $t_w$-optimal in single-landmark graphs. This suggests that decimation has good connectivity properties as well, and promotes reduced uncertainty volume [35].

Additionally, the modified, decimation-based policy `dec++` was proposed in Section 3.3, which addressed some of the minor implementation drawbacks of naive per-landmark decimation. Simulated SLAM results demonstrated empirically in Section 3.4 that decimation-style approaches can perform as well or better than more sophisticated methods, both in terms of KLD and resulting graph complexity. Given the negligible implementation and computational complexity of these policies, they present an effective and formidable measurement selection strategy available to even the most rudimentary landmark SLAM systems.

## 5.3 A robust, high-rate visual-inertial system for small, agile vehicles

The SAMWISE state estimator is designed to provide a robust, accurate, high-rate state-estimation and navigation solution for agile robots. Developed under the DARPA Fast Lightweight Autonomy (FLA) program, SAMWISE fuses inertial and monocular-video data streams into a smoothing-SLAM framework to provide maximal accuracy and robustness. By leveraging the state-of-the-art iSAM2 [32] incremental solver algorithm, SAMWISE efficiently estimates the vehicle trajectory, landmark positions, and a suite of calibration parameters simultaneously.

As described in Chapter 4, SAMWISE incorporates several key innovations which make feasible high-rate, low-latency closed-loop control. Decoupled IMU propagation and rebasing allows constant-time access to the latest state estimate (Section 4.1.3). Keypose publishing facilitates local-frame planning for robust trajectory tracking (Section 4.1.7).

In Section 4.2, SAMWISE was evaluated on the open-source EuRoC MAV dataset [5], demonstrating good performance in spite of the inherent challenges of this dataset. Furthermore, the proposed `dec++` policy was implemented within SAMWISE and tested on the EuRoC dataset under varying decimation rate parameters. Further flight results from indoor (Section 4.3.1) and outdoor (Sections 4.3.3 and 4.3.4) environments were presented as part of the DARPA FLA program, showing robustness in and applicability to challenging real-world conditions.

## 5.4 Future work

The insights presented in this thesis pose additional questions and expose some areas for future work.

The elimination complexity metric presented in Chapter 2 proved a useful tool for offline analysis of the computational cost represented by particular graph structure. Nonetheless, evaluation requires replicating the elimination process, and is therefore impractical for real-time use in measurement selection strategies. Methods of predicting computation directly from *local* graph structure, such as being able to evaluate the marginal cost of maintaining a particular measurement, could prove quite useful in implementing improved selection policies. Identification of additional sparse super-structures, similar to the partitioning structure discussed in Section 3.1, could inform implementation of inherently sparse graph architectures or novel measurement pruning policies.

This thesis showed that decimation essentially partitions the graph, which has the benefit of producing an inherently sparse super-structure. However, this partitioning intuitively comes at the cost of some form of connectivity, and empirically it was observed that significant accuracy can be lost. In a sense, decimation policies specify graph structure *a priori*. However, the realized sparsity in practice can depend significantly on the particular trajectory, availability of landmarks, and sensor limitations. Thus, more sophisticated selection policies which consider the current state of the graph could result in dramatic sparsity gains.

Ultimately, however, the arguments presented in this thesis show that decimation provides at the very least an effective primitive in the area of measurement selection. For computationally-constrained, high-rate systems, the simplicity of decimation can make it a formidable choice.

# Appendix A

# Proof of Lemmas

Lemma 1: $\mathcal{C}(G, \mathcal{P})$ is non-decreasing as edges are added to $G$

*Proof.* Let $G^+$ refer to the graph constructed by adding an edge to $G$. Following the elimination process described in Section 2.2, it is clear that the elimination neighborhood at each step $i$ cannot be smaller for $G^+$ than for $G$

$$d_s(i, G, \mathcal{P}) \leq d_s(i, G^+, \mathcal{P}) \qquad \forall i \tag{A.1}$$

and substituting this into the definitions of $\mathcal{C}(G, \mathcal{P})$ and $\mathcal{C}(G^+, \mathcal{P})$, it is clear that

$$\mathcal{C}(G, \mathcal{P}) \leq \mathcal{C}(G^+, \mathcal{P}) \tag{A.2}$$

□

Lemma 2: Properties of $t_w(G)$ for graph $G$ with positive weights

*Proof.* The definition of the weighted number of spanning trees is repeated here

$$t_w(G) \triangleq \sum_{T \in \mathcal{T}(G)} \mathbb{V}(T) \tag{A.3}$$

$$\mathbb{V}(T) \triangleq \prod_{e \in T} w_e \tag{A.4}$$

From the definition (A.3), $t_w(G)$ is a sum over all spanning trees of $G$. If $G$ is connected, $G$ must have at least one spanning tree. Additionally, because the edge weights are all positive, $\mathbb{V}(T) > 0$ for each $T \in \mathcal{T}(G)$. Thus, $t_w(G) > 0$. ✓

Because adding a new edge to graph $G$ to produce $\bar{G}$ cannot break any existing spanning trees, $\mathcal{T}(G) \subset \mathcal{T}(\bar{G})$. Using $\mathbb{V}(T) > 0$, it is clear that $t_w(G) \leq t_w(\bar{G})$. ✓

For any edge $(i, j)$ which exists in $G$, the set of spanning trees can be decomposed into two disjoint sets

$$\mathcal{T}(G) = \mathcal{T}_{\times(i,j)}(G) + \mathcal{T}_{\backslash(i,j)} \tag{A.5}$$

where $\mathcal{T}_{\times(i,j)}$ is the set of spanning trees which include edge $(i, j)$ and $\mathcal{T}_{\backslash(i,j)}$ is the set which lack edge $(i, j)$.

From the definition of $t_w$ in (A.3),

$$t_w(G) \triangleq \sum_{T \in \mathcal{T}(G)} \mathbb{V}(T) \tag{A.6}$$

$$= \sum_{T \in \mathcal{T}_{\times(i,j)}(G)} \mathbb{V}(T) + \sum_{T \in \mathcal{T}_{\backslash(i,j)}(G)} \mathbb{V}(T) \tag{A.7}$$

$$= t_{w,\{\times(i,j)\}}(G) + t_{w,\{\backslash(i,j)\}}(G) \tag{A.8}$$

□

Lemma 3: "Dangling chains"

*Proof.* For any graph $G^-$, consider a graph $G$ formed by attaching a chain of edge length $n > 1$ at one end to exactly one vertex $x_0$ in $G^-$, such that no new cycles are created. Refer to these added vertices as $\hat{X} = \{x_1, x_2, \ldots, x_n\}$, and the chain itself as $\hat{T}(\{x_0\} \cup \hat{X}, \mathcal{E}_{\hat{T}})$. Thus, for any node $x_i \in \hat{X}$, there exists exactly one path (comprising only of edges in $\mathcal{E}_{\hat{T}}$) connecting it to $x_0$ and the rest of $G^-$.

By definition, spanning trees of $G \in \mathcal{T}(G)$ must include all vertices $\hat{X}$ and be connected. Because exactly one path exists between poses in $\hat{X}$ and $x_0$, this entire path $\hat{T}$ must be maintained in every tree $T \in \mathcal{T}(G)$. Given any spanning tree $T^- \in \mathcal{T}(G^-)$, the corresponding spanning tree $T \in \mathcal{T}(G)$ can be generated by appending the full chain defined by $\hat{T}$, i.e. $T = T^{-} \cup \hat{T}$. Therefore,

$$t_w(G) \triangleq \sum_{T \in \mathcal{T}(G)} \mathbb{V}(T) \tag{A.9}$$

$$= \sum_{T^{-} \in \mathcal{T}(G^{-})} \mathbb{V}(\hat{T})\mathbb{V}(T^{-}) \tag{A.10}$$

$$= \mathbb{V}(\hat{T})t_w(G^{-}) \tag{A.11}$$

$$= vt_w(G^{-}) \tag{A.12}$$

□

Lemma 4: An optimal solution to Problem 1 exists and necessarily includes the observation edges associated with pose node $x_0$ and $x_n$.

*Proof.* Problem 1 is a maximization over the finite set of graphs $\bar{\mathcal{G}}_{m,n}$. Thus, an optimum must exist. ✓

Following the notation in Section 3.2, let $\mathcal{G}_{m,n} \subset \bar{\mathcal{G}}_{m,n}$ be this family of single landmark graphs which necessarily include the observation edges associated with the first and last pose nodes, $x_0$ and $x_n$. The fact that any optimizer $G^{\star}$ must belong to $\mathcal{G}_{m,n}$ will be shown by contradiction.

Because an optimum must exist, assume for the purpose of contradiction that $G_0 \notin \mathcal{G}_{m,n}$ is a maximizer. From the definition of $\mathcal{G}_{m,n}$, $G_0$ cannot simultaneously contain the first and last observation edges associated with $x_0$ and $x_n$, and therefore must include one or two dangling odometry chains. A possible example is shown in the left pane of Figure A-1.

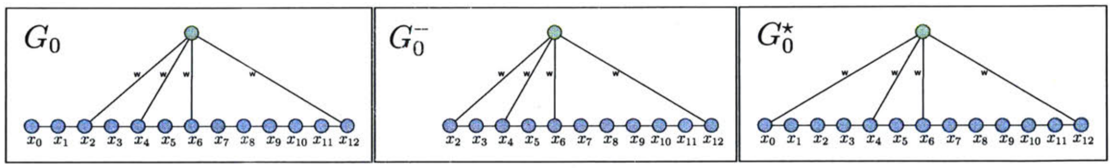

$G_0$
w
$x_0$ $x_1$ $x_2$ $x_3$ $x_4$ $x_5$ $x_6$ $x_7$ $x_8$ $x_9$ $x_{10}$ $x_{11}$ $x_{12}$
$G_0^-$
$x_2$ $x_3$ $x_4$ $x_5$ $x_6$ $x_7$ $x_8$ $x_9$ $x_{10}$ $x_{11}$ $x_{12}$
$G_0^\star$

Figure A-1: (left) An example graph $G_0 \notin \mathcal{G}_{m,n}$, with a dangling odometry chain of nodes $\{x_0, x_1\}$. (center) The graph $G_0^-$ produced by removing the chain characterized by nodes $\{x_0, x_1\}$. By Lemma 3, $t_w(G_0) = t_w(G_0^-)$. (right) The graph $G_0^\star$ produced from $G_0$ by moving the first (and generally last) observation edges to $x_0$ and $x_n$, respectively. In this case, $\{x_0, x_1\}$ have been "absorbed" into the left-most cycle of $G_0^\star$.

By Lemma 3, and using the fact that odometry edges are assigned unity weight, there exists a subgraph $G_0^- \in \mathcal{G}_{m,n^-}$ such that $t_w(G_0) = t_w(G_0^-)$. Additionally, we can construct a graph $G_0^\star$ based on $G_0$ by moving the first and last included observation edges to poses $x_0$ and $x_n$, respectively, such that $G_0^\star \in \mathcal{G}_{m,n}$. Examples of a possible $G_0$ and corresponding $G_0^-$ and $G_0^\star$ are shown in Figure A-1.

Because $t_w(G_0) = t_w(G_0^-)$, it is sufficient to compare $G_0^\star$ and $G_0^-$ directly. It is clear from inspection that $t_w(G_0^\star) > t_w(G_0^-)$ necessarily, and the conclusion follows. Thus, $t_w(G_0^\star) > t_w(G_0)$, and thus $G_0 \notin \mathcal{G}_{m,n}$ cannot be a maximizer of Problem 1. $\square$

# Appendix B

# Additive Noise in Continuous-Time Systems

Stochastic, time-varying systems are often modeled with additive Gaussian noise

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t), t) + \mathbf{w}(t) \tag{B.1}$$

with random process vector $\mathbf{x}(t)$, known input $\mathbf{u}(t)$, and time-varying mean dynamics $\mathbf{f}(\mathbf{x}, \mathbf{u}, t)$. Often, in inertial systems, the input $\mathbf{u}(t)$ is taken to be the raw sensor readings from the IMU. $\mathbf{w}(t)$ is a zero-mean, isotropic, white noise process, meaning that

$$\mathbb{E}[\mathbf{w}(\tau_1)] = 0 \qquad \mathbb{E}[\mathbf{w}(\tau_1)\mathbf{w}(\tau_2)^T] = \sigma^2\delta(\tau_2 - \tau_1)\mathbf{I} \tag{B.2}$$

for all $\tau_1, \tau_2 \in \mathbb{R}$, and identity matrix $\mathbf{I}$.

In order to be represented in digital systems, and particularly in a factor graph structure, the continuous time dynamics must be represented in discrete time. Between times $t_1$ and $t_2$, this involves determining $\Delta\mathbf{x} \equiv \mathbf{x}_{t_2} - \mathbf{x}_{t_1}$. For short time intervals $\Delta t \equiv t_2 - t_1 \ll 1$, the dynamics can be assumed constant

$$\Delta\mathbf{x} = \int_{t_1}^{t_2} \mathbf{f}(\mathbf{x}(\tau), \mathbf{u}(\tau), \tau) + \mathbf{w}(\tau)\, \mathrm{d}\tau \tag{B.3}$$

$$\approx \int_{t_1}^{t_2} \mathbf{f}(\mathbf{x}(t_1), \mathbf{u}(t_1), t_1) + \mathbf{w}(\tau)\,\mathrm{d}\tau \tag{B.4}$$

$$= \mathbf{f}(\mathbf{x}(t_1), \mathbf{u}(t_1), t_1)\Delta t + \int_{t_1}^{t_2} \mathbf{w}(\tau)\,\mathrm{d}\tau \tag{B.5}$$

Because $\mathbf{w}(t)$ is a random process, $\Delta\mathbf{x}$ is a random variable. Fortunately, its mean and covariance can be computed in a straightforward fashion

$$\mathbb{E}[\Delta\mathbf{x}] = \mathbf{f}(\mathbf{x}(t_1), \mathbf{u}(t_1), t_1)\Delta t \tag{B.6}$$

$$\begin{aligned}
\mathrm{Cov}(\Delta\mathbf{x}, \Delta\mathbf{x}) &= \mathbb{E}[(\Delta\mathbf{x} - \mathbb{E}[\Delta\mathbf{x}])(\Delta\mathbf{x} - \mathbb{E}[\Delta\mathbf{x}])^T] && \text{(B.7)}\\
&\approx \mathbb{E}\left[\int_{t_1}^{t_2}\int_{t_1}^{t_2} \mathbf{w}(\tau_1)\mathbf{w}(\tau_2)^T\,\mathrm{d}\tau_1\,\mathrm{d}\tau_2\right] && \text{(B.8)}\\
&= \int_{t_1}^{t_2}\int_{t_1}^{t_2} \mathbb{E}\big[\mathbf{w}(\tau_1)\mathbf{w}(\tau_2)^T\big]\,\mathrm{d}\tau_1\,\mathrm{d}\tau_2 && \text{(B.9)}\\
&= \int_{t_1}^{t_2} \sigma^2\mathbf{I}\,\mathrm{d}\tau_1 && \text{(B.10)}\\
&= \sigma^2\Delta t\mathbf{I} && \text{(B.11)}
\end{aligned}$$

Given the above assumptions, it is clear that uncertainty growth in the system (over short time scales) grows linearly with $\Delta t$, and is independent of the mean dynamics $\mathbf{f}(\cdot)$.

# References

[1] Martin Aigner, Günter M Ziegler, and Alfio Quarteroni. *Proofs from THE BOOK*, volume 274. Springer, 2010.

[2] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. *Computer vision–ECCV 2006*, pages 404–417, 2006.

[3] Bradley M Bell and Frederick W Cathey. The iterated kalman filter update as a gauss-newton method. *IEEE Transactions on Automatic Control*, 38(2):294–297, 1993.

[4] Dimitri P Bertsekas. *Nonlinear programming.* Athena scientific Belmont, 1999.

[5] Michael Burri, Janosch Nikolic, Pascal Gohl, Thomas Schneider, Joern Rehder, Sammy Omari, Markus W Achtelik, and Roland Siegwart. The euroc micro aerial vehicle datasets. *The International Journal of Robotics Research*, 2016.

[6] Nicholas Carlevaris-Bianco and Ryan M Eustice. Generic factor-based node marginalization and edge sparsification for pose-graph slam. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 5748–5755. IEEE, 2013.

[7] Luca Carlone and Sertac Karaman. Attention and anticipation in fast visual-inertial navigation. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 3886–3893. IEEE, 2017.

[8] Han-Pang Chiu, Stephen Williams, Frank Dellaert, Supun Samarasekera, and Rakesh Kumar. Robust vision-aided navigation using sliding-window factor graphs. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 46–53. IEEE, 2013.

[9] C Chow and Cong Liu. Approximating discrete probability distributions with dependence trees. *IEEE transactions on Information Theory*, 14(3):462–467, 1968.

[10] Timothy A Davis, John R Gilbert, Stefan I Larimore, and Esmond G Ng. Algorithm 836: Colamd, a column approximate minimum degree ordering algorithm. *ACM Transactions on Mathematical Software (TOMS)*, 30(3):377–380, 2004.

[11] Timothy A Davis, Sivasankaran Rajamanickam, and Wissam M Sid-Lakhdar. A survey of direct methods for sparse linear systems. *Acta Numerica*, 25:383–566, 2016.

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

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

[14] Frank Dellaert, Alexander Kipp, and Peter Krauthausen. A multifrontal qr factorization approach to distributed inference applied to multirobot localization and mapping. In *Proceedings of the national conference on artificial intelligence*, volume 20, page 1261. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2005.

[15] Tue-Cuong Dong-Si and Anastasios I Mourikis. Consistency analysis for sliding-window visual odometry. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 5202–5209. IEEE, 2012.

[16] Iain S Duff. On the number of nonzeros added when gaussian elimination is performed on sparse random matrices. *mathematics of computation*, 28(125):219–230, 1974.

[17] Jakob Engel, Vladlen Koltun, and Daniel Cremers. Direct sparse odometry. *arXiv preprint arXiv:1607.02565*, 2016.

[18] Jakob Engel, Thomas Schöps, and Daniel Cremers. Lsd-slam: Large-scale direct monocular slam. In *European Conference on Computer Vision*, pages 834–849. Springer, 2014.

[19] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[20] Christian Forster, Luca Carlone, Frank Dellaert, and Davide Scaramuzza. On-manifold preintegration theory for fast and accurate visual-inertial navigation. *IEEE Trans Robot*, pages 1–18, 2015.

[21] Christian Forster, Matia Pizzoli, and Davide Scaramuzza. Svo: Fast semi-direct monocular visual odometry. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 15–22. IEEE, 2014.

[22] W Nicholas Greene, Kyel Ok, Peter Lommel, and Nicholas Roy. Multi-level mapping: Real-time dense monocular slam. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 833–840. IEEE, 2016.

[23] Denis Grießbach, Dirk Baumbach, and Sergey Zuev. Stereo-vision-aided inertial navigation for unknown indoor and outdoor environments. In *Indoor Positioning*

*and Indoor Navigation (IPIN), 2014 International Conference on*, pages 709–716. IEEE, 2014.

[24] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision.* Cambridge university press, 2003.

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

[26] Joel A Hesch, Dimitrios G Kottas, Sean L Bowman, and Stergios I Roumeliotis. Observability-constrained vision-aided inertial navigation. *University of Minnesota, Dept. of Comp. Sci. & Eng., MARS Lab, Tech. Rep*, 1, 2012.

[27] David Garratt Hoag. *Apollo Navigation, Guidance, and Control Systems: A Progress Report.* MIT Instrumentation Laboratory, 1969.

[28] Guoquan Huang, Michael Kaess, and John J Leonard. Consistent sparsification for graph optimization. In *Mobile Robots (ECMR), 2013 European Conference on*, pages 150–157. IEEE, 2013.

[29] Viorela Ila, Josep M Porta, and Juan Andrade-Cetto. Information-based compact pose slam. *IEEE Transactions on Robotics*, 26(1):78–93, 2010.

[30] Simon J Julier and Jeffrey K Uhlmann. A new extension of the kalman filter to nonlinear systems. In *Int. symp. aerospace/defense sensing, simul. and controls*, volume 3, pages 182–193. Orlando, FL, 1997.

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

[32] Michael Kaess, Hordur Johannsson, Richard Roberts, Viorela Ila, John J Leonard, and Frank Dellaert. isam2: Incremental smoothing and mapping using the bayes tree. *The International Journal of Robotics Research*, page 0278364911430419, 2011.

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

[34] Rudolph Emil Kalman et al. A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, 82(1):35–45, 1960.

[35] Kasra Khosoussi, Shoudong Huang, and Gamini Dissanayake. Novel insights into the impact of graph structure on slam. In *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pages 2707–2714. IEEE, 2014.

[36] Kasra Khosoussi, Gaurav S Sukhatme, Shoudong Huang, and Gamini Dissanayake. Maximizing the weighted number of spanning trees: Near-$t$-optimal graphs. *arXiv preprint arXiv:1604.01116*, 2016.

[37] Georg Klein and David Murray. Parallel tracking and mapping for small ar workspaces. In *Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on*, pages 225–234. IEEE, 2007.

[38] Peter Krauthausen, Frank Dellaert, and Alexander Kipp. Exploiting locality by nested dissection for square root smoothing and mapping. Technical report, Georgia Institute of Technology, 2005.

[39] Mingyang Li and Anastasios I Mourikis. High-precision, consistent ekf-based visual–inertial odometry. *The International Journal of Robotics Research*, 32(6):690–711, 2013.

[40] Bruce D Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of the 7th Intl. Joint Conf. on Artificial intelligence (IJCAI)*, pages 674–679. ACM, 1981.

[41] Robert Luce and Esmond G Ng. On the minimum flops problem in the sparse cholesky factorization. *SIAM Journal on Matrix Analysis and Applications*, 35(1):1–21, 2014.

[42] Dmitry M Malioutov, Jason K Johnson, and Alan S Willsky. Walk-sums and belief propagation in gaussian graphical models. *Journal of Machine Learning Research*, 7(Oct):2031–2064, 2006.

[43] Agostino Martinelli. Vision and imu data fusion: Closed-form solutions for attitude, speed, absolute scale, and bias determination. *IEEE Transactions on Robotics*, 28(1):44–60, 2012.

[44] Mladen Mazuran, Wolfram Burgard, and Gian Diego Tipaldi. Nonlinear factor recovery for long-term slam. *The International Journal of Robotics Research*, 35(1-3):50–72, 2016.

[45] JMM Montiel, Javier Civera, and Andrew J Davison. Unified inverse depth parametrization for monocular slam. *analysis*, 9:1, 2006.

[46] Anastasios I Mourikis and Stergios I Roumeliotis. A multi-state constraint kalman filter for vision-aided inertial navigation. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 3565–3572. IEEE, 2007.

[47] Edwin Olson and Michael Kaess. Evaluating the performance of map optimization algorithms. In *RSS Workshop on Good Experimental Methodology in Robotics*, volume 15, 2009.

[48] ON Semiconductor. *Aptina MT9V034*, 1 2017. Rev. 7.

[49] Steve Paschall and Julius Rose. Fast, lightweight autonomy through an unknown cluttered environment. In *Aerospace Conference, 2017 IEEE*, pages 1–8. IEEE, 2017.

[50] Alex Pothen and Sivan Toledo. Elimination structures in scientific computing., 2004.

[51] William H Press. *Numerical recipes 3rd edition: The art of scientific computing.* Cambridge university press, 2007.

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

[53] David M. Rosen, Luca Carlone, Afonso S. Bandeira, and John J. Leonard. Sesync: A certifiably correct algorithm for synchronization over the special euclidean group, 2016.

[54] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *Computer Vision (ICCV), 2011 IEEE international conference on*, pages 2564–2571. IEEE, 2011.

[55] David Schleicher, Luis M Bergasa, Rafael Barea, Elena Lopez, Manuel Ocana, Jesus Nuevo, and Pablo Fernandez. Real-time stereo visual slam in large-scale environments based on sift fingerprints. In *Intelligent Signal Processing, 2007. WISP 2007. IEEE International Symposium on*, pages 1–6. IEEE, 2007.

[56] Jianbo Shi et al. Good features to track. In *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference on*, pages 593–600. IEEE, 1994.

[57] Gabe Sibley, Larry Matthies, and Gaurav Sukhatme. Sliding window filter with application to planetary landing. *Journal of Field Robotics*, 27(5):587–608, 2010.

[58] John Stalbaum and Jae-bok Song. Keyframe and inlier selection for visual slam. In *Ubiquitous Robots and Ambient Intelligence (URAI), 2013 10th International Conference on*, pages 391–396. IEEE, 2013.

[59] Ted J Steiner, Robert D Truax, and Kristoffer Frey. A vision-aided inertial navigation system for agile high-speed flight in unmapped environments. In *Aerospace Conference, 2017 IEEE*, pages 1–10. IEEE, 2017.

[60] Hauke Strasdat, JMM Montiel, and Andrew J Davison. Real-time monocular slam: Why filter? In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 2657–2664. IEEE, 2010.

[61] Yue Wang, Rong Xiong, Qianshan Li, and Shoudong Huang. Kullback-leibler divergence based graph pruning in robotic feature mapping. In *Mobile Robots (ECMR), 2013 European Conference on*, pages 32–37. IEEE, 2013.

[62] Mihalis Yannakakis. Computing the minimum fill-in is np-complete. *SIAM Journal on Algebraic Discrete Methods*, 2(1):77–79, 1981.