Robust and Efficient Robotic Mapping

by

Edwin B. Olson

Submitted to the Department of
Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2008

Author .....
Department of Electrical Engineering and Computer Science
May 16, 2008

Certified by .....
Seth Teller
Professor of Computer Science and Engineering
Thesis Supervisor

Certified by .....
John Leonard
Professor of Mechanical and Ocean Engineering
Thesis Supervisor

Accepted by .....
Terry P. Orlando
Chairman, Department Committee on Graduate Students

Robust and Efficient Robotic Mapping

by

Edwin B. Olson

Submitted to the Department of Electrical Engineering and Computer Science
on May 16, 2008, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Computer Science and Engineering

Abstract

Mobile robots are dependent upon a model of the environment for many of their basic functions. Locally accurate maps are critical to collision avoidance, while large-scale maps (accurate both metrically and topologically) are necessary for efficient route planning. Solutions to these problems have immediate and important applications to autonomous vehicles, precision surveying, and domestic robots.

Building accurate maps can be cast as an optimization problem: find the map that is most probable given the set of observations of the environment. However, the problem rapidly becomes difficult when dealing with large maps or large numbers of observations. Sensor noise and non-linearities make the problem even more difficult—especially when using inexpensive (and therefore preferable) sensors.

This thesis describes an optimization algorithm that can rapidly estimate the maximum likelihood map given a set of observations. The algorithm, which iteratively reduces map error by considering a single observation at a time, scales well to large environments with many observations. The approach is particularly robust to noise and non-linearities, quickly escaping local minima that trap current methods. Both batch and online versions of the algorithm are described.

In order to build a map, however, a robot must first be able to recognize places that it has previously seen. Limitations in sensor processing algorithms, coupled with environmental ambiguity, make this difficult. Incorrect place recognitions can rapidly lead to divergence of the map. This thesis describes a place recognition algorithm that can robustly handle ambiguous data.

We evaluate these algorithms on a number of challenging datasets and provide quantitative comparisons to other state-of-the-art methods, illustrating the advantages of our methods.

Thesis Supervisor: Seth Teller
Title: Professor of Computer Science and Engineering

Thesis Supervisor: John Leonard
Title: Professor of Mechanical and Ocean Engineering

Contents

1 Introduction **9**
- 1.1 Motivation . . . 9
- 1.2 Probabilistic Problem Formulation . . . 12
- 1.3 Loop Closing . . . 13
- 1.4 Pose/Feature Graphs . . . 15
- 1.5 Map Optimization . . . 16
- 1.6 Contributions . . . 16

2 Preliminaries **19**
- 2.1 Pose/Feature Graphs . . . 19
- 2.2 Graphical Models . . . 19
- 2.3 Geometrical Conventions . . . 20
- 2.4 Constraints . . . 22
  - 2.4.1 Common Constraint Forms . . . 23
  - 2.4.2 Linearization . . . 23
  - 2.4.3 Interpretation of the Chi-Squared Error . . . 25
  - 2.4.4 State Squared Error . . . 25

3 Loop Closing **29**
- 3.1 Introduction . . . 29
- 3.2 Method Overview . . . 32
- 3.3 Hypothesis Generation . . . 32
  - 3.3.1 Determining when poses overlap . . . 32
  - 3.3.2 Searching for a match . . . 34
- 3.4 Hypothesis Sets . . . 37
  - 3.4.1 Pairwise Consistency . . . 37
  - 3.4.2 Finding the optimal subset . . . 38
  - 3.4.3 The Eigenvectors of the Consistency Matrix . . . 40
  - 3.4.4 Discretization of the Indicator Vector . . . 40
  - 3.4.5 Improving Hypothesis Set Construction . . . 41
  - 3.4.6 Analysis of Robustness to Outliers . . . 42
- 3.5 Results . . . 43
  - 3.5.1 "Area C" Dataset . . . 44
  - 3.5.2 CSAIL Dataset . . . 46
  - 3.5.3 Killian Court Dataset . . . 48

3.5.4 Stanford Gates Building 49
3.5.5 Intel Research Center 49
3.6 Summary 49

4 Batch Optimization of Pose Graphs **55**
4.1 Prior Work 56
4.1.1 Recursive Linear Solutions 56
4.1.2 Graphical Models 58
4.1.3 Particle Filtering 58
4.1.4 Nonlinear Optimization 58
4.1.5 Hybrids 59
4.1.6 State Space 59
4.2 Overview and Intuition 60
4.3 Method 61
4.3.1 Incremental State Representation 61
4.3.2 Iterative optimization 63
4.3.3 Rigid-Body Constraints 66
4.3.4 The Error-Distributing Tree 68
4.3.5 Initial Learning Rate 71
4.3.6 Algorithm Summary 71
4.4 Results 72
4.4.1 CSW Dataset 73
4.4.2 High Noise Dataset 78
4.4.3 P5K20K Dataset 78
4.4.4 Intel Research Center Dataset 83
4.4.5 Stanford Gates Dataset 84
4.4.6 Dog Leg Configuration 85
4.4.7 Robustness 86
4.5 Summary 88

5 Incremental Optimization of Pose Graphs **89**
5.1 Method 90
5.1.1 Adding a new constraint 91
5.1.2 Processing existing constraints 92
5.1.3 Algorithm Summary 93
5.2 Partial Iterations 93
5.3 Learning Rate Data Structure 95
5.4 Analysis 97
5.5 Results 98
5.6 Summary 103

6 Conclusion **105**
6.1 Contributions 105
6.2 Future Work 106

A Sensor Processing **109**
A.1 Overview 109
A.2 Vision Processing 109
A.2.1 Data Collection Platform 109
A.2.2 Feature Detection 110
A.2.3 Feature Tracking and Landmark Initialization 110
A.2.4 Landmark matching 112
A.3 Laser Scan Processing 113
A.3.1 Challenges 114
A.3.2 Scan Matching 116
A.3.3 Loop Closing Hypothesis Generation 120
A.4 Summary 122

CHAPTER 1

Introduction

1.1 Motivation

Intelligent robots must answer a fundamental question: given a number of noisy sensor observations, what does the world actually look like? The quality of the world model is dependent on the quality of the sensors, of course, but as the environments that robots operate in get larger and more complicated, the limiting factor becomes *algorithmic*: the amount of computational effort required to process sensor observations grows quickly with the number of observations.

While robots like the Roomba vacuum can get by with limited world models (i.e., detecting obstacles by bouncing off of them), an autonomous vehicle travelling at highway speeds needs a more sophisticated obstacle avoidance strategy. High-quality maps are critical.

Many basic robot capabilities, such as collision avoidance and path planning, depend upon a map of the environment. Consequently, map building is a critical component of almost every autonomous robot, particularly if they must operate in unfamiliar environments. Many applications fall into this category, including cars in urban driving scenarios, military convoy vehicles, indoor delivery robots, and assistive robots for the elderly and infirm. Maps are also useful in non-autonomous applications: a human can become disoriented when teleoperating a search-and-rescue robot amidst rubble; a real-time map can make the operators more productive. In these cases, robots use maps in order to complete their missions. The accuracy and quality of the map directly affects their ability to effectively carry out their missions. If an obstacle is poorly localized, a robot may collide with it. If the topological structure of an environment is not properly modeled, a robot may take a needlessly circuitous route to a destination, or fail to find a route entirely.

The map itself is the goal of an increasing number of systems. These include capturing data for Geographic Information Systems (GIS) (both indoors and outdoors), along with other survey applications (such as measuring the sag of a bridge by driving a robot beneath it). The value of these systems is directly related to the accuracy of the map and the cost of constructing it.

Map building is hard. The real world is a complex, dynamic, and cluttered place. A sensor's view of this world is noisy and often ambiguous (different things often look

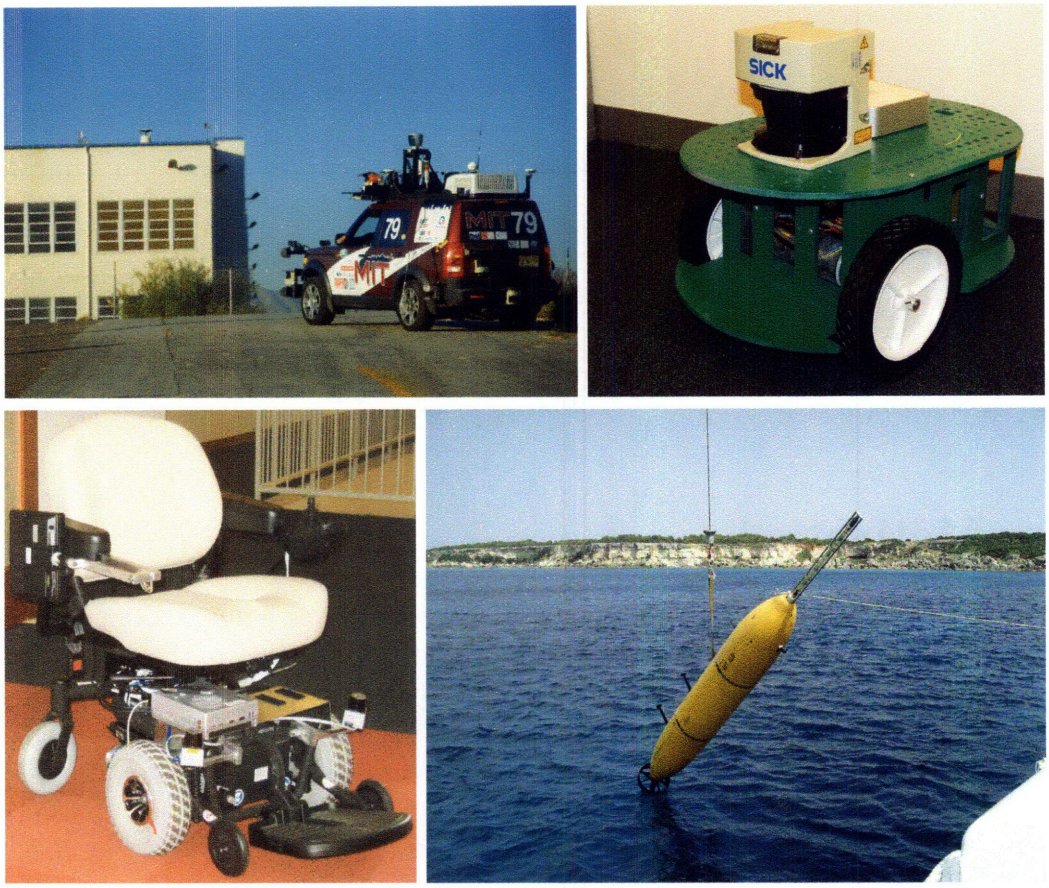

Figure 1-1: Robot Applications. Map building arises in many contexts, including autonomous cars [37], indoor surveying [68], underwater robots for ship-hull inspection [75], and human-centric assistive robotics [57].

about the same) and there is an economic incentive to use less expensive sensors; the limitations of these sensors often magnify the challenges. In short, the aim of mapping research is to accurately model large, complex, and cluttered environments using cheap sensors that produce poor data.

Mapping has been extensively studied by many researchers, serving as a central research topic for roboticists for decades. Early approaches were largely derived from the Extended Kalman Filter (EKF) [65] and it continues to be used in many applications. The EKF, however, has computational and memory costs that grow quadratically with the map size. The Extended Information Filter (EIF) [58] improves these costs, as do the many variants of Sparse Extended Information Filters (SEIF) [69, 70, 71, 13, 14, 74, 76, 76]. New variants, offering not only computational cost improvements but also interesting cost-versus-quality trade-offs, continue to be developed.

The mapping problem has also been approached by means of Rao-Blackwellized particle filters [44, 66, 22] and non-linear optimization [41, 12, 34, 72, 20, 16, 10, 24, 48, 50]. These methods offer additional cost/quality trade-offs, and have incorporated ideas and techniques from the machine learning and numerical computing community.

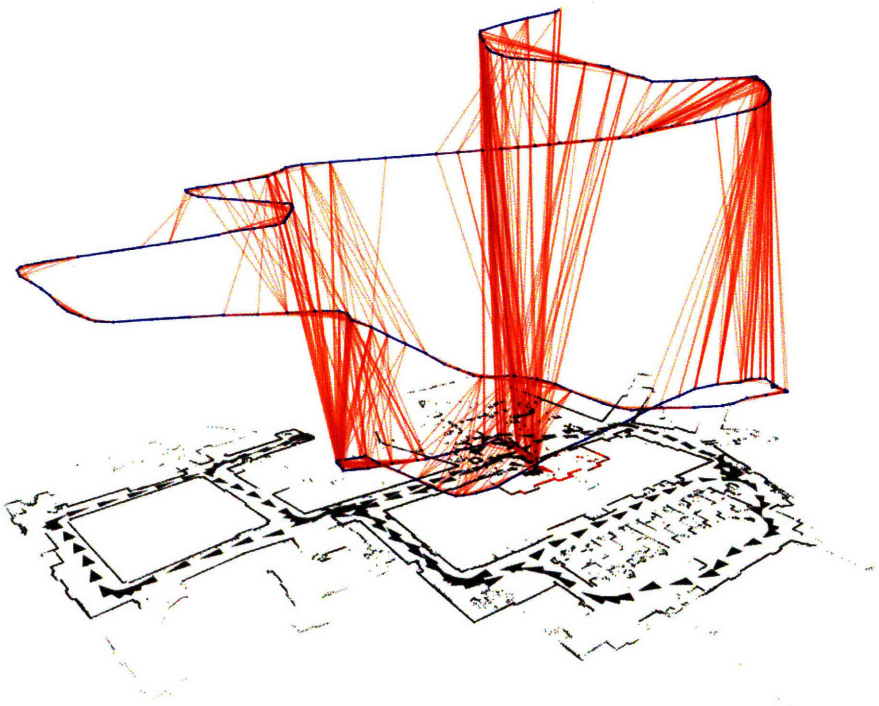

Figure 1-2: An automatically built map of an indoor office environment. Above the map, the trajectory (blue) is plotted with height corresponding to time. The large number of detected loop closures (shown in red) allow a high-quality posterior map to be computed.

However, the limited penetration of robots into our daily lives is a testament to the shortcomings of the state-of-the-art; at best, we see very expensive robots operating in rather limited environments. The challenge is to extend the state of the art by making these systems *faster* (so they can handle larger environments and greater numbers of sensor observations) and *more robust* to noise and ambiguity. Computational efficiency and robustness are the central themes of this thesis.

This thesis describes several new algorithms that extend the reach of mapping algorithms. We begin with a review of basic concepts in Chapter 2. The problem of identifying when a robot has revisited a particular place, loop closing, is a critical component of map building. Recognizing a place can be difficult because today's sensor processing algorithms can not always discriminate between similar (but distinct) places. We describe an algorithm that can automatically reject incorrect place recognitions that could otherwise result in poor maps in Chapter 3.

Given a set of correct place recognitions, the map building problem can be formally represented as a non-linear optimization: compute the map that is most consistent with all of the disambiguated sensor data. We describe a new algorithm for performing this optimization in Chapter 4, which is characterized by high speed and an unrivaled ability to escape local minima. As a result, our method can compute correct maps in cases where existing algorithms fail. In Chapter 5, we describe a variant of this algorithm that can be used in online settings— i.e., where the optimization problem is

constantly changing as new sensor data continuously arrives. Together, the methods of this thesis allow maps to be constructed even when sensor data is highly ambiguous, with computational costs that scale well with increasing environment size.

The remainder of this chapter provides a brief introduction to robot mapping and a high-level overview of this thesis's contributions.

1.2 Probabilistic Problem Formulation

In general terms, the map building problem can be described as finding the posterior probability distribution function of a map x given a set of sensor data z and a set of commanded robot motions u. We can divide the map into two parts: the trajectory of the robot s sampled at discrete times, and the location of other features f in the environment.

Unfortunately, most sensors do not unambiguously identify the features that they sense: they detect the presence of *some* feature, not the presence of a *particular* feature. The process of assigning feature identities to sensor observations is known as data association. Let d_i represent the identity of the i^{th} sensor observation z_i. The posterior probability distribution that we wish to discover is:

$$p(x \mid u, z, d) = p(s, f \mid u, z, d) \tag{1.1}$$

In the common case, sensors are mounted on the robot. Sensors observe the environment relative to the robot's current position, and the noise contaminating these robot-relative observations is independent. Projecting these observations into a global coordinate frame requires only the knowledge of the robot's position. In other words, sensor observations (in the global coordinate frame) are conditionally independent given the robot trajectory. This is the key insight behind FastSLAM[44], as it allows the posterior distribution (Eqn. 1.1) to be factored into terms corresponding to the robot trajectory and the individual position of each feature in the environment:

$$p(s, f \mid u, z, d) = p(s \mid u, z, d) \prod_i p(f_i \mid s, z, d) \tag{1.2}$$

The advantage of this factorization is that the conditional posterior distribution for a single feature $p(f_i \mid s, z, d)$ is easy to compute (e.g., with a Kalman filter). The posterior distribution for the trajectory is much harder: FastSLAM samples over this trajectory with a particle filter, whereas the approach in this thesis is to use non-linear optimization methods to compute it.

In short, once the trajectory of the map has been determined, a map can be easily computed. The critical role of the robot trajectory in map building has led to trajectory-centric approaches like the one presented in this thesis.

In principle, the data associations d are unknown and are correlated through the motion of the robot. Ideally, we would like to know the joint distribution:

$$p(d \mid u, z) \tag{1.3}$$

In this thesis, we make the common (but sub-optimal) approximation of computing d_i at about the time we observe z_i. Additionally, we do not revisit data associations based on subsequent sensor observations. Consequently, each data association d_i is a function of z_i and earlier sensor observations and their data associations.

$$p(d \mid u, z) \approx \prod_i p(d_i \mid u_{0\ldots i}, z_{0\ldots i}, d_{0\ldots i-1}) \tag{1.4}$$

The data-associations d assign particular observations to particular features. This is a discrete-valued assignment, in contrast to the continuously-valued variables comprising the map x. Since it has both discrete and continuous aspects, map building is a hybrid estimation problem. The discrete nature of the data associations is especially vulnerable to errors: assigning an observation to the incorrect feature can have a profound impact on the quality of the resulting map. Chapter 3 will illustrate a robust means of computing data-associations d, using an approximation similar to Eqn. 1.4.

Maintaining posterior *distributions* over the entire map is both computationally expensive and often unnecessary. The maximum likelihood map is generally the single most useful quantity for path planning, obstacle avoidance, and other practical applications. Some marginal distributions, such as the probability distribution of the robot's current position and some nearby feature ($p(s_{last}, f_i, |u, z, d)$), can be easily estimated without maintaining the full posterior distribution. Chapter 3 will show how to estimate marginal distributions of this form.

Consequently, while this thesis explores mapping from a probabilistic perspective, the primary goal is not to compute distributions, but instead to estimate the maximum likelihood map. Chapters 4 and 5 will then estimate the maximum-likelihood map x using non-linear optimization methods.

1.3 Loop Closing

As a robot drives around, its dead-reckoning error grows without bound. Suppose that a robot revisits a previously observed place, collecting similar sensor data each time. How can the robot determine that it is in fact in the same place it was before, versus some other place that looks about the same? How can the robot determine its position and orientation within that place? Loop closing algorithms seek to answer these questions.

The term "loop closing" arises from the common case where a robot drives in a large loop, returning to its starting point. If, after travelling around the loop, the robot can recognize that it is at its starting position, it "closes" the loop, creating a geometric relationship between the first and last position of the robot. This geometric relationship has two major effects:

- It establishes a *topological* relationship between temporally distant parts of the robot's trajectory. The robot can use this new topological link as a "short-cut" when planning routes.

- It provides a *metrical* relationship between the beginning and end of the robot's trajectory that can be used to refine the estimate of the robot's trajectory, improving upon the estimate provided by dead-reckoning.

In general, the larger the loop, the more difficult it is to recognize a closure. This is due to accumulating dead-reckoning error as the robot traverses the loop: when the robot returns to its starting point, it must determine that it is back at its starting point— not merely somewhere that *looks* like its starting place. Large loop closures may be more difficult, but they also are much more valuable in terms of refining the trajectory of the robot. Consequently, it is critical that a loop closing algorithm be able to close very large loops.

Loop closing is related to place recognition, which has applications beyond path planning and map building. For example, the same algorithms could be used by a robot to find a recharging station and properly align itself to it.

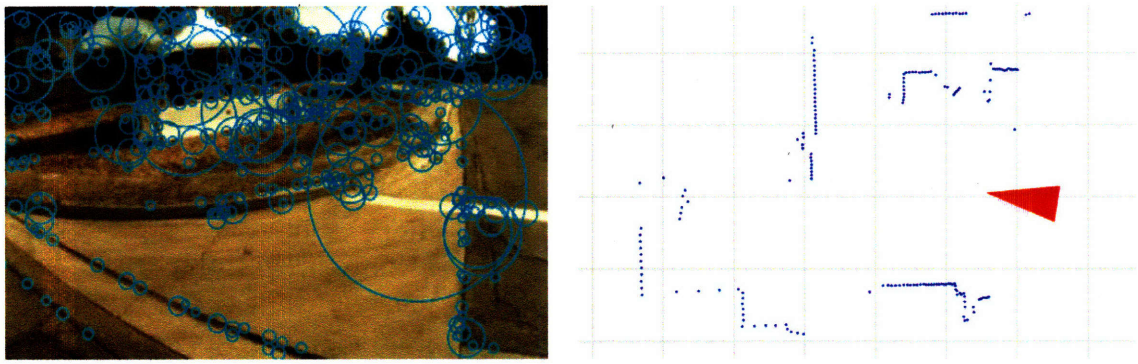

Figure 1-3: Sample sensor data. Different sensor modalities present different challenges when trying to "close the loop". Left: Camera-derived visual features can be richly descriptive, but are also cluttered and not consistently repeatable under changing lighting and viewpoint. Right: Laser-scan data from an indoor environment is composed principally of walls at 90 degree angles. While the scan is very precise, many indoor environments produce similar looking scans.

Loop closing is typically performed via explicit data association: recently observed landmarks are associated with previously mapped landmarks. Optimal data association is NP-hard, though a number of heuristics are often effective [46, 2]. Unfortunately, data association errors can lead to divergence of the map; this is combated by performing joint data association on large numbers of observations, which increases the computational costs.

An alternative approach relies on pose-to-pose matching, in which the global identities of locally-observed features are never explicitly computed. We consider two instances of this problem. The first is based on laser scan matching, in which case the match is the result of a numerical optimization that attempts to align two scans so that they overlap as much as possible. The second instance is based on visually tracked objects in the environment. Possible alignments between poses are recovered by attempting to align these locally tracked features.

In both of these cases, the computational burden of globally identifying all landmarks is eliminated. The trade-off is that these pose-to-pose alignments, while comparatively easy to compute, have a fairly high error rate. For example, failures can

be caused by matching two similar-looking (but physically distinct) features, or by getting stuck in a local minimum when attempting to align two laser scans.

This thesis describes a loop-closing verification algorithm that allows robust loop closing using these simple pose-to-pose correspondences. The algorithm does not attempt to globally register landmarks, and thus does not require their expensive-to-compute covariance matrix. The algorithm considers a set of pose-to-pose "hypotheses" and extracts the subset that is maximally self-consistent. The algorithm exploits the property that correct hypotheses must be mutually consistent (since there is only one true configuration), whereas incorrect hypotheses are often inconsistent.

In Chapter 3, we illustrate how this maximally self-consistent set can be computed from the dominant eigenvector of a carefully-constructed matrix of pair-wise consistencies. We evaluate our system using both laser and camera data, and briefly describe our method for obtaining pose-to-pose hypotheses.

1.4 Pose/Feature Graphs

Pose/Feature graphs are an intuitive representation of map optimization problems. The nodes of a pose/feature graph represent the positions whose exact location are of interest, while the edges represent information about the relative geometric position of those positions.

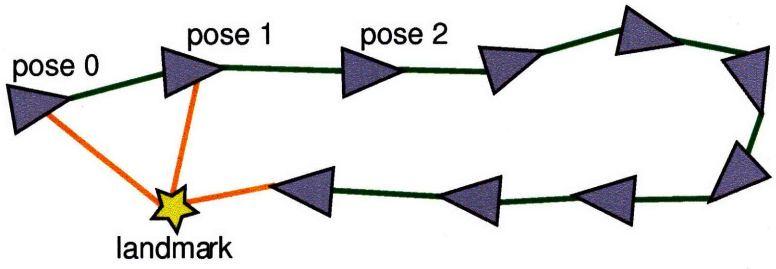

Figure 1-4: A Simple Pose/Feature graph. The robot's trajectory is discretized into a series of poses (triangles). These poses, as well as landmarks (stars), are nodes in the pose graph. Edges in the pose graph represent constraints between the positions of nodes, such as those derived from odometry (green) and a landmark detector (orange).

For example, the position of the robot at a particular point in time can be represented as a node in the graph. These types of nodes are known as *poses*. Features (or landmarks) in the environment can also be represented as nodes in the graph: when sensors detect a landmark, the sensor observation is represented as an edge connecting the landmark to the pose from which the observation was made. Fig. 1-4 illustrates a simple pose/feature graph; see Fig. 1-2 for a real-world example.

A graph containing both poses and features is a "pose/feature graph". A graph consisting of only poses and edges between poses is simply a "pose" graph.

1.5 Map Optimization

Loop closing adds edges to the graph, making it cyclic. Each edge relates the geometrical position of the two nodes it connects and has a cost function associated with deviations from that relationship. We now wish to find the maximum likelihood (minimum cost) position of each node. This is the map optimization step, and is what makes the map "line up" (see Fig. 1-5).

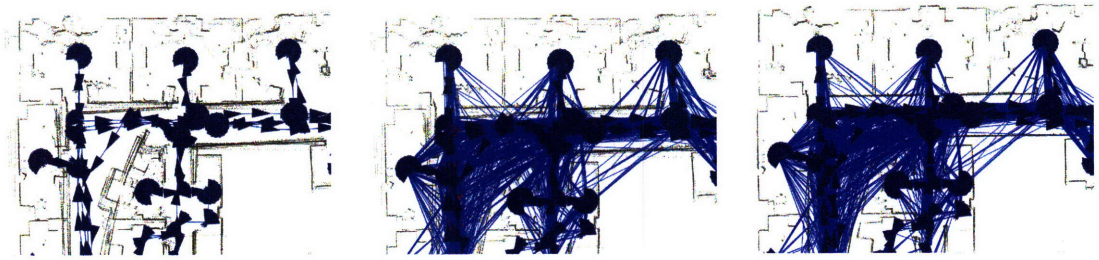

Figure 1-5: Map Building Example. Given the robot's trajectory (left), we must first identify loop closures (middle). Finally, the maximum likelihood positions of all the nodes, given the loop closures and the original trajectory, are computed (right). The result is a map that is much better than the map resulting from dead-reckoning.

The pose graph encodes a non-linear optimization problem: the goal is to estimate the position of all the nodes (the state variables) such that the constraints have the minimum possible error. The large number of state variables, the non-linear nature of the constraint equations, and the poor initial estimate (typically derived from dead-reckoning) combine to make the optimization problem difficult. Its importance and difficulty have made optimization a heavily researched topic, but performance on large problems (those with many nodes and edges) remain computationally difficult.

Robustness to noise is an important performance metric: if an algorithm converges to the wrong answer (a local minimum), it hardly matters how quickly it managed to compute it. Sensor noise increases the likelihood of an algorithm getting stuck in a local minimum, and the desire to use less expensive sensors (e.g., wireless RF ranging to base stations, radars, cameras) often results in noisier measurements. The robustness of optimization methods has not been a focus of previous research, despite its importance.

We describe an optimization algorithm based on stochastic gradient descent that computes updates to the state vector by considering a single constraint at a time. This yields fast updates that rapidly explore the state space, allowing the algorithm to escape local minima that slow or trap existing algorithms. A batch version of this algorithm (for off-line use) is described in Chapter 4, followed by a version suitable for on-line use in Chapter 5.

1.6 Contributions

We have identified two major challenges in robotic mapping: loop closing and map optimization. This thesis describes several contributions that demonstrably advance

the state of the art for both of these challenges. These contributions include:

- A place recognition system based on pair-wise correspondences that can robustly identify loop closures, even when many hypotheses are incorrect.
- A map optimization algorithm whose iterations depend on only a single constraint at a time. The algorithm is robust to optimization hazards such as local minima, producing good maps when existing algorithms fail. The algorithm is generally competitive with, or faster than, existing methods.
- An incremental on-line map optimization algorithm, based on the batch algorithm, that is well-suited for real-time use on robots.

These algorithms can construct maps with greater speed and robustness than existing algorithms. As a result, robots can successfully operate in larger, more complex, and more ambiguous environments than was previously possible.

CHAPTER 2

Preliminaries

This thesis is about how to construct accurate maps. There are several ways of describing this problem; in this thesis, we use the *pose graph* metaphor. This chapter describes pose graphs and then moves on to more mechanical matters such as our geometrical and mathematical conventions.

2.1 Pose/Feature Graphs

The position and orientation of a robot at a particular point in time is a *pose*, graphically denoted in this thesis as a triangle (see Fig. 2-1). Over time, the continuous trajectory of the robot is discretized into a set of *poses*. A landmark in the environment is graphically represented as a star.

Information relating the position of two nodes, along with the uncertainty of that information, is known as a *constraint*[1]. A constraint is represented as an edge in the pose graph. Constraints can express virtually any type of information, including a range constraint, bearing constraint, or a full rigid-body transformation. We model the uncertainty of these constraints as Gaussian, associating each with a covariance matrix.

This thesis focuses on two-dimensional pose graphs and rigid-body constraints, due to their ubiquity and applicability to many terrestrial mapping systems. We will also consider the case in which the robot is continuously moving and making new observations. This results in the addition of new nodes and edges to the graph. For simplicity, we assume that individual nodes do *not* move.

2.2 Graphical Models

Mapping problems can alternatively be described in terms of a graphical model of a Markov Random Field [30]. These graphical models are closely related to pose/feature

[1]The term "constraint" is used in some contexts to mean an inviolable criterion that any solution must satisfy exactly. In the map optimization context, a constraint is uncertain: violation of a constraint incurs a cost that is a function of the magnitude of the violation. It is usually impossible to satisfy all constraints exactly— instead, we wish to minimize the total cost of all constraints.

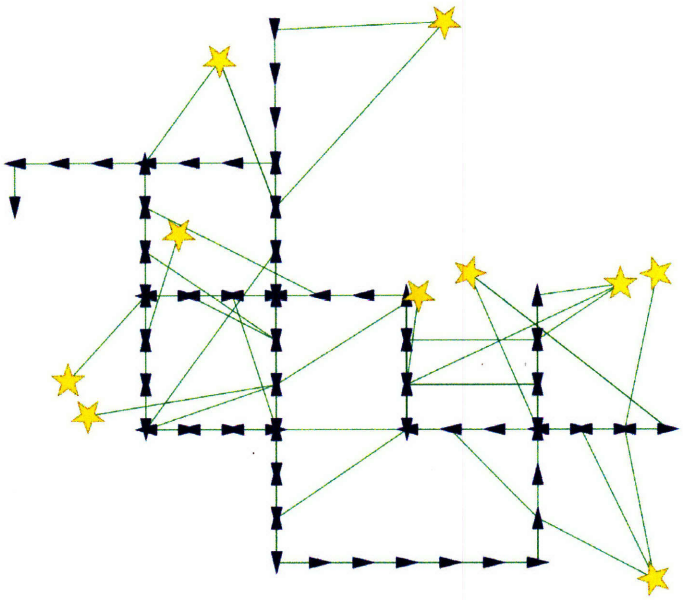

Figure 2-1: Sample Pose/Feature Graph. The robot trajectory is discretized into a series of poses (blue triangles) from which the robot observes landmarks (yellow stars). Both poses and landmarks are nodes in the graph. Graph edges represent information about the relative position and orientation of the two nodes.

graphs. The nodes of a pose/feature graph are the same as the random variables of a graphical model. A constraint between two nodes in a pose/feature graph implies a dependence between the same two nodes in a graphical model, which is also represented by an edge.

The difference between the two is a primarily an issue of connotation. A pose/feature graph describes the "experience" of a robot: its entire trajectory and its entire set of observations. It represents a complete statement of the structure of a particular mapping problem. For every such pose graph, there is an exactly equivalent graphical model.

In contrast, graphical models describe a joint probability distribution over the nodes in the graph. For example, the robot trajectory can be marginalized out. The resulting graphical model has no poses— only landmarks that are connected directly to each other. Naturally, landmarks cannot observe other landmarks, and so a marginalized graphical model does not correspond to a real set of observations. The use of a graphical model also implies a particular set of inference methods (e.g., belief propagation).

In this thesis, we use the term pose/feature graph to refer to a complete statement of the mapping problem (i.e., one in which no variables have been marginalized out).

2.3 Geometrical Conventions

A 2-D rigid body transformation is parameterized by three values, a translation in $\hat{x}$, a translation in $\hat{y}$, and a rotation θ. Subscripts will be used to identify particular sets

of parameters (e.g., x_A, y_A, and θ_A). These parameters can be used to compute a 3×3 transformation matrix, which we will identify by a capital letter. For example:

$$A = \begin{bmatrix} \cos(\theta_A) & -\sin(\theta_A) & x_A \\ \sin(\theta_A) & \cos(\theta_A) & y_A \\ 0 & 0 & 1 \end{bmatrix} \tag{2.1}$$

A point p, expressed in homogeneous coordinates [27] (e.g., $p = [x\ y\ 1]^T$) is transformed by A by simply $p' = Ap$. Transforming p first by B, then by A is computed by $p' = ABp$. When dealing with chains of rigid-body transformations, such as the path of a robot, we will write our transformations such that the rigid-body transformations project a point (whose coordinates are expressed relative to one pose) to a point whose coordinates are expressed relative to the *previous* pose. Consequently, the order of multiplication will be the same as the temporal order of the links. If the robot visits poses a and b (see Fig. 2-2), the matrix relating b to the global frame is $B = AT$ (not TA).

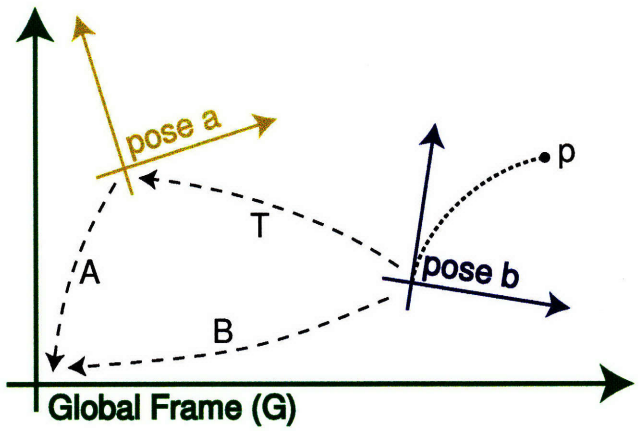

Figure 2-2: Coordinate transformations. Each robot pose represents a local coordinate system which can be related to the global coordinate by a rigid-body transform. Two local coordinate systems can also be related, e.g., $B = AT$. Point p, expressed with respect to coordinate frame B, can be projected to coordinate frame A by Ap and to the global coordinate frame by either Bp or ATp.

The product of two rigid-body transformations can be expressed directly in terms of their parameters. For example, consider $B = AT$:

$$\begin{aligned} x_B &= \cos(\theta_A)x_T - \sin(\theta_A)y_T + x_A \\ y_B &= \sin(\theta_A)x_T + \cos(\theta_A)y_T + y_A \\ \theta_B &= \theta_A + \theta_T \end{aligned} \tag{2.2}$$

It is also convenient to solve $B = AT$ for T; this will arise in the prediction steps of the Extended Kalman Filter and in the residual computations for non-linear

optimization:

$$\begin{aligned} x_T &= \cos(\theta_A)(x_B - x_A) + \sin(\theta_A)(y_B - y_A) \\ y_T &= -\sin(\theta_A)(x_B - x_A) + \cos(\theta_A)(y_B - y_A) \\ \theta_T &= \theta_B - \theta_A \end{aligned} \tag{2.3}$$

Finally, it is also useful to invert a rigid-body transformation. This can also be done directly via the parameters. Letting $T = A^{-1}$, we have:

$$\begin{aligned} x_T &= -\cos(\theta_A)x_A - \sin(\theta_A)y_A \\ y_T &= \sin(\theta_A)x_A - \cos(\theta_A)y_A \\ \theta_T &= -\theta_A \end{aligned} \tag{2.4}$$

Constraints are uncertain quantities, having arisen from noisy sensor observations. This uncertainty can be represented as a probability distribution over the parameters of the rigid-body transformation. We assume that the probability distribution can be well-approximated as a multi-variate Gaussian distribution. This common approximation is motivated both by the quality of fit and by the computational conveniences that result. This distribution can be written in terms of the parameters of the rigid-body transformation:

$$\Sigma = \begin{bmatrix} \Sigma_{xx} & \Sigma_{xy} & \Sigma_{x\theta} \\ \Sigma_{yx} & \Sigma_{yy} & \Sigma_{y\theta} \\ \Sigma_{\theta x} & \Sigma_{\theta y} & \Sigma_{\theta\theta} \end{bmatrix} \tag{2.5}$$

When performing the operations on transformations, we must also compute the covariance of the new transformations. This is done by computing the Jacobians of the output as a function of the inputs, and summing the scaled covariances of the inputs. For example, consider the composition of two rigid-body transformations, $C = AB$. Let Σ_A be the covariance of the transformation A, and J_A be the Jacobian of the parameters of C with respect to the parameters of A. The covariance of C is then:

$$\Sigma_C = J_A \Sigma_A J_A^T + J_B \Sigma_B J_B^T \tag{2.6}$$

Σ_A is the covariance of transformation A, J_A is the Jacobian of the parameters of C with respect to the parameters of A, and J_B is the Jacobian of the parameters of C with respect to the parameters of B.

2.4 Constraints

A constraint (or edge) is a set of simultaneous equations f that relates the state variables x to some observed quantity z. For a particular constraint i, we write:

$$z_i = f_i(x) \tag{2.7}$$

The difference between the observed and predicted values of an observation is the *residual* r_i:

$$r_i = z_i - f_i(x) \tag{2.8}$$

When the residual is scaled by the constraint's confidence Σ_i^{-1}, the result is the χ^2 error. Note that the confidence is the inverse of the covariance matrix.

$$\chi_i^2 = (z_i - f_i(x))^T \Sigma_i^{-1} (z_i - f_i(x)) \tag{2.9}$$

As with virtually all previous work on robotic mapping, we assume that each edge is uncorrelated with other edges in the graph. In many cases, this is a reasonable assumption: errors in sensors are often well-modelled as additive independent noise. In reality, however, sensor observations are often correlated, at least to a minor degree. For example, a laser range scanner might be sensitive ambient temperature fluctuations, and a vision-based system is sensitive to intrinsic calibration errors. In both of these cases, multiple observations are affected by a single error source. This makes the measurements correlated. Since this effect is typically minor, we neglect it.

2.4.1 Common Constraint Forms

Different types of sensors produce different types of geometrical information. A lidar, for example, can measure the relative position and orientation of features with respect to the robot. Such an observation fully specifies the position of the feature with respect to the robot, and is called a rigid-body constraint. The observation equations for a rigid-body constraint are given in Eqn. 2.3.

Monocular cameras can only measure the bearing to a feature. If the bearing of node B is observed from node A (a "bearing-only" constraint), we have:

$$z_i = \arctan2(b_y - a_y, b_x - a_x) \tag{2.10}$$

Various radio-frequency and acoustic sensors are capable of measuring the range between a feature and the robot, but cannot determine the bearing. A range-only constraint is described by:

$$z_i = ((b_x - a_x)^2 + (b_y - a_y)^2)^{1/2} \tag{2.11}$$

2.4.2 Linearization

As seen above, the observation equations are not generally linear in the state variables. However, a serviceable approximation can usually be obtained by linearizing

the equations around the current state estimate x_0:

$$F_i = f_i(x_0) \tag{2.12}$$

$$J_i = \left.\frac{\partial f_i}{dx}\right|_{x_0} \tag{2.13}$$

$$f_i(x) \approx F_i + J_i(x - x_0) \tag{2.14}$$

We can write the χ^2 error in terms of this linearization. First, we write the residual (error) of the constraint as $r_i = z_i - f_i(x_0) = z_i - F_i$. We also let $d = (x - x_0)$. These substitutions allow us to rewrite Eqn. 2.9 as:

$$\chi_i^2 \approx (J_i d - r_i)^T \Sigma_i^{-1} (J_i d - r_i) \tag{2.15}$$

The χ^2 error for the entire pose/feature graph is the sum of the χ^2 error of each edge, since each edge is assumed to be statistically independent:

$$\chi^2 = \sum_{i \in \text{edges}} \chi_i^2 \tag{2.16}$$

If we stack the J_i and r_i matrices, and create a block-diagonal matrix from the Σ_i^{-1} matrices, we can write:

$$\chi^2 \approx (Jd - r)^T \Sigma^{-1} (Jd - r) \tag{2.17}$$

where

$$J = \begin{bmatrix} \cdots & J_1 & \cdots \\ \cdots & J_2 & \cdots \\ & \vdots & \end{bmatrix} \quad r = \begin{bmatrix} r_1 \\ r_2 \\ \vdots \end{bmatrix} \quad \Sigma^{-1} = \begin{bmatrix} \Sigma_1^{-1} & & \\ & \Sigma_2^{-1} & \\ & & \ddots \end{bmatrix} \tag{2.18}$$

As a preview of the optimization methods to come, we can differentiate Eqn. 2.17 with respect to d to find the value of d that minimizes the χ^2 error:

$$\frac{\partial \chi^2}{\partial d} = 2J_i^T \Sigma_i^{-1} J d - 2J_i \Sigma_i^{-1} r_i = 0$$

$$d = (J^T \Sigma^{-1} J)^{-1} J^T \Sigma^{-1} r \tag{2.19}$$

The value d represents a change in the state variable that will minimize the χ^2 error. However, inverting $(J^T \Sigma^{-1} J)$ is impractical in many cases. This has spurred the development of many algorithms, including those in Chapter 4 and 5.

Due to the non-linearity of the constraints, computing d will generally not reduce the error to the global minimum in a single iteration: several iterations are often required.

2.4.3 Interpretation of the Chi-Squared Error

The χ_i^2 error is the square of the Mahalanobis distance, which has a natural geometric interpretation: it is the number of standard-deviations of error between the predicted and observed values. Mahalanobis distances are used as plausibility tests: values above three occur with a probability of about 0.2% and thus strongly indicate either an error in z (e.g., a measurement outlier), the confidence Σ_i, or the state.

The number of degrees of freedom k of a graph is equal to the number of independent components (the number of observation equations) minus the size of the state vector [73]. For example, a 2D pose graph with N nodes and M edges has $3M - 3N$ degrees of freedom, assuming that each constraint is a rigid-body transformation.

The chi-square distribution gives the probability of the maximum likelihood map having a given χ^2 error as a function of the number of degrees of freedom. The chi-square distribution for k degrees of freedom has mean k and variance $2k$.

We can compare our χ^2 error to the value predicted by the chi-square distribution. If our χ^2 error is highly improbable, it indicates one of two potential problems. The first possibility is a mismatch between the sensor noise model and the actual noise of the sensor. For example, if the sensor noise model indicates a much lower noise level than is present in the actual observations, the χ^2 error will be larger than expected. The second possibility is that our current state estimate is not close to the maximum likelihood map. In this case, it is an indication that we should attempt to further optimize the map.

When comparing the performance of optimization algorithms across a number of different optimization problems, the dependence of the χ^2 error on k makes it difficult for a human to determine whether the χ^2 error is probable or not. Instead, we consider the probability density function for χ^2/k: its expectation is always 1.0 and its variance is $2/k$. Consequently, for large k, the distribution will be ever-more tightly centered around 1.0.

If we trust our sensor noise model, a χ^2/k value near 1.0 indicates that the state estimate is near the maximum likelihood solution. Larger values indicate that additional optimization is needed.

Conversely, if the optimization algorithm has converged to a value that is far from 1.0, it indicates a problem with the sensor noise model. If the observation covariance matrices are incorrectly scaled by a factor α, the normalized χ^2 will be $1/\alpha$.

Because of these useful properties, the χ^2 values reported in this thesis are always normalized by the number of degrees of freedom.

2.4.4 State Squared Error

Different optimization algorithms will produce different estimates of the global minimum x_{opt} over time. Evaluating these algorithms requires us to be able to determine which estimates are better than others.

The χ^2 error gives us a means to compare different estimates, and it is a good measure in that it is the quantity that the optimization algorithms are explicitly attempting to minimize. However, from a practical perspective, we wish to know how

similar the *maps* are to the optimal map: the χ^2 error tells only about the errors of the constraints, not the positions of the nodes.

As Fig. 2-3 illustrates, χ^2 error is not necessarily a good indicator of map quality. In this figure, a map with a χ^2 error of 3.6 is of much lower quality than a map with a χ^2 error of 33.2.

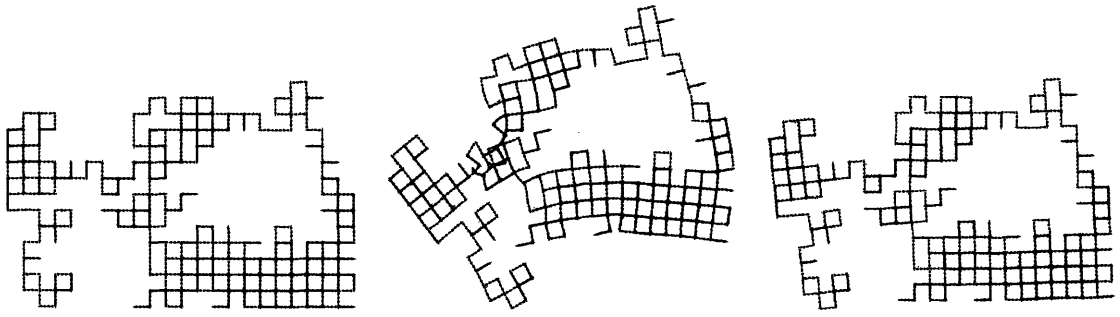

Figure 2-3: State Squared Error Example. χ^2 error is not always a good indicator of map quality. From a practical perspective, the right map is superior to the middle map, despite the fact that the middle map has one-tenth the χ^2 error. The SSE metrics, in contrast, clearly identify the map on the right as being better than the middle map.

The χ^2 error is not necessarily a good measure of map quality because different map distortions affect the χ^2 error to different degrees. In particular, the non-linear nature of the mapping problem results in local minima and long valleys in which the χ^2 error is both small and slowly changing. It is thus possible to have a small χ^2 error and yet have a map that is very different from the maximum likelihood map. As we will see in Chapter 4, this is one of the reasons that many map optimization algorithms either get stuck or converge very slowly. The opposite is also true: it is possible to have a high χ^2 error and be very close to the maximum likelihood map.

When x_{opt} is known, we can compute the distance of the current graph configuration x from the optimal value x_{opt}. The metric $||x - x_{opt}||^2$ is not a good choice, however, since it intermixes components of the state vector having different units (positions in meters and rotations in radians). Instead, we compute two different metrics so that positions and rotations are considered separately. In both cases, we use the mean squared error, but to avoid confusion with the χ^2 error (which is the squared error of the constraints), we will call these metrics the "state squared error", reflecting that they are the squared error of the state variables.

We define SSE_{xy} to be the mean squared error of the (x, y) positions of each node in the pose graph relative to x_{opt}, and SSE_θ to be the mean squared error of the θ positions of each node in the pose graph.

Implementing this error metric presents a minor wrinkle, however: in most map building applications, the position of the map has no global "anchor": translating and rotating the whole map has no effect on errors of node-to-node constraints, but has an obvious effect on the (x, y, θ) coordinates of those nodes. In other words,

projecting the whole map x by a rigid-body transform will not affect the χ^2 error but it will affect the SSE metrics as defined above.

Consequently, it is desirable to modify our definitions of the SSE metrics so that they are invariant to rigid-body transformations of the whole map. To do this, we introduce another rigid-body transformation T that we will apply to the map before computing the mean squared error of x with respect to x_{opt}. We will pick T to be the rigid-body transformation that minimizes SSE_{xy}. Minimizing SSE_{xy} has a tendency to minimize SSE_θ, but does not do so exactly. We have chosen to minimize SSE_{xy} rather than SSE_θ since all landmarks have a position, whereas some types of point features do not have an orientation.

The transform T can be computed in time proportional to the number of nodes using an algorithm developed by Horn [28]. Let a be the set of points corresponding to the positions of nodes in our current state estimate, and b be the positions of those points in the maximum-likelihood (or ground truth, if available) configuration. We write the position of the i^{th} point of a as (a_x^i, a_y^i). Let $\overline{a_x}$ be the average x coordinate of the points in set a. We can compute the rigid-body transformation T in terms of intermediate variables M and N:

$$M = \sum_i (a_x^i - \overline{a_x})(b_y^i - \overline{b_y}) - (a_y^i - \overline{a_y})(b_x^i - \overline{b_x}) \tag{2.20}$$

$$N = \sum_i (a_x^i - \overline{a_x})(b_x^i - \overline{b_x}) + (a_y^i - \overline{a_y})(b_y^i - \overline{b_y}) \tag{2.21}$$

The parameters of the rigid-body transform T, which optimally projects points b to align with a (minimizing the sum of squared distances between a_i and Tb_i), are then given as:

$$T_\theta = \arctan2(M, N) \tag{2.22}$$

$$T_x = \overline{b_x} - \overline{a_x}\cos(T_\theta) + \overline{a_y}\sin(T_\theta) \tag{2.23}$$

$$T_y = \overline{b_y} - \overline{a_x}\sin(T_\theta) - \overline{a_y}\cos(T_\theta) \tag{2.24}$$

The SSE error metrics are complementary to χ^2 error. Map optimization is defined in terms of minimizing the χ^2 error, and so it remains an important consideration. However, different optimization algorithms will take different paths through the state space as they minimize the χ^2 error: the SSE measures give us insight into how rapidly these paths lead to x_{opt}, independently of the χ^2 error of those paths.

CHAPTER 3

Loop Closing

3.1 Introduction

Place recognition is a central problem in many robot applications. Whether a robot is delivering a package or guiding a tour, being able to recognize a location is critical. With a place recognition system, a robot could be trained to identify rooms or even particular areas inside rooms (e.g., the robot's recharging station).

In mapping applications, place recognition is especially important. Without the ability to recognize previously-visited places, the position uncertainty of a robot increases without bound due to the ceaseless accumulation of dead-reckoning error. Place recognitions serve as constraints on the motion of the robot, allowing a correction of its dead-reckoning error. In the mapping context, place recognition is called "loop closing": if the robot drives in a loop and back to its starting position (and recognizes it), the robot "closes" the loop. The quality and accuracy of the map is a function of the quantity and quality of the loop closures.

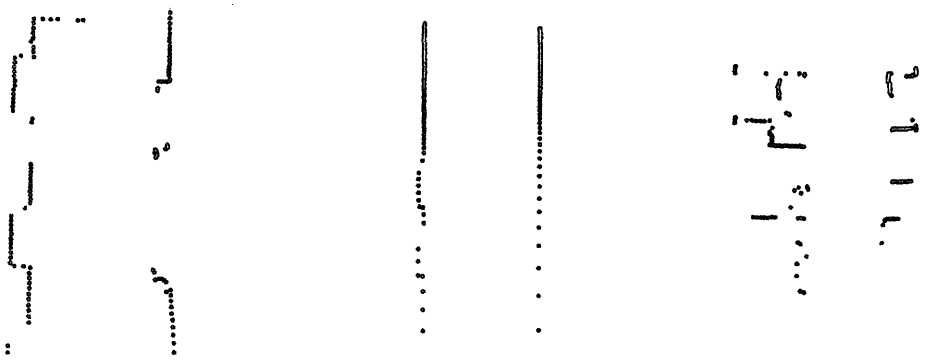

Figure 3-1: Three laser scans from the CSAIL dataset. Some environments present rich alignment cues (left: an elevator lobby). Incorrect matches arise from both spartan areas (middle: a corridor) and from repetitive/cluttered areas (right: office cubicles).

Loop closing is difficult for several reasons. Algorithms must be robust to modest changes in the environment caused by things like moving chairs or humans. Sensing limitations play a critical role. The data produced by laser scanners, for example, is metrically accurate (see Fig. 3-1), but it is often difficult to distinguish one room

from another: indoor environments tend to be composed of sets of straight walls and corners whose appearances are similar.

Camera-based systems, like those using the SIFT feature detector [38], provide a richer description of the environment (able to distinguish different posters on a wall, for example), but are still susceptible to ambiguity. Different offices, for example, may contain the same type of chair, and different outdoor environments contain the same types of street signs. The length of these feature vectors has a major impact on both total memory usage and the time required to index and search for similar features. Consequently, it is desirable to use shorter descriptors (such as PCA-SIFT [32]). Reducing the descriptor size generally increases the error-rate of matching: matching algorithms that are robust to higher error rates are thus very desirable.

Some authors have proposed using globally unique beacons, such as RFID tags [33], wireless networking base stations [25], or specialized sensor networks [54]. The resulting descriptors are concise and globally unique, but require the deployment of beacons and are not broadly applicable to all robotics applications.

In outdoor settings, GPS can provide a positional cue, however the accuracy of GPS (especially in consumer-grade devices) is insufficient for many applications. For example: while GPS can usually tell a car what *road* it is on, it can't reliably indicate which *lane* it is in. For many applications, greater accuracy is needed. This additional accuracy can be achieved by recognizing places: each recognition allows the map to be refined.

In a map-building context, place recognition is known as "loop closing", reflecting the common case in which a robot travels around a large loop and returns to its original position (thus "closing" the loop). However, any place recognition adds an edge to the pose graph, creating a new cycle— the robot does not need to physically travel in a circle in order for a loop closure to occur.

Loop closing has been approached in a variety of ways. The discovery of topological loop closures has been performed with odometry alone using hidden Markov models [61]. Folkesson [17] uses negative information to validate loop closure hypotheses, while Kuipers uses bootstrap learning to identify places [35]. Bosse validated his loop closures using a loop-consistency test [5].

In robotic mapping applications, place recognition has generally been cast in terms of explicitly associating landmarks with previously observed landmarks. This process is called "data association". Data association on one (or just a few) features at a time is highly susceptible to errors, and data-association errors can cause catastrophic divergence of the map.

In an attempt to decrease their error rate, data association algorithms typically perform association on groups of observations at once. In analogy, it is easier to recognize astronomical constellations than single stars. Given the mean and covariance of a set of tracked landmarks, the probability of a set of observations matching that set can be computed. However, as the number of landmarks grows, the number of possible assignments grows exponentially. Many authors have proposed solutions to this problem, providing reasonably efficient search heuristics that often perform well [46, 2]. Error rates can also be addressed by revisiting data associations previously made [26].

Even if the number of landmarks is kept small enough to ensure good performance, such methods still require that the position estimates and covariances of landmarks be maintained. This is a major imposition for robots that do not otherwise care about landmarks and would not otherwise maintain their covariance. It is also an imposition for mapping robots that use an optimization method in which landmark covariance data is not readily available, such as FastSLAM [44], information-form filters [70, 13, 76], non-linear optimization methods (Gauss-Seidel relaxation [12], Multi-Level Relaxation [20]), and the methods described in Chapters 3 and 4 and its derivatives [24, 23].

In this chapter, we describe an approach that attempts to close loops by performing a number of simple pose-to-pose matches. Given a number of naive matches ("hypotheses"), our goal is to identify those that are correct. A *set* of these hypotheses has an interesting property: the correct hypotheses all agree with each other (since there is only one true configuration), whereas the the incorrect hypotheses tend to be incorrect in different ways (and thus do not agree with each other). Our algorithm exploits this property by computing the subset of hypotheses that is most self-consistent. In comparison to the exponential cost of performing data association amongst N features, the cost of our approach is $O(N^2)$ in the number of hypotheses. Our approach is similar to CCDA [2], except that our notion of consistency is not limited to discrete boolean quantities.

Our approach does not require landmarks to be tracked, and thus does not require covariance information about the landmarks or a data association algorithm. All that is required is the ability to compute possible rigid-body transformations between two sensor scans. Our approach can be viewed as a outlier-rejection step, extending previous pose-to-pose methods [41].

A major advantage of our proposed method is that it computes the "second-best" set of mutually-consistent hypotheses as well. When using RANSAC [15] or Joint Compatibility Branch and Bound (JCBB) [46], the second-best solution is generally a trivial variation on the best solution: it is thus not very informative. In contrast, the second-best solution reported by our new approach is *orthogonal* to the best solution– i.e., represents a substantially different interpretation of the data. Our method allows the quality of these two solutions to be quantitatively compared, allowing ambiguous solutions to be safely discarded. (If the data can be equally well explained in two different ways, it is not safe to trust either explanation.) In these ambiguous cases, our method will occasionally reject hypotheses that are correct. Rejecting correct hypotheses has an impact on the quality of the final map (since some information is lost), but it does not result in the sort of rapid divergence that accepting an incorrect hypothesis can cause.

This chapter presents our algorithm and results using both camera-derived features and laser-derived features, illustrating the flexibility of the method.

3.2 Method Overview

Our loop-closing algorithm operates in several stages. We first generate a number of hypotheses. Given two robot positions (at different points in time), we can estimate the probability that those two positions are close enough together for them to have observed the same environmental features. If so, we attempt to generate a hypotheses relating the position of those poses. The particular hypothesis-generation algorithm varies according to the types of sensors being used: cameras and lidars, for example, produce very different types of data that require different matching algorithms.

We group the resulting hypotheses into "hypothesis sets". Our task is to identify those hypotheses that are correct. By considering pairs of these hypotheses at a time, we will identify which hypotheses are pairwise-compatible. Using an algorithm based on the spectral properties of the pair-wise compatibility matrix, we can identify the set of hypotheses that are most compatible with each other.

3.3 Hypothesis Generation

3.3.1 Determining when poses overlap

The first step in identifying loop closure hypotheses is to identify those poses that might contain overlapping sensor readings. Given a pose a, we wish to find all other poses in the graph whose sensor readings might overlap those from pose a. This could be determined from the posterior distribution (Eqn. 1.1) by marginalizing out pose a and any landmarks. Instead, we describe a conservative approximation that is much easier to compute. In particular, our method does not require knowledge of the posterior pose and landmark covariances.

Dijkstra Projection

We estimate the relative positional uncertainty between nodes a and b by searching for the minimum-uncertainty path through the pose/feature graph that connects them. This path can be found by using variant of Dijkstra's Algorithm [55]:

In order to compare the uncertainty of two paths, we use the determinant of the paths' covariance matrices. Geometrically, the determinant can be interpreted as the area of the covariance ellipse, and is thus proportional to the area of the region in which pose b is likely to be found given the position of pose a. Larger search areas generally result in more computational effort. Consequently, picking the path with the smaller determinant generally minimizes the amount of computation needed to find pose b.

The Dijkstra Projection is conservative in the sense that it will overestimate the uncertainty. For example, if a pose graph contains multiple independent paths between nodes a and b, the uncertainty of their relative positions will be less than that computed by the Dijkstra projection. Overestimating the uncertainty will result in larger search areas and thus greater computational effort when searching for loop closures.

Algorithm 1 Dijkstra Projection from node a

```
Initialize a set of paths P = {}
for all edges e leaving node a do
  P = P ∪ e
end for
visited(a) = true
while some nodes unvisited do
  find p, the minimum-uncertainty path in P
  if !visited(destination(p)) then
    optimal-path(b) = p
    visited(b) = true
    for all edges e leaving node b do
      P = P ∪ compose-transforms(p, e)
    end for
  end if
end while
```

However, even if the exact covariance could be easily computed, the search areas are often artificially enlarged in order to ensure that loop closures can be found whenever possible. Over-confident sensor-noise characterizations, for example, cause the "exact" covariance to be too small, and can result in missed loop closures. If loop closures are consistently missed, the quality of the map will suffer. In short, the conservatism of the Dijkstra algorithm does not typically cause any problems.

Sensor range

In addition to the relative positions (and uncertainties) of two nodes, we must know the sensor range in order to determine whether two sensor scans might overlap: large sensor ranges make it more likely that nodes will have overlapping fields of view. In indoor environments, the effective sensor range varies dramatically depending on the environment: a robot may only be able to sense a few meters when inside an office, but may be able to sense tens of meters outdoors.

We model the sensor range at a particular node a as a circle with center c_a (whose coordinates are expressed in terms of a) and radius r_a. We compute c_a as the centroid of the landmarks observed, with r_a being the average distance of the landmarks to the centroid. In order to test whether sensor observations at node b might overlap with those at node a, we similarly compute a circle for node b with center c_b and radius r_b, but project c_b into the same coordinate system as c_a using the Dijkstra projection between poses a and b. Let Σ_{ab} the uncertainty of the transformation from a to b as

given by the Dijkstra projection. We can write the Mahalanobis distance as:

$$d = (c_b - c_a) \tag{3.1}$$

$$s = \max(0, ||d|| - r_a - r_b)\frac{d}{||d||} \tag{3.2}$$

$$mahl = s^T \Sigma_{ab}^{-1} s \tag{3.3}$$

In the equations above, d is the Euclidean vector between the circle centers, and s is the separation vector between their sensor ranges (the vector d reduced in magnitude by the sum of the sensor ranges). If the resulting Mahalanobis distance is less than 3, it is reasonably likely that the two nodes have overlapping sensor data. In aggregate, our approach for computing the pair-wise "compatibility" is similar to the metric used by GraphSLAM [72], with the improvement that we explicitly consider sensor range. Our use of these pair-wise compatibilities differs, however.

3.3.2 Searching for a match

Given a prior on the relative position between a and b (as computed by the Dijkstra projection), and sensor observations from pose a and b, the task is now to compute a rigid-body transformation that is likely to correctly align a and b. The techniques for doing this are sensor-dependent; we have implemented both a vision-based method and a laser-based method. These are described in more detail in Appendix A, but are summarized here.

Vision-derived Local Landmark Matching

Our vision data sets were collected during the 2007 DARPA Grand Challenge National Qualifying Event by Team MIT's vehicle, Talos [37]. Our vision data sets use SIFT features extracted from an array of calibrated cameras providing a nearly 360 degree field-of-view. Note that these cameras were essentially monocular: their fields of view did not significantly overlap.

In addition to cameras, Talos had a high-end inertial measurement unit (IMU)/GPS system giving it very good positioning capabilities. In our experiments, we evaluate the performance of our algorithm on both the high-quality IMU data as well as artificially degraded data that is representative of a more typical vehicle.

SIFT features are extracted from each camera frame and are matched against a set of active "tracks". A track is comprised of a SIFT feature descriptor and a list of the observations of that feature (the position of the car and the bearing to the feature).

Periodically, the measurements for each track are examined to see if the position of the SIFT feature can be determined. Since each observation of a feature consists of only a bearing to the feature, the feature must be observed from several different positions in order to compute its position. Each bearing to the feature corresponds to a ray: when the rays from different observations intersect, we can estimate of the location of the feature. Once the feature is localized, the track is reset so that the

Figure 3-2: Overhead view of vision dataset area. The car made three counter-clockwise loops, with each loop approximately 610 m in length.

feature can be independently localized again and again. The result of this process is a sequence of independent rigid-body transformations relating the vehicle to the feature.

At this point, we have a list of SIFT landmark observations (rigid-body constraints) for every node in the graph. In order to find the rigid-body transformation between two nodes, we compute the matches between the SIFT landmarks of the two nodes. These SIFT correspondences are computed based on the SIFT descriptors. Given two sets of matching SIFT features, we compute a rigid-body transformation that best aligns those landmarks. We can score each rigid-body transformation by counting the total number of landmarks from node b that are projected near their counterpart in node a; using this metric, we find the best rigid-body transform using RANSAC [15].

The rigid-body transform output by RANSAC is a hypothesis relating nodes a and b. The quality of this hypothesis is strongly dependent on the reliability of matching SIFT features based on their descriptors. When the whole 128-dimensional SIFT descriptor is used, these SIFT matches are often correct, leading to good rigid-body transformations. As the descriptors are reduced in size, however, the transforms become increasingly unreliable.

Another source of error results from correct but undesirable SIFT matches. For example, portions of the car are visible in some cameras: these produce SIFT observations that move with the car. While these obvious parts of the car could be

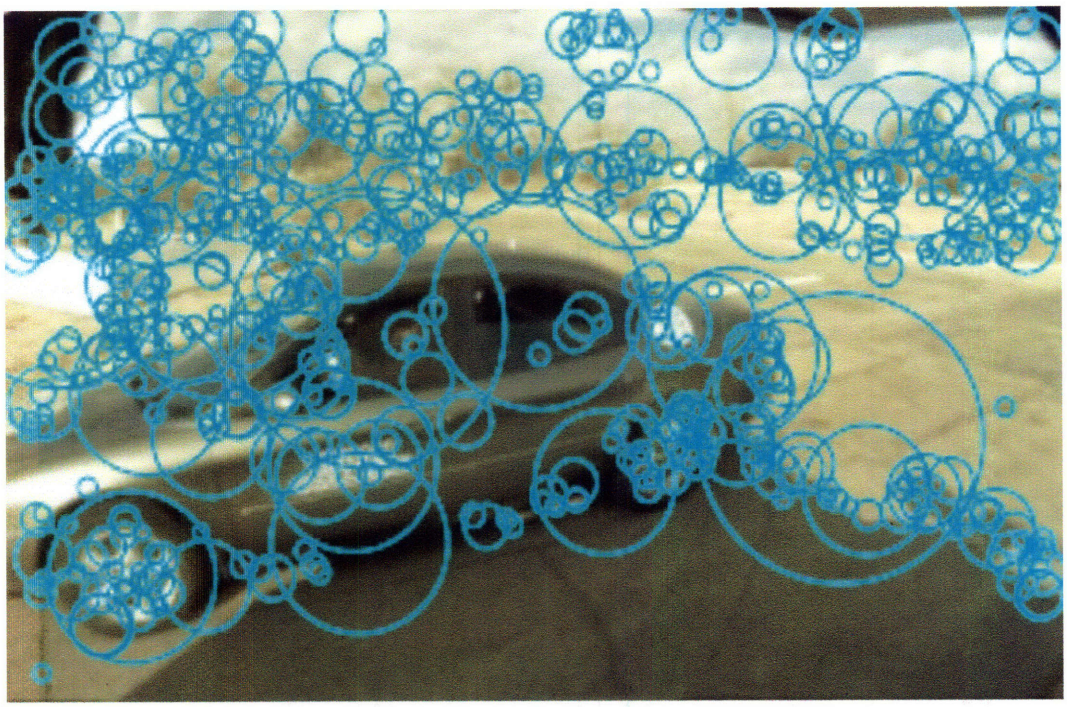

Figure 3-3: Sample SIFT detections. Cyan circles indicate SIFT features. Note how the other car, the shadow cast by our own car, and portions of our own car (black regions on left and top) cause undesirable SIFT detections.

masked out, other related effects are difficult to mask. For example, as the car drives around, it casts a shadow on the ground. This shadow generates many SIFT detections. Other moving cars can also cause undesirable SIFT matches. Since many of these sources of error are unavoidable, we did not attempt to mask any of them—instead, we let our proposed method detect and correct the resulting errors.

Laser Matching

We have processed a number of datasets with laser scanner data, including several common mapping benchmarks. These datasets were collected indoors using a single scanner with a 180-degree field-of-view. The complexity of these datasets varies significantly in terms of the noise of the range measurements, the rate at which observations are made, and the complexity and size of the environment.

Line features are extracted from individual point scans. Two correspondences between node a and b (two lines from each pose) are used to compute a rigid-body transformation. This rigid-body transformation is then scored by spatially correlating the points from b with a rasterized version of the points from a. The rigid-body transformation yielding the highest correlation is then subjected to a brute-force refinement step, in which the rigid-body transformation is numerically optimized.

The result is a rigid-body transformation T that causes the points from scan b to "line up" with the points from scan a. If the Mahalanobis distance of T is less than a threshold (typically 3) from the prior derived from the Dijkstra projection, T is output as a hypothesis.

In indoor environments (where there is often repeated structure), the probability of an incorrect match increases with the uncertainty of the prior. Failures can occur when the line correspondences are incorrect but happen to lead to a good correlation.

3.4 Hypothesis Sets

A single hypothesis by itself is impossible to validate. However, if two hypotheses are considered jointly, it is possible to test them against each other, even if those two hypotheses relate different pairs of nodes.

3.4.1 Pairwise Consistency

Given two hypotheses, we can construct a closed loop comprised of two hypotheses plus two additional rigid-body transformations obtained from Dijkstra projections (see Fig. 3-4).

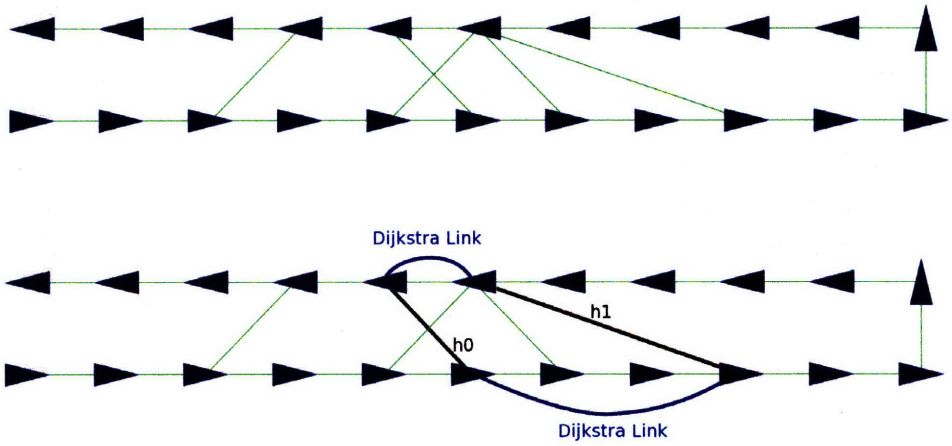

Figure 3-4: Hypothesis Set Loop. A hypothesis set contains a number of hypotheses (top). Given a pair h_0 and h_1 of them (bottom), a loop can be formed by adding a pair of Dijkstra links. The rigid-body transformation around this loop is compared to the identity transform. If either hypothesis is incorrect, it is unlikely that the resulting transformation will be close to the identity transform.

Naturally, the rigid-body transformation around any closed loop *should* be the identity matrix. Due to measurement errors (even for "good" hypotheses), it isn't ever *quite* the identity matrix, but this uncertainty is captured by the uncertainties associated with the hypotheses. Thus we can compute the rigid-body transformation around the loop (including its net covariance), and compute the probability that the resulting rigid-body transformation is actually the identity matrix. This probability is the pair-wise consistency of the two hypotheses. If both hypotheses are correct, it is likely that the probability will be large. However, if either hypothesis is wrong, the rigid-body transformation of the loop will not be the identity matrix; this results in a low probability.

3.4.2 Finding the optimal subset

Given a set of N hypotheses, we compute the pair-wise consistency for each pair, yielding an $N \times N$ "consistency" matrix A. We now wish to find the subset of hypotheses that is maximally self-consistent. In other words, we wish to find a subset of hypotheses whose pair-wise consistency is, on average, the greatest.

A hypothesis set can be viewed as a graph, with each hypothesis represented as a node. The (weighted) adjacency matrix of this graph is the pair-wise consistency matrix A.

Our solution is based on our previous work, an algorithm called "Single Cluster Graph Partitioning" (SCGP) [51, 49, 4]. SCGP is a graph partitioning method that attempts to find a single cluster of well-connected nodes, rejecting other nodes.

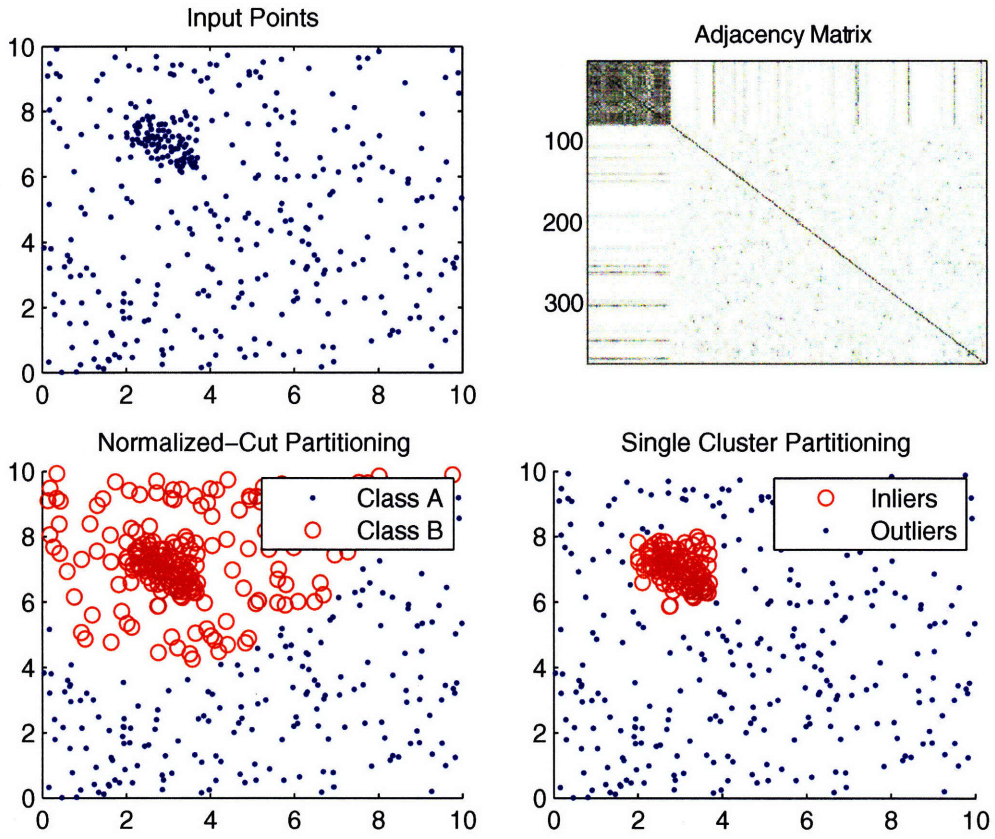

Figure 3-5: SCGP versus Normalized Cuts. Given a set of (x, y) points, each representing a hypothesis (top left), the goal is to identify the subset of hypotheses that forms the maximally self-consistent cluster. The adjacency matrix is plotted such that the desired points have small indices: their higher-than-average pair-wise consistency is clearly visible (top right). Our method, SCGP, produces the desired segmentation (lower right). Normalized-Cuts, also based on spectral clustering, uses a different clustering metric which results in a poor solutions on this type of problem.

Previous spectral clustering methods like Normalized-Cuts attempt to divide the graph into two parts such that clusters have both high intra-cluster consistency and low inter-cluster consistency. This metric does not work well on hypothesis filtering problems since incorrect hypotheses do not generally form a coherent cluster.

Consider a toy hypothesis filtering problem in which samples from a Gaussian distribution are mixed with uniform noise (see Fig. 3-5). Each point represents a hypothesis of the center of the Gaussian distribution: pair-wise consistency of hypotheses is simply a function of distance. As the figure illustrates, algorithms like Normalized-Cuts do not produce the desired output.

Let v be an $N \times 1$ *indicator vector* such that $v_i = 1$ if the i^{th} hypothesis should be accepted, and $v_i = 0$ otherwise. The sum of the compatibilities between the accepted hypotheses is $v^T A v$, and the number of accepted hypotheses is $v^T v$. Thus, the average pair-wise consistency $\lambda(v)$ for a subset of hypotheses v is simply:

$$\lambda(v) = \frac{v^T A v}{v^T v} \tag{3.4}$$

As an intuition-building exercise, suppose we have a set of accepted hypotheses v_0, and we want to decide whether to add another hypothesis to it. Adding a new hypothesis will add N pair-wise consistencies (of differing weights) to the numerator, but will also increase the denominator by 1. If adding those pair-wise consistencies increases the average amount of pair-wise consistency per node, then we should add that hypothesis. Note that our actual algorithm, unlike this exercise, does not greedily select hypotheses one by one.

Now, our task is to find an indicator vector v that maximizes $\lambda(v)$. Unfortunately, this problem has an exponentially-large search space. In the tradition of spectral graph-cutting methods like MinMaxCuts [11] and NormalizedCuts [62], we relax v to allow continuous values. We will find an optimal answer for continuous-valued v, and then later discretize v back into a proper indicator vector.

With v now a continuous-valued vector, we can look for extrema of $\lambda(v)$ by differentiating $\lambda(v)$ with respect to v and setting the result to zero:

$$\frac{d\lambda(v)}{dv} = \frac{Avv^T v - v^T Avv}{(v^T v)^2} = \frac{Av - \lambda(v)v}{v^T v} = 0 \tag{3.5}$$

Note that we have exploited the fact that A is symmetric. Rearranging terms, we get:

$$Av = \lambda(v)v \tag{3.6}$$

By inspection, we see that the value of v that maximizes $\lambda(v)$ is the dominant eigenvector of A (and $\lambda(v)$ will be the dominant eigenvalue).

We note that the densest subgraph can be found in polynomial time by performing a number of max-flow computations on the hypothesis graph [21]. Our spectral approach is faster and (perhaps more importantly) leads to a confidence metric, as described below.

3.4.3 The Eigenvectors of the Consistency Matrix

As we have shown, the dominant eigenvector of the consistency matrix A yields the optimal continuous-valued indicator vector v. The second-most dominant eigenvector of A, however, is also useful.

Recall that A is symmetric and that the eigenvectors of a symmetric matrix are orthogonal [67]. Intuitively, this means that the hypothesis sets described by the first and second eigenvalues represent *different explanations of the data.* Thus, the ratio of the first and second eigenvalues conveys the relative confidence of the solution.

In algorithms like JCBB or RANSAC, it is also possible to keep track of the second-best answer, but the second-best answer is almost invariably a trivial variation on the best answer. SCGP's ability to evaluate the next-best *orthogonal* solution is unique, and extremely useful.

The first two dominant eigenvectors e_1 and e_2 of a symmetric matrix A can be efficiently computed using the Power Method as follows:

Algorithm 2 Power Method for extracting the two dominant Eigenvectors and Eigenvalues of a symmetric matrix A. A total of K power iterations are performed, with $K = 5$ being a reasonable value.

for $i = 1; i <= 2; i = i + 1$ **do**
  e_i = a random $N \times 1$ positive vector, $||e_i|| = 1$
  for $iters = 0; iters < K; iters = iters + 1$ **do**
    for $j = 1; j < i; j = j + 1$ **do**
      $\alpha = e_j \cdot e_i$
      $e_i = e_i - \alpha e_j$
    end for
    $e_i = e_i / ||e_i||$
    $e_i = A e_i$
    $\lambda_i = ||e_i||$
  end for
  $e_i = e_i / ||e_i||$
end for

This algorithm performs five iterations of the power method per eigenvector, which is adequate in virtually all cases. (When the eigenvalues are closely spaced and convergence is slow, the confidence test based on the ratio of the eigenvalues would reject the eigenvectors anyway.) The inner loop in j removes any component of earlier eigenvectors from the current eigenvector.

We reject the whole hypothesis set when the confidence ratio λ_1/λ_2 is less than a threshold; we use a threshold of 2.

3.4.4 Discretization of the Indicator Vector

In an earlier step, we allowed v, an indicator vector nominally consisting of only zeros and ones, to take on arbitrary continuous values. We now need to discretize v back

into an indicator vector. We will denote the new discretized version of v as w. Given a threshold t, we write w as:

$$w_i(t) = \begin{cases} 1 & \text{if } v_i \geq t \\ 0 & \text{otherwise} \end{cases} \tag{3.7}$$

There are at least two reasonable ways of selecting t: picking t such that it maximizes the dot product $\hat{w} \cdot v$, or so that it maximizes Eqn. 3.4. Both work well, often producing the same answers. In our experiments, we have used the dot-product method.

The dot-product method can be implemented in $O(N \log(N))$ time. First, note that there are at most N different w vectors, since there are at most N distinct values in v. If the values of v are sorted, the dot product $w(t) \cdot v$ for increasing values of t can be incrementally computed in $O(1)$ time. Thus the total time complexity is $O(N + N \log(N)) = O(N \log(N))$.

We note that the discrete indicator vector w is not necessarily the globally optimal indicator vector. Instead, it is a discretized version of the optimal continuous indicator vector. While v is an approximation, its performance is very good.

Once the discrete-valued indicator vector has been computed, we add the hypotheses with $v_i = 1$ to the graph, discarding the others. (We assume that low-confidence hypothesis sets, as measured by λ_1/λ_2, have already been discarded.) Adding edges to the graph allows an improved posterior to be computed; the computation of this posterior is the subject of Chapters 4 and 5.

3.4.5 Improving Hypothesis Set Construction

The computation of the pair-wise consistencies incorporates information about existing edges in the graph, since these edges are used to compute the Dijkstra Links shown in Fig. 3-4. As a reminder, the rigid-body transformations of the two hypotheses and the Dijkstra links are composed together, with the resulting rigid-body transformation compared to the identity matrix. However, these Dijkstra links are not known with certainty, and errors in those links can distort the resulting probability calculations.

In general, the error (and uncertainty) between two poses is related to the length of the trajectory between them: longer trajectories allow for greater accumulation of error. With this in mind, we can construct hypothesis sets so that the Dijkstra links will be derived from relatively short trajectory segments.

Each hypothesis relates a pair of nodes, which we denote as (a, b). Our strategy is to restrict hypothesis sets to contain hypotheses of the form $(a + k_i, b + k_j)$ where each k is a relatively small number ($|k| \leq k_{max}$). Given two hypotheses $(a+k_1, b+k_2)$ and $(a + k_3, b + k_4)$, we can construct a loop using two Dijkstra links, $(a + k_1, a + k_3)$ and $(b + k_2, b + k_4)$. This strategy keeps the Dijkstra links short (at most $2k_{max}$ poses long), allowing us to limit the amount of error they introduce.

The value of parameter k_{max} is based on the local-navigation accuracy of the robot. In our systems, we set k_{max} to 8 poses, which typically translates to about 8 m of

trajectory for our indoor robots, or about 16 m of trajectory for our full-sized car. Larger values of k_{max} increase the number of opportunities to generate hypotheses for any given hypothesis set: this is important since we require a set to have a minimum number of hypotheses before it is processed (we used a minimum size of 4 in our experiments).

Large values of k_{max} can lead to inaccurate Dijkstra links. In general, this causes pair-wise consistency values to be smaller than ideal, not larger. (It is more likely that an error will result in a bad loop than a good loop.) Our method is robust to these types of errors: the algorithm continues to identify sets of hypotheses that are correct, but there is an increase in the number of falsely rejected hypotheses. In general, the performance of our method is good over a broad range of k_{max} values.

3.4.6 Analysis of Robustness to Outliers

At the heart of the SCGP is the pairwise consistency test which is used to construct the adjacency matrix. The behavior of the pairwise test can have a profound effect on the success or failure of the algorithm. SCGP can be applied to a large class of outlier rejection problems [51]; in the current context, correct hypotheses can be viewed as inliers, while incorrect hypotheses are outliers.

The most often encountered failure mode of SCGP is that SCGP selects most or all of graph, rather than just the subset of inliers. This happens when there are many outliers and those outliers have fairly high consistency with each other. In this section, we show how much outlier consistency SCGP can tolerate while still returning the correct answer.

Consider a scenario in which there are N_t true hypotheses and N_f false hypotheses. Assume that the pairwise consistency test reliably assigns edge weights of C_t to pairs of true hypotheses, and C_f to all other pairs. We assume that $0 \leq C_f < C_t$. The adjacency matrix (with inliers indexed first) has the following structure:

$$A = \left[\begin{array}{c|c} C_t & C_f \\ \hline C_f & C_f \end{array}\right] \tag{3.8}$$

By inspection, we see that an adjacency matrix of the form described above is rank 2, and has two non-zero eigenvalues. One non-trivial eigenvector corresponds to the subset of true hypotheses, while the other corresponds to the set consisting of the entire graph. The two non-zero eigenvalues are easily shown to be:

$$\lambda(v_t) = \frac{C_t N_t^2}{N_t} \tag{3.9}$$

$$\lambda(v_t + u_f) = \frac{C_t N_t^2 + C_f N_f^2 + 2C_f N_t N_f}{N_t + N_f} \tag{3.10}$$

We want SCGP to identify the set of true hypotheses, rather than the whole graph. This will happen so long as $\lambda(v_t) > \lambda(v_t + u_f)$. With some algebra, we see that this

constraint is satisfied when:

$$\frac{C_t}{C_f} > \frac{N_f}{N_t} + 2 \tag{3.11}$$

In other words, the ratio C_t/C_f determines how many outliers can be tolerated. Thus, a good consistency metric should try to make C_f as close to zero as possible, in order to maximize C_t/C_f. In the case of loop closure hypothesis filtering, C_f decreases exponentially as the rigid-body transformation of the loop deviates from the identity matrix. This helps ensure that the ratio C_t/C_f is large, allowing our algorithm to operate correctly even when the number of outliers greatly exceeds the number of inliers.

A different consideration for the pairwise consistency test is the "weight" of a hypothesis with itself. If we change the self-consistency of hypotheses, it is equivalent to adding a multiple of the identity matrix to the adjacency matrix.

$$\begin{aligned} (A + \alpha I)u &= \lambda(v)u \\ Au &= (\lambda(v) + \alpha)\, u \end{aligned} \tag{3.12}$$

We see immediately that the eigenvectors (and thus the SCGP solution) are unaffected. However, α shifts the eigen*values*, which can affect the convergence speed of Power Method-style algorithms for estimating the eigenvectors. Such a shift will also affect the confidence value λ_1/λ_2. Both the confidence threshold and the self-consistency value are under the control of the user, however, so they can be chosen in a compatible manner.

3.5 Results

We evaluated our loop-closing method on both vision-based and laser-based datasets. Our vision data set is a 1.9km trajectory comprised of three complete loops in an outdoor urban environment. Our lidar-based data sets include standard map building benchmark datasets, as well as several new real and synthetic datasets.

Loop closure algorithms are difficult to evaluate quantitatively: there are no standardized datasets for this purpose. The benchmark datasets that we have processed are composed of raw sensor data, but because sensor processing methods vary between researchers, the resulting sets of hypotheses are not the same. Naturally, the quality and reliability of the sensor processing (and hypothesis-generating) systems has huge impact on the difficulty of the loop-closing problem.

We address this problem in two ways. First, we evaluate the performance of our system repeatedly on the same dataset, with the quality of hypotheses purposefully degraded to varying degrees. This allows us to determine the point (if any) at which our algorithm can no longer produce useful output. Second, our hypothesis datasets have been made available so that other researchers can perform a direct comparison to our method.

3.5.1 "Area C" Dataset

During the DARPA Urban Challenge, "Area C" was a testing area in which the ability of autonomous cars to queue at intersections (waiting for other cars) was tested. Team MIT's vehicle logged a great deal of sensor data, including vision data, during its test. This logged data now serves as a vision-based mapping test[1]

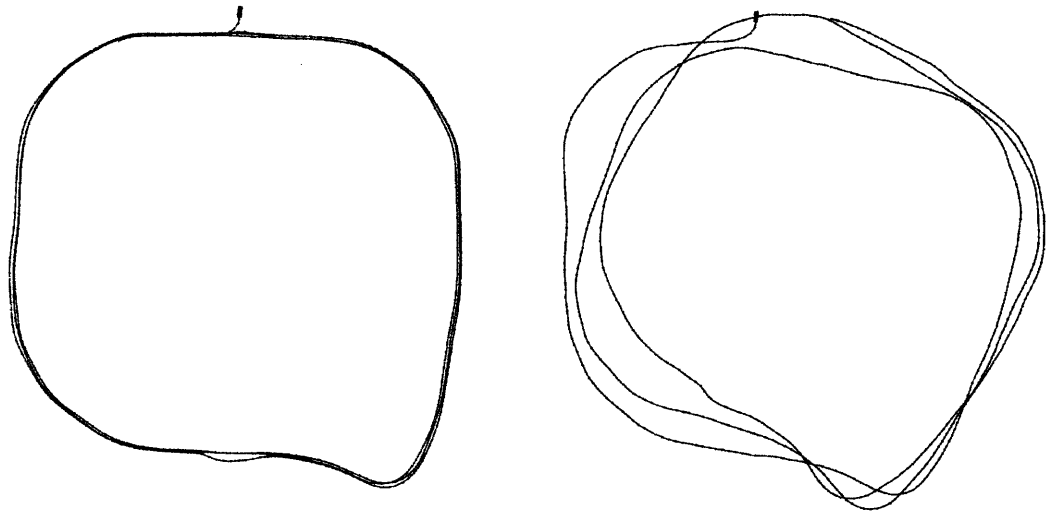

Figure 3-6: Area C dead-reckoning performance. Left: the dead-reckoning estimate, output by the high-end IMU, has an SSE of 1.02 versus the GPS-augmented ground truth. To simulate a more typical vehicle, additional noise was added (right): the initial configuration has an SSE of 487.

As described in Appendix A, we used SIFT features to initialize local landmarks. These local landmarks were then associated with each other in order to compute a rigid-body transformation relating two poses. This rigid-body transformation served as a hypothesis.

SIFT descriptors are 128-dimensional vectors. At their full size, they provide a fairly robust means of matching landmarks. However, hundreds of SIFT features can be detected in a single camera frame, making the storage of these descriptors problematic. Data structures to perform matching on these features also becomes more complicated. Other authors [32, 64] have described ways of reducing the dimensionality of SIFT descriptors, but the risk is that the discriminative power of the descriptor might be reduced. When designing new descriptors, it is important to know how much discriminative capability the descriptors must have in order to be able to close loops.

We answer this question by reducing the dimensionality of SIFT vectors in a simple way; this causes an increase in the hypothesis error rate. We then measure the ability of our approach to identify correct loop closures and reject incorrect hypotheses.

We reduce the dimensionality of SIFT vectors by dividing the descriptor into equally sized blocks and summing the elements in each block to form a shorter descriptor. While simplistic, our purpose is not to explore how to optimally reduce the

[1]The method described here was not used by MIT during the DARPA Urban Challenge itself; the data here has been post-processed.

dimensionality of SIFT descriptors, but to explore what happens when descriptors become ambiguous. Our technique is a simple and easily repeatable means of doing this.

In Fig. 3-7, two sets of SIFT features observed at different times (but near the same place) are shown. The two sets of features are matched based on descriptor's nearest-neighbors; these correspondences are shown as lines. The RANSAC method described in Appendix A is used to identify a set of "good" correspondences from which a hypothesis is generated. As the descriptor is decimated, the matching performance drops as expected, producing increasingly poor hypotheses.

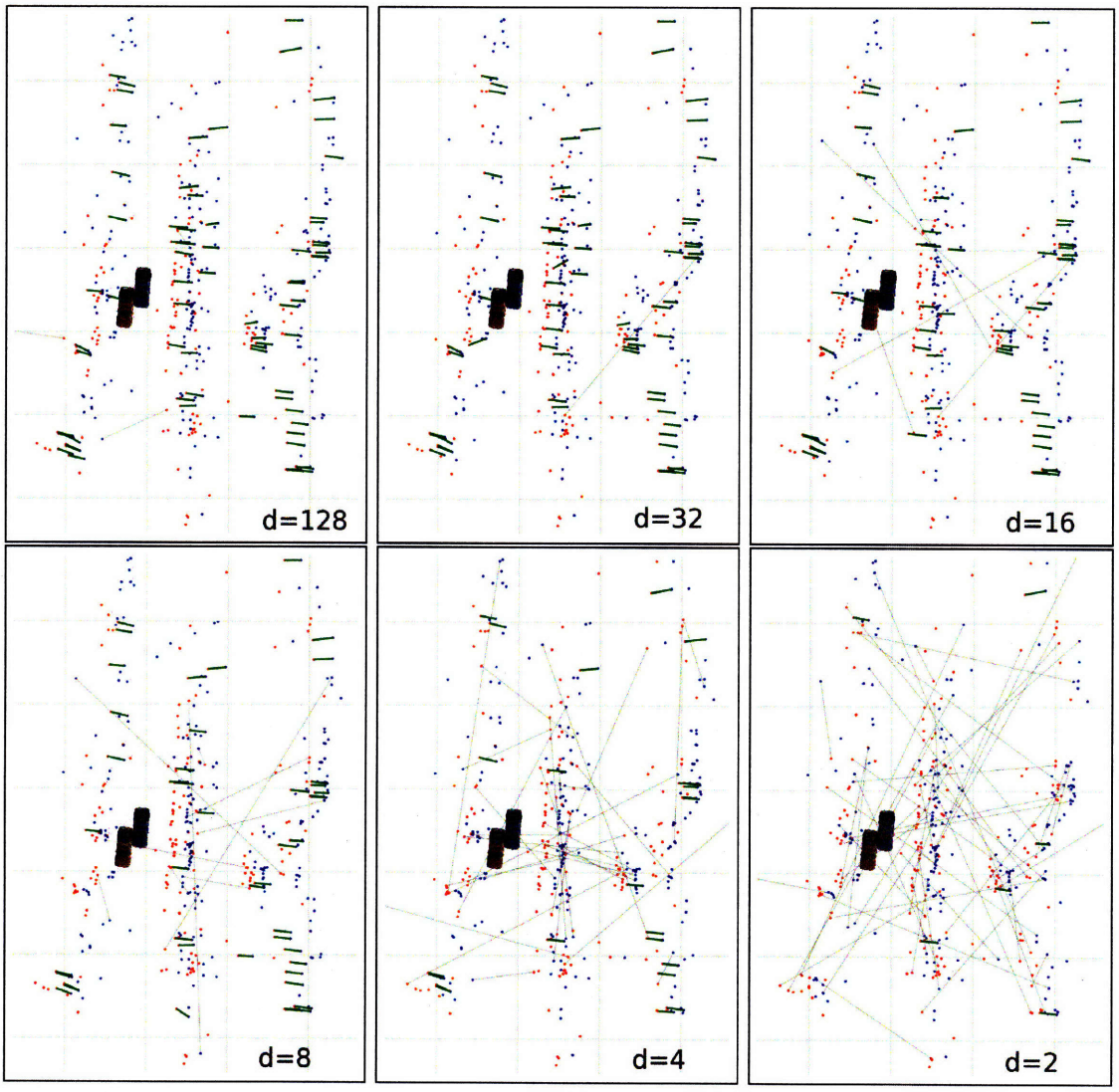

Figure 3-7: Matching decimated SIFT features. The SIFT features from two different visits to the same place are shown: dots correspond to features detected from the correspondingly-colored vehicle. Thick green lines indicate RANSAC-accepted correspondences, thin gray lines are rejected matches. As the size of the SIFT descriptor is reduced in size, the error rate increases. Our proposed loop closing method can effectively handle the increased error rate.

Significantly, our algorithm is able to identify correct hypotheses even when generated hypotheses are of very low quality (due to severe SIFT feature decimation).

In fact, our algorithm is still able to identify correct hypotheses even when those hypotheses are the result of scalar-valued SIFT descriptors.

As expected, our algorithm rejects a large fraction of the hypotheses when the SIFT descriptor is small (see Fig. 3-8); the acceptance rate rapidly increases with increasing SIFT descriptor size. Most importantly, the posterior maps that result from the filtered hypotheses are of high-quality (see Fig. 3-9).

The posterior map resulting from our vision-based approach, based on degraded odometry, was *better* (SSE=0.87) than the map resulting from the map derived from the expensive IMU alone (SSE=1.02). If we combine the high-quality IMU data with our vision method, the SSE drops to 0.07— even if using SIFT descriptors of size 3.

SIFT feature extraction and tracking were implemented using relatively slow methods and ran at about 10% of real time. Other authors have described ways of accelerating these steps [63]. Loop closing and the algorithms described in this chapter, in contrast, took between 7-9 s (depending on the size of the SIFT feature descriptors)— about 62 times faster than real-time.

3.5.2 CSAIL Dataset

We collected the CSAIL dataset in the Computer Science and Artificial Intelligence Laboratory's 7th floor (see Fig. 3-10). Its relatively short length coupled with a significant amount of clutter and fairly long loops make it an interesting test case. Like all of the datasets that follow, its primary sensor is a 180-degree field-of-view laser scanner. The robot trajectory was estimated via incremental scan matching, and loop closure hypotheses generated using a combination of local feature matching and scan matching refinement. The particulars of these methods are described in greater detail in Appendix A.

We will use this dataset to examine an actual hypothesis set in detail. The hypothesis set occurred in the area shown by Fig. 3-11. This hypothesis set contained 46 hypotheses, 34 of which were accepted as correct. Six representative samples of both inliers and outliers are shown in Fig. 3-12. The outliers (on top) are subtly misaligned; in contrast, the inliers (bottom) are well-aligned.

The pair-wise consistency matrix of those 46 hypotheses is shown as a graph in Fig. 3-13. In this figure, each node represents a hypothesis (e.g., one of the panels from Fig. 3-12): the *length* of a line represents the consistency between two hypotheses, with shorter lines indicating greater consistency. The cluster of inliers is readily discernible on inspection, and indeed, SCGP identifies it automatically.

A total of 5.3 s of CPU time was required to automatically generate loop-closure hypotheses and filter them using the mechanisms in this chapter. The graph had 201 poses, and a total of 1057 hypotheses were generated. Of these, 710 were accepted. (184 were rejected because they were members of hypothesis sets that had fewer than four hypotheses; 63 were rejected because they were members of a hypothesis set whose first two eigenvalues were too similar; and 100 were explicitly classified as outliers.)

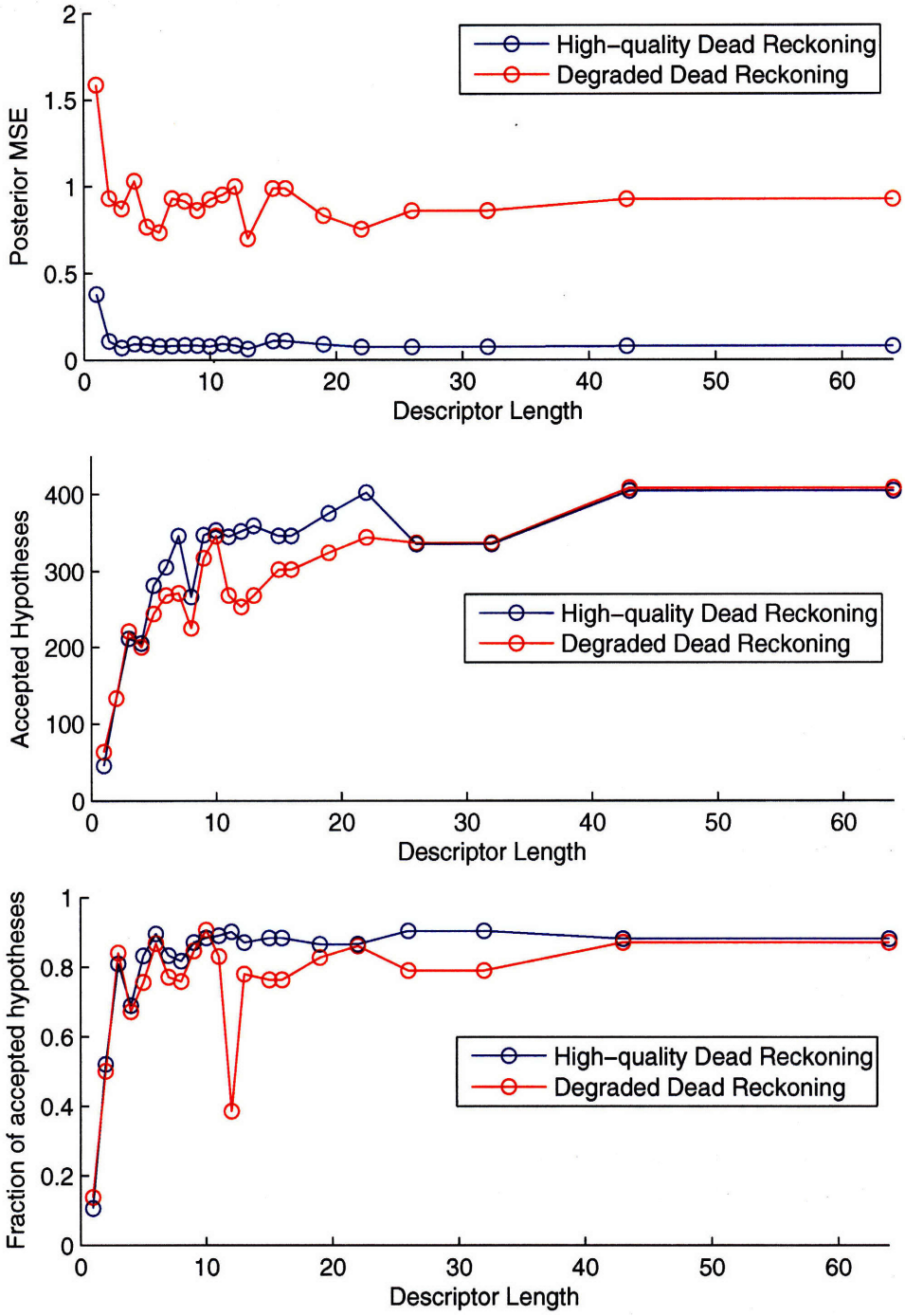

Figure 3-8: Area C Robustness. Shorter SIFT descriptors present greater perceptual ambiguity, and are more likely to produce incorrect matches. However, our method produces good maps in each case, and most surprisingly, can do almost as well with 3-dimensional vectors as with 128-dimensional vectors. The downward blip at $x = 12$ (bottom) was due to an usually large number of ambiguous hypothesis sets, in which the ratio of the first to second eigenvalues was less than the threshold of 2.0.

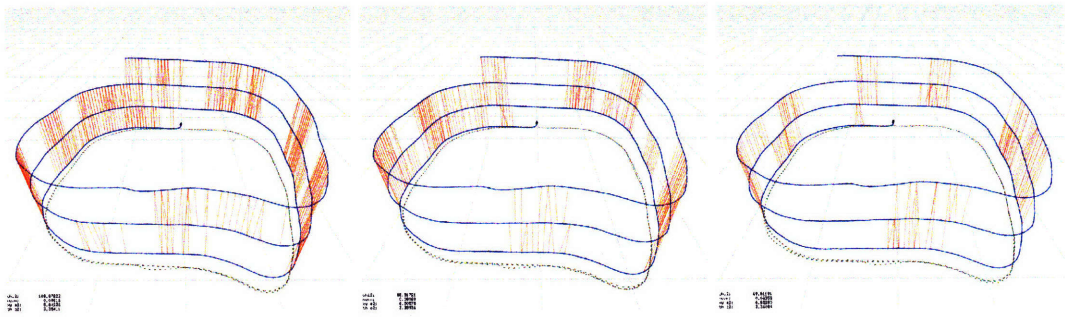

Figure 3-9: Area C Pose Graphs versus Descriptor Length. The pose/feature graph for the Area C dataset using 128-dimensional descriptors, 8-dimensional descriptors, and 1-dimensional descriptors. As the size of the descriptor goes down, ambiguity increases, resulting in fewer loop closures (red lines).

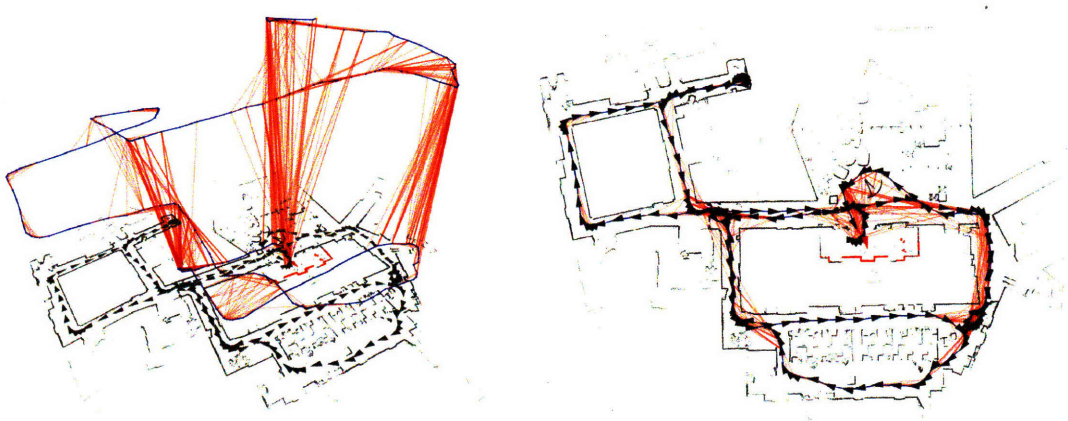

Figure 3-10: CSAIL Dataset Overview. Left: the pose graph is shown, with the trajectory in blue and loop closures in red. Right: the same pose graph, shown from above.

3.5.3 Killian Court Dataset

The Killian Court dataset, first collected and processed by Bosse [5], is challenging due to several long loops (see Fig. 3-15). Significant amounts of error can accumulate while the robot is traversing a long loop, which makes closing the loop harder. Another interesting characteristic of the Killian dataset is that it was recorded with a laser scanner with relatively high noise. This sensor noise makes it more difficult to recognize areas based on fine details. As a result, the hypothesis generator produced a greater number of incorrect hypotheses. The dataset also suffers from particularly poor wheel-derived odometry, as seen in Fig. 3-15.

As a partial compensation for this greater noise, loop closing was performed on larger sets of laser scans; i.e., instead of matching a single pose to another single pose, a small group of poses was matched at a time. This provided greater context for the hypothesis generator and reduced the rate of false positives.

In total, the Killian dataset consists of 1462 poses. These generated 6091 hypotheses, of which 4142 were accepted. 1187 were rejected because they were members of small hypothesis sets, 342 were rejected due to ambiguous eigenvalues, and 420 were

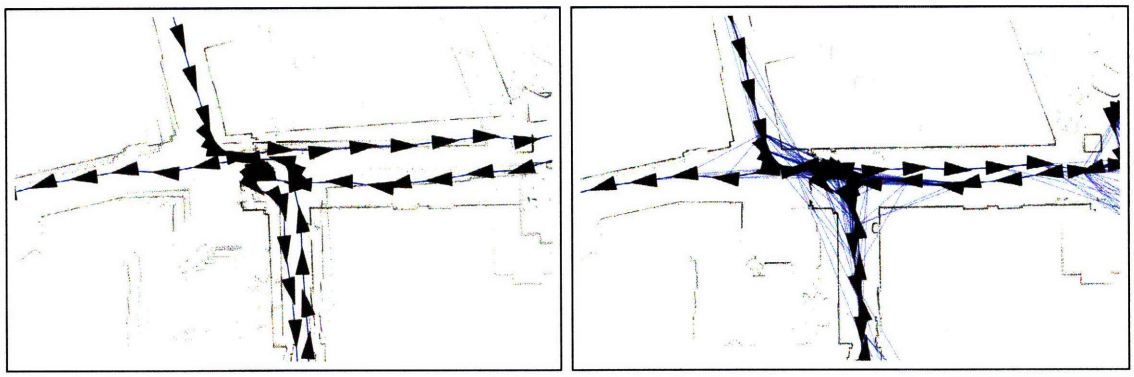

Figure 3-11: CSAIL loop closure close-up. Before loop-closing (left), laser scans show significant discrepancies. After recognition (right), not only are the discrepancies resolved, but the topological relationships are discovered.

classified as outliers. A total of 82 s of CPU time was required to generate and filter the hypotheses. The dataset corresponds to 2 hours and 9 minutes of robot driving.

3.5.4 Stanford Gates Building

Like many of the other datasets, the Stanford Gates dataset (a standard benchmark) was recorded with a single 180-degree field-of-view lidar. The dataset is reasonably large, and contains several often-visited areas. These result in a large number of loop closures.

The Stanford dataset consists of 679 poses, which generated 7549 hypotheses. Of these, 6398 were accepted, with 470 rejected because their hypothesis set was too small, 253 rejected due to poor eigenvalue confidence, and 428 classified as outliers. Processing time was 64 s; the dataset represents 34.5 minutes of data.

3.5.5 Intel Research Center

The Intel Research Center is a common benchmark dataset characterized by a large number of loop closures. The robot travels several times around the entire lab space, and visits each room. Of the datasets we processed, this dataset has the most loop closures.

The Intel dataset consists of 875 poses, spanning 44.9 minutes of robot driving. A whopping 15611 loop closure hypotheses were generated, and 12900 were accepted. Of those rejected, 758 were members of small sets, 715 were members of sets that failed the confidence test, and 1238 were classified as outliers. Total processing time was 180 s.

3.6 Summary

Recognizing places is a critical capability for many robot systems. Loop closing is a form of place recognition that is central to the task of map building: it prevents the

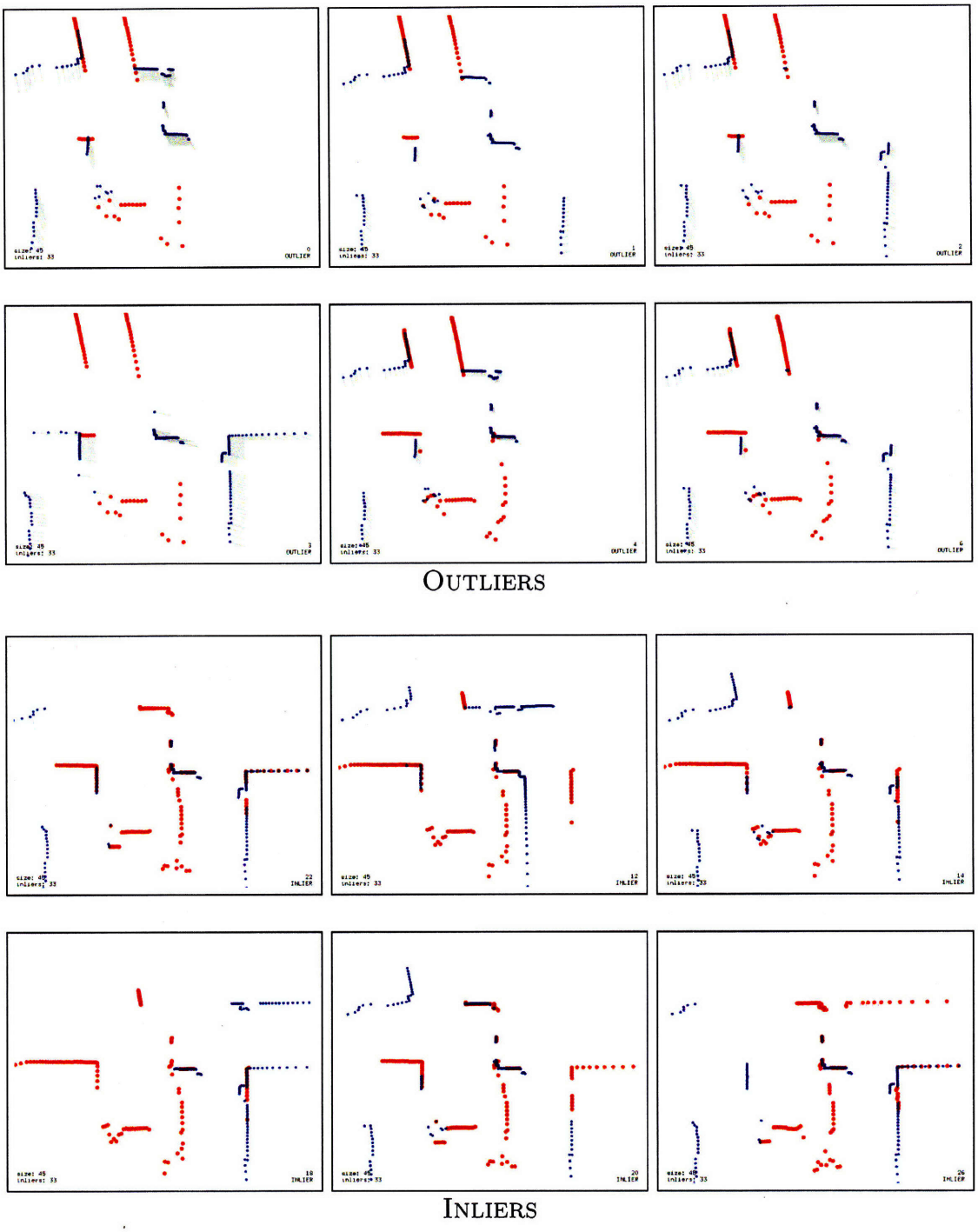

Figure 3-12: Filtered loop-closure hypotheses. Twelve representative hypotheses from a hypothesis set of size 45 are displayed. The outliers (top) and inliers (bottom) were automatically labelled using Single-Cluster Graph Partitioning. The alignment errors (derived from the posterior) are shown as gray lines: these errors are apparent in the outlier set.

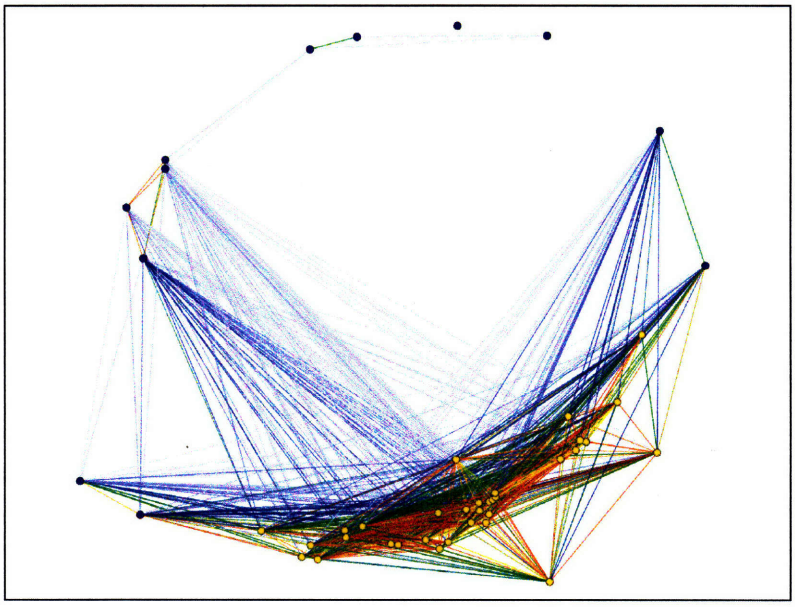

Figure 3-13: Adjacency graph for a set of 45 hypotheses. Each node represents a pose-to-pose hypothesis: the inlier set is indicated by yellow nodes (towards the bottom), with outliers in blue. Brightly-colored edges indicate greater pair-wise compatibility. The distance between two nodes is roughly proportional to the joint probability of the two hypotheses.

unbounded growth of dead-reckoning error. However, it is a difficult task due to both perceptual ambiguities and the large potential number of data associations.

In this chapter, we described an automatic loop closure system that can process a set of unreliable loop closure hypotheses and produce a set of correct hypotheses. It exploits the property that correct hypotheses generally agree with each other, whereas incorrect hypotheses tend to disagree with each other. The set of correct hypotheses is identified by examining the dominant eigenvectors of the pair-wise consistency matrix.

A critical capability of our method is the ability to determine the confidence of the solution. The second-best solution is not trivial variation on the best solution (as is the case in other algorithms), but is an orthogonal explanation of the data. The ratio of the eigenvalues of these two solutions serves as a useful confidence metric.

The method is also very fast, running much faster than real-time on all of the datasets we examined. It is also adaptable to a variety of sensing modalities: we demonstrated the system using local vision-based landmarks and laser-based scan matching.

The robustness of the system was demonstrated by decimating SIFT features. Originally 128-dimensional vectors, these were purposefully degraded to 1-dimensional scalars. Even though SIFT matching performance was severely affected, our system was able to identify correct loop closure hypotheses. The resulting loop closures allowed better posterior maps to be computed.

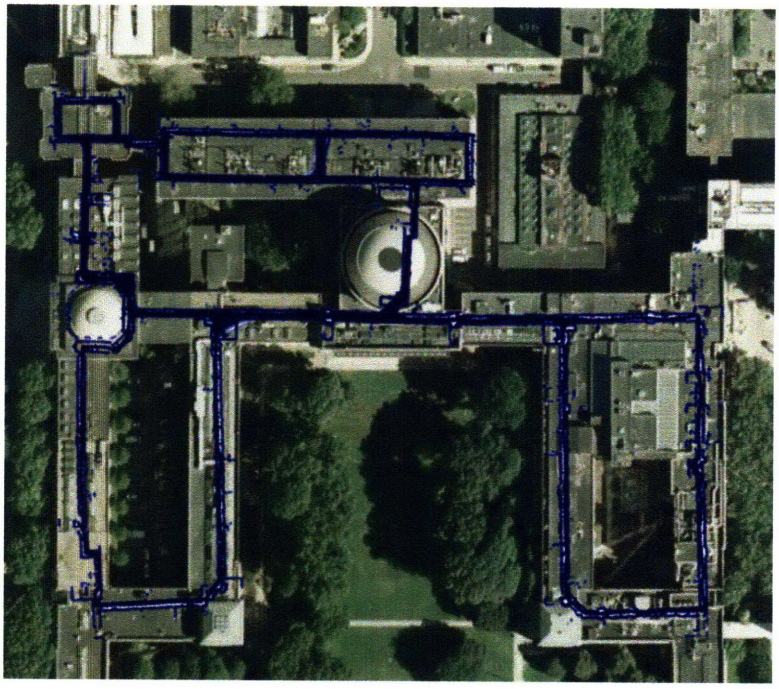

Figure 3-14: Killian Satellite Overlay. The long loops provide few opportunities for loop closing, leading to small distortions near the bottom of the figure. Macroscopically, however, the map aligns well with the satellite imagery.

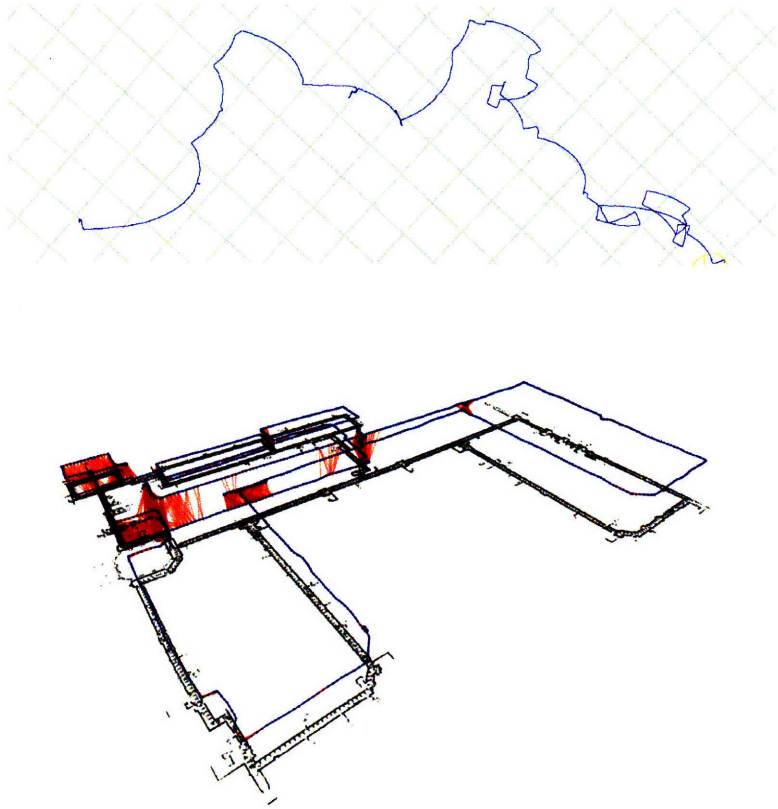

Figure 3-15: Killian Odometry and Pose Graph. Top: the raw wheel odometry (plotted on a 50 m grid). Bottom: The pose graph shows the relatively few areas in which loop closures are possible.

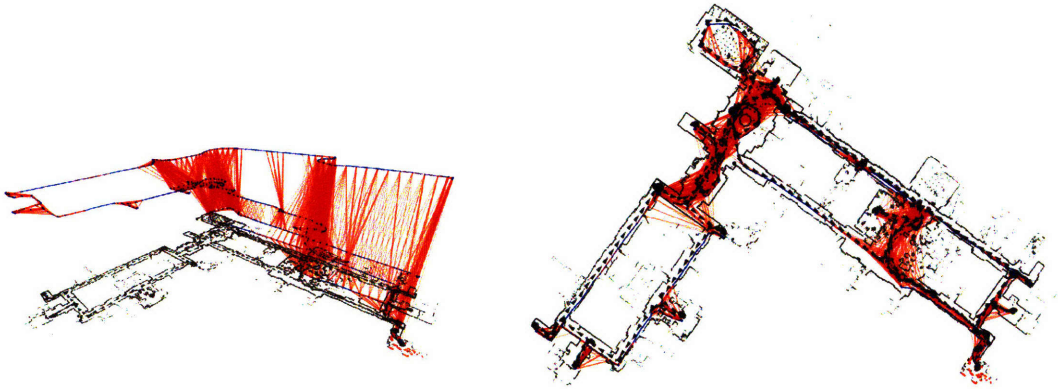

Figure 3-16: Stanford Gates Building. Left: the pose graph resulting from our algorithm. Right: the optimized map, viewed from above.

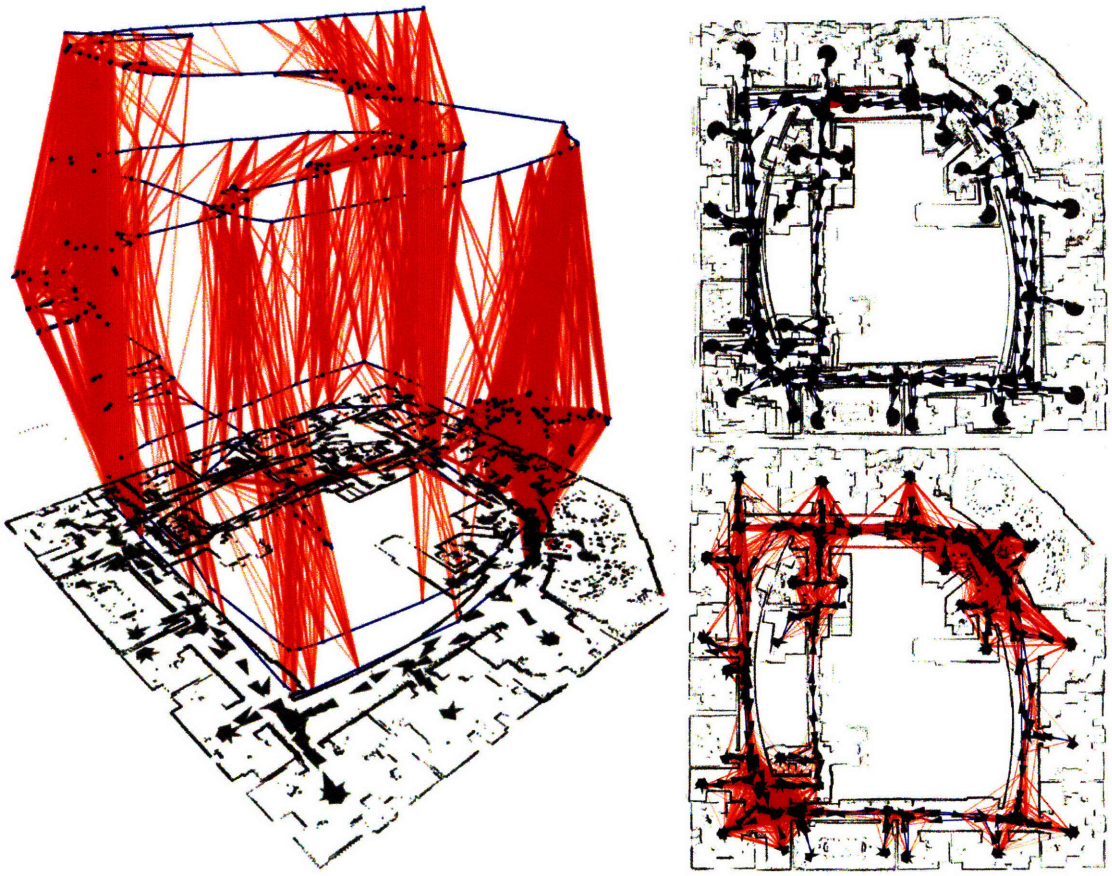

Figure 3-17: Intel research center. Left: the pose graph resulting from our algorithm. Top right: the uncorrected map. Bottom right: the optimized map, viewed from above.

CHAPTER 4

Batch Optimization of Pose Graphs

In this chapter, we describe our approach to optimizing pose graphs like those constructed in Chapter 3. The nodes of a pose graph represent particular places in the environment: these nodes can represent physical things (such as landmarks that have been sensed by the robot) or can represent the current or previous position of the robot. The edges of a pose graph relate the positions of two nodes and describe a cost as a function of the deviation from their preferred configuration. Pose graph optimization is the task of computing positions for each node such that the edges have minimum cost.

This optimization problem is critically important: the lowest-cost solution represents the best (maximum likelihood) map of the world. Whether a robot is avoiding obstacles, planning routes to some goal, or is explicitly trying to map an environment, the robot is dependent upon the fast and reliable optimization of the pose graph.

However, pose graph optimization is difficult for three central reasons:

- The state space is large. The size of the state vector is proportional to the number of nodes in the graph; a problem with thousands of state elements is *small.*
- The constraints are non-linear. As a result, the cost surface contains optimization hazards like local minima and valleys.
- The initial estimate is often poor. Most robots have meager dead-reckoning accuracy— as a result, the initial configuration of the pose graph can be far away from the best configuration.

This chapter describes a new optimization method that is both fast and robust. It can rapidly find the best configuration of nodes, and is robust to optimization hazards that trap existing methods. Our method also has low memory requirements and is relatively easy to implement.

Our method adapts Stochastic Gradient Descent (SGD) [56], commonly used in training artificial neural networks, to the problem of pose graph optimization. SGD has not previously been applied to map optimization, but has been used for localization [53].

We use a novel state-space representation, solving for the global-relative motions between poses, rather than solving for the absolute positions of the nodes. This representation allows faster reductions of error at each iteration, while a specialized data structure allows each iteration to be computed very quickly.

We demonstrate that our approach is not only very fast (competitive with or faster than other state-of-the-art methods), but is also significantly more robust: it finds correct solutions in cases that other methods get stuck in local minima.

4.1 Prior Work

4.1.1 Recursive Linear Solutions

If the constraints of a pose graph are approximated as linear and the noise is assumed to be Gaussian, pose graph optimization can be performed recursively using the Extended Kalman Filter (EKF) [65] or its dual, the Extended Information filter [58][1]. In these approaches, constraints are incorporated into the optimization one at a time. If the constraints are exactly linear, the resulting solution is optimal [43].

The Kalman filter maintains the covariance Σ and mean μ of the posterior distribution, while the Information filter maintains the information matrix Σ^{-1} and the information vector η. The information matrix and covariance are simply the inverses of each other, while the information vector is $\eta = \Sigma^{-1}\mu$.

These two representations provide an interesting trade-off in terms of the computational complexity of incorporating new information versus the computational complexity of extracting the posterior mean and covariance.

	Kalman Filter	Information Filter
Incorporate Constraint	$O(N^2)$	$O(1)$
Recover posterior mean/covariance	$O(1)$	$O(N^2)$

A large fraction of mapping applications, however, need to perform both operations. Even batch applications, in which the maximum likelihood solution is only needed at the very end, often need to recover the mean and covariance matrix at intermediate steps in order to perform data association. In other words, in order to compute the constraints from raw sensor data, the mean and covariance are often required. As such, in their naive forms, neither the Kalman filter nor Information filter is clearly superior.

However, the information matrix has a key property that has motivated a significant amount of research: sparsity. Whereas the covariance matrix rapidly becomes filled with significant non-zero values, the information matrix does not. The information matrix can be interpreted as the adjacency graph of a graphical model: its elements are non-zero only when variables are conditionally dependent (connected by an edge). If no variables have been marginalized out, the information matrix is

[1] As opposed to the Kalman filter and the Information filter, the extended versions provide for non-linear constraints.

naturally sparse [13, 14]. This is because the observation range of a robot is limited, and thus a robot can observe only a small fraction of the environment from any given pose. A sparse information matrix can be stored in less memory than a covariance matrix (which is almost always dense), and the sparsity allows much faster recovery of the posterior mean and covariance.

In feature-based mapping methods, past robot positions are typically marginalized out. This marginalization creates new edges in the graphical model and makes the information matrix dense. However, the new edges tend to have small weights [18]. Thrun et al. described a method called Sparse Extended Information Filters (SEIF) in which these small weights are truncated. The result is a sparse approximation of the information matrix [69, 70, 71].

However, truncating elements of the information matrix can lead to over confidence [74]. An alternative method called Exactly Sparse Extended Information Filters (ESEIF) [75, 76] reimposes sparsity in a conservative manner by periodically relocalizing the robot. These relocalizations sever the conditional dependence on new robot positions from old positions. The resulting information matrix is truly sparse (hence "Exactly Sparse"), but the posterior does not exactly match that of the full information matrix: information about the previous position of the robot is discarded whenever the robot is relocalized.

Unfortunately, all recursive formulations suffer from linearization error. When a non-linear constraint is processed, it is linearized around the current state estimate—the non-linear aspects of the constraint are discarded. If the state estimate is of poor quality (i.e., far from the truth) when a constraint is linearized, the resulting linearization error is injected into the solution. Worse, the resulting uncertainty estimate is ignorant of this extra error. As a result, the uncertainty estimates are too low.

Some of the impact of this linearization error can be ameliorated by using the Unscented Kalman Filter (UKF). The Unscented Kalman Filter allows a more accurate projection of the covariance matrix after each constraint update by employing a small particle filter. Each particle is updated according to the non-linear form of the constraint; the covariance computed from these particles can be more accurate than the projection of the covariance based on only the first derivatives of the constraint.

The Iterated Kalman Filter [3] addresses linearization error in a different way. When a constraint is incorporated, that constraint might result in a large change in the state estimate. Since the constraint presumably brings the state estimate closer to the truth than it was before, it would be preferable to linearize around the new state estimate, rather than the previous state estimate. The Iterated Kalman filter allows this: the linearization point for each constraint is iteratively updated so that each constraint is linearized around a state estimate that reflects the contributions of that constraint. However, as new constraints continue to arrive and change the state estimate, older constraints are *not* relinearized.

4.1.2 Graphical Models

When a graphical model has a tree structure, the posterior distribution can be computed quite quickly [30]. Unfortunately, the graphical models corresponding to pose graphs are cyclic in virtually all interesting cases. Paskin's Thin Junction Tree Filters (TJTF) [52] and Frese's TreeMap [19] impose a tree structure on the graphical model by pruning edges. An advantage of this approach over SEIFs and ESEIFs (which similarly sacrifice edges in the graphical model for computational efficiency) is that constraints can be relinearized whenever inference is performed. Since edges are removed from the model, these methods compute an approximation of the correct posterior. These approximations can result in noticeable map artifacts.

4.1.3 Particle Filtering

Particle filters approximate the posterior distribution using a set of discrete samples. The best known method is Montemerlo's FastSLAM [44]. FastSLAM particles are samples of just the robot trajectory: Montemerlo showed that the position of landmarks can be trivially recovered once the trajectory is known. (Landmarks are conditionally independent given the robot trajectory, since landmarks can only be observed from the robot trajectory).

The primary challenge inherent in particle filtering is accurately approximating the posterior: this can require large numbers of samples, especially when the uncertainty is large. Maintaining sufficient sample density and diversity has been an ongoing challenge [66, 22].

A unique advantage of particle filter methods is that data association is simplified: since each particle represents a trajectory, and the position of landmarks are independent given a trajectory, data association can be performed using a simple nearest-neighbor rule. Each particle can represent a different set of data association decisions.

4.1.4 Nonlinear Optimization

A growing family of iterative optimization methods can also be used to compute map posteriors. These approaches operate on the whole pose graph, and do not require any information to be discarded. Further, at each iteration, they can relinearize the constraints.

A brute-force nonlinear least squares implementation was suggested by Lu and Milios [41]. Their approach is similar in formulation to more modern approaches, but is generally too slow. Gradient Descent and Conjugate Gradient Descent [34, 72] have also been applied to map building.

Duckett *et al.* described an optimization method [12] using Gauss-Seidel relaxation. The basic approach is to repeatedly solve for the position of one node at a time, fixing the position of all the other nodes. They assumed absolute knowledge of the robot's orientation, essentially making the problem linear. Fortunately, their approach can be extended in a straight-forward manner to account for unknown ro-

tation, and the resulting algorithm is quite good at fine-tuning a state estimate that is already close to the global minimum. Unfortunately, when the state estimate is far from the minimum, convergence can be impractically slow. Frese, Larsson, and Duckett attempted to improve convergence speed with the Multi-Level Relaxation (MLR) algorithm [20]. Folkesson's Graphical SLAM method [16] is also based on Gauss-Seidel relaxation: it is accelerated by explicitly grouping nodes into "star nodes", rather than automatically partitioning them as in MLR. Star nodes effectively reduce the size of the state space, accelerating convergence.

Konolige proposes a method [34] for accelerating convergence by reducing the graph to poses that have a loop constraint attached, solving for the other nodes separately. This can save considerable CPU time, but requires the graph to have low connectivity. This optimization is broadly applicable to most mapping algorithms.

Dellaert's $\sqrt{SAM}$ method [10] performs a sparse Cholesky decomposition on the sparse information matrix. The factorization is greatly accelerated by selecting a good variable reordering (which is equivalent to picking an order to marginalize features in the corresponding graphical model). A key insight is that explicitly marginalizing out variables can only increase memory usage and decreases the effectiveness of the on-demand variable reordering algorithm. The method is generally one of the fastest methods available for full nonlinear optimization. Its performance, however, is very sensitive to the structure of the information matrix and the fill-in of the resulting Cholesky factors.

All non-linear optimization methods run the risk of getting stuck in local minima. Unfortunately, this risk is not hypothetical for map-building problems: non-linear approaches are susceptible to producing substantially different results depending on the initial conditions, as previously shown [24] and further demonstrated in this chapter.

The linearizations of recursive formulations like the EKF and EIF guarantee a quadratic and convex optimization surface with an easy-to-find minimum, but the resulting answer may not be a good solution to the actual problem. Nonlinear optimization methods generally offer better solutions, even if they do "get stuck". This chapter describes non-linear optimization methods based on earlier work [48, 50].

4.1.5 Hybrids

Bosse's Atlas [5] uses linearized (EKF-based) submaps but stitches them together using nonlinear optimization. During this stitching operation, the submaps are treated as rigid. The resulting "stitched" map generally has small errors where submaps do not align exactly. The decoupling of large maps into smaller maps, however, requires less computation.

4.1.6 State Space

Most map building algorithms use an obvious state space representation in which the state variables corresponds to the *positions* of the nodes. However, this is not the only possibility. Previous authors have used local submaps whose coordinate system is defined by features in that submap [36], while others have used state space

representations in which the *relative* position of nodes is used. This chapter will introduce a new representation that encodes the *motions* of the robot as it moves from pose to pose.

4.2 Overview and Intuition

Our approach is motivated by a simple intuition. Consider a pose graph constructed by a robot as it travels around a large loop. The trajectory of the robot is periodically sampled, with each sample point represented by a pose (a "node") and connected to its predecessor by an estimate of the robot's motion (an "edge"). Note that every edge has an uncertainty associated with it, so it is possible to compute its χ^2 error.

Suppose the robot returns to its starting place and recognizes it; this adds an additional edge between the first and last pose, creating a cycle (see Fig. 4-1). This edge is commonly called a "loop closure", since it closes a large loop of nodes. Recognizing these loop closures was the subject of Chapter 3.

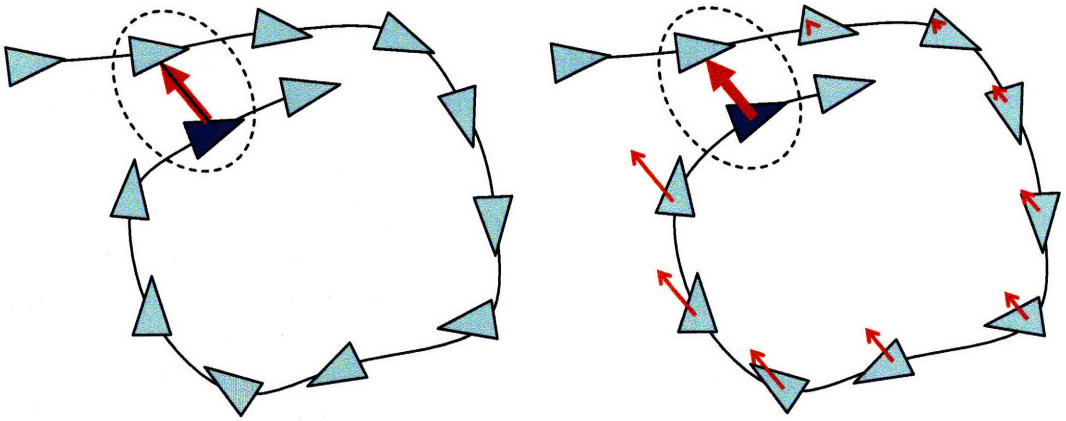

Figure 4-1: Map Optimization Intuition. The χ^2 error of a graph with a poorly-satisfied constraint (left, red arrow) can be reduced by distributing small adjustments around the nodes in the loop (right).

The χ^2 cost of any constraint grows as the square of its residual; a net reduction in cost can be obtained by reducing the error of the high-error constraint at the expense of adding a small amount of error to other constraints. We can do exactly this, adding small motions to each of the constraints so that the final loop closure is more closely satisfied. All of the constraints now have some error, but the χ^2 error has been reduced.

Konolige [34] pointed out that this simple problem can be solved "exactly" (up to linearization error) in $O(N)$ time by linearizing the constraints and recursively dissecting the loop and optimizing each half using a trivial EKF. (We note that this method is not immune to linearization error; in fact, this linearization error can cause divergence when large errors are present.)

In general, pose graphs do not consist of a single loop, but rather a complex web of interconnected constraints. Konolige's method, which simultaneously optimizes N

constraints, cannot be extended to this case, because the recursive dissection relies on the nodes having only two neighbors.

However, each constraint can be viewed as part of a loop in which the constraint forms a loop relative to the current state estimate. The error of the constraint can be reduced by adjusting the position of elements of the state vector between the beginning and end of the loop, just as in Fig. 4-1.

Unfortunately, it is generally impossible to satisfy all the constraints at the same time. How do we know which constraints to satisfy at the expense of other constraints? Determining this is equivalent to performing a least-squares iteration, and is a slow operation. Instead, our method iteratively finds an equilibrium in which the conflicting updates of antagonistic constraints are in balance. This equilibrium corresponds to our desired solution.

4.3 Method

In this section, we derive our method. The first step is to introduce a novel state space representation that accurately encodes the cumulative nature of robot motion: the position of a robot at time t is the integral of its motions over times $[0, t]$. This state space representation is a key contribution, and is a critical factor in the algorithm's performance.

Second, we derive an iterative update to the state vector using a single constraint. This update is based on stochastic gradient descent. It is simple, fast to compute, and highly effective at reducing both the χ^2 error and SSE of the graph. These traits make the algorithm competitive with (and often superior to) other state-of-the-art methods. However, our iterative update is particularly good at escaping local minima, giving our algorithm an ability to find good solutions when other algorithms get stuck.

Finally, we describe a data structure that reduces the computational cost of each update from $O(N)$ to $O(\log N)$, where N represents the number of poses in the graph.

In this work, we assume that all constraints between poses are full-rank rigid-body transformations. Rank-deficient constraints can, in principle, be treated as well; however, we do not examine this case. (We note that rank-deficient constraints, such as the bearing-only constraints arising from vision data, can often be combined to form full-rank constraints; this is precisely the technique we use on our vision based dataset.) We also limit ourselves to 2D, though our methods also apply in principle to 3D.

4.3.1 Incremental State Representation

The choice of state representation has a large impact on the performance of our optimization algorithm. Picking a state space in which the state variables echo the structure of the underlying random process results in significant performance improvements.

Consider a pose graph consisting of a single robot trajectory sampled at particular times. The conventional state space is the *global absolute* (x_{global}) state space— i.e.,

a list of (x, y, θ) positions for each node. However, the position of a robot at time t is the integral of all of its motions from time 0 through time t. The underlying random processes of the robot trajectory are the *motions* between each pose, not the positions.

Using the x_{global} state space tends to obscure this underlying structure. Suppose a and b are two poses from a robot trajectory and that their positions are constrained by a rigid-body transformation. In the x_{global} state space representation, the gradient of this constraint is zero everywhere except for nodes a and b. In other words, while the constraint gives information about the motion of the robot between a and b, this information is not reflected in the position of other nodes, whose positions are a function of this motion.

Virtually all contemporary algorithms incorporate the effects of *all* the constraints at each update step, which avoids this problem. By including all the constraints (most obviously the odometric constraints), their solutions reflect the underlying structure of the problem. This comes at a cost: it takes more time to incorporate the effects of all the constraints.

However, it is possible to use a state space that more directly reflects the structure of the problem. This allows good state updates *without* having to consider the effects of all the constraints at every iteration.

$$\underset{\mathbf{x_{global}}}{\begin{bmatrix} x_0 \\ y_0 \\ \theta_0 \\ x_1 \\ y_1 \\ \theta_1 \\ x_2 \\ y_2 \\ \theta_2 \\ \cdots \end{bmatrix}} \quad \underset{\mathbf{x_{rel}}}{\begin{bmatrix} x_0 \\ y_0 \\ \theta_0 \\ \cos(\theta_0)(x_1 - x_0) + \sin(\theta_0)(y_1 - y_0) \\ -\sin(\theta_0)(x_1 - x_0) + \cos(\theta_0)(y_1 - y_0) \\ \theta_1 - \theta_0 \\ \cos(\theta_1)(x_2 - x_1) + \sin(\theta_1)(y_2 - y_1) \\ -\sin(\theta_1)(x_2 - x_1) + \cos(\theta_1)(y_2 - y_1) \\ \theta_2 - \theta_1 \\ \cdots \end{bmatrix}} \quad \underset{\mathbf{x_{incr}}}{\begin{bmatrix} x_0 \\ y_0 \\ \theta_0 \\ x_1 - x_0 \\ y_1 - y_0 \\ \theta_1 - \theta_0 \\ x_2 - x_1 \\ y_2 - y_1 \\ \theta_2 - \theta_1 \\ \cdots \end{bmatrix}}$$

Figure 4-2: State Space Representations. The commonly used state space x_{global}, the rigid-body transformation state space x_{rel}, and our proposed "global incremental" state space x_{incr}.

One way of "capturing" the cumulative nature of robot motion is to represent the robot trajectory as a series of rigid body transformations. In this case, the position of the trajectory node i in the trajectory is the composition of all the rigid-body transformations $T_0 \oplus T_1 \oplus ... \oplus T_{i-1}$, and the state space is a list of the rigid-body transformations. We call this the "relative" (x_{rel}) state space (see Fig. 4-2).

This state space, while accurately modeling the motion of the robot, is not stable. In a well-behaved optimization problem, a change to the state vector should have a commensurate effect on the cost function: small steps should lead to small changes in cost, for example. This is not the case when using a relative state space representation. A tiny change to a rotational component in one part of the graph can have a very

large projective effect on another (distant) part of the graph. This can cause enormous increases in the χ^2 error of other constraints. Because these projective effects are not well-modeled due to linearization error, a state change that is predicted to reduce the χ^2 error might actually increase it.

Instead, we propose a state space representation that approximates the cumulative motion of the robot like x_{rel}, but eliminates the projective effects of the relative representation. We call it the "global incremental" state space representation, x_{incr} : the state vector contains the difference between the successive global positions. In the absence of rotations ($\theta_i = 0$), it is identical to the relative state space. With this state space representation, there are no projective effects (the position of a node is a simple sum of global motions, not the composition of rigid-body transformations), and so updates are more stable.

4.3.2 Iterative optimization

Our approach is a hybrid of nonlinear least squares and stochastic gradient descent; we begin by reviewing the "vanilla" versions of these algorithms.

Nonlinear Least Squares

Recall from Chapter 2 that the χ^2 error for a graph, evaluated at a distance d from the current state estimate, can be written in terms of the Jacobians of the constraints J, the residuals r, and the block-diagonal information matrix Σ^{-1} of the constraints:

$$\chi^2 = (Jd - r)^T \Sigma^{-1} (Jd - r) \tag{4.1}$$

We can compute the d that minimizes the χ^2 error by differentiating with respect to d and setting the result to zero:

$$\begin{aligned} \frac{\partial \chi^2}{\partial d} &= 2J_i^T \Sigma_i^{-1} Jd - 2J_i \Sigma_i^{-1} r_i = 0 \\ (J^T \Sigma^{-1} J)d &= J^T \Sigma^{-1} r \end{aligned} \tag{4.2}$$

Note that Eqn. 4.2 is of the form $Ax = b$, with $A = J^T \Sigma^{-1} J$ (the information matrix) and $b = J^T \Sigma^{-1} r$. This equation can be solved for d via a number of methods such as inversion of A, LU decomposition, or Cholesky decomposition.

Algorithm 3 NonlinearLeastSquares

repeat
  compute J and r at the current state estimate.
  $d = (J^T \Sigma^{-1} J)^{-1} J^T \Sigma^{-1} r$
  $x = x + d$
until $||d|| < \epsilon$

Nonlinear-least squares computes state updates by simultaneously considering the interaction of all the constraints. Often, only a few iterations are required for a very

good solution. Unfortunately, Alg. 3 can be impractically slow.

Stochastic Gradient Descent

In Stochastic Gradient Descent (SGD), a constraint is selected and a gradient descent step is taken to reduce the error of that constraint. However, a graph generally contains antagonistic constraints, whose steps tend to conflict. Stochastic Gradient Descent suggests a solution: introduce a learning rate λ that modulates the step size, and reduce λ over time. Provided that a few technical conditions (described below) are satisfied, the resulting method will find an equilibrium between antagonistic constraints. This equilibrium indicates that a minimum has been found.

Naturally, we expect that single iteration of stochastic gradient descent will not reduce the χ^2 error as much as an iteration of non-linear least squares. The promise of SGD is that each iteration is so easy to compute that the minimum will be found faster, despite requiring more iterations.

The learning rate must satisfy two conditions in order to guarantee convergence [6]; the first ensures that the algorithm can enough "energy" to explore the whole state space, while the second ensures that the gradients eventually vanish when the solution approaches the optimal value:

$$\sum \lambda_t = \infty \tag{4.3}$$

$$\sum \lambda_t^2 < \infty \tag{4.4}$$

We use Robbins' original suggestion [56], of a harmonically-decreasing learning rate (i.e., $\lambda_t = 1/t$), which satisfies these properties. Other authors have explored more complex (and adaptive) learning rates [7], but our experiments with these methods did not consistently improve performance over a simple harmonic progression.

Consider the χ^2 error of a single constraint i, evaluated at the current state estimate plus d (see Chapter 2 for more details):

$$\chi_i^2 = (J_i d - r_i)^T \Sigma^{-1} (J_i d - r_i) \tag{4.5}$$

The gradient is:

$$\frac{\partial \chi_i^2}{\partial d} = 2 J_i^T \Sigma_i^{-1} J d - 2 J_i^T \Sigma_i^{-1} r_i \tag{4.6}$$

Evaluated at the current state estimate, where $d = 0$, gives just:

$$\frac{\partial \chi_i^2}{\partial d} = -2 J_i^T \Sigma_i^{-1} r_i \tag{4.7}$$

At time step t, let the learning rate λ be $1/t$. The gradient descent step moves in the opposite direction of the gradient, and can be written as:

$$d = \lambda 2 J_i^T \Sigma_i^{-1} r_i \tag{4.8}$$

Algorithm 4 Stochastic Gradient Descent Algorithm

$\lambda = 1/3$
while not converged **do**
  select a constraint i at random
  compute J_i and r_i at the current state estimate.
  $d = 2\lambda J_i^T \Sigma_i^{-1} r_i$
  $x = x + d$
  $\lambda = \lambda/(\lambda + 1)$
end while

At each iteration, Stochastic Gradient Descent (see Alg. 4) reduces the error of a single constraint (often unintentionally increasing the error of other constraints). In early iterations, it will take large steps in order to reduce the residual, which causes the state estimate to jump around the state space. This behavior allows SGD to escape local minima. Intuitively, while many constraints may be satisfied at a local minimum, there are generally one or more constraints that remain poorly satisfied. These ill-satisfied constraints will cause a comparatively large jump away from that local minimum. Conversely, near the global minimum, constraints tend to be better satisfied: this means smaller jumps. In short, ill-satisfied constraints cause the state estimate to jump around, but the state estimate tends to "stick" near the global minimum once it is found.

The ability to rapidly explore the state space not only allows SGD to escape local minima, but is conducive to rapid reductions in the χ^2 error. This is a valuable feature to the application of map optimization, where the initial state estimate can be very poor. The disadvantage, however, is that this exploration never ends. Even if the global minimum has been found, SGD continues to jump around (though the jumps shrink over time due to the deceasing learning rate). In other words, while convergence is very fast at the beginning, it is ill-suited towards "fine tuning" the solution. As we will illustrate, while the effect is real, the practical consequences are often negligible.

SGD is sensitive to the scaling of the Σ_i^{-1} matrices: this scale proportionately affects the size of the step d. Two optimization problems that differ only by the scaling of their Σ_i^{-1} matrices are equivalent, yet Eqn. 4.7 will yield different steps for them. We can eliminate this sensitivity by normalizing the set of Σ_i^{-1} matrices according to the largest example.

Our Hybrid Method

Our method combines features of nonlinear least squares and stochastic gradient descent. We also incorporate additional domain-specific knowledge to prevent the state updates from making harmfully large steps.

First, note that the nonlinear least squares step (Eqn. 4.2) can be written in terms

of contributions from each individual constraint:

$$d = (J^T\Sigma^{-1}J)^{-1}J^T\Sigma^{-1}r \tag{4.9}$$
$$= \sum_i d_i \tag{4.10}$$
$$= \sum_i (J^T\Sigma^{-1}J)^{-1}J_i^T\Sigma_i^{-1}r_i \tag{4.11}$$

The d_i's computed by nonlinear least squares and stochastic gradient descent are quite similar. The difference is essentially scaling: nonlinear least squares rescales and reshapes the step according to the inverse of the information matrix. Intuitively, this scaling accomplishes two things: it determines the relative importance of each constraint, and it determines how error should be distributed around a loop. (Since some nodes are more uncertain than others, a larger decrease of χ^2 error can be obtained by distributing more error to less-confident nodes.)

Nonlinear Least Squares	Stochastic Gradient Descent
$d_i = (J^T\Sigma^{-1}J)^{-1}J_i^T\Sigma_i^{-1}r_i$	$d_i = 2\lambda J_i^T\Sigma_i^{-1}r_i$

Our hybrid solution approximates this scaling by finding an $M^{-1} \approx (J^T\Sigma^{-1}J)^{-1}$. We use a relatively low-fidelity approximation, computing M as the block-diagonal of $J^T\Sigma^{-1}J$. This approximation is easy to compute and does a reasonably good job of adjusting how error is distributed around a loop (i.e., it estimates the relative confidence of each node), but it is nearly useless as a means of estimating the absolute importance of each constraint. In other words, the *shape* of M is useful, but the *scale* is not. To negate the effect of M's scale, we constrain d_i so that it has the same magnitude as $J_i^T\Sigma_i^{-1}r_i$. In summary, we can compare the three update formulas:

Nonlinear Least Squares	Stochastic Gradient Descent	Hybrid
$d_i = (J^T\Sigma^{-1}J)^{-1}J_i^T\Sigma_i^{-1}r_i$	$d_i = 2\lambda J_i^T\Sigma_i^{-1}r_i$	$d_i = \lambda M^{-1}J_i^T\Sigma_i^{-1}r_i$

As mentioned earlier, the scale of Σ_i^{-1} has an impact on the step size. This effect is removed by scaling the steps by largest Σ_i^{-1} in the graph.

Our algorithm, like stochastic gradient descent, takes steps whose size are proportional to the errors of the constraints. In map optimization, however, we can often compute a *maximum* permissible step size. Stepping farther than this maximum would only start increasing the error again, and thus should be avoided. Consequently, we clamp d such that it will not overshoot the constraint's minimum. This improves both the performance and stability of the algorithm.

4.3.3 Rigid-Body Constraints

We now illustrate how to apply the optimization method to a rigid-body constraint. Suppose we have a rigid-body constraint T connecting poses a and b. In x_{global}

coordinates, the observation equations for the rigid-body constraint are:

$$\begin{aligned} x_T &= \cos(\theta_A)(x_B - x_A) + \sin(\theta_A)(y_B - y_A) \\ y_T &= -\sin(\theta_A)(x_B - x_A) + \cos(\theta_A)(y_B - y_A) \\ \theta_T &= \theta_B - \theta_A \end{aligned} \tag{4.12}$$

Let us write the incremental motions between consecutive x coordinates as $\dot{x}$, such that $\dot{x}_i = x_i - x_{i-1}$. We do the same with $\dot{y}$ and $\dot{\theta}$ such that we can write $x_{incr} = [\dot{x}_1\ \dot{y}_1\ \dot{\theta}_1\ \dot{x}_2\ \dot{y}_2\ \dot{\theta}_2\ ...]^T$. We can now write Eqns. 4.12 in terms of the x_{incr} state variables.

$$\begin{aligned} x_T &= \cos\left(\sum_0^a \dot{\theta}_i\right)\sum_{a+1}^b \dot{x}_i + \sin\left(\sum_0^a \dot{\theta}_i\right)\sum_{a+1}^b \dot{y}_i \\ y_T &= -\sin\left(\sum_0^a \dot{\theta}_i\right)\sum_{a+1}^b \dot{x}_i + \cos\left(\sum_0^a \dot{\theta}_i\right)\sum_{a+1}^b \dot{y}_i \\ \theta_T &= \sum_{a+1}^b \dot{\theta}_i \end{aligned} \tag{4.13}$$

The Jacobian of the rigid-body transformation T with respect to the state variables x_{incr} is:

$$\frac{\partial[x_T\ y_T\ \theta_T]}{\partial[\dot{x}_i\ \dot{y}_i\ \dot{\theta}_i]} = \begin{cases} \begin{bmatrix} 0 & 0 & -(x_b - x_a)\sin(\theta_A) + (y_b - y_a)\cos(\theta_a) \\ 0 & 0 & -(x_b - x_a)\cos(\theta_A) - (y_b - y_a)\sin(\theta_a) \\ 0 & 0 & 0 \end{bmatrix} & 0 \le i \le a \\ \begin{bmatrix} \cos(\theta_a) & \sin(\theta_a) & 0 \\ -\sin(\theta_a) & \cos(\theta_a) & 0 \\ 0 & 0 & 1 \end{bmatrix} & a < i \le b \\ \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} & otherwise \end{cases} \tag{4.14}$$

Given some thought, the block of Jacobians corresponding to $0 \le i \le a$ is somewhat odd. Intuitively, this Jacobian suggests that a translational error in the relative positions of a and b can be reduced by rotating the nodes $[0, a-1]$. This is an artifact of our state representation, which (purposefully) does not model the projective effect of robot motion. If we consider the projective nature of the robot trajectory, we know that rotating a node $0 \le i < a$ does not affect the relative positions of nodes a and b: such a rotation will have the same projective effect on both nodes, leaving their relative positions unchanged. If we instead computed the Jacobian of the constraint between a and b in the x_{rel} state space (a more accurate, if less stable representation), we would find that the Jacobian for nodes $[0, a-1]$ is exactly zero. In other words, over the interval $0 \le i \le a$, the Jacobian is usually better approximated by zero than by Eqn. 4.14.

In summary, the Jacobian that we use is:

$$\frac{\partial[x_T\ y_T\ \theta_T]}{\partial[\dot{x}_i\ \dot{y}_i\ \dot{\theta}_i]} \approx \begin{cases} \begin{bmatrix} \cos(\theta_a) & \sin(\theta_a) & 0 \\ -\sin(\theta_a) & \cos(\theta_a) & 0 \\ 0 & 0 & 1 \end{bmatrix} & a < i \leq b \\ \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} & otherwise \end{cases} \tag{4.15}$$

In order to compute a step, we also need to compute the block-diagonal approximation M of the information matrix $J^T\Sigma^{-1}J$. Since each constraint is independent, Σ^{-1} is a block-diagonal matrix composed of the individual Σ_i^{-1} matrices. Thus, we can build up our approximation for M by considering each constraint one at a time:

$$J^T\Sigma^{-1}J = \sum_i J_i^T\Sigma_i^{-1}J_i \tag{4.16}$$

Let R_i be the repeated block of the Jacobian J_i for $a \leq i \leq b$, i.e.:

$$R_i = \begin{bmatrix} \cos(\theta_a) & \sin(\theta_a) & 0 \\ -\sin(\theta_a) & \cos(\theta_a) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.17}$$

From the repeated-block structure of J_i it is easy to see that the block diagonals will be $R_i^T\Sigma_i^{-1}R_i$ for blocks $a < i \leq b$, and zero elsewhere. M can be computed by adding up these blocks for each constraint, discarding all but the values on the diagonal.

4.3.4 The Error-Distributing Tree

In the previous sections, we showed how the state update d can be computed for a constraint. This computation requires the Jacobian of the constraint, which we derived for the ubiquitous case of a rigid-body constraint. A simple implementation could compute d literally and add it to the state vector x. For a graph with N nodes, this would result in a runtime cost of $O(N)$ per constraint.

However, the Jacobian for a rigid-body constraint has an interesting structure: it is *constant* over the interval $a < i \leq b$, and zero elsewhere. This means that the error is distributed uniformly over the nodes in the loop, subject to the reshaping effect of M. This section describes a specialized data structure for the state that allows updates (including the effects of M) to be performed in $O(\log N)$ time.

D-Trees

This data structure makes extensive use of a simpler novel data structure we call a *distribution tree*, or dtree. A dtree conceptually maintains an array of values v_i $(0 \leq i < N)$ and supports two operations: compute the value at a particular index

i, and increase all values at indices $\geq i$ by some amount. These operations could be implemented using a trivial array in $O(N)$, but the dtree we describe here is $O(\log N)$.

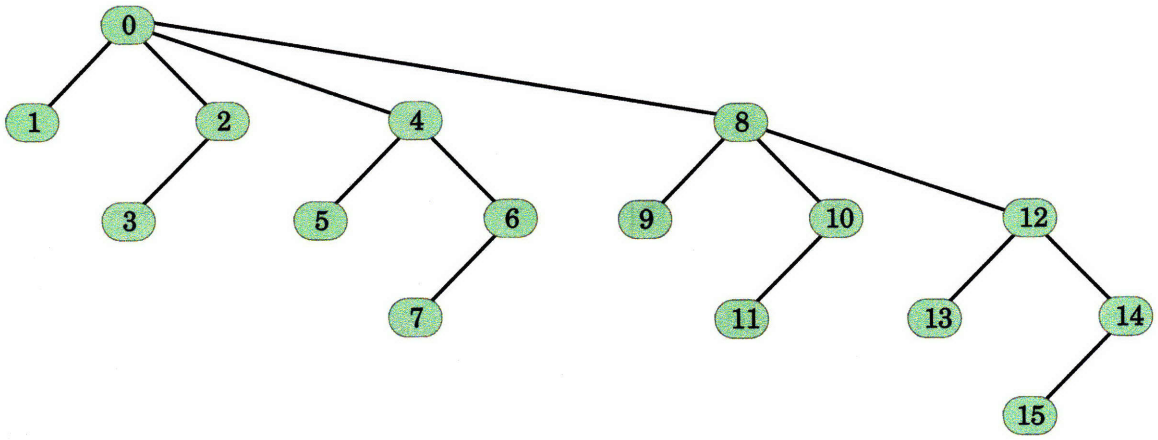

Figure 4-3: Distribution Tree. A dtree (a binomial tree) stores an array of data, allowing an amount to be added to any contiguous portion of the array in $O(\log N)$ time. The value of an array element i is the sum of the node values along the path from node i to the root. The indices of each node are shown.

A dtree is a binomial tree: it has 2^{k-1} nodes (such that $2^k \geq N$) and height k. We give each node an index such that the parent node of node i is $i\&(i-1)$ (see Fig. 4-3). Each node also maintains a value, n_i. We then define the value of v_i to be the sum of the node values n_j along the path from the root to node i. The value v_i is computed in at most $O(\log N)$ time, as seen in Alg. 4.3.4.

Algorithm 5 dtree-get(i)

```
value = 0
loop
  value = value + n_i
  if i = 0 then
    return value
  end if
  i = i&(i − 1)
end loop
```

Adding an amount δ to all nodes $[i, N-1]$ is possible by adding δ to a set of nodes such that every path from the root to node $j \geq i$ goes through exactly one adjusted node. There is only one such set, which can be found by starting at node i and repeatedly traveling to the right sibling, or to the right sibling of the parent if there is no right sibling.

For example, adding δ to all nodes ≥ 3 can be accomplished by adjusting nodes n_3, n_4, n_8, n_{16}, etc. For any given node j, the next node in the sequence is $j + j\&(j-1)$. This process guarantees that the right-most k bits of the node index are zero after k steps, thus there can be at most $O(\log N)$ steps (see Alg. 4.3.4).

Algorithm 6 dtree-add(i, δ)

while $i < MAX$ **do**
  $n[i] = n[i] + \delta$
  if $i = 0$ **then**
    return
  end if
  $i = i + (i \oplus (i-1))$
end while

We now note that we can add a δ to any contiguous *subset* of nodes $[a, b]$ by simply performing two updates: one which adds δ to each element after a and another which subtracts δ from each element after $b+1$. The operation remains $O(\log N)$.

Error Distribution Tree

The error distribution tree will allow a value Δ to be distributed over a set of buckets δ_i for $i \in [a, b]$ in proportion to the "weights" w_i of those buckets. In our map optimization application, we are distributing error around a loop in proportion to the positional uncertainty of each node[2].

We define the *cumulative weight* C_i of a node to be $C_i = \sum_{j=0}^{i} w_j$, which allows us to write δ_i as:

$$\delta_i = \delta_i + \begin{cases} 0 & i \leq a \\ \Delta * \frac{C_i - C_a}{C_b - C_a} & a < i < b \\ \Delta & i \geq b \end{cases} \tag{4.18}$$

A simple dtree could easily implement the steps for $i \leq a$ and $i \geq b$, since the quantities are constant over their ranges of indices. The interval from $a < i \leq b$ is more difficult, since it is a function of C_i, which is different for every element in the interval.

Let us implement Eqn. 4.18 in terms of *two* dtrees, *offset* and *scale* such that:

$$\delta_i = \text{offset}(i) + \text{scale}(i) * C_i \tag{4.19}$$

We can now rewrite Eqn. 4.18 so that each update is of the form $K_1 + K_2 * C_i$, such that K_1 and K_2 are constant over their intervals:

$$\delta_i = \delta_i + \begin{cases} 0 + 0 * C_i & i \leq a \\ -\Delta * \frac{C_a}{C_b - C_a} + \Delta \frac{1}{C_b - C_a} * C_i & a < i < b \\ \Delta + 0 * C_i & i \geq b \end{cases} \tag{4.20}$$

We are now done; we can implement Eqn. 4.20 by updating the *offset* dtree according to K_1 and updating the *scale* dtree according to K_2. These operations require a constant number of *dtree-add* operations, each of which is $O(\log N)$. When

[2]Each error-distribution supports *scalar* operations. Since each node in the graph has three degrees of freedom, we will use three separate trees in order to be able to distribute error over x, y, and θ.

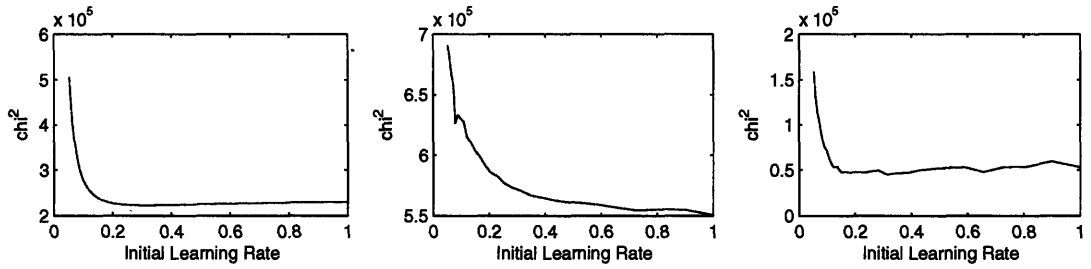

Figure 4-4: Initial Learning Rate effect on χ^2. The initial learning rate has an impact on the convergence rate of the algorithm, as illustrated here by the χ^2 value after 20 iterations. Very low initial learning rates (less than about 0.1) incur a severe penalty, but the effect of larger learning rates is fairly modest. Left: CSW data set, middle: Freiburg, right: Intel research center.

reading from the error-distribution tree, we must read from both the offset and scale dtrees, multiplying the latter value by C_i as in Eqn. 4.19. Our total computational cost for each of these operations is $O(\log N)$.

By using this data structure, we must pay $O(\log N)$ to compute the position of any pose (i.e., to compute Eqn. 4.19). However, a constraint connects only two nodes, and so requires only two lookups.

The total computational cost of processing a constraint is thus $O(\log N)$, a significant improvement over the naive $O(N)$ implementation.

4.3.5 Initial Learning Rate

A free parameter of the algorithm is the initial, maximum learning rate. Learning rates that are too large result in large steps that often clamp. If two antagonistic constraints clamp in an iteration, little progress is made towards finding an equilibrium between them, since the constraint processed last will override the constraint processed earlier.

Conversely, a learning rate that starts off too small will exhibit slow convergence, and is less likely to escape a local minimum.

In Fig. 4-4, we have plotted the χ^2 error after 20 iterations as a function of the initial learning rate. Since the convergence behavior is also dependent on the particular optimization problem, we have plotted results for three representative data sets. It is quickly evident that low initial learning rates are far more detrimental to convergence than higher learning rates: low initial learning rates have much higher χ^2 error than higher learning rates, and there is virtually no penalty for having "too large" a learning rate. We consequently recommend an initial learning rate of about 1/3.

4.3.6 Algorithm Summary

In the previous sections, we have described the essential elements of the algorithm. In this section, we provide a consolidated and concise description of the algorithm (see Alg. 7). We also elaborate on some implementation details.

Algorithm 7 SGD-Optimize-Graph

$\lambda = 1/3$
repeat
  Randomly permute order of *Constraints*
  {Update approximation $M = J^T\Sigma^{-1}J$. Let M_i be the i^{th} diagonal block.}
  for all $\{a, b, t_{ab}, \Sigma_{ab}\}$ in *Constraints* **do**
    compute R using Eqn. 4.15
    $W = R^T\Sigma_{ab}^{-1}R$
    for all $i \in [a+1, b]$ **do**
      $M_i = M_i + W$
      Update error-distribution trees with M
    end for
  end for
  $\Gamma = \max(\Sigma_i^{-1})$

  {Modified Stochastic Gradient Descent}
  for all $\{a, b, t_{ab}, \Sigma_{ab}\}$ in *Constraints* **do**
    $r = X_a^{-1} \oplus X_b$
    $r(3) = \text{mod2pi}(r(3))$
    compute R using Eqn. 4.15
    $s = -(b-a)\lambda R^T\Gamma^{-1}\Sigma_{ab}^{-1}r$
    $s_{max} = X_b - (X_a \oplus T_{ab})$
    $s = \text{clamp}(s, s_{max})$
    distribute$(a+1, b, s)$
  end for
  $\lambda = \lambda/(\lambda+1)$
until converged

The essential strategy is to compute the total correction s to the map, i.e., the amount by which node b will be brought closer to the position predicted by the constraint. We can then distribute this total motion over the nodes $[a+1, b]$, in proportion to the confidence of each individual node (i.e., redistributed according to M.)

The total correction s is easy to compute because of the block structure of the Jacobian: a total of $(b-a)$ state variables will each move by the same amount (subject to subsequent reshaping via matrix M). Since the state variables are in the x_{incr} state space, the total motion is the sum of these individual motions.

4.4 Results

We now consider the performance of our algorithm versus other algorithms. We begin with synthetic data sets, which allow us to more easily manipulate the characteristics of the problem (and provide ground truth), then demonstrate the algorithm on real

data.

Our first dataset, the CSW dataset, will include comparatively exhaustive analyses of the performance of several algorithms (including our method). Much of this analysis generalizes to other datasets.

4.4.1 CSW Dataset

The CSW dataset named after the CSAIL Student Workshop at which it was first used. It has since become a common synthetic benchmark for non-linear optimization algorithms. It has 3500 nodes in a "grid world" configuration with a total of 5598 edges (see Fig. 4-8). This gives it an average node degree of about 1.6, making it as a relatively sparse problem.

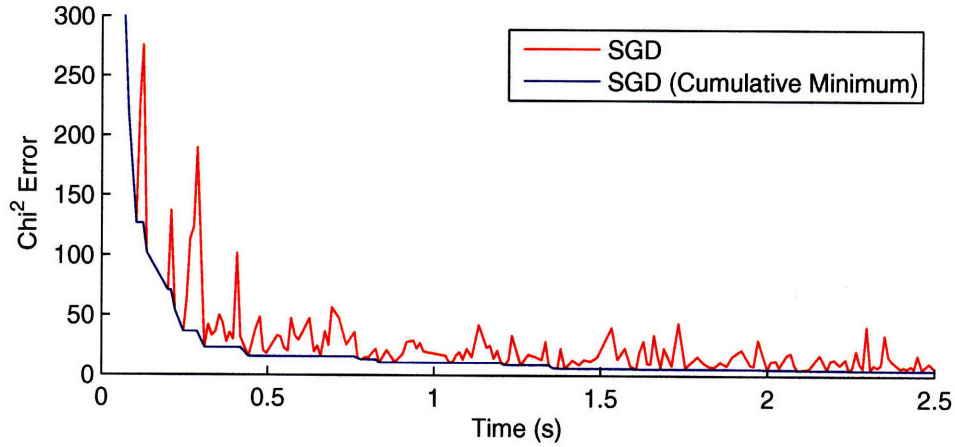

Figure 4-5: CSW Dataset, SGD Algorithm. The stochastic nature of the optimization method causes the error estimate to fluctuate near the minimum. An application that used SGD, however, would use the best-cumulative map (blue).

A typical application using SGD would not use the most recent state estimate (which is constantly exploring new parts of the state space), but would instead use the best known map to-date (as measured by χ^2 error; SSE cannot be computed when the global minimum is not yet known). The quality of the cumulative-best map versus the stochastically optimized map is shown in Fig. 4-5. Note that since SGD is a randomized algorithm, the behavior varies from run to run. We will use the cumulative minimum plot for the remainder of the figures.

As we will see, our algorithm converges to lower-error configurations significantly faster than several representative existing methods (see Fig. 4-6), with the exception of sparse Cholesky Decomposition which achieved similar performance.

Extended Kalman Filter

We also processed the CSW dataset using an Extended Kalman Filter. The total elapsed time was 2.5 hours, more than three orders of magnitude slower than either the sparse Cholesky decomposition or our method. The quadratic growth in computational complexity can be seen in Fig. 4-7. At the end of the dataset, the EKF required

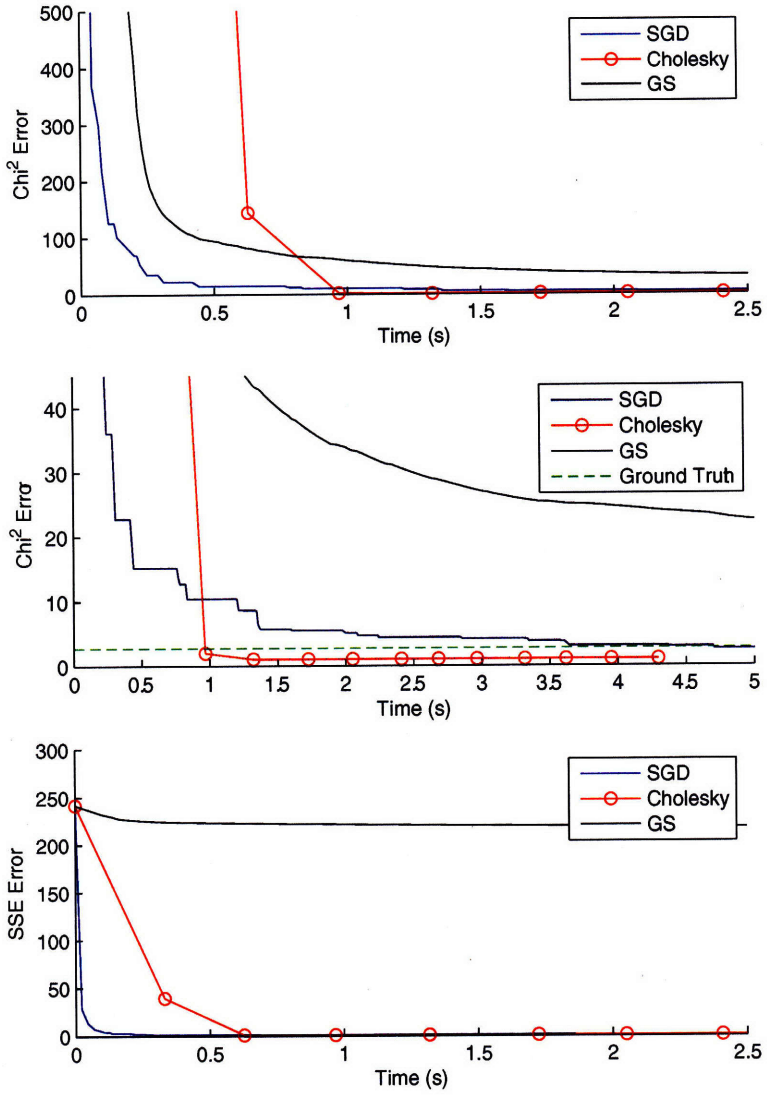

Figure 4-6: CSW Dataset Convergence Performance. The χ^2 error (top, and closeup in middle) and state squared error (SSE, bottom) paint two complementary pictures of optimization performance. Our stochastic gradient method (SGD) converges reduces both the χ^2 error and SSE error very quickly. Cholesky Decomposition converges more slowly in early steps, but finds the globally optimal solution after about one second. Gauss-Seidel relaxation, while appearing to reduce the χ^2 error reasonably well, has actually produced a very poor map estimate, as seen in the SSE plots and in Fig. 4-8.

841 MB of memory to store the covariance matrix, which grows as $O(N^2)$. The large amount of memory usage caused our Java implementation to experience more garbage collection operations, which caused some jitter in the timing measurements (readily visible beginning around constraint 2500).

Cholesky Decomposition

We have implemented a sparse Cholesky decomposition routine very similar to Dellaert's $\sqrt{SAM}$ algorithm. It differs in that instead of using COLAMD [8] for deter-

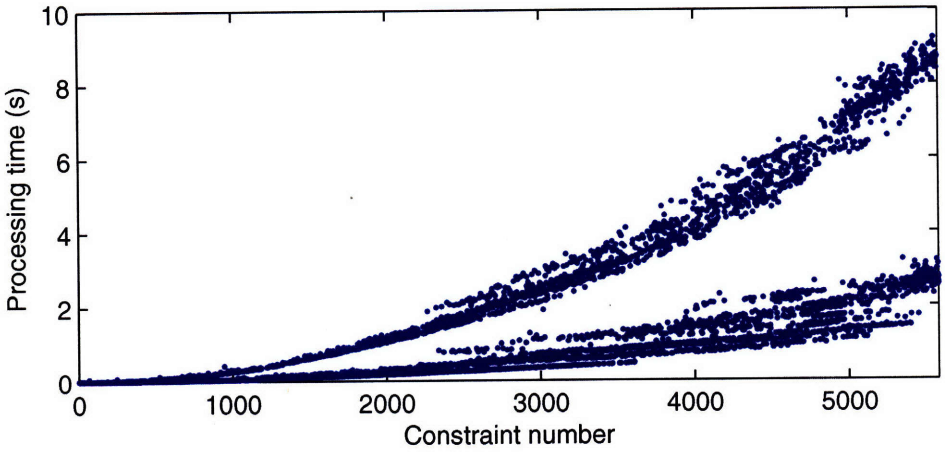

Figure 4-7: CSW Dataset, EKF Performance. Seen above are the CPU time per constraint (total elapsed time was 2.5 hours). Two trends are visible: the trend on top corresponds to the cost of performing an observation step, while the lower trend corresponds to performing a time-projection step (in which the state and covariance matrix are grown to accommodate a new state).

mining a variable ordering, it uses an exact minimum-degree ordering[3] that yields less fill-in during factorization at the expense of taking longer to compute the ordering. Since the ordering needs to be computed only once (even if the Cholesky decomposition is iterated many times), and since reduced fill-in has a significant impact on the cost of Cholesky factorization, this trade-off is often a net win. A careful comparison of different variable ordering algorithms in the context of robot mapping problems has not been conducted, however, and remains an area for future study.

In this experiment, the sparse Cholesky Decomposition processed an information matrix of size 10500×10500. The information matrix was exceptionally sparse (0.081% non-zero entries), due to the low average node degree. The resulting Cholesky factor contained 0.166% non-zero entries. This sparsity allows significant memory savings (only non-zero entries need be stored), and greatly accelerates the actual factorization and subsequent back-solves.

Sparse Cholesky decomposition is one of the fastest map optimization algorithms known, and represents a state-of-the-art algorithm. As the results show in Fig. 4-6, our algorithm reduces the error (both χ^2 and SSE) faster than Cholesky decomposition during the early iterations of optimization. However, once the Cholesky decomposition lands near the global minimum, it exhibits quadratic convergence. In cases where Cholesky decomposition finds the correct minimum, it generally finds a lower χ^2 error. However, this additional reduction in error often represents over-fitting to the data, and is not representative of an increase in map quality. This is illustrated in Fig. 4-8b, in which the χ^2 error of the ground truth is significantly higher than the χ^2 error found by Cholesky decomposition. As with machine learning algorithms in general, over-fitting can occur when a model (in this case a map) is fit to a rel-

[3]Finding the optimal variable ordering is NP-hard; the minimum-degree ordering is an effective heuristic. COLAMD is, in turn, an approximation of the minimum-degree ordering which can be significantly faster to compute.

atively small amount of data (the constraints). In the map optimization context, small average node degrees are fairly common, making optimization algorithms prone to over-fitting.

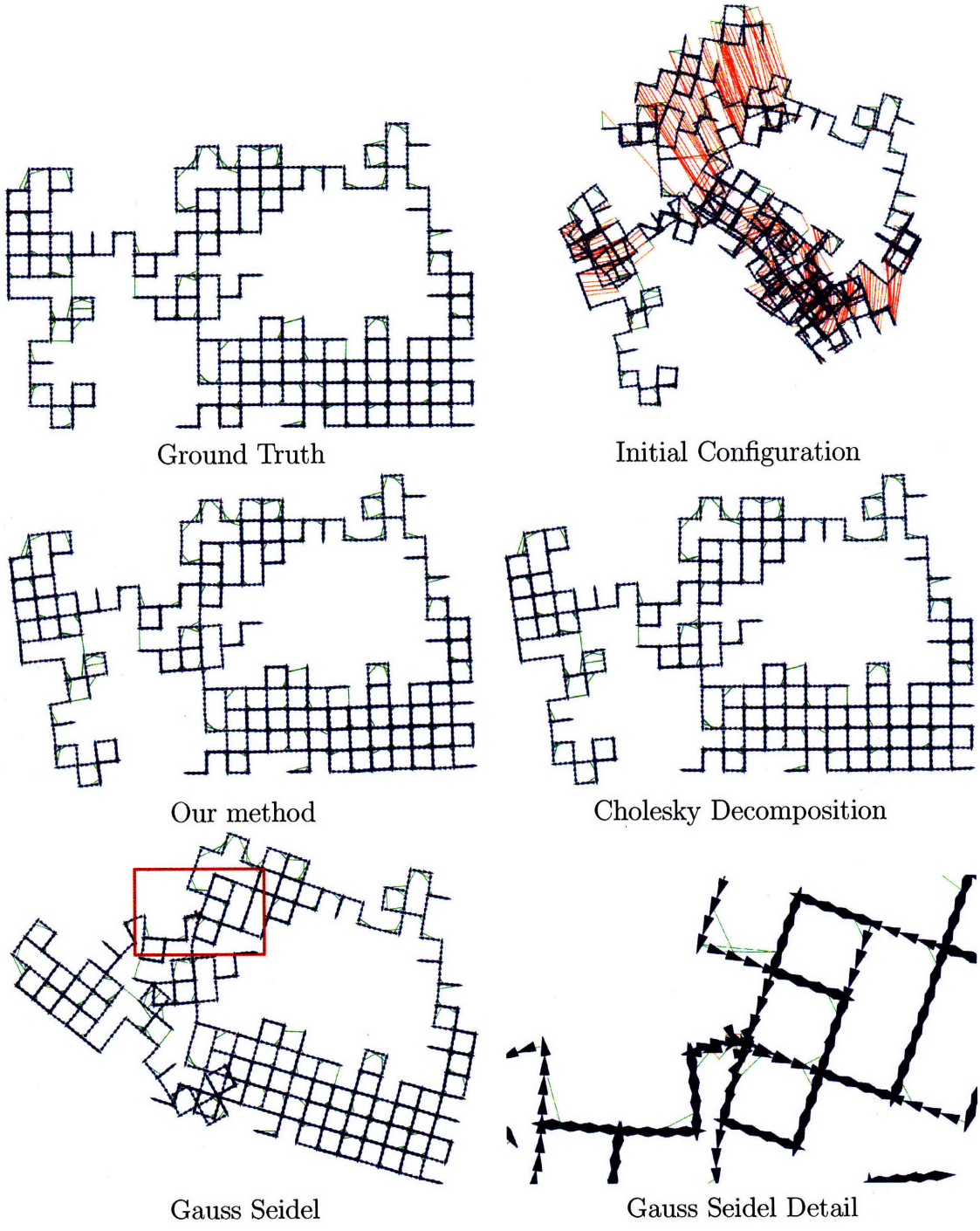

Figure 4-8: CSW Dataset Graphs. Our method and Cholesky Decomposition produce nearly indistinguishable output. Gauss Seidel converges exceptionally slowly, producing the figure above after 1.5 hours of CPU time. The detail region indicates a "knot" which is irreversibly induced early in the optimization.

Gauss Seidel Relaxation

Gauss Seidel typically exhibits poor performance on large datasets that have significant errors in their initial configuration. Intuitively, this is because each node is only affected by its immediate neighbors, but an error in one part of the graph often requires an adjustment to a distant part of the graph. With Gauss Seidel relaxation, it takes a long time for these adjustments to propagate through the graph.

This effect is significant on the CSW dataset, which is has two sub-graphs connected by a "thin" chain of poses and edges. Individually, the sub-graphs have a relatively small diameter (due to the nodes' high average node degree). Consequently, information can flow fairly rapidly throughout the sub-graph. As a result, the sub-graphs converge to low-error configurations fairly quickly. In contrast, information about the relative alignment of these two sub-graphs must pass through the thin chain of poses, which slows the rate at which the two sub-graphs can find a mutually-consistent configuration.

Consider a node j at the junction of the chain and one of the larger pieces: it has many neighbors belonging to the locally-consistent sub-graph and only one neighbor belonging to the chain. Gauss-Seidel iterations for node j will heavily favor the sub-graph's current configuration, since all but one of its neighbors support that configuration. Thus, any new information propagated along the chain is rapidly attenuated when it enters a sub-graph. As a result, the two sub-graphs converge to a mutually-consistent arrangement very slowly.

The resulting graph can have very low χ^2 error even when the two sub-graphs are poorly aligned with respect to each other. This is because almost all of the edges in the graph belong to one of the two large sub-graphs (which have found locally consistent configurations)— the error is concentrated along just a handful of edges belonging to the chain. The total χ^2 error for the graph is thus quite low, even though the positions of the nodes are far from the global minimum. This illustrates why χ^2 error cannot be used as a quality metric by itself, and why our proposed SSE metric is important for quantitative evaluations.

In early steps of Gauss-Seidel, large errors can cause nodes in the graph to move dramatically. These large state updates generally reduce the χ^2 error dramatically, which leads to smaller steps later on. However, these initial large steps can introduce large errors into the positions of some nodes that later (smaller) steps cannot undo the damage. In the CSW dataset, a node near the junction of the right and left halves of the graph experiences such a large update that it effectively ties a "knot" into the robot trajectory. However, this knot has so much less χ^2 error than the initial configuration that subsequent Gauss Seidel iterations do not have the energy to escape it. In other words, Gauss Seidel has become stuck in a local minima of its own creation. (This knot is shown in more detail in Fig. 4-8.) Given a week of CPU time, the Gauss-Seidel solution macroscopically resembles the solutions given by our method and Cholesky decomposition— however, the knot remains.

A typical way of accelerating Gauss Seidel relaxation is by performing *over-relaxation*. It addresses the fact that nodes tend to move too slowly in response to distant constraints. The intuition is that moving *past* the locally-optimal solution

may in fact move the state estimate closer to the globally-optimal solution. Indeed, over-relaxation *can* increase the convergence rate of Gauss-Seidel (modestly, in our experiments), but the larger step sizes also greatly increase the risk of "knots" and divergence.

4.4.2 High Noise Dataset

Our second dataset illustrates the effect of noisy odometry and loop closures (see Fig. 4-9). It is a synthetic dataset of 501 nodes with 544 edges, i.e., reflecting situations in which loop closures are difficult to obtain.

In other datasets presented in this thesis, it is usually possible to recover a map that subjectively looks like the ground truth. In this high-noise dataset, this is not the case: even the best map is visibly imperfect. Producing the best map *possible*, even under high noise situations, is a key test of an optimization algorithm. Success in these cases allows robots to operate in more difficult environments and use less expensive (noisier) sensors.

In this dataset, the only algorithm to produce a good result is our method. The convergence rate of Gauss Seidel, while initially rapid, quickly drops off and leads to a poor solution in both terms of χ^2 and SSE.

The Extended Kalman Filter solution, which took 17 s to compute, suffered from significant linearization error, leading to both the worst χ^2 score (31.9) and the worst SSE score (45.6). While EIFs, SEIFS, and ESEIFs might be faster than the EKF, they will exhibit similar error. This illustrates the importance of methods that can relinearize its constraints.

The solution computed by Cholesky decomposition is visibly distorted (see Fig. 4-9), and has a significantly higher SSE error (see Fig. 4-10) than our solution. The first few iterations of Cholesky decomposition were very chaotic; the violent updates to the state estimate resulted in a "knot" in the center of the map, much like the knot that resulted when using Gauss-Seidel on the CSW data set. Additional iterations of Cholesky decomposition have no effect on this knot.

The similarity of the χ^2 errors (3.5 for Cholesky, 3.06 for our method), illustrates that χ^2 error is not always a good indicator of solution quality; SSE is more revealing in this case.

4.4.3 P5K20K Dataset

P5K20K is a synthetically generated dataset with 5000 nodes and 20000 edges. This gives it a fairly high average node degree, which, with the fairly large size of the state vector, makes it is a challenging optimization problem (see Fig. 4-11 and Fig. 4-12).

The Cholesky factorization is much slower on this problem than on previous examples due to the fairly high average node degree and relatively large sensing range. The result is that the Cholesky factor is an order of magnitude more dense than on the CSW dataset: the information matrix has a similar density (0.086%), but the Cholesky factor has a density of 1.96% (versus 0.166% on the CSW dataset).

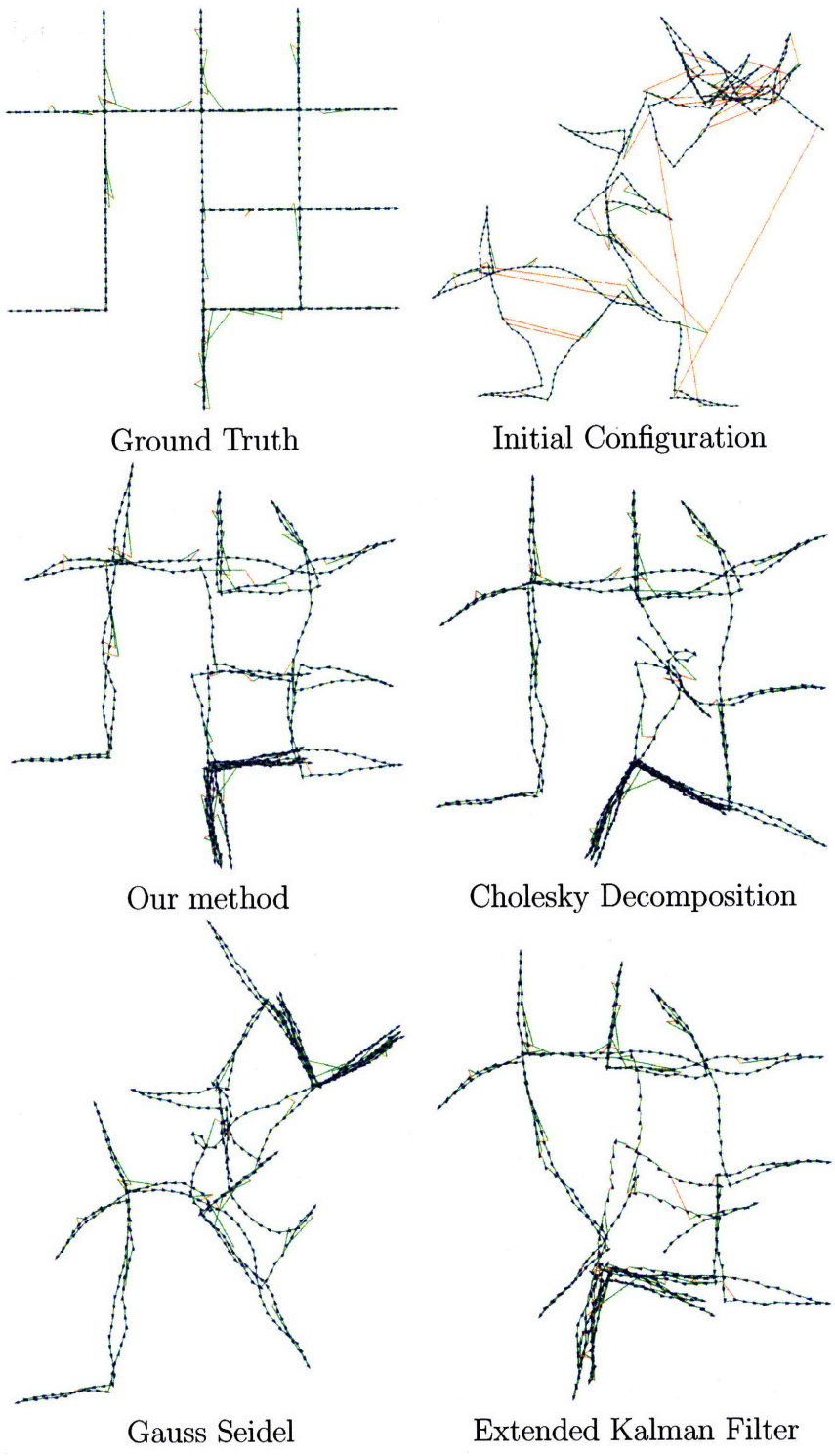

Figure 4-9: High Noise Dataset. The true configuration (top left) and odometry-derived initial configuration (top right) are shown. In this high-noise case, only our method finds a reasonable solution. Cholesky Decomposition has become stuck in a local minimum (note the knot in the center of the map), while Gauss Seidel and the Extended Kalman Filter have both produced poor maps.

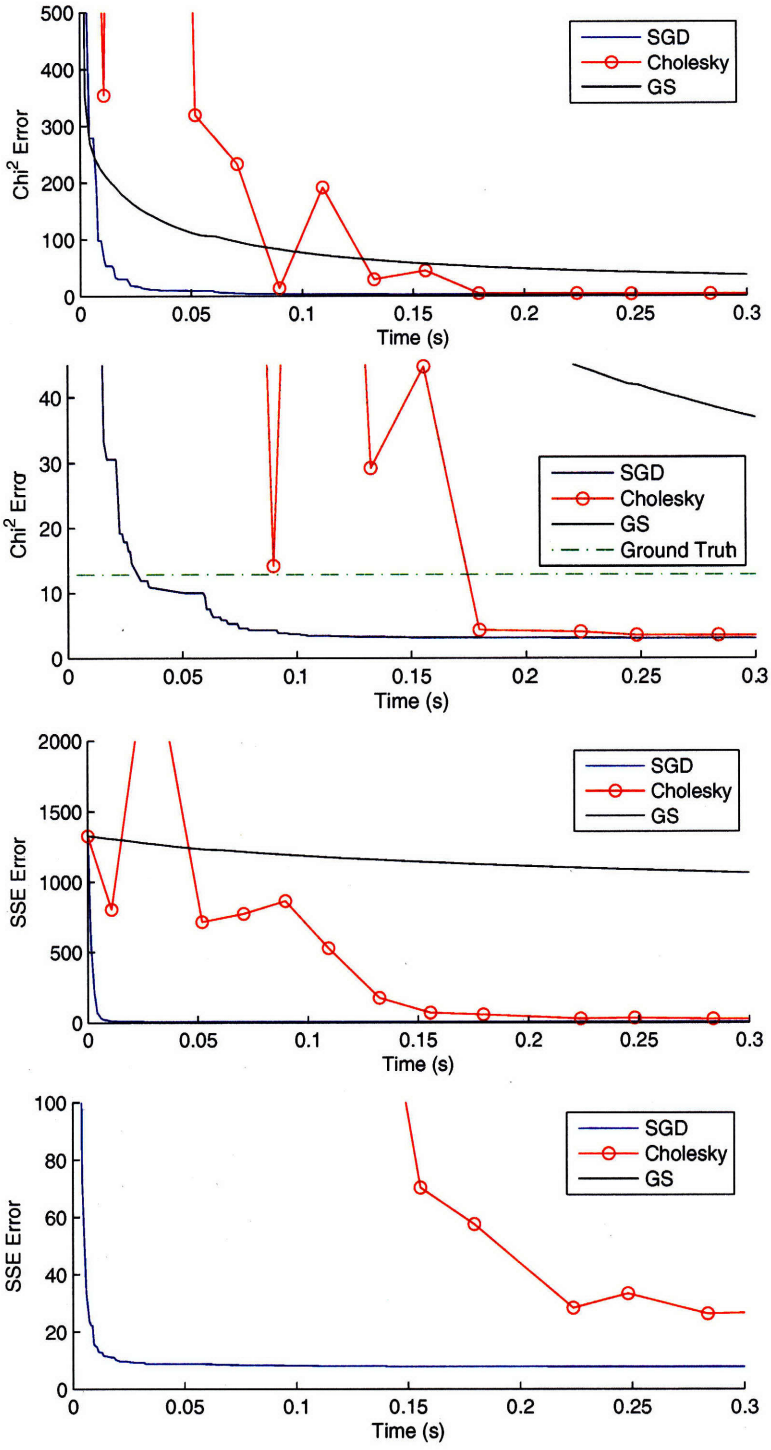

Figure 4-10: High Noise Results. The χ^2 error and SSE error are plotted for three different algorithms. Each error metric is plotted twice: the second plot is an enlargement of the first to show detail. Only our method finds the global minimum. Cholesky decomposition gets stuck in a local minimum with fairly low χ^2, but large SSE. (Not shown: EKF took 17.2 s, and had $\chi^2 = 31.9$ and $SSE = 45.6$.)

It takes Cholesky Decomposition three iterations totalling 91 seconds of computa-

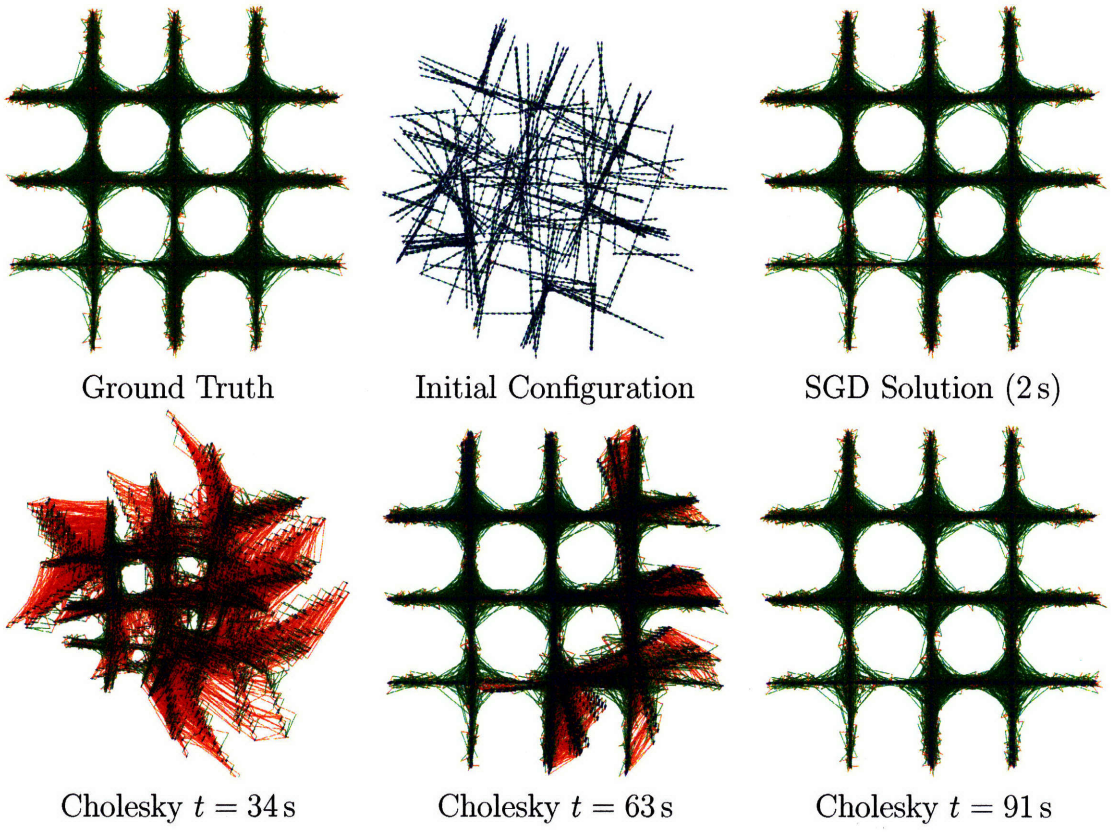

Figure 4-11: P5K20K Graphs. The true configuration (top left) and an odometry-derived initial configuration (top middle) are shown. The configuration resulting from three successive iterations of Cholesky Decomposition are shown (bottom, left to right). After 91 seconds, its solution is comparable to our solution (top right), which was obtained in 2 seconds.

tion to produce a solution ($\chi^2 = 1.47$, SSE=0.0387) similar to the solution produced by our method in 2 seconds ($\chi^2 = 2.9$, SSE=0.0307). Given a fourth iteration, Cholesky Decomposition improves the estimate to $\chi^2 = 0.9983$ and SSE=0.00744, but with a total elapsed time of 124 s. Interestingly, this solution is numerically optimal. Our method cannot typically recover the numerically optimal solution due to its stochastic exploration of the state space and the effects of approximations made in the algorithm. However, as this dataset illustrates, our method produces good maps in a fraction of the time.

The χ^2 error of the solution produced by our algorithm can be reduced significantly by using a few iterations of Gauss-Seidel relaxation. Such a hybrid method combines SGD's ability to rapidly recover the basic structure of the graph with Gauss-Seidel's ability to fine-tune local structure (see Fig. 4-13).

The hybrid SGD/Gauss-Seidel method can noticeably reduce the χ^2 error, but has little effect on SSE error. Tiny perturbations in the positions of individual poses can have a significant impact on the error of individual constraints, but so long as these perturbations do not have a cumulative effect, their effect on SSE is small. SGD's stochastic exploration of state space tends to add noise to the position of each node in the graph: this causes an increase in χ^2 error of the constraints connected

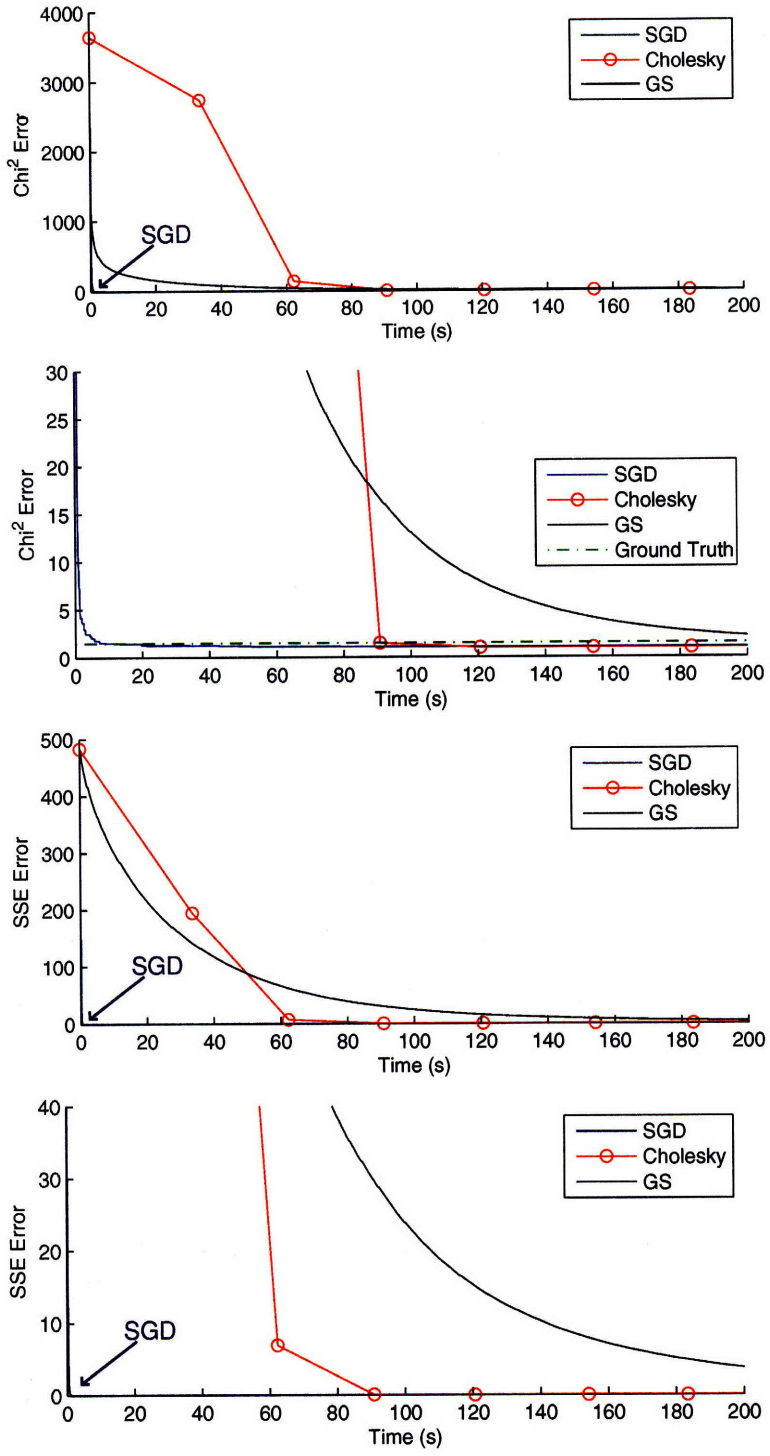

Figure 4-12: P5K20K Results. χ^2 and SSE error are plotted (with enlarged versions for each) for several algorithms. SGD converges much more quickly than the other methods. Unlike previous data sets, however, all methods ultimately converge on a good solution.

to those nodes, even though those nodes are very close to their optimal position. Running Gauss-Seidel allows the noise to be corrected, significantly reducing the χ^2 error by making minute adjustments to the position of each node. However, these

minute adjustments have little effect on SSE. This is precisely the behavior we see in Fig. 4-13. This behavior also illustrates that χ^2 error, by itself, is not necessarily indicative of map quality.

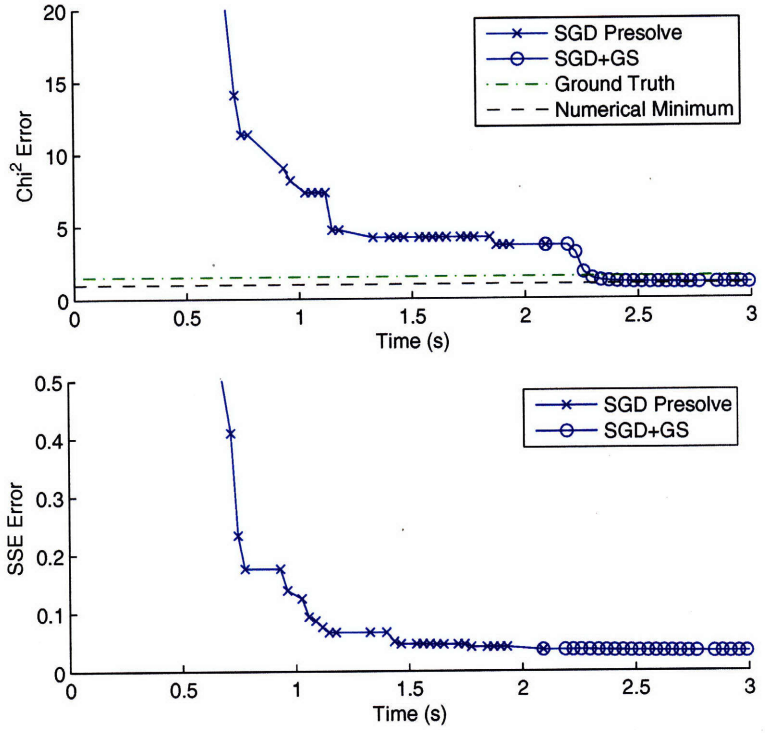

Figure 4-13: Hybrid Method Results on P5K20K Dataset. Our method can be combined with Gauss-Seidel to produce results with lower χ^2 error by using Gauss-Seidel to refine a configuration produced by our algorithm. In contrast, the effect on SSE error is minimal; this shows that SGD produces graphs whose quality is higher than χ^2 alone would indicate.

4.4.4 Intel Research Center Dataset

The Intel Research Center dataset is composed of 875 nodes connected via 15605 edges, and represents a real data set spanning 44.9 minutes [29]. Cholesky, Gauss-Seidel, and our method all perform well on this dataset. Although our method does very well with respect to SSE error[4], our method lags behind in terms of χ^2 error. Our method is an approximation that neglects certain projective effects (as seen in Fig. 4-16). That, coupled with its stochastic exploration of the state space, limits its ability to find extremely low-error configurations. In this case, all the methods achieved very low errors ($\chi^2 < 1$), and so SGD's larger χ^2 error is more conspicuous. However, as we have seen in earlier datasets, this does not mean that the map is worse. If minimizing χ^2 error is critical, Gauss-Seidel relaxation can be used to fine-tune the solution computed by SGD.

[4]In the absence of an actual ground truth, we compute SSE with respect to the global minimum solution.

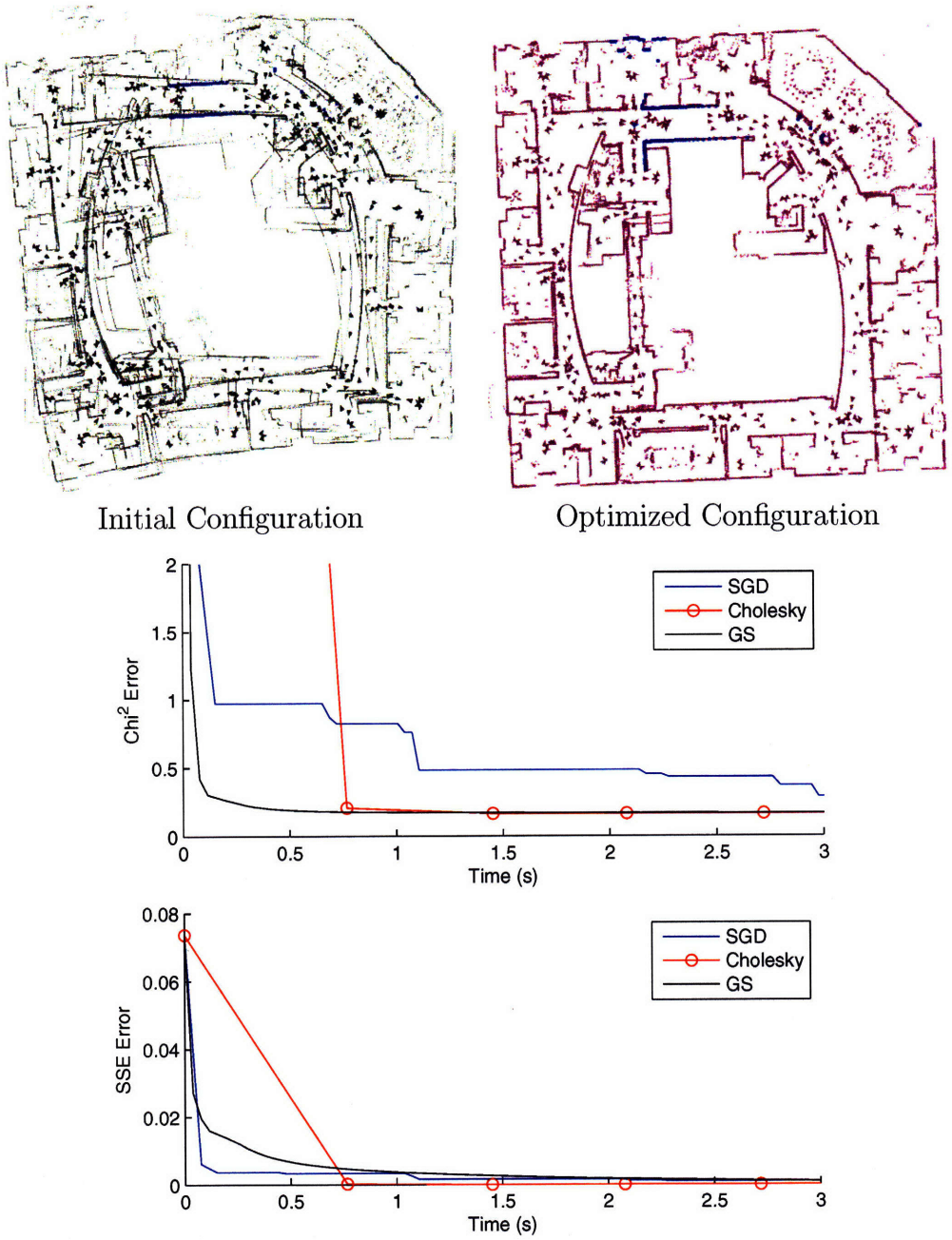

Figure 4-14: Intel Research Center. All the methods achieve very low error, though their convergence rates differ. Note that our method takes a quick and early lead in SSE error, though it lags in χ^2 error.

4.4.5 Stanford Gates Dataset

The Stanford Gates dataset includes 34.6 minutes of data. As processed by our lidar loop-recognition algorithms (described in Chapter 3), the raw data resulted in a graph containing 679 nodes and 7121 edges. Fig. 4-15 shows the initial configuration and optimized configuration, with timing results for three algorithms.

Only Gauss-Seidel produces a poor result: Cholesky Decomposition, EKF, and

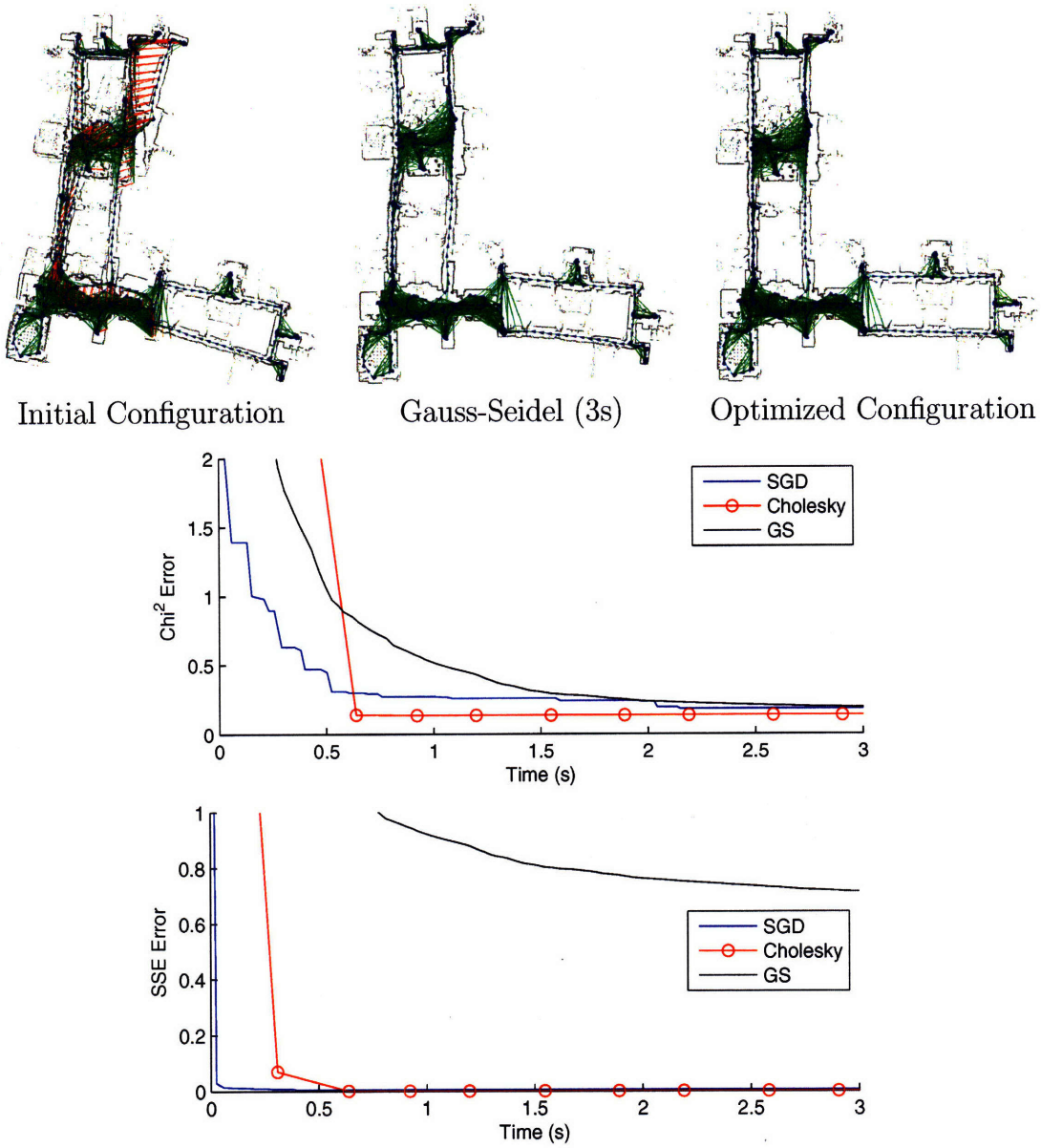

Figure 4-15: Stanford Gates Building Results. Once again, Gauss-Seidel quickly reduced the χ^2 error, but results in a map with noticeable distortions (and large SSE). Both Cholesky decomposition and our method quickly produced good maps, with our method producing low-error maps earlier. (Not pictured: a map produced by an EKF with $\chi^2 = 0.134$ and SSE=0.00031 in 881 s.)

our method all produce very good maps. Our method converges faster than other methods initially, though Cholesky Decomposition ultimately produces a graph with lower χ^2 error. The Extended Kalman Filter is orders of magnitude slower than either Cholesky Decomposition or SGD.

4.4.6 Dog Leg Configuration

In certain contrived cases, errors due to our approximation of the Jacobian can be observed. We have purposefully eliminated many projective effects in both our state

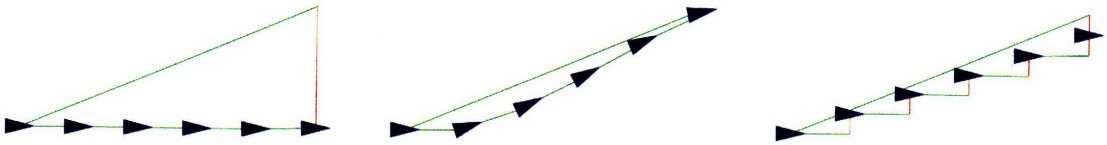

Figure 4-16: Dog leg configuration. The dog leg initial condition (left) has a single ill-satisfied constraint; the optimal solution is shown in the middle. Our proposed algorithm converges to an approximate solution that does not reflect all of the projective constraints in the configuration. This effect is minor in non-contrived cases, however.

representation and in our Jacobians. These allow fast and robust updates, but they are not exact.

Fig. 4-16 shows the "Dog Leg" configuration[5]. The nodes are initialized along a straight line. A constraint is added between the first and last node that suggests a different position (but *not* a different orientation) of the last node. In the initial configuration, the orientation error for every constraint is zero, and so the state updates computed by our method include no rotational component. A lower χ^2 error can be obtained by rotating the intermediate nodes, as seen in the middle of the figure.

Real-world datasets rarely exhibit artifacts as noticeable as those in this contrived case. This type of error is significant only when there are large rotational errors between physically distant nodes and there are no other constraints available to help guide the orientation of the intermediate nodes.

4.4.7 Robustness

Compared to other algorithms, our proposed method is substantially more robust to initialization error. Specifically, our method can find the global minimum even when other methods get stuck in local minima.

This robustness can be measured. The likelihood of an algorithm getting stuck in a local minimum increases with the noise of the observations in the graph: noisy observations are more likely to cause the algorithm to erroneously step in a sub-optimal direction. We can test the robustness of an optimization algorithm by generating a large number of synthetic datasets with varying degrees of noise, and counting the number of graphs that each algorithm successfully solves. We detect failure by comparing the normalized χ^2 to 1.0, its expected value at the global minimum: convergence to significantly larger values indicates a local minimum.

Using a synthetic graph generator, we generated about 15000 graphs. Each graph consisted of 1000 nodes and 200 landmarks; nodes were connected by odometry measurements (whose noise was the independent variable). Finally, 5000 landmark observations were generated with constant-magnitude noise.

We expect, a priori, that algorithms will have an increasing tendency to get stuck in a local minimum with increasing noise. For this study, we continue to optimize until either a generous time bound is exceeded or until the algorithm's convergence

[5]We thank Udo Frese for suggesting the Dog Leg configuration.

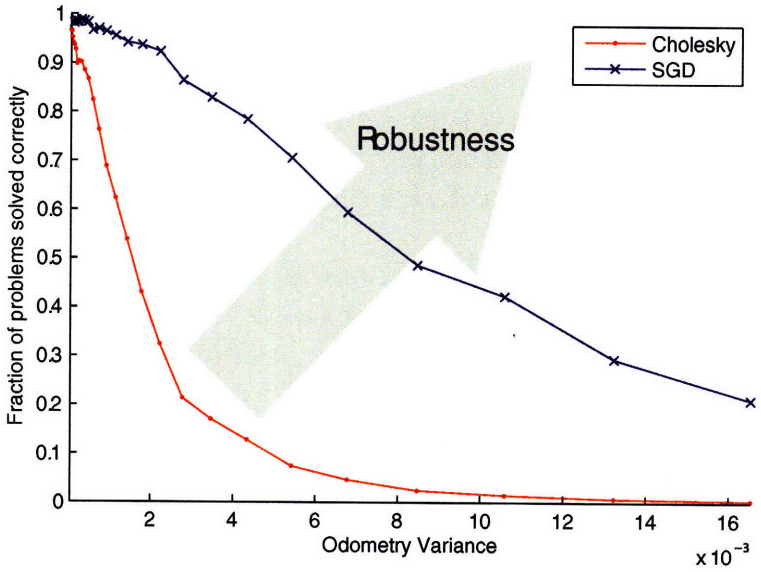

Figure 4-17: Convergence Robustness. On problems with significant noise and poor initial estimates, our algorithm finds the global minimum much more often than a Cholesky-Decomposition based method.

rate has slowed dramatically. We cannot reliably distinguish between a true local minimum versus a long valley that ultimately does lead to the global minimum, but from a practical standpoint, these two cases are equally problematic to applications.

We compare our approach to a Cholesky Decomposition-based method similar to Dellaert's $\sqrt{SAM}$. This Cholesky Decomposition based approach is one of the most stable and fastest performing algorithms that we have tested (and generally outperforms Gauss-Seidel and non-iterated forms of the Kalman filter and Information Filter). However, our algorithm consistently produces better results (normalized χ^2 values near the expected value of 1.0) at much higher noise levels than Cholesky Decomposition. In practical terms, this means that our algorithm can produce correct answers when Cholesky Decomposition (and other similar methods) produce unusable maps.

Our algorithm selects constraints in a pseudo-random way, taking care to ensure that every constraint is processed before any constraint is processed a second time. Processing the constraints in random order increases the algorithm's ability to explore the state space (and thus to escape local minima).

We have observed that when our algorithm fails to produce a good map, the optimization often diverges: the χ^2 increases by orders of magnitude. Since χ^2 error has an expected value of 1.0, such divergences are easy to detect. A robot could conceivably perform some fail-safe behavior when failure was detected. In contrast, when Cholesky Decomposition produces a bad map, the χ^2 error is often large, but not terribly so (e.g., less than 100). In cases where sensor noise covariances have been poorly modelled, it could be difficult to differentiate between map divergence and covariance modelling error.

4.5 Summary

We have presented a new map optimization that is not only fast, but remarkably robust to poor initial conditions and noisy observations. Our approach adapts stochastic gradient descent, a learning algorithm previously used in training neural networks, and introduces a state-space representation that approximates the cumulative nature of robot motion.

We compared our method to the previous fastest non-linear method, which is based on sparse Cholesky factorization. Numerous experiments demonstrate that our method is competitive with, or faster than, existing methods. More critically, we demonstrated that Cholesky Decomposition and similar methods are significantly more susceptible to getting stuck in local minima than our algorithm. In short, these methods produce poor maps in cases where our algorithm succeeds.

We also introduced a new error metric, the mean squared error of the state estimate (SSE). Our results show that it provides a useful measure of the graph quality, and that χ^2 error alone does not paint a complete picture of optimization performance.

CHAPTER 5

Incremental Optimization of Pose Graphs

When a robot obtains new information about the environment, this information must be rapidly incorporated into the model of the world so that the robot's obstacle avoidance and path planning activities can take that information into account. An incremental optimization algorithm allows new information (nodes and edges) to be added to a pose graph, and provides for an efficient incremental update of the posterior map.

Most existing optimization methods are inherently incremental[1]. Recursive linear solutions (e.g., Extended Kalman Filter (EKF), Extended Information Filter (EIF), and their many variants) explicitly compute the importance of a new observation in relation to all previous observations: this allows new information to be incorporated in a single step. However, the cost per step increases quadratically in the size of the map, and poor maps can result due to the effects of linearization error.

Non-linear optimization methods like Gauss-Seidel [12] and Multi-Level Relaxation (MLR) [20] maintain no state aside from the state estimate itself. The graph can be updated between iterations without any complications. TreeMap [19] and Thin Junction Tree Filters [52] do maintain some auxiliary state (a pruned tree), but adding new nodes is relatively straightforward. Once nodes are added to the tree, the actual optimization algorithm is unchanged.

Dellaert's $\sqrt{SAM}$ [10] method also maintains additional state, namely a variable reordering. More problematic, however, is the fact that the information matrix must be factored at each iteration— a potentially slow operation. Kaess has proposed modifications that make incremental usage more efficient, including incremental variable reordering strategies and using QR (instead of Cholesky) decomposition [31].

Our batch algorithm, based upon stochastic gradient descent, has been shown to be both fast and robust. Naturally, these attributes would be very useful in an incremental context. Stochastic gradient descent (SGD), however, assumes that the

[1]The terms *iterative* and *incremental* describe distinct concepts. An *iterative* algorithm is one in which the quality of the solution can be improved by running the algorithm repeatedly. An *incremental* algorithm is one in which the underlying problem can be modified without requiring the algorithm to restart from initial conditions. They are orthogonal features of an algorithm. Both the batch algorithm described in Chapter 4 and the algorithm described here are iterative, but this chapter adds an incremental capability.

underlying optimization problem does not change over time. SGD uses a learning rate that varies as a function of time: this learning rate modulates the step size, allowing an equilibrium between all the constraints to be found.

Quickly adapting the posterior estimate to reflect new information can require large steps, which are hampered by small learning rates. Consequently, the learning rate may need to be increased when new information is added to the graph.

A simple strategy would be to reset the learning rate to its initial value, but this would be equivalent to restarting the batch optimization. This is undesirable: even though SGD rapidly decreases the error of the posterior, the robot may have already expended significant computational resources optimizing the map. Turning the learning rate back up will undo much of this work, since *all* the constraints will once again take large steps.

In this chapter, we describe an incremental version of stochastic gradient descent. The central contribution is a spatially-varying learning rate that allows us to incorporate new information without obliterating the results of previous iterations. To our knowledge, this is the first instance of stochastic gradient descent being used to estimate the parameters of a dynamic optimization problem.

We also describe a technique for performing iterations on only the parts of the graph that are most poorly optimized. New information most often perturbs areas near the robot much more than distant areas. Our method allows computational effort to be focused on optimizing the area near the robot, without needlessly refining already-optimized areas farther away.

5.1 Method

The learning rate scales the update step produced by each constraint. If we wish to rapidly incorporate information from a new constraint, the learning rate may need to be increased. In the case of the batch algorithm, in which there is a single (global) learning rate, an increase of the learning rate will cause *every* constraint to start taking larger steps. As a result, even if the effect of the new constraint is limited to a small portion of the graph, the whole graph will be perturbed away from the minimum.

Our solution is to allow different parts of the graph to have different learning rates. When a new constraint is added to the graph, the learning rate can then be selectively increased. This allows unaffected parts of the graph to retain their low-error configurations while volatile parts of the graph are allowed to rapidly find a new equilibrium.

Specifically, instead of a single global learning rate λ, we will give each node i its own learning rate Λ_i. These learning rates represent the volatility of each node in the graph. The bulk of this chapter is devoted to a set of rules describing how these learning rates should be adjusted in order to accommodate new information. The batch algorithm can be described as a special case of the incremental algorithm: in the absence of new observations, the learning rates Λ_i will all equal the learning rate λ of the batch algorithm.

In general, we note that the maintenance of learning rates is a matter of *convergence speed*, rather than correctness. Even if the learning rate is *never* increased, stochastic gradient descent will ultimately arrive near a local minimum— but it might take a long time to do so. The methods in this chapter comprise an effective set of heuristics for managing the learning rates; they model the interactions of constraints so that the learning rate can be increased sufficiently to allow rapid convergence. Simultaneously, they attempt to avoid unnecessary increases in learning rates, but as we saw in Chapter 4, there is generally a bigger performance penalty associated with picking too small a learning rate than picking too large a learning rate.

There are three basic operations that affect the learning rates: adding a new constraint (which can selectively increase learning rates), processing existing constraints (which may propagate the effects of previous learning rate increases to additional learning rates), and reducing the learning rates according to a generalized harmonic progression.

5.1.1 Adding a new constraint

When adding a new constraint, we must estimate how large a step should be taken to reduce its residual. Once determined, we can compute the learning rate that will permit a step of that size.

The graph's current state estimate already reflects the effects of a number of other constraints. The step resulting from the addition of a new constraint should reflect the certainty of the new constraint relative to the certainties of the constraints already incorporated into the graph. Let gain β be the fraction of a full-step that would optimally fuse the previous estimate and the new constraint. β can be derived by differentiating the χ^2 cost of two Gaussian observations of the same quantity, or manipulated from the Kalman gain equation:

$$\beta = \Sigma_i^{-1}(\Sigma_i^{-1} + \Sigma_{graph}^{-1})^{-1} \tag{5.1}$$

Computing β exactly would be a computationally expensive operation (equivalent to performing a Kalman step). Instead, we estimate Σ_{graph}^{-1} from the diagonals of the information matrix. We have already approximated these uncertainties in our diagonal approximation to the information matrix M, which we described in Chapter 4. This is an approximation because, in truth, the states are correlated. However, it still yields a serviceable approximation.

We now compute the learning rate that will allow a total correction s of size βr_i. Consider the state update equation (unchanged from Chapter 4), where s represents the total correction:

$$s_i = \lambda(b-a)R_i^T\Sigma_i^{-1}r_i = \beta r_i \tag{5.2}$$

Using Eqn. 5.2, we can solve for the learning rate that would result in a step of size $s = \beta r_i$. Because there are three degrees-of-freedom per pose, we obtain three simultaneous equations for λ; we could maintain separate learning rates for each, but

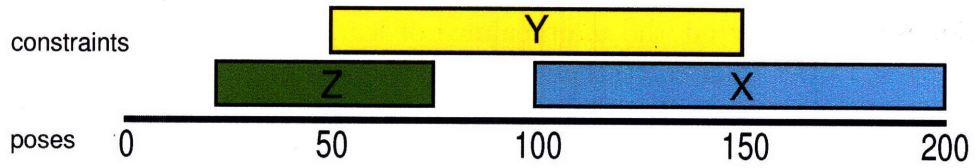

Figure 5-1: Three overlapping constraints. A disturbance in constraint X will be propagated through Y to Z, even though X and Z do not overlap. At each step, the effect of the disturbance is attenuated.

we use the maximum value for all three. With $\oslash$ representing row-by-row division, we can write:

$$\lambda = \text{maxrow}\left(\frac{1}{b-a}\left(\beta r_i \oslash \Omega R \Sigma_i^{-1} r_i\right)\right) \tag{5.3}$$

This value of λ is then propagated to all of the poses after pose a, since the resulting state update will adjust all of those nodes. This is due to our use of the x_{incr} state space: we are estimating the robot's motion at each time, and the position of the robot is the sum of its previous motions. Note that monotonicity of learning rates Λ_i is important since an adjustment to an incremental state variable i has an effect on all the subsequent positions $j \geq i$:

$$\Lambda_i' = \max(\Lambda_i, \lambda) \quad \text{for } i > a. \tag{5.4}$$

5.1.2 Processing existing constraints

When processing an old constraint, we must determine what *effective learning rate* should be used when calculating its step size. If a new constraint has been added to the graph, it may upset existing constraints. In order for all of these constraints to rapidly find a new equilibrium, the effective learning rate of other constraints needs to increase in response to the newly added constraint.

The learning rates for each node are set in proportion to how much they are perturbed by new constraints. Thus, the average amount of perturbation affecting a constraint between a and b is the average of learning rates Λ_i for $i \leq a \leq b$. Constraints that overlap with a new constraint will see an increase in their effective learning rate in proportion to their overlap with the new constraint. The resulting state update will perturb all subsequent poses (again due to our use of the x_{incr} state space); as before, we update the learning rates so that all the affected poses have a learning rate at least as large as the effective learning rate.

Consider an example with three constraints (see Fig. 5-1) with a newly-added constraint X between poses 100 and 200, and existing constraints Y (between poses 50 and 150) and Z (between poses 25 and 75). The learning rate increase caused by constraint X will cause an increase in the effective learning rate for constraint Y. On the next iteration, constraint Z will also see an increase in its effective learning rate, due to the learning rate increases caused by Y. In other words, constraint Z will be affected by constraint X, even though they have no poses in common. Their

interaction is mediated by constraint Y. Each interaction attenuates the learning rate increase in a manner related to the overlap between the constraints. This "percolating" effect is important in order to accommodate new information, even though the effect is generally small.

5.1.3 Algorithm Summary

In summary, adding a new constraint (or constraints) to the graph requires the following initialization:

1. Compute the constraint's effective learning rate λ_{eff} using Eqn. 5.3 and perform a step.

2. Increase the learning rates, as necessary:

$$\Lambda_i' = \max(\Lambda_i, \lambda_{eff}) \quad \text{for } i > a \tag{5.5}$$

Updating an existing constraint between poses a and b involves three steps:

1. Compute the constraint's effective learning rate λ_{eff} by computing the mean value of the learning rates for each pose spanned by the constraint:

$$\lambda_{eff} = \frac{1}{b-a} \sum_{a+1}^{b} \Lambda_i \tag{5.6}$$

2. Compute and apply the constraint step, using learning rate λ_{eff};

3. Update the learning rates of the affected poses, as in Eqn. 5.5.

After processing all of the constraints, the learning rates Λ_i are decreased according to a generalized harmonic progression, e.g.:

$$\Lambda_i' = \frac{\Lambda_i}{1 + \Lambda_i} \tag{5.7}$$

These rules guarantee the increasing monotonicity of Λ_i with i. I.e., at any given time step, $\Lambda_i \leq \Lambda_j$ for all $i < j$.

5.2 Partial Iterations

In the case of the batch algorithm, a "full iteration" includes a step for *every* constraint in the graph. The learning rate is controlled globally, and the graph converges more-or-less uniformly throughout the graph.

In the incremental algorithm that we have described, the learning rate is not uniform and different parts of the graph can be in dramatically different states of convergence. In fact, it is often the case that older parts of the graph have much

lower learning rates (and are closer to the minimum-error configuration) than newer parts, which are far from a minimum. This is because new constraints are generally added near the robot.

Suppose a robot explores two buildings, first A then B. While the robot is in B, the map for A is more-or-less unaffected, whereas the map for building B may be undergoing dramatic changes as new loop closures are identified.

A reasonable goal for an optimization algorithm is to reduce χ^2 error per CPU time used. Building B, in this simple example, represents low-hanging fruit. It would be highly advantageous if we spend more CPU time optimizing building B than A.

Suppose we have a graph with a maximum learning rate of λ_{max}. If we performed a full iteration (without adding any new constraints), the maximum learning rate in the graph would be reduced to $\lambda'_{max} = \lambda_{max}/(\lambda_{max}+1)$. Now suppose that there is a constraint whose effective learning rate is less than λ'_{max}. Processing that constraint will not significantly improve the state estimate, since other constraints are taking larger steps (which will have a dominant effect on χ^2 error). That constraint will also not reduce the largest learning rate in the graph. Consequently, this constraint will have virtually no effect on the state estimate and can be skipped. In short, processing constraints whose effective learning rate is less than λ'_{max} is unproductive.

The following algorithm exploits this property to optimize only a recent subgraph of the pose graph. It does not actually compute the effective learning rate for each constraint; instead, it identifies constraints whose effective learning rate *must* be lower than λ'_{max} by virtue of not including any nodes whose learning rate is higher. As a result, some constraints that could be skipped will be processed anyway, but the resulting algorithm is simpler.

1. Look up the maximum learning rate in the graph, e.g., $\lambda_{max} = \Lambda_{nposes-1}$. If we performed a full iteration, the maximum learning rate after the iteration would be $\lambda'_{max} = \lambda_{max}/(1+\lambda_{max})$.

2. Find the oldest pose p such that $\lambda_p \geq \lambda'_{max}$

3. Perform steps only on those constraints that involve a pose $\geq p$.

4. Set $\lambda_i = \lambda'_{max}$ for all $i \geq p$

In other words, this algorithm reduces the maximum learning rate in the graph by a full harmonic progression by computing and operating on only a subset of the graph's constraints. The procedure above conservatively identifies the set of constraints that should be considered.

In many cases, large reductions in learning rate can be achieved by considering only a handful of constraints. In the Freiburg data set, the technique is very effective, with an average of only 6.73% of the constraints updated at each iteration (see Fig. 5-2). Since computational cost is linear in the number of constraints processed, this yields a speed-up of almost 15x. The χ^2 error is naturally somewhat higher (see Fig. 5-6) in absolute terms, since older parts of the graph are not being fine-tuned when there are high learning-rate subgraphs. However, when the total amount of

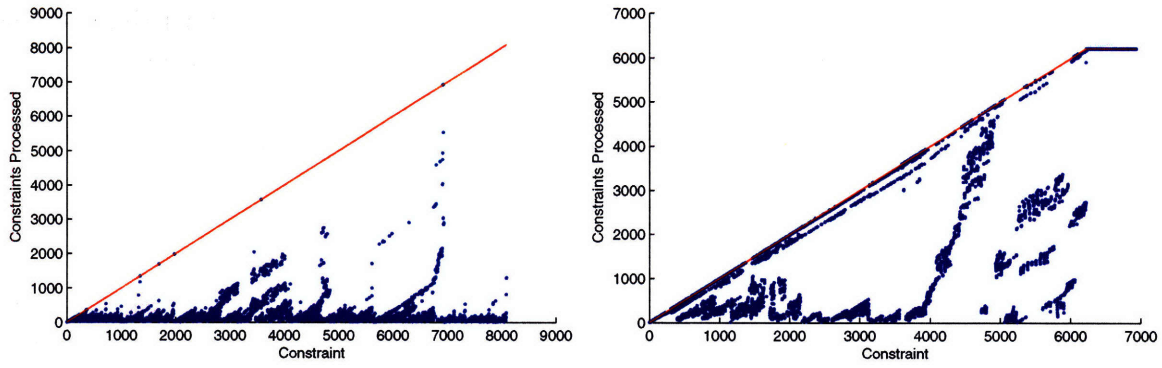

Figure 5-2: Partial Updates. Left: Freiburg, Right: Intel Research Center. Selective processing of constraints leads to large speed-ups. In the figure, the line indicates the total number of constraints in the graph; the points show the actual number of constraints that were processed. In the Intel dataset, more constraints tend to be processed since the robot repeatedly revisits the same areas, causing new constraints to disturb larger portions of the graph.

CPU time is taken into account, the selective method is substantially more efficient at reducing the χ^2 error.

The same approach on the Intel Research Center data set processes 59% of the constraints on average. The lower performance is due to the fact that the robot is orbiting the entire facility: many constraints perturb virtually the whole graph. Still, on this near worst-case data set, a significant speedup of 69% is achieved.

The approach outlined here is somewhat greedy: it attempts to reduce the worst-case learning rate to Λ'_{max}. This goal determines both the oldest pose p and thus the number of constraints that are necessary to perform the partial update. On very-well connected graphs with many large loop closures, the learning rate tends to be relatively flat. This leads to very small values of p and thus large numbers of constraints to be processed. It is possible that picking a larger Λ'_{max} (i.e., settling for less than a full harmonic progression) would result in a more economical update—this is an area of future work.

Similarly, if a robot knew that it was leaving an area and that it would not return for some time, it would be reasonable to expend additional iterations performing complete updates, lowering the learning rate throughout the graph and thus improving the quality of the map.

5.3 Learning Rate Data Structure

An obvious data structure for maintaining the values Λ_i is an array. The operations required by Eqn. 5.5 and 5.6 would run in $O(N)$ time for a graph with N poses. Since every constraint step requires updating the learning rates, this would increase the time complexity per constraint from $O(\log N)$ to $O(N)$.

An augmented binary tree can be used to implement both of these operations in $O(\log N)$ time. The learning rates for each pose correspond to the leaves of the tree. Each non-leaf node in the tree nominally stores the minimum, maximum, and sum of

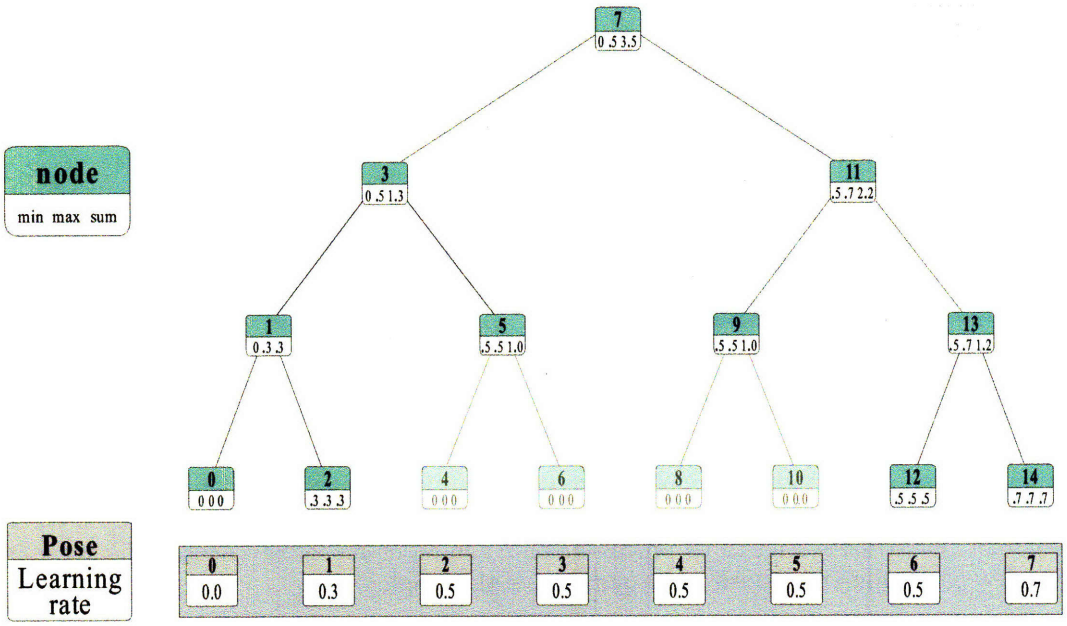

Figure 5-3: Learning-Rate Data Structure. Our learning-rate data structure, an augmented binary tree, allows all critical learning-rate functions to be implemented in $O(\log N)$ time. Each node in the tree tracks the minimum, maximum, and sum of all the elements beneath it; however, any parent can override all of its children by having min=max. The node indexing scheme allows the tree to be stored in-place in an array.

its children. However, a parent overrides the value of its children when its minimum field equals its maximum field. The values stored in the descendant nodes of such a parent are invalid.

The tree supports a variety of operations, each of which runs in $O(\log N)$ time. Since any node can override its children, operations begin at the root and descend towards the leaves; if they encounter a node at which $min == max$ (which signals that the parent is overriding the data stored in its descendants), the tree descent ends.

Most operations terminate at the first overriding node. (Note that leaves always have $min == max$ and thus can be considered "overriding" nodes, even though they have no children.) The procedure *setLowerLimitRecurse* is somewhat more complicated, since it must traverse all the way to a leaf, possibly passing an overriding node along the way. During its recursion, any overridden value is passed along and used to "patch up" the tree as it goes.

The *fixup* procedure recomputes a nodes min, max, and sum values from its children. The procedure *mean* can be trivially computed from *cumulativeSum*.

The node indexing scheme (see Fig. 5-3) permits computation of parent, child, sibling, and number of children using constant time arithmetic expressions. For example, the arithmetic mean of two child node ids gives the parent node id.

Aging the learning rates requires $O(N)$ time: the learning rates are extracted from the tree into an array (via a recursive traversal of the tree), aged, and then reinserted back into the tree. However, aging the learning rates only occurs after all the constraints have been updated. The amortized time is $O(N/M)$ for N nodes

Algorithm 8 getValue(node) : *Return the value* v_{node}

```
i = ROOT
loop
  if min_i == max_i then
    return min_i
  end if
  i =descendTowardsNode(i, node)
end loop
```

Algorithm 9 cumulativeSum(idx) : *Compute* $\sum_0^{idx} v_i$

```
sum = 0
i = ROOT
loop
  if min_i == max_i then
    return sum + min_i * (idx−leftMostDescendant(i) + 1)
  end if
  i =descendTowardsChild(i, idx)
  if isRightChild(i) then
    sum = sum + sum_leftSibling(i)
  end if
end loop
```

and M constraints; since M must increase *at least* linearly with N, this runtime is typically sub-linear.

5.4 Analysis

Our approach for updating the learning rates in order to perform incremental stochastic optimization has a number of appealing properties.

- When the robot travels to a new location, a new pose and a new constraint connecting that pose to the graph are created. Since the residual of this new constraint is zero (because there is only a single constraint to that node), no learning rate increase results.
- When a new loop closure is identified, but the residual is small (perhaps because the area is already well-mapped), little or no learning rate increase results.
- When a new loop closure is identified, and the residual is large (closing a very large loop for the first time, for example), a larger learning rate increase results.

Algorithm 10 setUpperLimit(ulimit) : *For all i, set* $v_i' = \min(v_i, ulimit)$

```
setUpperLimitRecurse(ROOT, ulimit)
```

Algorithm 11 setUpperLimitRecurse(i, ulimit)

if $max_i \leq ulimit$ **then**
  return
end if
if $min_i == max_i$ **then**
  $min_i = ulimit$
  $max_i = ulimit$
  $sum_i = ulimit * (\text{numberOfChildren}(i) + 1)$
end if
setUpperLimitRecurse(leftChild(i), $ulimit$)
setUpperLimitRecurse(rightChild(i), $ulimit$)
fixup(i)

Algorithm 12 setLowerLimit(idx, llimit) : *For all* $i \geq idx$, *set* $v_i' = max(v_i, llimit)$

setLowerLimitRecurse(idx, ROOT, llimit, false, 0)

This is to be expected, since the positions of many nodes must be updated. However, any portion of the graph prior to the loop closure will be insulated from this increase.

On a graph with N poses and M constraints, memory usage of the algorithm is $O(N + M)$, the same as the batch algorithm. Runtime per constraint also remains $O(\log N)$.

The actual CPU requirements, as with the batch algorithm, are difficult to specify. The nature of the stochastic optimization provides no guarantees that the error will go down at each iteration. It is also the case that the amount of map optimization required for any particular application varies. Consequently, we cannot prescribe a specific number of iterations to compute after adding a constraint.

That said, the convergence of the algorithm is typically quite rapid, especially in correcting gross errors. Further, the fact that individual iterations are inexpensive affords users a great deal of granularity in controlling their CPU requirements versus their map quality requirements.

5.5 Results

We compared the runtime performance characteristics of our approach to that of LU decomposition (LU), the Extended Kalman Filter (EKF), and Gauss-Seidel Relaxation (GS). Our assumption is that the robot requires a full state estimate after every observation; this makes the performance of the EKF comparable to that of an information-form filter. Fig. 5-6 shows the cumulative computational time versus step (with one constraint added per step). For the Gauss-Seidel implementation, every state is iteratively updated once after each constraint. Our method is plotted twice: first, using all the constraints in the graph ("Whole"), and second, using only a subset of the constraints ("Partial").

Algorithm 13 setLowerLimitRecurse(idx, i, llimit, overriding, overrideValue)

if overriding **then**
  $min_i = overrideValue$
  $max_i = overrideValue$
  $sum_i = overrideValue * (\text{numberOfChildren}(i) + 1)$
end if
if descendants of i are all $< idx$ **then**
  return
end if
if $min_i == max_i$ and descendants of i are all $\geq idx$ **then**
  $min_i = llimit$
  $max_i = llimit$
  $sum_i = llimit * (\text{numberOfChildren}(i) + 1)$
  return
end if
if $min_i == max_i$ and not $overriding$ **then**
  $overriding = true$
  $overrideValue = min_i$
end if
setLowerLimitRecurse(idx,leftChild(i), $llimit, overriding, overrideValue$)
setLowerLimitRecurse(idx,rightChild(i), $llimit, overriding, overrideValue$)
fixup(i)

Our approach, especially with partial updates enabled, is significantly faster than any of the other methods. In terms of quality (as measured by χ^2 error), our approach produced somewhat worse maps. However, the difference is very subtle. When using the constraint selection algorithm (plots labeled "partial"), our algorithm is significantly faster than the other approaches. The CPU time and χ^2 results on the Freiburg data set are shown in Fig. 5-6. Similar behavior occurred on the Intel Research Center dataset (see Fig. 5-7).

A brute force LU decomposition at each iteration is the slowest method by far. The EKF's $O(N^2)$ time complexity amounted to 588 s, making it an unappealing approach. Gauss-Seidel took 110 s, and our approach took 15.1 s. The EKF, Gauss-Seidel (GS), and our method (not using the partial updates optimization) had similar χ^2 errors at each step, tracking the actual minimum quite well. When selectively processing constraints, our method's χ^2 is naturally somewhat higher; subjectively, the quality of the maps was acceptable. We note that Gauss-Seidel relaxation does not perform this well in all cases, as was demonstrated in [48] and in Chapter 4.

The algorithm, even if all constraints must be processed, is very fast. Processing all 8000 constraints on the Freiburg graph with 906 poses required a total of 16.8 ms. When using the partial updates algorithm, only a small fraction of these constraints need to be processed: the algorithm took an average of 1.1 ms to add each of the final 10 constraints on the Freiburg graph: for each, it only needed to consider an average of 73 constraints. Several Freiburg maps are illustrated in Fig. 5-5, including

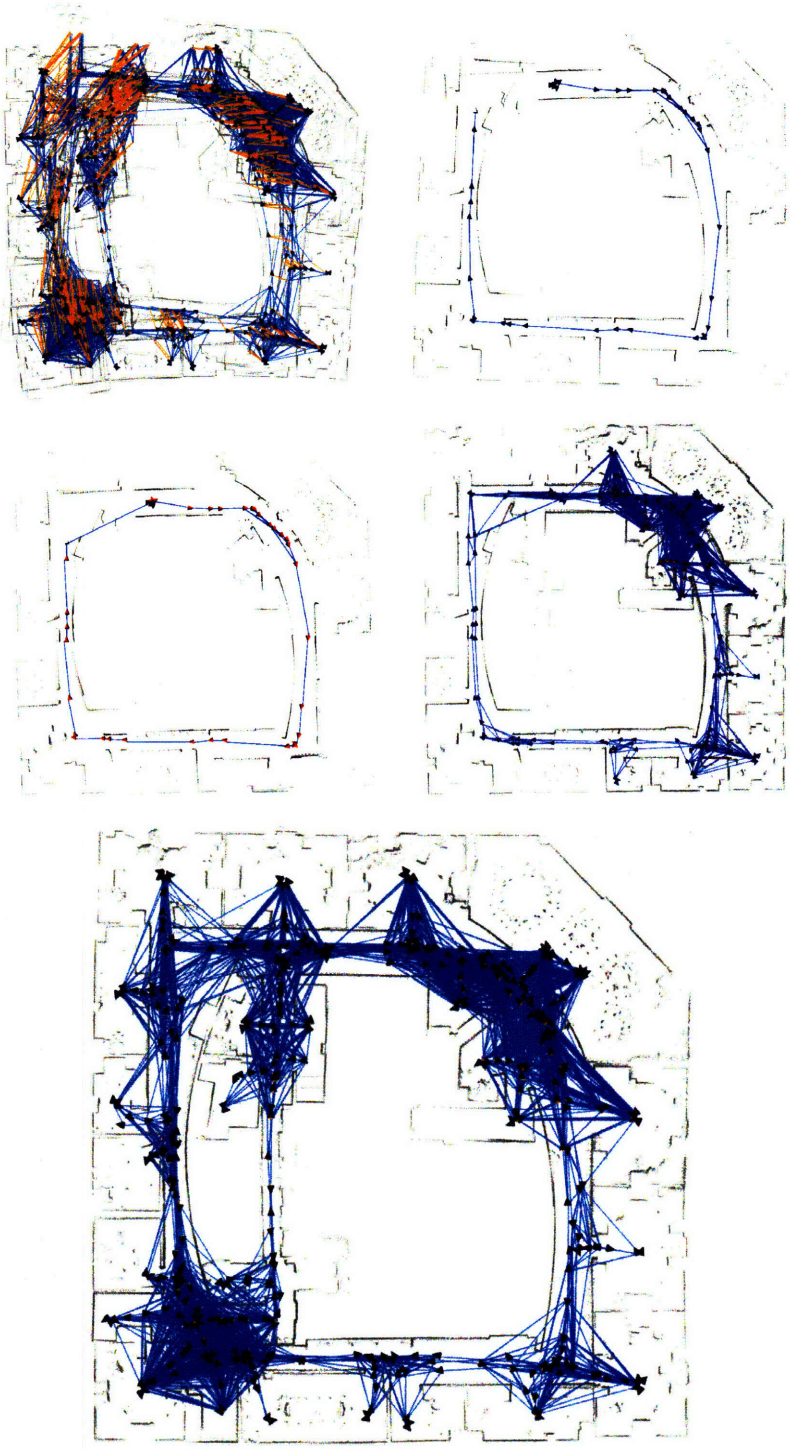

Figure 5-4: Intel Research Center. Top-left: open loop trajectory; top-right: just before closing the first loop, middle-left: just after closing the first loop; middle-right: about half way through the data set; bottom: the posterior map.

the learning rates (as a function of pose).

Putting these numbers in perspective, the Intel Research Center dataset (Fig. 5-4) represents 45 minutes of data; incorporating observations one at a time (and out-

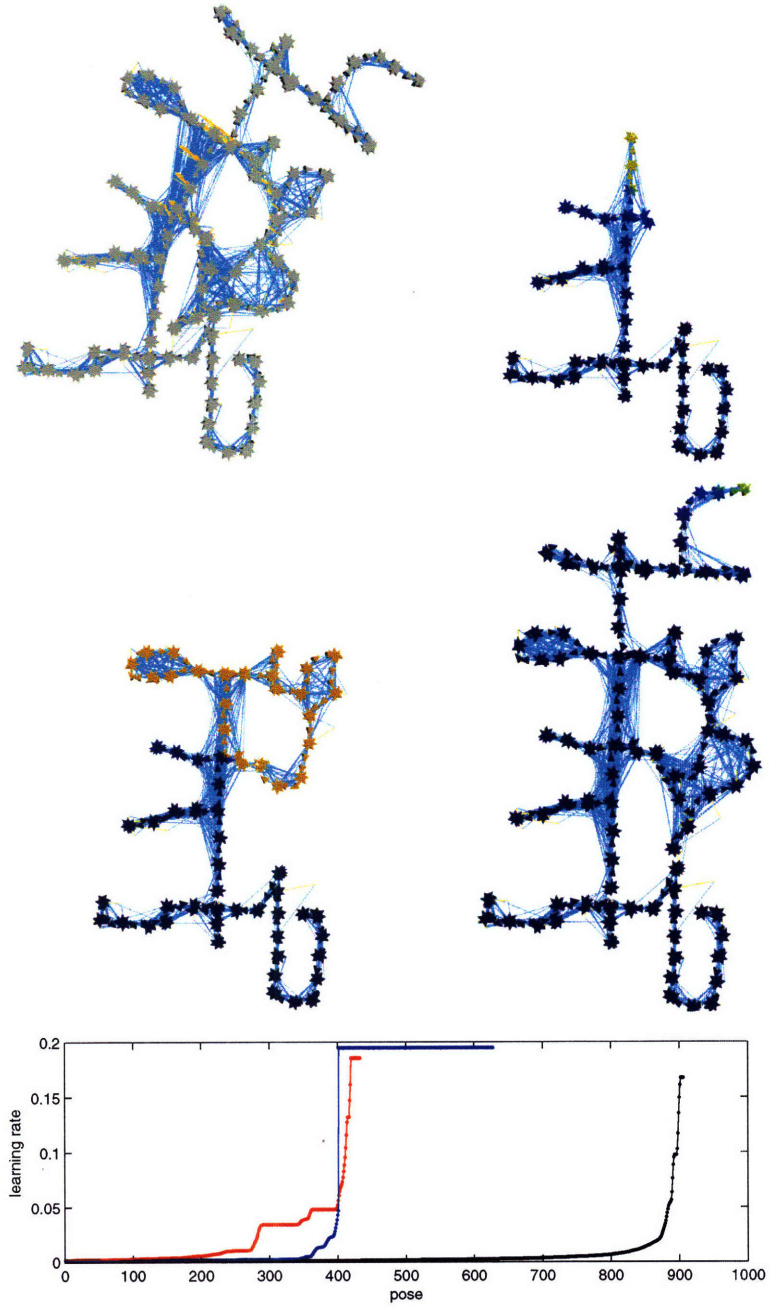

Figure 5-5: Incremental Processing of Freiburg dataset. The open-loop graph (top-left) is incrementally optimized; the state of the graph is shown at two intermediate configurations and the final configuration. The colors used in the maps indicate the learning rates Λ_i, which are also plotted on the bottom as a function of pose number; earlier parts of the graph are clearly insulated from learning rate increases affecting poses closer to the robot. When closing large loops (middle-left figure), the learning rate is increased over a larger portion of the graph.

putting a posterior map after every observation) required a total cumulative time of 27.9 s with partial updates enabled, and 50.9 s without.

Several maps from the Intel Research Center are shown in Fig. 5-4. The open-loop trajectory is shown, as well as several intermediate maps. In the first map, a

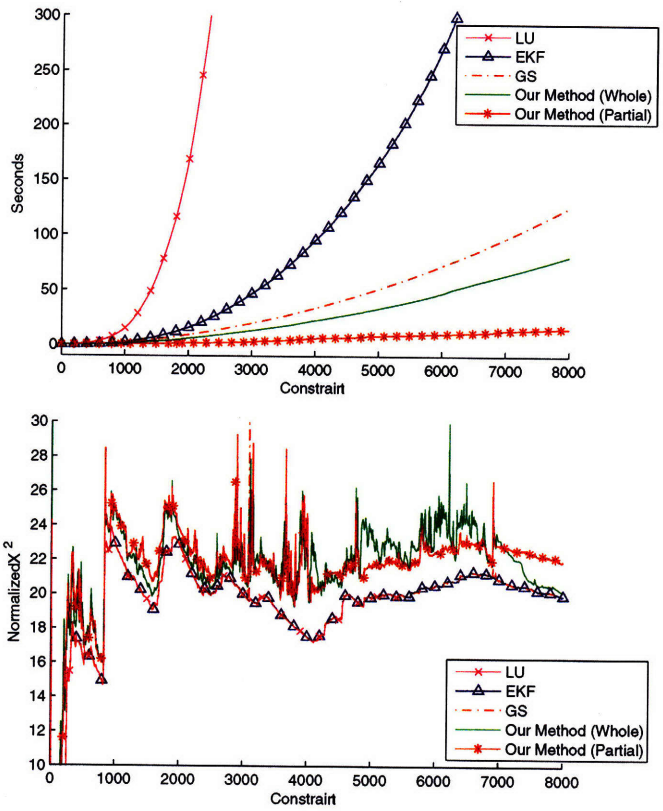

Figure 5-6: Cumulative Runtime and Error Comparisons, Freiburg dataset. Each constraint was added one at a time. EKF and LU computational time dwarfs the others. Our proposed method (with partial updates) is by far the fastest at 21 s; our method (without partial updates) beats out Gauss-Seidel relaxation. In terms of quality, LU, EKF, and Gauss-Seidel all produce nearly optimal results; our methods have marginally higher χ^2 error, as expected, but the maps are subjectively difficult to distinguish. (Partial: 15.5 s; Whole: 82.5 s; Gauss Seidel: 128.3 s, EKF: 650 s.)

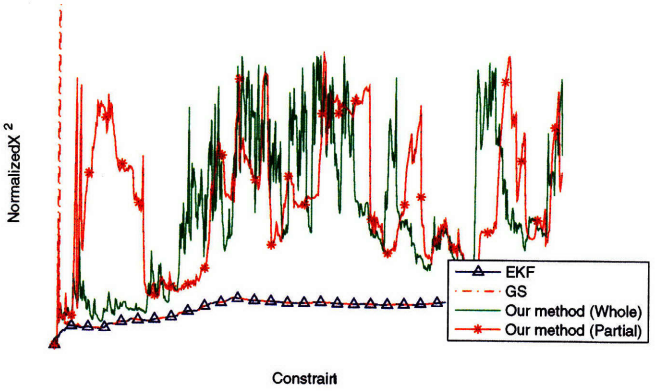

Figure 5-7: Error Comparisons, Intel Research Center. Our methods are able to stay relatively close to the nearly-optimal χ^2 error produced by the EKF and Gauss-Seidel, however, did so at a fraction of the run time. (Partial: 14.5 s, Whole: 45.4 s, Gauss-Seidel: 65.2 s, EKF: 132.3 s)

loop closure is just about to occur; prior to this, the learning rate is low everywhere. After the loop closure, the learning rate is high everywhere. The final map exhibits

sharp walls and virtually no feature doubling; the incremental algorithm matches the quality of the batch algorithm. We also validated our algorithm on countless synthetic datasets with consistently good results.

As with the batch algorithm, the optimization rapidly finds a solution *near* the global minimum, but once near the minimum, other algorithms can "fine tune" more efficiently. On a large synthetic data set, our method requires 24.18 s for incremental processing; after a post-processing using 2.0 s of Gauss-Seidel relaxation, the normalized χ^2 error drops to 1.18. This is far better than Gauss-Seidel can do on its own: it requires 168 s to reduce the χ^2 error by the same amount.

5.6 Summary

We have presented an incremental map optimization algorithm based upon stochastic gradient descent. Our algorithm provides very fast updates, rapidly incorporating new data into the map estimate. We introduced the idea of spatially-dependent learning rates as a means of isolating different parts of the graph. This allows distant and less volatile parts of the graph to be largely unaffected by the addition of new constraints. We also described a data structure that allows the spatially-dependent learning rates to be efficiently updated over time.

Finally, we showed how to optimize only the subgraph that has the largest learning rate. This allows the χ^2 error of the graph to be rapidly decreased by focusing computational effort on the part of the graph that needs it most, making the algorithm even more computationally efficient.

CHAPTER 6

Conclusion

Building maps is a fundamental capability for robots operating in unknown or changing environments. In some instances, the map itself is a useful output of the robot, while in others, the map allows the robot to avoid obstacles and plan paths.

Map building is difficult. Environments are often cluttered and ambiguous, and are perceived only through imperfect sensor systems. Existing methods can construct maps in many environments, but when environments are large, or when the sensor data is ambiguous, those methods can become slow or produce incorrect answers.

Map building can be decomposed into two basic pieces, both of which are critical to success. The first is loop closing: recognizing when a robot has revisited a place it has been before. Each loop closure represents a constraint on the trajectory of the robot; the more over-constrained the trajectory is, the more accurately the trajectory—and thus the map—can be estimated. Loop closing is difficult, however, due to ambiguities in sensor data. Many environments can appear similar, making it hard to determine whether two sets of sensor readings correspond to the same or different places.

The second challenge is to compute the maximum-likelihood map once the loop closures have been detected. This optimization problem is difficult due to the size of the state vector, the non-linear nature of the constraints, and the poor quality of the initial state estimates.

6.1 Contributions

This thesis has described algorithmic improvements for both loop closing and map optimization. We show how relatively easy-to-generate loop-closure hypotheses can be filtered so that only correct loop-closure hypotheses remain. Our approach is based on the spectral properties of the pair-wise consistency matrix. A key feature of our approach is that the ratio of the first and second eigenvalues can be interpreted as a confidence metric, allowing perceptually ambiguous sensor data to be detected. In this thesis, we simply reject ambiguous hypothesis sets; a natural direction for future work would be to resolve ambiguous hypothesis sets by collecting additional data.

The second major contribution of this thesis is a new family of map optimization algorithms. These algorithms are based on stochastic gradient descent, which

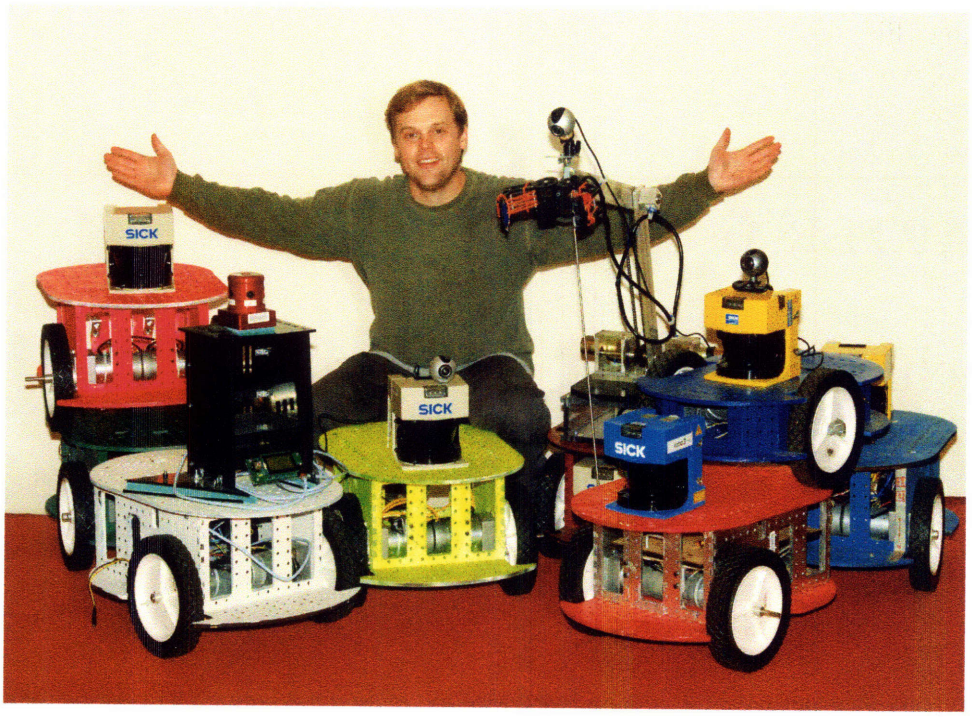

Figure 6-1: The author with several indoor robots.

improves the state estimate by considering a single constraint at a time. Stochastic Gradient Descent has previously used in machine learning (particularly in the training of neural networks), but we showed how the method could be adapted to the map optimization problem. Our resulting algorithm rapidly explores the state space, giving it two key advantages: it escapes local minima that trap existing methods, and it can rapidly find solutions near the global minimum. This thesis describes both a batch and online version of the algorithm and evaluates the algorithms on a variety of real and synthetic data sets.

6.2 Future Work

The methods described in this thesis allow robots to operate in larger and more perceptually ambiguous environments than before. Still, the map building problem has not been definitively solved. Both loop-closing and map optimization continue to offer difficult challenges.

Loop closing is primarily difficult due to perceptual ambiguity. Robots are unable to recognizes places as well as a human, even though a robot's sensors are superior in some important respects. For example, planar lidars can easily measure distances with centimeter precision—far better than an unaided human. Humans make up for this, at least in part, by having a better understanding of the environment. A human might recognize an office copier room by noting the presence of a copy machine, whereas to a lidar, a copier is essentially indistinguishable from any other piece of furniture. In the short term, there is room for improvement in both low-level sensor

processing (like feature detectors) as well as higher-level processing (like the loop-closing method described in Chapter 3).

Current optimization methods, augmented by the techniques described in this thesis, can often successfully solve map optimization problems. The problem can be made arbitrarily harder, of course, by increasing the size of the problem: more nodes and more sensor observations will tend to bog down any method. However, with several fast methods now available, *robustness* becomes a distinguishing factor. If an algorithm arrives at the wrong answer (by getting stuck in a local minimum, for example), the speed of that algorithm is largely irrelevant.

Problems in which there is a great deal of noise make the optimization harder, but the *type* of observation also has an effect on difficulty. This thesis, for example, focused on edges that fully constrain the position of two nodes. Range-only and bearing-only observations do not fully constrain the position of two nodes: they constrain the position of one node to lie on either a circle or ray (respectively).

Range-only observations have proven to be fairly easy to incorporate [47, 49]. Bearing-only observations, in contrast, are more difficult. Bearing-only measurements, such as those naturally obtained from monocular cameras, exhibit a singularity: if the predicted position of a landmark is near the robot, small changes in the robot's position cause large changes in the predicted bearing to the landmark. In the worst case, if the robot happens to be on top of the landmark's predicted position, the predicted bearing to the landmark is undefined. While these effects can be mitigated to some degree (by skipping observations that are near a singularity, for example), the likelihood of the optimization failing increase. Failure modes include both local minima and divergence.

It is important that map building performance continue to improve. Better mapping algorithms will allow robots to operate in environments that are currently too complex. For example, autonomous cars have the potential to reduce the number of traffic fatalities— with humans at the wheel, more than 40,000 die every year on roadways in the United States [1]. Other potential applications, from charting abandoned (and hazardous) mines to helping the elderly move about their homes, all illustrate the practical usefulness of robotic map building.

APPENDIX A

Sensor Processing

A.1 Overview

This appendix describes some of the implementation details of the sensor processing methods used in this thesis. The central contributions of this thesis are largely sensor agnostic, but an understanding of the underlying systems can build an appreciation for how complexities that arise from real data.

The bulk of this thesis uses two sensor modalities: vision and laser scanners (lidars). This appendix discusses each in turn.

A.2 Vision Processing

A.2.1 Data Collection Platform

Our vision data set was collected by MIT's DARPA Grand Challenge vehicle (see Fig. A-1) during an autonomous run of the National Qualifying Event. Five calibrated cameras recorded a nearly 360-degree field of view. For these experiments, each camera's data stream was reduced to 376×240 at 10 fps. These cameras were essentially monocular since their fields of view did not significantly overlap. In other words, the range to objects in the environment could not be determined from a single set of camera images.

The motion between poses is estimated by the inertial measurement unit (IMU) attached to the robot. This IMU, coupled with a three-antenna GPS system, provides a good ground truth estimate. For many of our experiments, we purposefully degraded the IMU data in order to mimic the lower-quality odometry data available on a typical vehicle.

Figure A-1: Talos. Our vision test-bed was MIT's DARPA Grand Challenge vehicle. The vehicle has five cameras (two forward, left, right, rear), and a high-quality IMU/GPS system for both dead-reckoning and ground-truthing.

A.2.2 Feature Detection

We applied Lowe's Scale-Invariant Feature Transform (SIFT) [38, 39] detector to our images, resulting in a large number of SIFT detections (see Fig. A-2)[1]. SIFT features occur at so-called *interest points*, which corresponding to extrema of the difference-of-Gaussian function applied over a multi-resolution image pyramid. Each feature is described by a 128-dimensional vector that allows features to be reliably associated over time.

Using known camera calibrations, the ray on which the SIFT feature lies can be computed.

A.2.3 Feature Tracking and Landmark Initialization

Unlike some previous SIFT-based feature tracking and mapping work [59, 60], we did not use stereo/trinocular vision to determine the range to detected SIFT features: the minimal overlap of our cameras' fields-of-view precluded this. This means that individual observations constrain the position of a feature to a ray— the distance to that feature cannot be determined from a single observation. Davison accounted for unknown range by tracking features with a particle filter whose particles were distributed all along the ray [9]. Our approach is to triangulate the position of a feature by using several observations of the same collected while the robot moves[2].

Whenever a camera frame arrives, a new set of SIFT features are extracted. These features are matched against a set of "tracks": each track maintains a "reference" SIFT descriptor and a list of observations matching that descriptor. Each observation includes both a vehicle position and the bearing to the SIFT feature.

[1]My thanks to Albert Huang for generating the SIFT detections and for guidance on how to process them.

[2]Initializing features based on bearing-only observations shares similarities with using range-only observations [49].

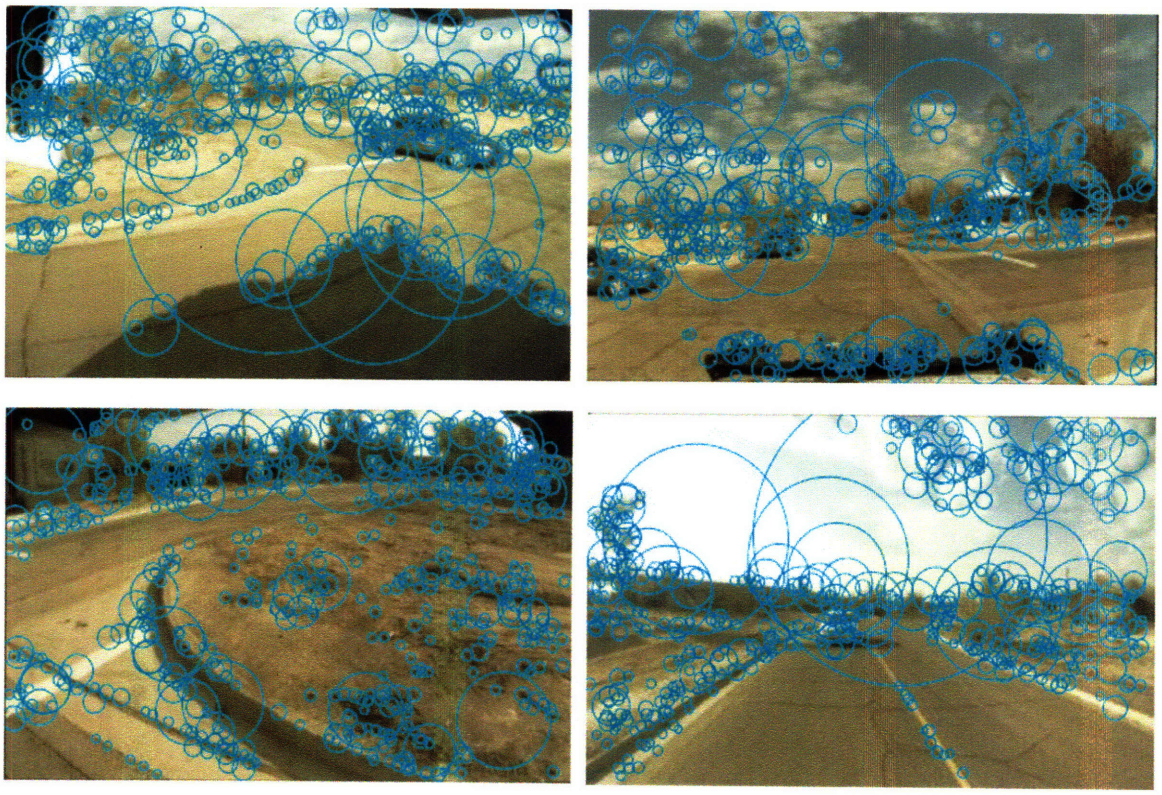

Figure A-2: Extracted SIFT features. Clockwise: left camera, forward camera, right camera, rear camera. Cyan circles represent extracted SIFT features: the size of the circle encodes the scale of the feature.

Observations are matched to tracks using the SIFT descriptor alone. We use commonly-accepted matching parameters: in order for a match to be valid, the mean squared error corresponding to the best match must be less than 60% of the mean squared error of the next best match. Additionally, if two newly detected features match to the same track, both matches are discarded. (This is similar to "marriage", in which two descriptors a and b match only if a is the best possible match for b, *and* b is the best possible match for a. Our faster rule is "monogamous", in that it requires that a is the only feature whose best match is b.)

The set of observations for each track are periodically tested to see if they can be used to localized the landmark. The position of the landmark is estimated by taking the pair-wise intersection of every observation ray, with a maximum distance set to 100 m. For N observations, this yields up to $N(N-1)/2$ points. The mean and covariance of these points is used as the landmark position estimate and its uncertainty. If the landmark's covariance is under a threshold (2 m in both $\hat{x}$ and $\hat{y}$), and the landmark has been observed over at least 5 degrees of motion, a landmark is initialized (see Fig. A-3). When a landmark is initialized, the track is reset so that the landmark can be independently observed and estimated again. The result is that the same landmarks are repeatedly reobserved with respect to the car's current position.

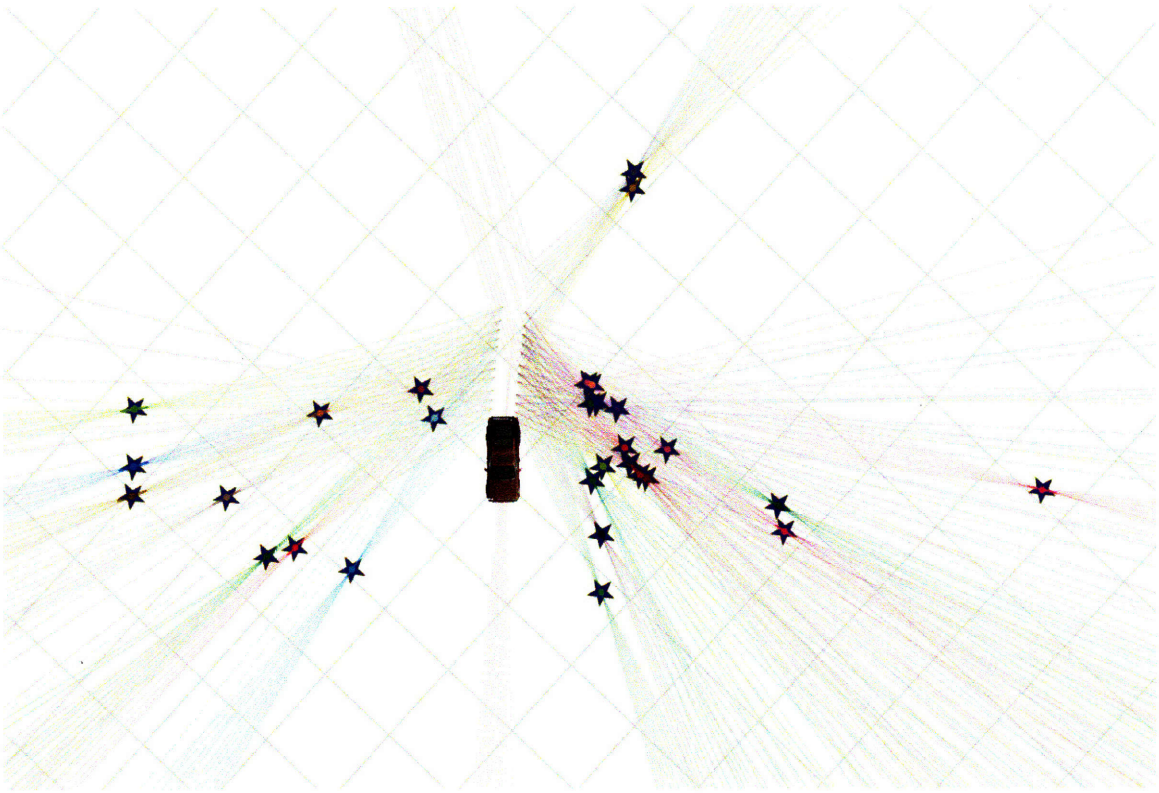

Figure A-3: SIFT Landmark Initialization. Landmarks are computed by tracking SIFT features over short motions of the robot: when the intersection of the bearings converges near a single point, a landmark is initialized.

A.2.4 Landmark matching

When initializing landmarks, a fairly restrictive set of parameters are used to ensure that features are not erroneously associated. When closing loops, however, the robot may be viewing the features from a different angle, distance, and under different lighting conditions. While SIFT features are somewhat robust to these variations, a conservative set of matching requirements (like those used in tracking) can result in very few matches. Instead, we independently compute the best match for each feature, without any additional restrictions such as "marriage" or "monogamy".

Suppose that a robot has initialized a set of landmarks A from one pose, and a different set of landmarks B from a different pose. Using this greedy matching process, we find the best matches between these two sets by comparing the SIFT descriptors of each landmark. Suppose the indices of the landmarks A and B are assigned such that (A_i, B_i) represent a matched pair.

Given two correspondences, e.g. $(A_i, B_i), (A_j, B_j)$, we can compute a rigid-body transformation that would approximately align the landmarks. For example, one correspondence could be used to establish the translational component of the rigid-body transformation, while the second correspondence could be used to compute the rotational component. (In practice, a lower-error rigid-body transformation can be found using Horn's algorithm [28]).

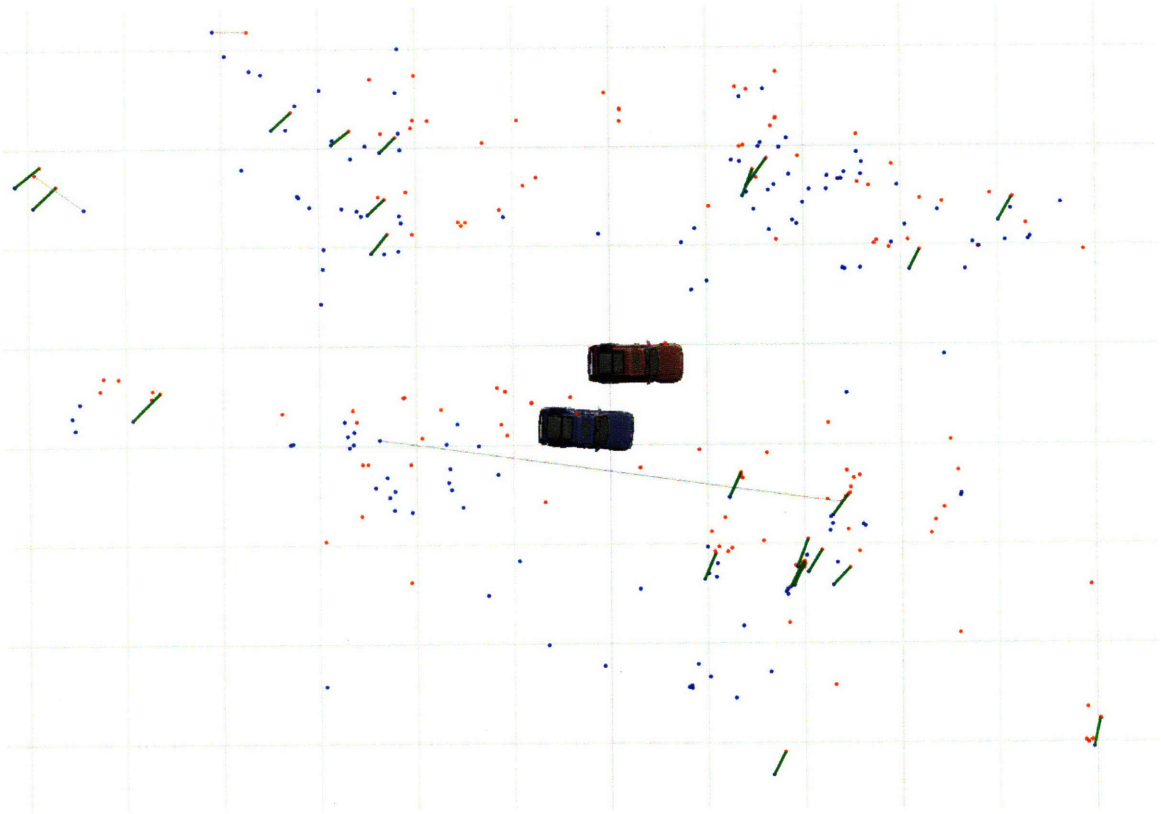

Figure A-4: SIFT Landmark Matching. The SIFT features observed from two different visits to the same place are shown: blue dots correspond to SIFT features observed from the blue pose, red dots correspond to SIFT features observed from the red pose. Matched SIFT features are connected by lines: heavier lines indicate matches that were accepted by the RANSAC validation.

The best rigid-body transformation is then found via RANSAC (see Fig. A-4). Each putative rigid-body transformation is scored by counting the number of features B_i that, when projected, lie near their counterpart A_i. We used a distance threshold of 1 m in our implementation. The rigid-body transformation with the largest score is output as a candidate loop closure hypothesis, which can be filtered using the techniques in Chapter 3.

A.3 Laser Scan Processing

Laser scanners measure the range and distance to other objects in the environment. A typical SICK laser scanner (see Fig. A-5) captures 180 points at one degree intervals with a range accurate of a few centimeters. Bearings are accurate to a fraction of a degree.

We use laser scanners (often called "lidars") in two different ways. First, we use lidars to produce an initial estimate of the robot's trajectory by *scan matching* consecutive scans. Our approach is to search for a rigid-body transform that aligns two scans by using a brute-force search (i.e., by considering a range of translations

Figure A-5: SICK Laser Scanner on a small indoor robot.

and rotations). This incremental scan matching generally produces a much better initial estimate of the robot trajectory than the robot's wheel-derived odometry.

We also use laser scanners as a means of generating loop closure hypotheses (see Chapter 3). In this case, we attempt to determine the alignment of two scans taken at very different times: the initial uncertainty is much larger than in the incremental scan matching case, making a brute-force search impractical. Our system uses a feature detector in order to generate rough candidate alignments; these candidate alignments are then refined by a much smaller brute-force search.

In this section, we describe our both aspects of our lidar data processing. First, however, we begin by describing some of the challenges inherent in lidar data.

A.3.1 Challenges

Laser data is often thought to be a panacea, providing precise and accurate data that makes map building trivial. While laser range finders produce accurate metrical descriptions of the environment, they also present a number of challenges.

The most significant limitation of laser scanners is that they measure only the *shape* of the environment (and even that, only on a single plane). In many common cases, such as in office buildings, matching laser scans can be ambiguous. This is due to the fact that the environments are composed of objects that are hard to distinguish from laser data alone. For example, in an office environment, virtually all rooms contain a similar-looking set of walls and corners.

Lidar data, often consisting of only a single cross-sectional slice of the environment, often lacks cues useful for navigation. In a long straight hallway, for example, longitudinal motion can be difficult to detect since the cross-sectional profile of the environment does not change (see Fig. A-6).

It is often difficult to exploit the subtle cues that *are* present, since algorithms must be robust to small changes in the environment. Algorithms must continue to work, for example, even if a chair has been moved or if a human is walking down a

hallway with the robot.

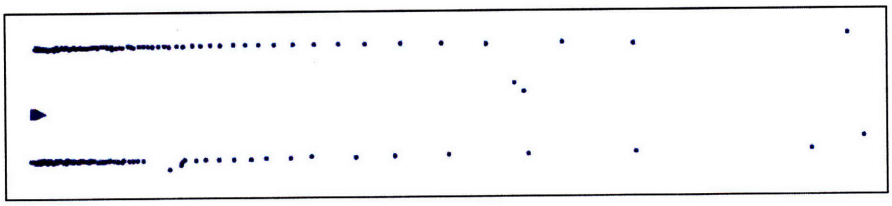

Figure A-6: Ambiguous long corridor. As a robot travels down a long corridor, the data from the lidar changes very little; this makes it difficult for the lidar to estimate the robot's motion from the lidar data. This data also includes a pedestrian (two dots near the center of the figure).

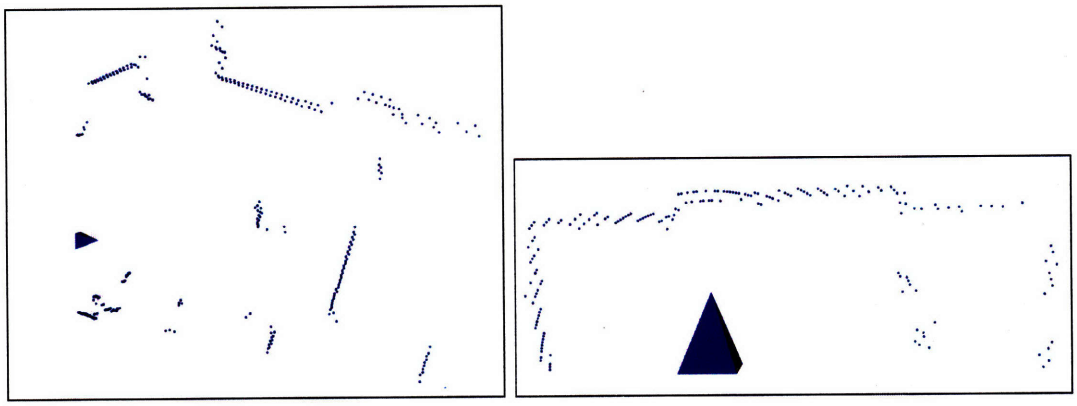

Figure A-7: Laser challenges. Left: an interlaced scan providing 0.5 degree resolution is combined from two separate scans taken 13.3 ms apart. When the robot is moving, visible artifacts result. Right: coarse range quantization creates false "scalloped" textures.

Planar laser scanners are also difficult to use in cluttered outdoor scenes. Since they scan in only a single plane, any pitching or rolling of the sensor results in different cross-sectional views of the environment. Not only can this make the appearance of objects change (since they are being sampled at different heights above the ground), it introduces range distortions when the scanning plane is no longer horizontal.

Laser scanners do not directly capture the shape of the room: instead, they sample the shape at discrete points. The position of these sample points varies as the robot moves around, meaning that the set of points from a scan taken at time $t+1$ may not all be near the points taken from a scan at time t. Consequently, it is necessary to infer the actual contours of an environment so that the "closeness" of two scans can be meaningfully computed. Since laser scanners sample points at a fixed angular intervals, the *spatial* resolution decreases rapidly with range: this makes any inferences about the geometry of the environment increasingly error-prone with range. Partially transparent surfaces complicate this inference (see Fig. A-8).

One common error made with laser scanners manufactured by SICK is to configure them to produce *interleaved* sub-degree resolution scans. A typical SICK sensor can only produce samples at one degree angular spacing; in order to produce a scan with quarter degree spacing, four scans (each with an angular spacing of one degree) are performed sequentially. Each scan is offset a quarter degree from the previous scan,

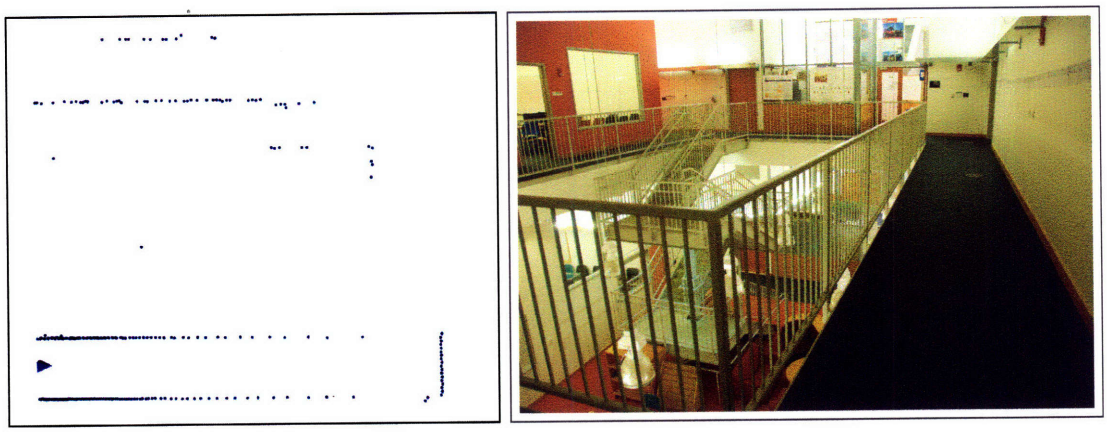

Figure A-8: Partially transparent surfaces. Inferring the correct geometry of a scene is complicated by partially transparent surfaces. Left: the laser data can partially see through the guard rail, detecting both the guard rail and wall on the opposite side. Right: a camera's view of the same environment.

and the results from all four scans are then interleaved. Motion of the robot between these scans, especially rotation, can cause artifacts in the interleaved scan (see Fig. A-7). Instead, it is better to operate SICK sensors in "dithered mode": each successive scan is still offset by a quarter degree, but each scan is individually output. This allows the motion of the robot to be incorporated for each individual scan, avoiding interlacing artifacts. This is the ideal mode for SICKs, as it increases the effective resolution of the sensor when the robot is stationary, but does not create artifacts if it is moving.

A.3.2 Scan Matching

Scan matching is the process of finding a rigid-body transformation that best aligns two scans. Incremental scan matching is the special case of examining temporally-consecutive scans: it is used to enhance or replace conventional odometry sensors.

In Fig. A-9, two scans acquired approximately 133 ms apart are shown in an poorly-aligned configuration (the alignment corresponds to the odometry-derived estimate).

Scan matching is an optimization problem with three free parameters corresponding to the parameters of a rigid-body transformation. For each rigid-body transformation, a cost function is computed that indicates the quality of the alignment. An optimization algorithm then attempts to find the minimum-cost alignment. This section describes these steps.

Contour Extraction

As previously discussed, each lidar scan samples the shape of the environment at a set of discrete points. When matching two scans, the point samples do not generally correspond to the same points in the environment. Consequently, matching points directly can produce poor results. Instead, we explicitly estimate the shape of the

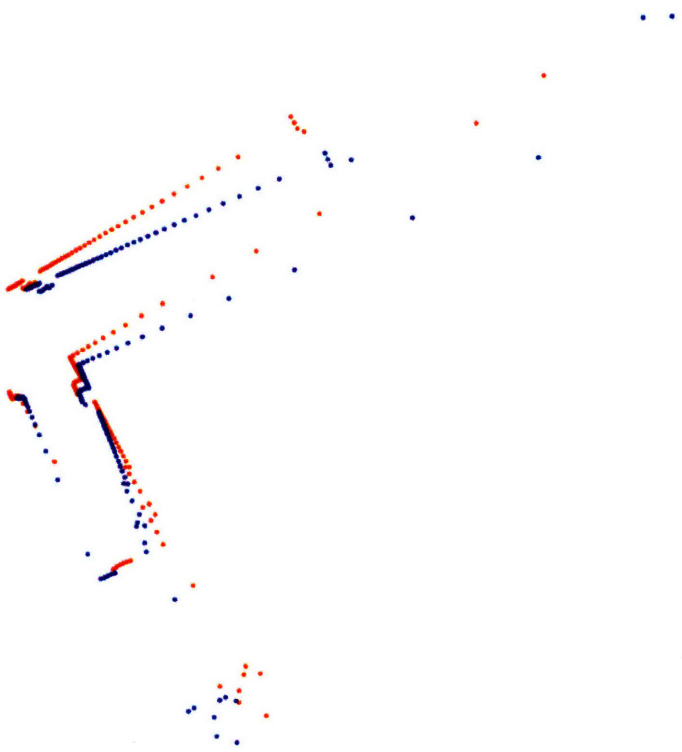

Figure A-9: Scan Matching. Two scans taken 133 ms apart are poorly aligned. Scan-matching is the process of finding a rigid-body transformation that improves the alignment of two scans. This alignment corresponds to the motion of the robot.

environment that led to a set of samples. Given a shape estimate, we can then test if a second set of point samples is consistent with that shape. We represent the shape of the environment with a set of poly-lines we call contours. This section describes our contour extraction algorithm.

Initially, each point forms a zero-length contour. Contours are agglomeratively merged by repeatedly finding the pair of contours whose endpoints are the closest. If these two endpoints are within a threshold distance, the contours are merged. This process repeats until there are no more nearby endpoints. This algorithm preferentially connects nearby points, and produces reasonable contours even in environments with partially-transparent surfaces (as in Fig. A-8).

Our implementation maintains a MinHeap [55] of candidate contour merges, sorted according to the distance between their endpoints. We repeatedly extract the next best candidate from the MinHeap: if the endpoints of the candidate's contours have not already been merged with other contours, then the candidate's contours are merged. The algorithm is very fast: on a typical scan with 180 points, it takes about 0.2 ms to extract the contours.

Computing the Probability of an Alignment

We wish to determine the cost associated with a particular alignment of one laser scan with another. The noise model of the laser scanner involves uncertainty in

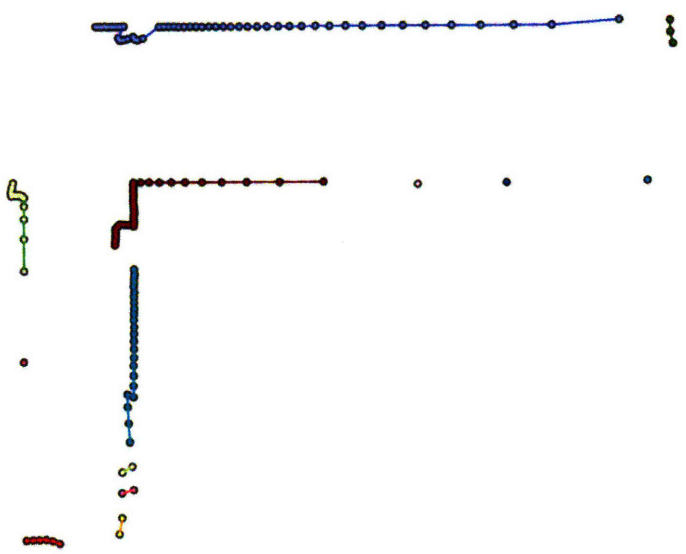

Figure A-10: Contour Extraction. A laser scanner measures only a discrete set of points; it is necessary to infer the *contours* of the surface being sampled as shown here.

both range and bearing. Range is typically more uncertain than bearing, and so the probability distributions resemble radially-oriented ellipses. In other words, the probability distribution of an observed point is dependent upon the position of that point relative to the scanner. In practice, we approximate the distribution as a circle (regardless of its position). Specifically, given a laser scan point (x, y) and the distance d to the nearest contour C, we approximate the probability of a single lidar point as:

$$P(x, y \mid C) \propto e^{-d^2/\sigma^2} \tag{A.1}$$

The variance parameter σ is determined by the sensor's noise characteristics. This approximation yields good results, and allows a lookup table for $P(x, y)$ to be easily constructed. This lookup table greatly accelerates scan matching since each scan matching operation evaluates the probability for a large number of rotations and translations.

We compute a look-up table by rendering each of the extracted contours with an intensity pattern corresponding to the log-likelihood of Eqn. A.1. The log-likelihood of a scan with N points is proportional to the sum of the N lookups.

Laser scanners are strictly line-of-sight, and are susceptible to occlusions. In general, better scan matching results can be obtained by building the cost lookup table using *several* previous scans. This increases the completeness of the lookup table (and thus the likelihood of getting a good match) at the expense of potentially introducing errors into the lookup table if the alignment of those earlier scans is incorrect.

While Carmen's "Vasco" scan matcher[45] assumes a good initial estimate and relies on gradient descent to achieve a good solution, our approach is more brute-force, computing the cost function over a wide range of translations and rotations. The cost surface can be very complicated, even in simple environments as seen in

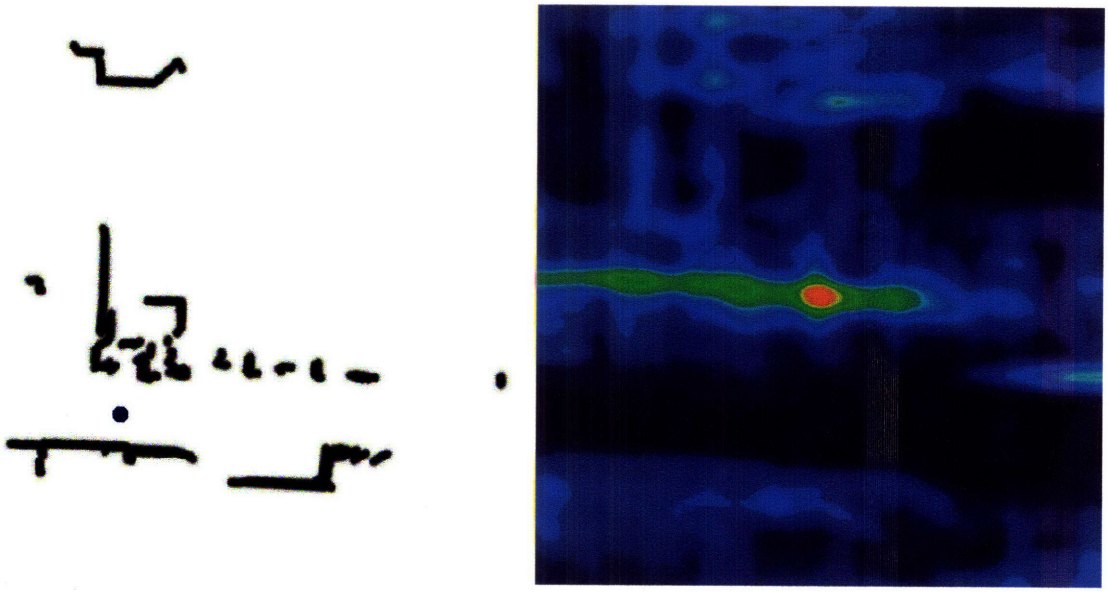

Figure A-11: Scan Matching Cost Lookup Table. The cost function is precomputed into a 2D lookup table (left) allowing the probability of a particular alignment to be computed very quickly. The cost surface for an incoming scan, parameterized by the rigid-body transformation, can very complicated (right). This cost surface is one slice of a 3-dimensional surface, corresponding to pure translations given the correct rotation.

Fig. A-11. This complexity can cause gradient-descent methods to get stuck in local minima. The primary motivation behind our brute-force approach is to improve the scan matcher's ability to find better solutions. This robustness comes at the cost of increased computational complexity, but our method can still run much faster than real-time on modern hardware if it is implemented carefully.

Scan Matching Optimization

Our brute-force approach to scan matching is robust, but is also fairly computationally expensive. This section describes an optimization that helps make the algorithm run faster than real time.

Our goal is to compute the cost function for a range of rigid-body transformations, outputting the rigid-body transformation with the minimum cost. Each rigid-body transformation corresponds to a point in a three-dimensional parameter space; our algorithm will evaluate the cost function on a 3D grid of points in that parameter space.

A straight-forward implementation would independently consider each rigid-body transformation, projecting each lidar point in order to find the appropriate cell in the lookup table and summing the resulting values. However, the computational cost of repeatedly projecting the lidar points can be substantial. Our optimization reduces the number of projections by simultaneously considering all the translations for a particular rotation.

Let us consider one slice of the 3D grid of rigid-body transformations: this 2D slice corresponds to the set of possible translations for a fixed rotation. For a single lidar

point, the cost of a translation is computed by projecting the point into the lookup map and sampling the value. Importantly, translating the lidar point is equivalent to translating the lookup table by the same amount. This means that the 2D cost map for that point can be obtained by simply copying data out a rectangular window of the lookup table. This operation is particularly fast if the 2D search grid is the same resolution as the lookup table.

The cost for all of the translations can be determined as a group by summing the rectangular windows corresponding to each point. The best translation for that particular rotation can then be determined. Our scan-matching algorithm repeats this process for each candidate rotation.

Naturally, there are an infinite number of possible translations and rotations. A prior limits the range of rotations and translations that we search, while a quality parameter determines how densely we sample the cost function within that region. The fact that the cost function is the sum of a large number of lookups (one for each lidar point) helps to reduce the effects of quantization noise stemming from the finite resolution of the lookup table. Our experiments use a lookup table with 2.5 cm resolution, but the accuracy is typically *better* than this.

A.3.3 Loop Closing Hypothesis Generation

In contrast to the incremental scan-matching case, the positional uncertainty between two scans when closing a loop can be very large. Gradient-Descent methods are more prone to failure since their initial estimate may be far from the optimal answer. With the scan matching method described above, large uncertainties require large (and slow) searches.

Instead, we extract features from the laser scans and match the features between scans. If a feature match can be found, an approximate alignment between the scans can be computed. This rough solution greatly reduces the uncertainty, making it possible to use scan matching to refine the match.

Feature Extraction

Line segments are easy to detect and are common in most indoor environments. Even in outdoor environments, lines can be fit around non-linear features like trees and shrubs: despite their poorer quality-of-fit, these lines can still be used to bootstrap the scan-matcher in order to compute good loop-closure hypotheses.

Line fitting is performed on one contour at a time: in cases of semi-transparent objects like those in Fig. A-8, this allows us to fit lines to each guard rail individually.

Our line extraction algorithm is similar to our contour extraction method. A set of lines are initialized for each pair of adjacent points within a contour (we constrain our line features to include only contiguous sets of points from the contours). Nearby line features are then repeatedly merged until the quality-of-fit of the merged lines exceeds a threshold. When merging two line features, the resulting line represents the optimal line fit of all the points associated with the input lines. Fortunately, the optimal line can be computed based on a set of sufficient statistics of constant

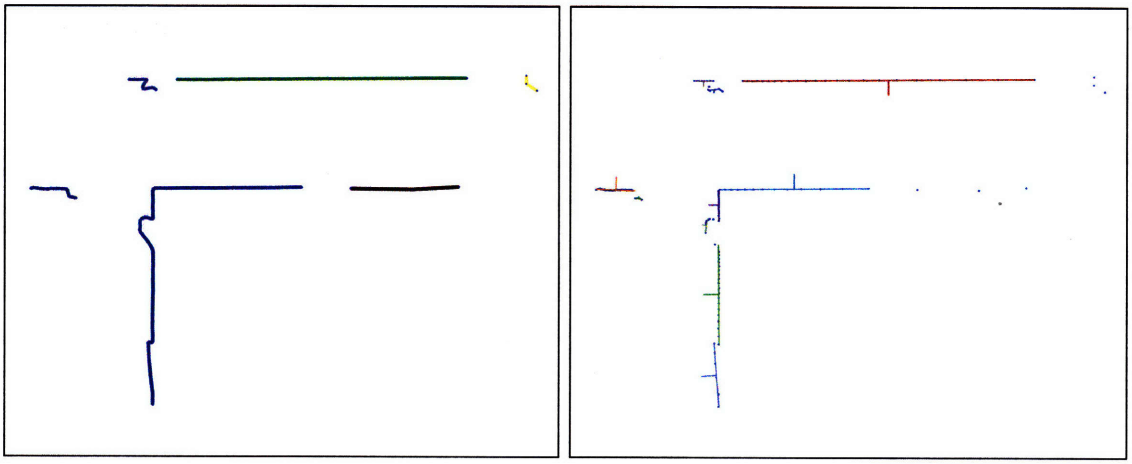

Figure A-12: Feature Extraction. Beginning with contours, lines are agglomeratively clustered until the mean squared error of the line exceeds a threshold. Left: the input contours. Right: the extracted line features (normals have been added for visualization purposes).

size [40, 42]: this makes the computational complexity of merging two lines $O(1)$ instead of linear in the number of points.

As with contour extraction, we use a MinHeap to keep track of the relative merits of all possible line merges. Given that the computational complexity of inserts and removals for a MinHeap is $O(\log N)$, and that there can be (at most) $N-1$ merges, the runtime is $O(N \log N)$. On a 2.4GHz Intel Core2 system, line extraction takes about 0.8 ms for a 180 point scan: i.e., its runtime is negligible.

Feature Matching and Refinement

Suppose that we extract line features from two different lidar scans. Given two lines from each scan (and correspondences between those lines), a rigid-body transformation can be computed that approximately aligns those lines. The intersection of the two lines serves as a common anchor point (determining the translational component), and the angles of the lines with respect to that intersection allow the rotational component to be determined.[3]

However, we do not initially know the correspondences. Instead, we select the K best lines from each scan and compute the rigid-body transformation that results from all possible correspondences amongst those lines. The number of possible pairs of correspondences grows as $O(K^4)$, making it necessary to keep K relatively small. The best lines are selected based on a heuristic incorporating the length of the line and the number of points on the line. This heuristic favors lines resulting from long and straight walls. In our experiments, we set K to 10.

The cost of each rigid-body transformation is computed using the lookup table; the lowest-cost transformation is then refined using a brute-force scan-matching operation. The line-based alignment greatly reduces the search space, making the run-time

[3]In 3D, there is an additional "flip" ambiguity, but in 2D, we can force a unique solution by forbidding reflections.

cost manageable. The result of this refinement becomes a loop closure "hypothesis", as discussed in Chapter 3.

A.4 Summary

This appendix described our low-level sensor processing algorithms for both cameras and laser scanners. The goal of these algorithms is to compute rigid-body transformations that relate the positions of the robot at different times.

Sensor processing algorithms tend to be difficult to design and implement, since they must simultaneously minimize both their error rate and their runtime costs. The methods described in this appendix are computationally efficient and generally produce good results (i.e., have a low error rate). However, they can still produce incorrect rigid-body transformations; the methods of Chapter 3 show how the error rate can be reduced further.

Bibliography

[1] National Highway Traffic Safety Administration. Fatality analysis reporting system (FARS), 2007.

[2] Tim Bailey. *Mobile Robot Localisation and Mapping in Extensive Outdoor Environments.* PhD thesis, Australian Centre for Field Robotics, University of Sydney, August 2002.

[3] Y. Bar-Shalom, X. Rong Li, and T. Kirubarajan. *Estimation with Applications to Tracking and Navigation.* John Wiley & Sons, Inc., New York, 2001.

[4] Kostas E. Bekris, Max Glick, and Lydia E. Kavraki. Evaluation of algorithms for bearing-only SLAM. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 1937–1943. IEEE, 2006.

[5] M. Bosse, P. Newman, J. Leonard, and S. Teller. Simultaneous localization and map building in large-scale cyclic environments using the Atlas framework. *International Journal of Robotics Research*, 23(12):1113–1139, December 2004.

[6] Léon Bottou. Stochastic learning. In Olivier Bousquet and Ulrike von Luxburg, editors, *Advanced Lectures on Machine Learning*, number LNAI 3176 in Lecture Notes in Artificial Intelligence, pages 146–168. Springer Verlag, Berlin, 2004.

[7] Christian Darken, Joseph Chang, and John Moody. Learning rate schedules for faster stochastic gradient search. In *Proceedings of Neural Networks for Signal Processing 2.* IEEE Press, 1992.

[8] Timothy A. Davis, John R. Gilbert, Stefan I. Larimore, and Esmond G. Ng. A column approximate minimum degree ordering algorithm. *ACM Transactions on Mathematical Software*, 30(3):353–376, 2004.

[9] A.J. Davison. Real-time simultaneous localisation and mapping with a single camera. In *Proceedings of the IEEE International Conference on Computer Vision*, volume 2, pages 1403–1410, 2003.

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

[11] Chris Ding, XiaoFeng He, Hongyuan Zha, Ming Gu, and Horst D. Simon. A Min-MaxCut spectral method for data clustering and graph partitioning. Technical Report 54111, Lawrence Berkeley National Laboratory, 2003.

[12] T. Duckett, S. Marsland, and J. Shapiro. Learning globally consistent maps by relaxation. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, volume 4, pages 3841–3846, San Francisco, CA, 2000.

[13] R. Eustice, H. Singh, and J. Leonard. Exactly sparse delayed-state filters. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 2428–2435, Barcelona, Spain, April 2005.

[14] R.M. Eustice, H. Singh, J.J. Leonard, and M.R. Walter. Visually mapping the RMS Titanic: Conservative covariance estimates for SLAM information filters. *International Journal of Robotics Research*, 25(12):1223–1242, December 2006.

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

[16] John Folkesson and H. I. Christensen. Graphical SLAM - a self-correcting map. *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 791–798, 2004.

[17] John Folkesson and Henrik I. Christensen. Closing the loop with graphical SLAM. *IEEE Transactions on Robotics*, 23(4):731–741, 2007.

[18] U. Frese. A proof for the approximate sparsity of SLAM information matrices. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 331–337, Barcelona, Spain, April 2005.

[19] U. Frese. Treemap: An $O(\log n)$ algorithm for simultaneous localization and mapping. In C. Freksa, editor, *Spatial Cognition IV*, pages 455–476. Springer–Verlag, 2005.

[20] U. Frese, P. Larsson, and T. Duckett. A multilevel relaxation algorithm for simultaneous localization and mapping. *IEEE Transactions on Robotics*, 21(2):196–207, April 2005.

[21] G. Gallo, M. D. Grigoriadis, and R. E. Tarjan. A fast parametric maximum flow algorithm and applications. *SIAM J. Comput.*, 18(1):30–55, 1989.

[22] G. Grisetti, C. Stachniss, and W. Burgard. Improving grid-based SLAM with Rao-Blackwellized particle filters by adaptive proposals and selective resampling. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 2432–2437, Barcelona, April 2005.

[23] Giorgio Grisetti, Dario Lodi Rizzini, Cyrill Stachniss, Edwin Olson, and Wolfram Burgard. Online constraint network optimization for efficient maximum likelihood map learning. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2008.

[24] Giorgio Grisetti, Cyrill Stachniss, Slawomir Grzonka, and Wolfram Burgard. A tree parameterization for efficiently computing maximum likelihood maps using gradient descent. In *Proceedings of Robotics: Science and Systems (RSS)*, Atlanta, GA, USA, 2007.

[25] Andreas Haeberlen, Eliot Flannery, Andrew M. Ladd, Algis Rudys, Dan S. Wallach, and Lydia E. Kavraki. Practical robust localization over large-scale 802.11 wireless networks. In *MobiCom '04: Proceedings of the 10th annual international conference on Mobile computing and networking*, pages 70–84, New York, NY, USA, 2004. ACM.

[26] D. Hähnel, W. Burgard, B. Wegbreit, and S. Thrun. Towards lazy data association in SLAM. In *Proceedings of the International Symposium of Robotics Research (ISRR)*, Sienna, Italy, 2003. Springer.

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

[28] Berthold K. P. Horn. Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America. A*, 4(4):629–642, Apr 1987.

[29] Andrew Howard and Nicholas Roy. The robotics data set repository (radish), 2003.

[30] M.I. Jordan. *Learning in Graphical Models.* MIT Press, 1999.

[31] Michael Kaess, Ananth Ranganathan, and Frank Dellaert. Fast incremental square root information smoothing. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2129–2134, Hyderabad; India, 2007.

[32] Yan Ke and Rahul Sukthankar. PCA-SIFT: A more distinctive representation for local image descriptors. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, 02:506–513, 2004.

[33] Alexander Kleiner, Johann Prediger, and Bernhard Nebel. RFID technology-based exploration and SLAM for search and rescue. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4054–4059, October 2006.

[34] Kurt Konolige. Large-scale map-making. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 457–463, 2004.

[35] Benjamin Kuipers and Patrick Beeson. Bootstrap learning for place recognition. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 174–180, Edmonton, Canada, 2002.

[36] J. Leonard and P. Newman. Consistent, convergent, and constant-time SLAM. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1143–1150, Acapulco, Mexico, August 2003. IJCAI.

[37] John Leonard, David Barrett, Jonathan How, Seth Teller, and et al. Team MIT DARPA urban challenge technical report. Technical report, Massachusetts Institute of Technology, April 2007.

[38] David G. Lowe. Object recognition from local scale-invariant features. In *International Conference on Computer Vision*, pages 1150–1157, Corfu, Greece, 1999.

[39] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, November 2004.

[40] F. Lu and E. Milios. Robot pose estimation in unknown environments by matching 2d range scans. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 935–938, 1994.

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

[42] F. Lu and E. Milios. Robot pose estimation in unknown environments by matching 2D range scans. *Journal of Intelligent and Robotic Systems*, 18(249-275), 1997.

[43] Peter S. Maybeck. *Stochastic Models, Estimation, and Control*, volume 1. Academic Press, New York, NY, 1979.

[44] Michael Montemerlo. *FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem with Unknown Data Association.* PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, July 2003.

[45] Michael Montemerlo, Nicholas Roy, and Sebastian Thrun. Perspectives on standardization in mobile robot programming: The Carnegie Mellon navigation (CARMEN) toolkit. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 3, pages 2436–2441, Las Vegas, NV, October 2003.

[46] Jose Neira and Juan D. Tardos. Data association in stochastic mapping using the joint compatibility test. *IEEE Transactions on Robotics and Automation*, 17(6):890–897, December 2001.

[47] Paul M. Newman and John Leonard. Pure range-only sub-sea SLAM. In *Proceedings of the IEEE Conference on Robotics and Automation (ICRA '02)*, pages 1921–1926, 2003.

[48] Edwin Olson, John Leonard, and Seth Teller. Fast iterative optimization of pose graphs with poor initial estimates. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 2262–2269, 2006.

[49] Edwin Olson, John Leonard, and Seth Teller. Robust range-only beacon localization. *IEEE Journal of Oceanic Engineering*, 31(4):949–958, October 2006.

[50] Edwin Olson, John Leonard, and Seth Teller. Spatially-adaptive learning rates for online incremental SLAM. In *Proceedings of Robotics: Science and Systems*, Atlanta, GA, USA, June 2007.

[51] Edwin Olson, Matthew Walter, John Leonard, and Seth Teller. Single cluster graph partitioning for robotics applications. In *Proceedings of Robotics Science and Systems*, pages 265–272, 2005.

[52] M.A. Paskin. Thin junction tree filters for simultaneous localization and mapping. Technical Report UCB/CSD-02-1198, University of California, Berkeley, September 2002.

[53] Son Bao Pham, Bernhard Hengst, Darren Ibbotson, and Claude Sammut. Stochastic gradient descent localisation in quadruped robots. In Andreas Birk 0002, Silvia Coradeschi, and Satoshi Tadokoro, editors, *RoboCup*, volume 2377 of *Lecture Notes in Computer Science*, pages 447–452. Springer, 2001.

[54] Nissanka Priyantha, Anit Chakraborty, and Hari Balakrishnan. The cricket location-support system. In *Proceedings of the ACM International Conference on Mobile Computing and Networking (MobiCom)*, pages 32–34, Boston, MA, August 2000.

[55] Ronald L. Rivest and Charles E. Leiserson. *Introduction to Algorithms*. McGraw-Hill, Inc., New York, NY, USA, 1990.

[56] H. Robbins and S. Monro. A stochastic approximation method. *Annals of Mathematical Statistics*, 22:400–407, 1951.

[57] Nicholas Roy and Joelle Pineau. *Gerontechnology: Growing Old in a Technological Society*, chapter Robotics and Independence for the Elderly, pages 209–242. Charles C. Thomas Publisher Ltd., 2007. Co-authors: Nicholas Roy (MIT).

[58] Brad Schumitsch, Sebastian Thrun, Gary Bradski, and Kunle Olukotun. The information-form data association filter. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Advances in Neural Information Processing Systems (NIPS)*, pages 1193–1200, Cambridge, MA, 2006. MIT Press.

[59] S. Se, D. Lowe, and J. Little. Vision-based mobile robot localization and mapping using scale-invariant features. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 2051–2058, Seoul, Korea, May 2001.

[60] S. Se, D. Lowe, and J. Little. Mobile robot localization and mapping with uncertainty using scale-invariant visual landmarks. *International Journal of Robotics Research*, 21(8):735–758, 2002.

[61] H. Shatkay and L. Kaelbling. Learning topological maps with weak local odometric information. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 920–929, 1997.

[62] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 888–905, August 2000.

[63] Sudipta Sinha, Jan-Michael Frahm, Marc Pollefeys, and Yakup Genc. Feature tracking and matching in video using programmable graphics hardware. *Machine Vision and Applications*, November 2007.

[64] J. Sivic and A. Zisserman. Video Google: Efficient visual search of videos. In J. Ponce, M. Hebert, C. Schmid, and A. Zisserman, editors, *Toward Category-Level Object Recognition*, volume 4170 of *LNCS*, pages 127–144. Springer, 2006.

[65] R. Smith, M. Self, and P. Cheeseman. A stochastic map for uncertain spatial relationships. In O. Faugeras and G. Giralt, editors, *Proceedings of the International Symposium of Robotics Research (ISRR)*, pages 467–474, 1988.

[66] C. Stachniss, G. Grisetti, and W. Burgard. Recovering particle diversity in a Rao-Blackwellized particle filter for SLAM after actively closing loops. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 667–672, Barcelona, Spain, 2005.

[67] Gilbert Strang. *Introduction to Linear Algebra.* Wellesley-Cambridge Press, 1993.

[68] J.D. Tardós, J. Neira, P.M. Newman, and J.J. Leonard. Robust mapping and localization in indoor environments using sonar data. *International Journal of Robotics Research*, 21(4):311–330, 2002.

[69] S. Thrun and Y. Liu. Multi-robot SLAM with sparse extended information filters. In *Proceedings of the International Symposium of Robotics Research (ISRR)*, Sienna, Italy, 2003.

[70] S. Thrun, Y. Liu, D. Koller, A.Y. Ng, Z. Ghahramani, and H. Durrant-Whyte. Simultaneous localization and mapping with sparse extended information filters. Technical report, Carnegie Mellon University, Pittsburgh, PA, April 2003.

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

[72] S. Thrun and M. Montemerlo. The GraphSLAM algorithm with applications to large-scale mapping of urban structures. *International Journal of Robotics Research*, 25(5-6):403–430, May-June 2006.

[73] Walker. Degrees of freedom. *Journal of Educational Psychology*, 31(4):253–269, 1940.

[74] M. Walter, R. Eustice, and J. Leonard. A provably consistent method for imposing exact sparsity in feature-based SLAM information filters. In S. Thrun, R. Brooks, and H. Durrant-Whyte, editors, *Proceedings of the International Symposium of Robotics Research (ISRR)*, pages 214–234, San Francisco, CA, October 2005. Springer.

[75] M. Walter, F. Hover, and J. Leonard. SLAM for ship hull inspection using exactly sparse extended information filters. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Pasadena, CA, May 2008.

[76] Matthew R. Walter, Ryan M. Eustice, and John J. Leonard. Exactly sparse extended information filters for feature-based SLAM. *Int. J. Rob. Res.*, 26(4):335–359, 2007.