

**ATLAS:**  
**A Framework for Large Scale**  
**Automated Mapping and Localization**

by

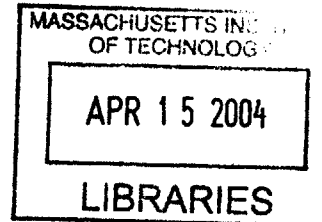
Michael Carsten Bosse

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

Feb 2004



© Massachusetts Institute of Technology 2004. All rights reserved.

Author .....  
Department of Electrical Engineering  
and Computer Science  
Feb 2, 2004

Certified by .....  
Seth Teller  
Associate Professor  
Thesis Supervisor

Certified by .....  
John J. Leonard  
Associate Professor  
Thesis Supervisor

Accepted by .....  
Arthur C. Smith  
Chairman, Department Committee on Graduate Students



**ATLAS:**  
**A Framework for Large Scale**  
**Automated Mapping and Localization**

by

Michael Carsten Bosse

Submitted to the Department of Electrical Engineering  
and Computer Science  
on Feb 2, 2004, in partial fulfillment of the  
requirements for the degree of  
Doctor of Philosophy in Computer Science and Engineering

**Abstract**

This thesis describes a scalable robotic navigation system that builds a map of the robot's environment on the fly. This problem is also known as Simultaneous Localization and Mapping (SLAM). The SLAM problem has as inputs the control of the robot's motion and sensor measurements to features in the environment. The desired output is the path traversed by the robot (localization) and a representation of the sensed environment (mapping). The principal contribution of this thesis is the introduction of a framework, termed *Atlas*, that alleviates the computational restrictions of previous approaches to SLAM when mapping extended environments. The *Atlas* framework partitions the SLAM problem into a graph of submaps, each with its own coordinate system. Furthermore, the framework facilitates the modularity of sensors, map representations, and local navigation algorithms by encapsulating the implementation specific algorithms into an abstracted module. The challenge of loop closing is handled with a module that matches submaps and a verification procedure that trades latency in loop closing with a lower chance of incorrect loop detections inherent with symmetric environments. The framework is demonstrated with several datasets that map large indoor and urban outdoor environments using a variety of sensors: a laser scanner, sonar rangefinders, and omni-directional video.

Thesis Supervisor: Seth Teller  
Title: Associate Professor

Thesis Supervisor: John J. Leonard  
Title: Associate Professor



## Acknowledgments

First of all, I would like to thank my family, Steffi, Erika, and Carsten for their ongoing support and encouragement;

I want to thank my advisors, Seth and John, for all the emergency meetings, advice, and for going easier on me at the end; My thesis committee, Eric and Leslie, for their invaluable comments on the early drafts; Paul, the best person I have ever worked with, for bringing out my productive side; Larry, for his writing tutelage and letting me crash at his place during the most difficult weeks of the writing; Elise's mother, for counseling me with regards to my writers' block; My friends, Erik, Chris and Matt, for brainstorming with me and letting me bounce so many ideas off of them; Jill and Bryt, for taking care of all the administrative details that had me running in circles;

And most of all, I thank my beloved, Elise, for standing by me even when I was at my most abject, for reading, organizing, and giving input on a topic she knows little about, and for her love.



# Contents

<b>1</b>	<b>Introduction</b>	<b>19</b>
1.1	The Simultaneous Localization and Mapping Problem . . . . .	19
1.2	Challenges . . . . .	21
1.3	Related Research . . . . .	25
1.4	Atlas Fundamentals and System Overview . . . . .	28
1.5	Working Hypothesis and Contributions . . . . .	31
1.6	Road Map . . . . .	32
<b>2</b>	<b>Atlas Framework</b>	<b>33</b>
2.1	Introduction . . . . .	33
2.1.1	Basic Principles . . . . .	33
2.1.2	Loop Closing and Uncertainty . . . . .	34
2.2	Atlas Components . . . . .	37
2.2.1	Uncertainty Projection . . . . .	37
2.2.2	Genesis . . . . .	40
2.2.3	Map-Matching . . . . .	41
2.2.4	Cycle Verification . . . . .	42
2.2.5	Traversal with Competing Hypotheses . . . . .	45
2.3	Atlas Modules . . . . .	48
2.3.1	Local SLAM . . . . .	51
2.3.2	Map-Matching . . . . .	51
2.4	Global Map Frame Optimization . . . . .	52
2.5	Summary and Running Time Analysis . . . . .	53

<b>3</b>	<b>Atlas with 2D Laser Lines</b>	<b>55</b>
3.1	Introduction . . . . .	55
3.2	Line Feature Estimation . . . . .	57
3.2.1	Line Splitting and Fitting . . . . .	60
3.3	Landmark Kalman Filter . . . . .	62
3.3.1	Odometry Model . . . . .	62
3.3.2	Advanced Odometry Model . . . . .	65
3.3.3	Measurement Models and Processing Summary . . . . .	67
3.3.4	Robot Relocalization . . . . .	77
3.4	Map Matching . . . . .	79
3.4.1	Line Signature Strings . . . . .	80
3.4.2	Repetitive Structure . . . . .	81
3.4.3	Alignment Optimization . . . . .	81
3.4.4	Summary . . . . .	83
3.5	Experimental Results . . . . .	84
3.5.1	MIT's Killian Court . . . . .	85
3.5.2	Ten loops in a small-scale environment . . . . .	96
<b>4</b>	<b>Atlas with Wide Angle Sonar</b>	<b>101</b>
4.1	Introduction . . . . .	101
4.2	Sonar Features . . . . .	102
4.2.1	Point Features . . . . .	102
4.2.2	Line Features . . . . .	104
4.3	RANSAC Data Association . . . . .	104
4.3.1	Adjacency preprocessing . . . . .	105
4.4	Multiple Vantage-Point Kalman Filter . . . . .	108
4.4.1	Saved robot poses . . . . .	108
4.4.2	Feature updates . . . . .	110
4.4.3	Feature Initialization . . . . .	112
4.4.4	Performance Metric . . . . .	113



4.5	Map Matching and Signatures . . . . .	113
4.6	Experimental Results: Killian Court . . . . .	115
<b>5</b>	<b>Atlas with 2D Laser Scan-matching</b>	<b>119</b>
5.1	Introduction . . . . .	119
5.2	Iterative Closest Point Scan-matching . . . . .	120
5.2.1	Normals . . . . .	120
5.2.2	Transformation Update . . . . .	122
5.3	Pose Snapshot Kalman Filter . . . . .	127
5.3.1	Relocalization . . . . .	129
5.3.2	Performance Metric . . . . .	129
5.4	Map Matching . . . . .	130
5.5	Experimental Results . . . . .	131
5.5.1	Killian Court . . . . .	131
5.5.2	Ten Loops . . . . .	132
5.5.3	Victoria Park . . . . .	137
5.5.4	Pennsylvania Coal Mine . . . . .	141
<b>6</b>	<b>Atlas with Omnidirectional Video</b>	<b>145</b>
6.1	Introduction . . . . .	145
6.1.1	<i>Atlas</i> requirements . . . . .	146
6.1.2	Assumptions . . . . .	147
6.2	Relationship to Previous Work . . . . .	148
6.2.1	Feature Geometry . . . . .	149
6.2.2	Chapter Summary . . . . .	150
6.3	Single View Geometry . . . . .	150
6.3.1	Omnicam projection model . . . . .	151
6.3.2	Image Points . . . . .	152
6.3.3	Image Lines . . . . .	153
6.3.4	Vanishing Points . . . . .	154
6.3.5	Line estimation . . . . .	155

6.3.6	RANSAC for Vanishing Points . . . . .	156
6.3.7	Expectation Maximization for Vanishing Points . . . . .	158
6.4	Multiple View Geometry . . . . .	160
6.4.1	Point Triangulation . . . . .	160
6.4.2	Line Triangulation . . . . .	161
6.4.3	Line Segment Matching with Dynamic Programming . . . . .	162
6.4.4	Bundle Adjustment . . . . .	163
6.5	Local Mapping Data flow . . . . .	166
6.5.1	Preprocessing . . . . .	166
6.5.2	Feature Tracking . . . . .	168
6.5.3	Bundle Adjustment . . . . .	168
6.6	Map Matching . . . . .	169
6.7	Experimental Results . . . . .	170
6.7.1	Hardware Setup . . . . .	170
6.7.2	Lounge Sequence . . . . .	171
6.7.3	Library Sequence . . . . .	172
6.7.4	Atrium Sequence . . . . .	173
6.7.5	Tech Square with Omnibike . . . . .	186
<b>7</b>	<b>Conclusion</b>	<b>191</b>
7.1	Summary . . . . .	191
7.1.1	Implementation Comparison . . . . .	192
7.1.2	Connectivity Metric . . . . .	193
7.2	Failure modes . . . . .	196
7.3	Future Work . . . . .	199
7.3.1	Relocation with partial observability . . . . .	199
7.3.2	On-line map fusion . . . . .	200
7.3.3	Global state estimation . . . . .	200
7.3.4	Improved implementation of <i>Atlas</i> modules . . . . .	201
7.3.5	Multi-robot Mapping . . . . .	202

7.3.6 Exploration and Long-term Autonomy . . . . . 202



# List of Figures

1-1	A graph of map-frames . . . . .	29
2-1	Ambiguity condition . . . . .	36
2-2	The Dijkstra Projection using two different source nodes. . . . .	38
2-3	Dijkstra Shortest Path Tree . . . . .	40
2-4	Cycle Verification . . . . .	44
2-5	Hypothesis state transition diagram . . . . .	48
2-6	The spatial and topological evolution of an <i>Atlas</i> graph, Part I . . . .	49
2-7	The spatial and topological evolution of an <i>Atlas</i> graph, Part II . . . .	50
2-8	Algorithm Summary . . . . .	54
3-1	Typical Laser Scan . . . . .	56
3-2	Line Parameterization . . . . .	58
3-3	Line Splitting . . . . .	61
3-4	Seeding the robot position for a juvenile hypothesis . . . . .	78
3-5	Line Signature Elements . . . . .	81
3-6	Map-Matching as a search for a transformation between maps . . . .	84
3-7	Killian Court topology . . . . .	88
3-8	Killian laser maps using uncorrected odometry . . . . .	89
3-9	Uncorrected odometry bias . . . . .	90
3-10	Odometry trajectories . . . . .	91
3-11	Killian laser maps . . . . .	92
3-12	Killian laser adjacency matrix . . . . .	93
3-13	Killian laser Kalman filter residuals . . . . .	94

3-14 Killian global optimization residuals . . . . .	94
3-15 Killian laser processor load . . . . .	95
3-16 Building 6 and odometry path . . . . .	97
3-17 Tenloops laser maps . . . . .	98
3-18 Tenloops adjacency and map times . . . . .	99
3-19 Tenloops laser Kalman filter residuals . . . . .	100
3-20 Tenloops global optimization residuals . . . . .	100
4-1 Sonar Features . . . . .	103
4-2 Sonar Point Matching . . . . .	107
4-3 Sonar Line Matching . . . . .	107
4-4 Killian sonar maps . . . . .	116
4-5 Killian sonar adjacency matrix . . . . .	117
4-6 Killian sonar Kalman filter residuals . . . . .	118
4-7 Killian global optimization residuals . . . . .	118
5-1 Iterative Closest Point . . . . .	121
5-2 Scan point normal . . . . .	122
5-3 Scan-match Diagram . . . . .	128
5-4 Killian scan-match optimized map . . . . .	132
5-5 Killian scan-match adjacency and map times . . . . .	133
5-6 Killian scan-match Kalman filter residuals . . . . .	134
5-7 Killian global optimization residuals . . . . .	134
5-8 Ten loops scan-match optimized map . . . . .	135
5-9 Ten loops adjacency and map times . . . . .	136
5-10 Victoria Park scan-match map . . . . .	138
5-11 Victoria Park adjacency and map times . . . . .	139
5-12 Victoria Park scan-match Kalman filter residuals . . . . .	140
5-13 Victoria Park global optimization residuals . . . . .	140
5-14 Mines scan-match optimized map . . . . .	141
5-15 Mines scan-match processing time . . . . .	142

5-16	Mines scan-match adjacency and map times . . . . .	143
5-17	Mines scan-match Kalman filter residuals . . . . .	144
5-18	Mines global optimization residuals . . . . .	144
6-1	Omnicam Geometry . . . . .	152
6-2	Projection of a 3-D line . . . . .	157
6-3	Line ordering . . . . .	163
6-4	Data Flow Diagram . . . . .	166
6-5	Omnicam Hardware . . . . .	171
6-6	Omnicam sample image . . . . .	172
6-7	Lounge residual point errors . . . . .	174
6-8	Lounge residual line errors . . . . .	174
6-9	Lounge map match scores . . . . .	175
6-10	Lounge plan view . . . . .	176
6-11	Lounge oblique view . . . . .	176
6-12	Lounge plan view of path . . . . .	177
6-13	Lounge oblique view of path . . . . .	177
6-14	Library residual errors . . . . .	178
6-15	Library map match scores . . . . .	179
6-16	Library plan view . . . . .	180
6-17	Library oblique view . . . . .	180
6-18	Library plan view of path . . . . .	181
6-19	Library oblique view of path . . . . .	181
6-20	Atrium reprojection errors . . . . .	182
6-21	Atrium map match scores . . . . .	183
6-22	Atrium plan view . . . . .	184
6-23	Atrium oblique view . . . . .	184
6-24	Atrium plan view of path . . . . .	185
6-25	Atrium profile view of path . . . . .	185
6-26	Camera mounted on bicycle . . . . .	186

6-27 Omnibike residual point errors . . . . .	187
6-28 Omnibike residual line errors . . . . .	187
6-29 Omnibike map match scores . . . . .	188
6-30 Omnibike oblique view . . . . .	189
6-31 Omnibike overlay plan view . . . . .	189
6-32 Omnibike plan view . . . . .	190
6-33 Omnibike oblique path view . . . . .	190
7-1 Killian Court robot pose comparison . . . . .	194
7-2 Pose Errors . . . . .	195
7-3 Time Adjacency Matrix . . . . .	197



# List of Tables

1.1	Related Research . . . . .	26
2.1	<i>Atlas</i> hypothesis types . . . . .	47
4.1	Map signature elements definitions . . . . .	113
6.1	Image sequences for experiments. . . . .	171
7.1	Standard deviation of global robot pose errors. . . . .	193
7.2	Topological Connectivity Performance. . . . .	196



# Chapter 1

## Introduction

The problem of simultaneous localization and mapping (SLAM) is to enable a mobile robot to build a map of its environment, while simultaneously using this map to navigate. This thesis addresses two key shortcomings of previous SLAM algorithms: scale and modularity. It demonstrates a SLAM algorithm that can cope with large-scale, cyclic environments, and a single, unified SLAM architecture that has been successfully applied to a variety of different input sequences, including laser, sonar, and omnidirectional video data.

### 1.1 The Simultaneous Localization and Mapping Problem

The Global Positioning System (GPS) has revolutionized localization. Developed in the 1970s and 80s, GPS provides the user with an earth relative position and velocity estimate to sub-meter accuracy under certain conditions. However, knowing one's location in an arbitrary coordinate system has little value without a map that links these coordinates to a representation of the environment.

Together, a localization system and maps allow individuals to navigate successfully. Nevertheless, there are limitations to the efficacy of the integration of maps and localization. GPS functions fully only in outdoor locations with good visibility

to satellites. It is not practical for use indoors, underground, underwater, in urban canyons, or on planets lacking GPS satellites. Maps are highly dependent on intensive human labor for their construction and are limited by the models used to represent the features of the environment on any given map. Thus, while localization and mapping are basic aspects of successful navigation, their effective integration and application remains a significant challenge in many areas.

In mobile robotics, localization and mapping are fundamental requirements for applications such as exploration, path planning, and dynamic object tracking. In order to effectively accomplish tasks in which they interact with their environments, robots need to know where they are. However, this knowledge of position is meaningless without a map that relates the robot's location to its environment. Autonomous vacuum cleaners, supermarket floor washers, and tour guide robots [7] are examples of existing mobile robots that navigate based upon maps.

These maps can be provided to the robot ahead of time or simultaneously generated by the robot while localizing. This latter form of mapping is both more desirable and more challenging. *A priori* maps often do not exist or are incomplete, inaccurate, or inadequate for a robot's needs, while robots with the ability to self-generate maps are able to explore previously unmapped areas. Furthermore, an internally generated map will generally be more up-to-date than one previously produced. The challenge ensues when a robot must localize concurrently with mapping. Individually, mapping and localization are straightforward tasks accomplished with bounded complexity. When coupled, the complexity growth of these tasks can escalate in an unbounded manner. Overcoming this hurdle is a key to expanding the future of mobile robotics.

In the robotics community, the task of a robot building a map and simultaneously localizing from it is commonly referred to as Simultaneous Localization and Mapping (SLAM). Most current SLAM implementations are limited by the size and type of the environments they can handle. There are few existing algorithms that work well in large-scale environments; however, there are several notable active projects that seek to tackle large-scale mapping, including the mapping of abandoned coal mines [55], searching the ocean for explosive mines [43, 33], and capturing models of

entire cities [52].

This thesis approaches large-scale SLAM from a different perspective. Rather than develop a system in which all components are tightly integrated and therefore dependent upon each other, it is possible to separate the dependencies into modules. This method of operation allows for flexibility and variety in implementing SLAM solutions. This thesis proposes a system for encapsulating existing SLAM solutions for small-scale environments within a framework that allows them to be applied to large-scale SLAM problems. This modular framework also precludes a dependence upon a single choice for sensor or map representation, further facilitating the implementation of different SLAM systems. To accomplish this, each of the major challenges to achieving real-time SLAM must be addressed.

## 1.2 Challenges

The challenges to achieving successful SLAM are varied and often interconnected. Sensor, model, and mapping variety have lead to a hodgepodge of methods for approaching SLAM. The need to map large-scale environments complicates SLAM and further exacerbates issues such as coupled uncertainty, loop closing, and loop validation. An effective large-scale SLAM framework must address these challenges.

Model variety makes SLAM more difficult. Differences in sensor measurement, robot dynamics, and map representation models convolute efforts to implement a unified and systematic approach to SLAM. Two basic classes of sensors, proprioceptive and mapping, are utilized in SLAM. Proprioceptive sensors directly measure a robot's relative motion using gyroscopes, accelerometers, wheel odometers, and steer angle sensors. Robots like ground rovers and water surface crafts move only in 2D with rotations. Robots with 6 degrees of freedom (DOF) of motion capabilities, such as aircraft, helicopters, underwater vehicles, and hand-held cameras, can access more environments. Because proprioceptive sensors measure only relative motion of the robot, they lead to unbounded growth in localization error.

Mapping sensors measure properties of the environment relative to the robot.

They can be used to correct the unbounded dead-reckoning errors of proprioceptive sensors by re-observing previously mapped features. These mapping sensors may be active or passive and measure bearing, range, or both. For example, a camera is a passive, bearing-only sensor, whereas a sonar ranger is an active, range-only sensor. The most useful sensors measure both range and bearing. A common example of a fully observable sensor is a laser scanner, which can measure all of the modeled degrees of freedom of an environment features from a single pose. However, sensors that observe only partial DOF of the environment's features require that multiple measurements from different positions be combined in order to obtain a complete view of structure in the scene. With mapping sensors it is necessary to interpret the data correctly, which means having an appropriate model for observations of features in the environment.

A final form of model variety inherent within SLAM is map representation. The environment and the set of available sensors determine the choice of map representation. Feature-based maps [47] work best in structured or sparse environments in two or three dimensions. Evidence grids [40, 46] are most effective in small unstructured 2D environments, particularly when using fully observable sensors [12].

Each particular implementation of SLAM will have its own criteria for determining which models will be used. These criteria must take into account the types of sensors available and environment to be mapped. Given that the process of mapping environments with robots is not deterministic, uncertainties exist in all of the models described above. These uncertainties themselves must be modeled with probabilities and reasonable assumptions must be made on the chosen error distributions. If errors in sensor measurements, robot dynamics and map fidelity are kept small, then it is possible to use a Gaussian noise assumption with linearized models. Unfortunately, in the real world, models are nonlinear and errors are often large or not Gaussian. Although Gaussian distributions can be fit to nonlinear models when the errors are small, the linearizations are no longer accurate when these errors become large. Non-Gaussian errors include multi-modal errors that occur, for example, from mistakes in matching during data association.

Another major challenge in SLAM arises from the coupling of sensor and robot uncertainties. Coupling occurs when noisy sensor measurements are taken from uncertain robot poses. This coupling is most apparent between two mapped features. If there were no uncertainty in the robot pose when mapping the features, then the errors of the features would be independent from one another. However, since both features are mapped from the same uncertain robot, their errors are linked. For example when the robot makes a subsequent measurement on one feature, it improves the knowledge of the robot’s pose and hence consequently improves also the position of the second feature. This coupling is represented, when using Gaussian error distributions, as non-zero off-diagonal covariance matrix elements which cannot be ignored.

Large scale environments exacerbate the obstacles to SLAM by increasing the computational requirements. Often SLAM algorithms do not have constant time complexity per update. (Other constant time SLAM algorithms exist; however, they either don’t close loops [39] or make global linearization assumptions [56]) As new parts of the environment are mapped, their errors are coupled through the uncertain robot position to the errors of all the other mapped parts. Subsequently, when re-observing old parts of the map, all the coupled parts of the map will also need to be updated. Thus the computation required to process each measurement increases as the size of the map increases, unless some steps are taken to limit the coupling.

As noted above, the map elements and the robot poses are statistically coupled, and even in the simple Gaussian case, all the cross-covariances need to be maintained. (The cross-covariances are significant even if the features are not co-visible). Since the estimated errors of all the map elements are correlated, processing each sensor measurement results in at least a quadratic growth in memory and computation as the size of the map increases. The challenge is to develop valid approximations such that not all correlations among robot and map elements need to be maintained. Ideally, the computational time should remain nearly constant per sensor measurement, and the memory required should grow nearly linearly with the size of the map.

Another challenge inherent with mapping extended environments is the need to

close loops when revisiting previously mapped areas. The difficulty arises since the robot may take a long time and travel a significant distance before revisiting the area from which it started. Hence the open-loop uncertainty can be quite large and it will perhaps even be difficult for the robot to recognize that it is in a previously mapped region.

There are two algorithmic elements to loop closing. The first is data association, or, recognizing when a loop can be closed. The second element is incorporating the effects of a loop closure into the SLAM system state. The difficulty in data association is that the recognition of revisited areas in a particular environment is dependent upon both the growth of uncertainty in local mapping and the richness of the local map representation. A slower uncertainty growth enables the robot to traverse longer paths before a mismatch between two nearby ambiguous regions occurs. Additionally, richer maps are less likely to exhibit local symmetry and global ambiguity. The update of a loop closure event is also challenging, especially if the prior uncertainty is large. The update must be distributed in some manner around the loop if the map is to be expressed in a single coordinate system. The error accumulated around the loop is typically large and poorly modeled, therefore the process of distributing the residuals of the loop-closing constraint can lead to inconsistencies or tears in the map. Also, if all the map elements around an arbitrarily long loop must be updated, then the computational burden for processing sensor measurements can no longer be bounded by a constant factor.

Loop closure validation is necessary to mitigate the errors associated with loop closing. There are two types of loop-closing errors: false positive matches and missed detected matches. False positive matches are situations in which the robot erroneously asserts that a loop has been closed; however, the inferred match is false. When the environment contains repetitive structure or local symmetries, the detected loop-closing map-matches may not be unique. Missed detections occur when a loop closure has been missed due to failure to successfully match the current map with a previously mapped area. Furthermore, the prior uncertainty of the robot before closing the loop may be too large to identify the correct match. Even when there is only one candidate



match, it is possible that the correct match has simply not been recognized yet and the lone existing match is incorrect. The challenge for the algorithm is to verify loop closure decisions before they are committed, to be certain that no mistakes are made.

## 1.3 Related Research

A review of related research reveals common issues in the field of SLAM. How various methods address map and probabilistic representations, non-linearities, data association, scaling, and loop closing highlights the strengths and weaknesses of the different methods. Table 1.3 summarizes the key references in terms of the aforementioned categories in a concise format. These previous research efforts will be referenced throughout the remainder of the thesis.

Probabilistic techniques have proven vital in attacking the large-scale simultaneous localization and mapping problem. A variety of approaches have been proposed for representing the uncertainty inherent to sensor data and robot motion, including topological [30], particle filter [53, 39], and feature-based [48] models. Several successful SLAM approaches have been developed based on the combination of laser scan matching with Bayesian state estimation [27, 53]. All of these methods, however, encounter computational difficulties when closing large loops.

The Kalman filter provides the optimal linear recursive solution to SLAM when certain assumptions hold, such as perfect data association, linear motion and measurement models, and Gaussian error models [48]. The convergence and scaling properties of the Kalman filter solution to the linear Gaussian SLAM problem are now well-known (Dissanayake *et al.* [17]). Considerable recent research effort has been extended toward mitigation of the  $\mathcal{O}(n^2)$  complexity (where  $n$  is the number of features) of the Kalman filter SLAM solution. Efficient strategies for SLAM with feature-based representations and Gaussian representation of error include postponement [14], decoupled stochastic mapping (DSM) [31], the compressed filter [26], sequential map joining [50], the constrained local submap filter [61], and sparse extended information filters (SEIFs) (Thrun *et al.* [56]; Thrun *et al.* [55]). Each of these methods employs

Table 1.1: Related Research

<i>Researchers</i>	<i>Map Rep.</i>	<i>Nonlinear</i>	<i>Data Association</i>	<i>Scale</i>	<i>Global Coord.</i>	<i>Loop closing</i>
Smith, Self, Cheesman [47]	points	EKF	nearest	no	yes	no
Semantic Map [30]	metric topological semantic	<i>n/a</i>	<i>n/a</i>	yes	no	no
Gutmann & Konolige [27]	laser scans	full optim	ICP	yes except loops	yes	yes
Sequential Map Joining [50]	points & lines	SP Map	Joint Compatibility	yes	yes	no
SEIFs [56]	points	EIF	given	yes	yes	no
FastSLAM [39]	points	particle filter EKF	given	yes	yes	no
Sum of Gaussians [19]	points	sum of Gaussians	bayesian	no	yes	no
Choset's Voronoi [12]	voronoi graph	procedural	<i>n/a</i>	no	yes	yes
Postponement Filter [14]	points	EKF	nearest	no	yes	no
Compressed Filter [26]	points	EKF	nearest	yes	yes	no
Constrained Filter [61]	points	EKF	nearest	yes	no	yes
DSM [31]	points	EKF	nearest	yes	yes	no
Chong & Kleeman [11]	points & lines	EKF	nearest	yes	no	no
<i>Atlas</i> Laser Lines	lines	EKF	nearest	yes	no	yes
<i>Atlas</i> Sonar	points & lines	EKF	nearest	yes	no	yes
<i>Atlas</i> Scan-match	scans	EKF	ICP	yes	no	yes
<i>Atlas</i> Omni-video	points & lines	full optim	nearest	yes	no	yes

a single, globally-referenced coordinate frame for state estimation. The Kalman filter can fail badly, however, in situations in which large angular errors and significant data association ambiguities invalidate the Gaussian error assumptions. The large-scale linearization inherent in methods that use a single, global coordinate system for error representation, such as SEIFs or DSM, will fail when closing large loops with unbounded linearization errors.

In outdoor and underground environments, several SLAM algorithms have been implemented for large scale datasets. Guivant and Nebot have published results for a dataset acquired in Victoria Park, Sydney, Australia, using the compressed filter [26], with an implementation that employs trees as “point” features. More recently, Wang *et al.* [60] and Hähnel *et al.* [28] have achieved full 3D mapping of urban areas with dynamic objects using scan-matching. Thrun *et al.* [55] have demonstrated large-scale SLAM in a cyclic underground mine, also using scan-matching.

Some of the most successful experiments for autonomous loop closure in indoor environments have been performed by Gutmann and Konolige [27]. One notable feature of their work is the use of a hybrid metrical/topological map representation:

“A map is represented as an undirected graph: nodes are robot poses with associated scans and links are constraints between poses obtained from dead-reckoning, scan-matching, or correlation. [27]”

One of the appealing aspects of a hybrid metrical/topological approach to mapping and localization [9, 30, 3, 12] is that uncertain state estimates need not to be referenced to a single global reference frame. The strategy of partitioning a large map into multiple smaller maps is intuitively appealing, both for its computational efficiency and robustness. Chong and Kleeman [10] assert that “. . . the local mapping strategy is devised . . . to improve efficiency and to curb the accumulated ‘inevitable errors’ from propagating to other local maps continuously.” The hybrid metrical/topological approach models errors using Gaussian distributions only over local regions where linearization works well, rather than representing the entire environment with one

Gaussian distribution.

An alternative to the use of local linearization is the adoption of a fully nonlinear formulation of the SLAM problem, such as FastSLAM [39] or SLAM using a sum of Gaussians model [19]. The computational requirements of these methods, however, remain poorly understood and uncharacterized in large cyclic environments. In future research, it may be possible to implement one of these techniques as the local mapping strategy within the *Atlas* framework.

## 1.4 Atlas Fundamentals and System Overview

This thesis introduces a SLAM framework, termed *Atlas*, for the metaphor of a book with many adjacent maps. *Atlas* is a framework in which existing small-scale mapping algorithms are used to achieve real-time performance in large-scale, cyclic environments. The approach does not maintain a single, global coordinate frame, but rather an interconnected set of local coordinate frames, analogous to atlas pages. The representation consists of a graph of multiple local maps of limited size. Each vertex in the graph represents a local coordinate frame (and a local map expressed with respect to that frame), and each edge represents the transformation between adjacent local coordinate frames. In each local coordinate frame, a map is built that captures the local environment and the current robot pose along with associated uncertainties. Together, the map and the coordinate frame within which it is defined are referred to as a *map-frame*. Figure 1-1 depicts a graph of map-frames in which two hypotheses maintain the robot position.

The spatial extent of each map is not predefined. Instead, *Atlas* limits the *complexity* of each map. An intrinsic performance metric of the local SLAM processing determines whether to transition to an existing adjacent map-frame or generate a new one. The map's complexity is easily bounded by placing a hard limit on the

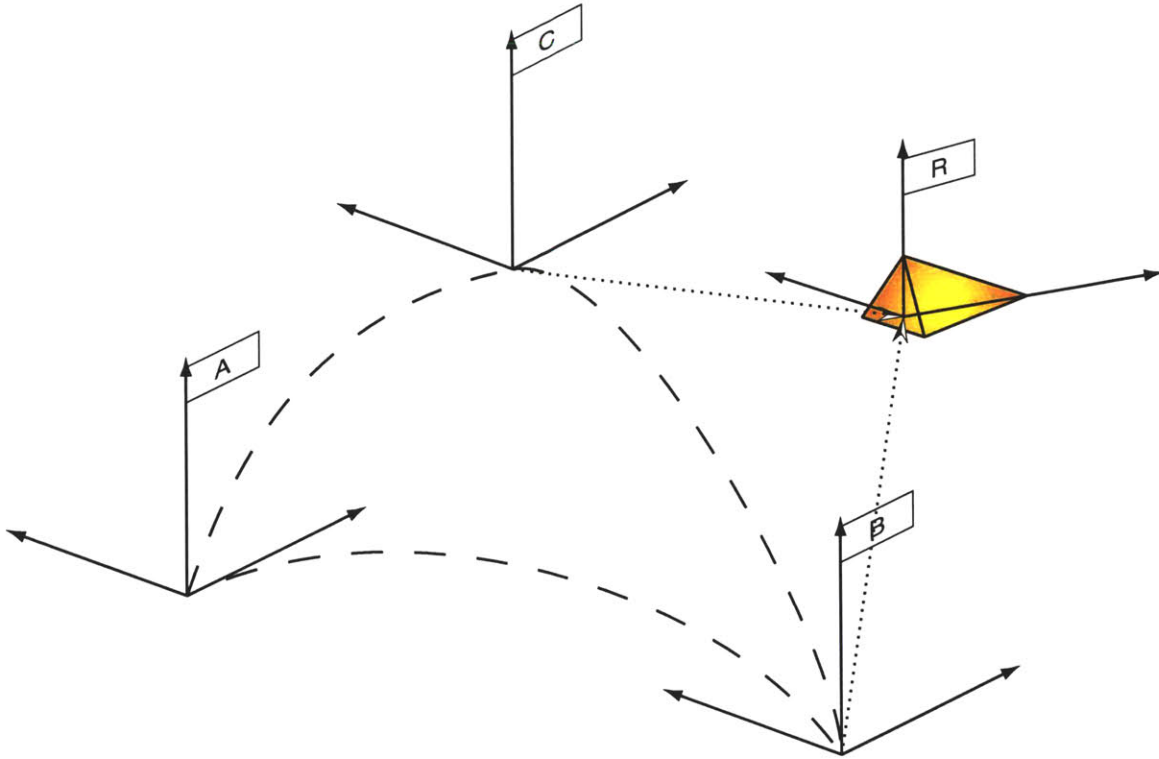


Figure 1-1: A graph of three map-frames in which two maps (B and C) can express the current robot position (R).

maximum number of features that can be inserted into the map. The performance metric is based on the certainty of the current robot pose by measuring how well the current robot pose and map are consistent with the current sensor readings.

*Atlas* is a generic framework which incorporates a variety of techniques as the local mapping module. The approach assumes that a suitable local SLAM algorithm exists that can produce consistent maps in small-scale (non-cyclic) regions with a fixed amount of computation for each new sensor observation. The local SLAM method may not incur an ever-growing computational burden, otherwise efficient global performance is not possible. For example with laser scan-matching, if local processing were based on the matching of a new sensor scan with all of the scans obtained in a local region, then local map complexity, which grows linearly with time, would not be bounded. Only a finite set of scans may be retained in any local

region to ensure that the local SLAM processing time can be bounded by a constant factor.

Each map’s uncertainties are modeled with respect to its own local coordinate frame. The uncertainty of the edges (adjacency transformations) in the *Atlas* graph are represented by a Gaussian random variable and are derived from the output of the SLAM algorithm running in a local region. The framework limits the per-map computation by defining a measure of complexity for each map-frame, which is not allowed to exceed a threshold (the map capacity). Rather than operating on a single map of ever-increasing complexity, the *Atlas* framework simply switches its focus to a new or adjacent map-frame, when the current map frame grows too large or performs poorly.

The *Atlas* graph encompasses new edges to close cycles via an efficient map-matching algorithm. All the map representation based dependencies for matching two sub-maps are contained in an implementation specific module, the Map-Matching module, whereas the general framework indicates potential map matches for map-frames that fall within an approximate uncertainty bound of the current map. The framework computes these uncertainty bounds by integrating the coordinate transformations and associated error estimates along a path formed by the edges between adjacent map-frames. A background map projection process computes these paths and error bounds by using a variant of Dijkstra’s shortest path algorithm [16].

The *Atlas* framework employs a verification step before integrating updates to the graph. The verification delays the acceptance of a map-match until additional map-matches are discovered that form a small consistent cycle in the graph. This procedure effectively increases the area of overlap considered to make sure that the map matches are not the result of ambiguous symmetries in the environment.

## 1.5 Working Hypothesis and Contributions

The *Atlas* framework abstracts and decouples the details of each sensor suite and environment characteristics from the mechanism for handling scale and loops. The *Atlas* framework for large scale SLAM has been implemented for a variety of sensors and tested in a variety of environments.

The first implementation uses extracted line features from a 2D scanning laser ranger on a B21 robot and models the environment as a collection of 2D line segments. A second implementation utilizes line and point features measured by monaural ultra-sonic ranggers. The third implementation again utilizes the laser ranger, but processes the raw laser data directly, using the technique of scan matching to relate data gathered from different positions without explicitly extracting any features from scans. A final implementation employs omni-directional video to model the environment with vanishing directions, 3D lines, and 3D points. This implementation uses a 3D world model and batch optimization on each local map rather than a recursive navigation strategy.

These implementations have been tested in a variety of environments which can be categorized into three groups based on the type of robot and sensors employed. The first group (Sections 3.5, 4.6, 5.5) uses combinations of the laser and sonar sensors from a B21 robot traveling through the corridors of MIT’s main campus. The second group (Section 6.7) also used the B21 robot, but mounted with an omnidirectional video camera. These data sets were also taken in and around MIT’s campus, including the CSAIL Reading Room, the Building 200 Second Floor Lounge, the Atrium in the same building, and also along similar routes as Group One about the “Infinite Corridor”. Additionally, data sets from third party sources taken in Victoria Park, Sydney [41] and coal mines in Pennsylvania [55], (Section 5.5) were processed using the Scan-Match implementation.

Perhaps the most significant contribution of the *Atlas* framework is its address-

ing of the Challenges described in Section 1.2. The issue of model variety is tackled through the modular framework where the details of the local SLAM module and Map-Matching module are abstracted from the general framework. Since the navigation module need only work with a limited sub-map of the environment, it can employ non-scalable SLAM algorithms that fully address and model the coupled uncertainty among mapped features and the current pose. Similarly the sub-map approach can map large scale environments efficiently, since only a few submaps need to be active at any particular time. The *Atlas* framework allows constant time processing complexity under the condition that the open-loop uncertainty of the robot does not diverge. *Atlas*'s Map-Matching module is employed to recognize loop closure events in large scale environments even when there are significant open loop errors. The *Atlas* framework also validates detected loops before they are closed to address the errors inherent in ambiguous environments.

## 1.6 Road Map

Chapter 2 will cover the details of the general *Atlas* framework whereas Chapters 3–6 cover details of specific implementations. The implementation using line features extracted from a 2D laser scanner is described in Chapter 3. Chapter 4 covers the implementation using range measurements from sonar. Chapter 5 revisits the laser sensor, but processes the scans without extracting features by matching scans, and Chapter 6 presents the implementation for processing omnidirectional video. Chapter 7, the conclusion, compares the implementations, discusses failure modes, and presents future work.



# Chapter 2

## Atlas Framework

### 2.1 Introduction

The *Atlas* framework achieves SLAM through a modular approach. By providing flexible access to a variety of small-scale solutions, *Atlas* successfully accomplishes real-time, large-scale SLAM.

#### 2.1.1 Basic Principles

The design of the *Atlas* framework rests on four basic principles: the elimination of a dependence on a global coordinate frame; the elimination of loop constraints; the elimination of predefined map extents, and the elimination of a dependence upon a single lowlevel mapping sensor and navigation strategy.

The *Atlas* framework maintains no single, global coordinate frame, but rather, an interconnected set of local coordinate frames. Each frame contains a local map of limited extent, called a *map-frame*. Two coordinate frames are considered to be connected (adjacent) if their map-frames possess shared mapped structure. This adjacency is represented by a approximate coordinate transformation (ACT) between frames.

Local maps handle short sensor excursions; however, in an extended mapping task previously mapped regions will be revisited and loops need to be closed. After recognizing the closure of an extended loop, the composition of adjacency transformations is not constrained to be the identity transformation. This strategy is essential to achieving constant time performance in the *Atlas* framework since no global updates are required. (Nevertheless, the identity constraint can be applied off-line to refine the global arrangement of the multiple coordinate frames.)

The spatial extent of the map-frames is not predefined, but rather, determined by an intrinsic performance metric on the map-frame’s associated map. Since the map complexity is bounded and the mapped features are in proximity to one another, the robot will exhibit a high performance metric when near the poses from which the features were mapped. The performance metric naturally drops when the robot is constrained from adding new features and leaves the proximity of mapped features, at which time the *Atlas* framework will invoke either a transition to an adjacent frame or the genesis of a new one.

The *Atlas* framework does not depend on a particular local navigation and mapping strategy. Instead, local navigation and mapping are abstracted into a module that need only provide an estimate of the robot’s current pose (along with its uncertainty) and a metric describing how well the current map explains the current sensor measurements. Likewise, the data association engine used to match maps is also modularized. By incorporating different sensor platforms and map representations, the modular design allows *Atlas* to scale a variety of SLAM implementations.

### **2.1.2 Loop Closing and Uncertainty**

Loop closing is one of the most difficult issues in SLAM research. Two different types of errors can occur in loop closing: false positive matches and missed detected matches. The former refers to situations in which the robot, based upon a false

match, erroneously asserts that a loop is closed. The latter case occurs when a loop closure is missed due to failure to successfully match the current map with a previously mapped area. *Atlas* adopts a conservative loop closing and verification strategy which attempts to avoid false positive matches at the expense of increased loop closing latency and missing some genuine loop closure events. It is possible, however, to present an adversely designed environment with highly repetitive structure and a path in which the accumulated uncertainty is so large that the *Atlas* technique, as well as any known SLAM loop closing algorithm, will fail.

Given a particular environment, the difficulty of loop closure is dependent on the growth of uncertainty in the local SLAM method and on the richness of the local map representation. A smaller growth of uncertainty allows for longer paths to be followed before confusing two nearby ambiguous regions, and richer maps are less likely to be ambiguous. Additionally, an environment with less repetitive global structure will present fewer challenges to loop closing, since there will be fewer ambiguous regions.

Uncertainty presents another challenge to SLAM. Local uncertainty results from noise introduced into individual measurements. Global uncertainty arises from the need to discern whether the region currently being observed by the robot consists of newly explored or previously visited territory. Both local and global uncertainty are exacerbated by repeated low-level features (e.g. periodic arrays of doors or windows) or high-level structure (e.g. nearby corridors that are nearly indistinguishable).

Uncertainty is a challenge to SLAM, because it is antagonistic both to achieving correctness and efficiency. When the mapping algorithm mistakes one local feature for another, or one region for another, it performs an incorrect data association and produces an incorrect map. On the other hand, if the algorithm expends excessive effort in an attempt to avoid such misassociations, it sacrifices efficiency. For algorithm designers, then, a key challenge is to construct a correct map with reasonably high probability, while bounding the amount of computation expended.

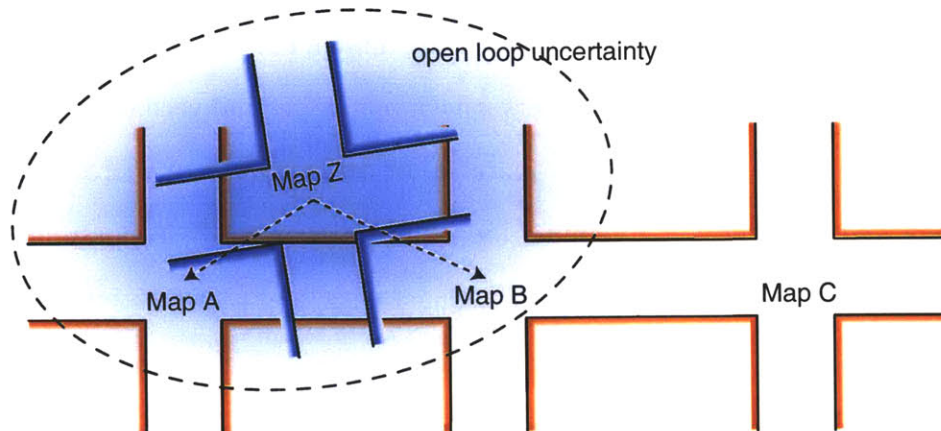


Figure 2-1: The open loop uncertainty of Map Z with respect to the Maps A, B, and C is large. Due to the scale of the symmetry in environment there are two ambiguous matches for Map Z.

No mapping system, including human cartography, can guarantee a perfect map. However, provided that the robot pose error accrued by the navigation subsystem along any loop (cyclic path) encompasses at most one region similar to the current sub-map, our algorithm will efficiently produce a correct map. In other words, the use of submaps allows *Atlas* to handle ambiguity at small scales (feature sizes). For *Atlas* to handle ambiguity at large scales, the environment must satisfy the condition that the ambiguity scale must be larger than the error ellipsoid accrued by the robot around any loop traversed prior to encountering the ambiguous region. (See Figure 2-1.)

To ensure that the system can correctly closes loops, the distance between possible matches must be less than the open loop uncertainty of the robot. The loop verification procedure has the effect of increasing the distances between possible matches by considering only those matches whose transformations are consistent with other maps in a local cycle of the graph.

After recognizing the closure of an extended loop, the *Atlas* framework does not constrain the composition of adjacency transformations to be the identity transfor-

mation. This policy precludes the need for global updates during the robot’s motion and is essential to achieving efficient, real-time performance. The identity constraint can be applied off-line, however, to refine the global arrangement of the multiple coordinate frames. (See Section 2.4.)

## 2.2 Atlas Components

The core components of the *Atlas* framework include uncertainty projection, genesis, map-matching, cycle verification, and traversal with competing hypotheses. Uncertainty Projection (Section 2.2.1) is used to relate the coordinate systems and uncertainties of map-frames that are not directly connected in the *Atlas* graph. Genesis (Section 2.2.2) is the process of creating new map-frames and adding nodes to the *Atlas* graph. Map-matching (Section 2.2.3) adds new edges to the graph by matching common structure between previously unconnected map-frames. The Cycle Verification procedure (Section 2.2.4) validates the potential loops created by map-matching. The Traversal process, employing competing hypotheses (Section 2.2.5), governs the decisions for which map-frame to transition to and whether to initiate the genesis process.

### 2.2.1 Uncertainty Projection

*Atlas* edges contain the information necessary to relate two map-frames. The uncertainty of a transformation edge is used to project a stochastic entity, such as the robot position, from one map-frame into another. However, if the map-frames are not adjacent, these transformations and their uncertainties must be composed along a path of edges that link the *Atlas* nodes. Due to cycles in the graph, there may be more than one path from one node to another. Since these cycles are not constrained online, distinct paths will not, in general, produce the same composite transforma-

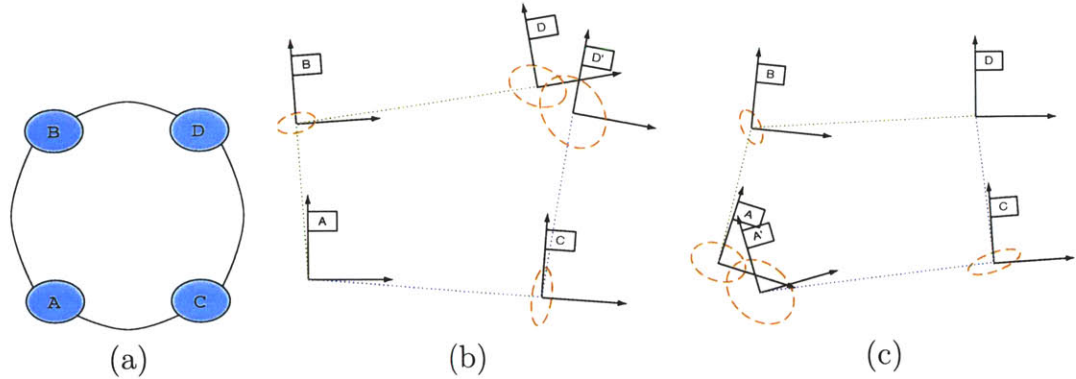


Figure 2-2: The Dijkstra Projection using two different source nodes. (a) depicts the topological arrangement of the *Atlas* graph. (b) uses map-frame A as the source of the projection. (c) uses map-frame D as the source. The ellipses on the coordinate frames represent the accumulated projection error. The shortest path from map-frame A to D is clearly via map-frame B.

tion. In Figure 2-2(a), frame D is reachable from A via B or C, resulting in the two possible projections of frame D relative to A, shown in Figure 2-2(b).

To overcome this ambiguity the *Atlas* framework chooses at any instant a single path. The path should be “short”, in terms of error, to minimize the accumulated error when composing multiple uncertain transformations. There are various possible definitions for what makes a path “short”. The particular choice for a shortness metric is not critical; rather, it is only necessary to have a reasonable metric to facilitate the determination of unique paths in the *Atlas* graph.

Two algorithms have been implemented for determining short paths: breadth-first search (BFS) and Dijkstra’s shortest path [16]. The two algorithms are very simple, differing only in choice of distance metric and running time. Both algorithms compute shortest paths from a single source node (usually the current map-frame) to all other nodes in the graph; however, BFS uses the count of nodes on the path and Dijkstra’s algorithm uses a non-negative edge weight to compute the length of each path. BFS runs in  $\mathcal{O}(N + E)$  time on a graph of  $N$  nodes and  $E$  edges. It is assumed that  $E = \mathcal{O}(N)$ , since the nodes are spatially localized and edges exist only between

nearby nodes; thus the simplified expression for the running time is  $\mathcal{O}(N)$ . Dijkstra's shortest path algorithm adopts a more sophisticated distance metric, but at a cost of a slightly higher running time. The metric is a statistical distance  $\rho$  based on the uncertainty of the transformation in *Atlas* edges, defined as the determinant of the covariance matrix of the composite transformation:

$$\begin{aligned}
T_a^c &= T_a^b \oplus T_b^c \\
\Sigma_{ac} &= J_1(T_a^b, T_b^c) \Sigma_{ab} J_1(T_a^b, T_b^c)^T + \\
&\quad J_2(T_a^b, T_b^c) \Sigma_{bc} J_2(T_a^b, T_b^c)^T \\
\rho &= \det(\Sigma_{ac})
\end{aligned}$$

where  $T_a^b$  is a coordinate transform from frame-*b* to frame-*a*,  $J_1(T_a^b, T_b^c)$  is the Jacobian of the coordinate frame composition  $T_a^b \oplus T_b^c$  with respect to  $T_a^b$ , and  $\Sigma_{ac}$  is the covariance matrix for the uncertainty in the transformation  $T_a^c$ .

The determinant of the covariance is a measure of the volume of the n-sigma hyper-ellipsoid of probability mass for a Gaussian distribution [22]. It is a good metric to use since it is also invariant to deterministic coordinate transformations. For example, if the uncertain transformation  $\hat{T}_a^c$  has the covariance  $\Sigma_{ac}$  and is composed with the deterministic transformation  $T_w^a$ , then the determinant of the resulting covariance  $\Sigma_{wc}$  equals the determinant of  $\Sigma_{ac}$ . This results from the Jacobian  $J_2$  of the transformation having a determinant of one.

$$\begin{aligned}
T_w^c &= T_w^a \oplus T_a^c \\
\Sigma_{wc} &= J_2(T_w^a, T_a^c) \Sigma_{ac} J_2(T_w^a, T_a^c)^T \\
\det(\Sigma_{wc}) &= \det(\Sigma_{ac})
\end{aligned}$$

The Dijkstra projection of an *Atlas* graph is defined with respect to a given source

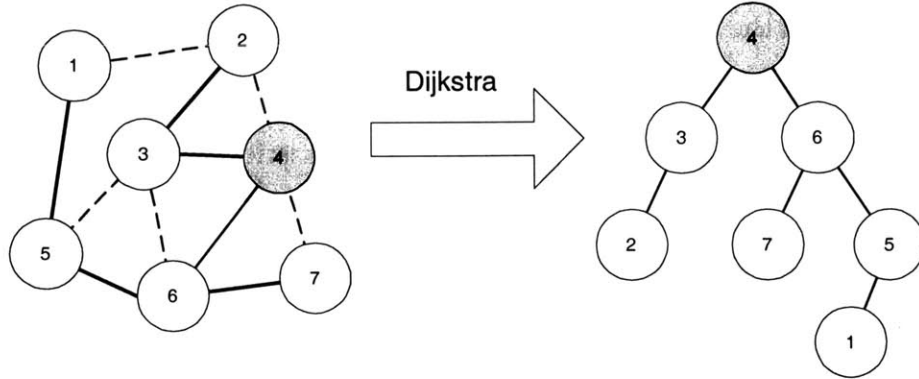


Figure 2-3: The Dijkstra projection from a given node in a graph transforms the graph into a tree with the source node as a root. Here node 4 is taken as a source. Solid lines correspond to links that are used in the tree representation. Note how in this example the uncertainty between link 4 and 7 is larger than that accrued via traversing links 4-6 and then 6-7. Hence there is no direct link between 4 and 7 in the tree.

vertex as the global arrangement of frames using compositions along Dijkstra shortest paths. This projection transforms the *Atlas* graph into a tree of transformations with the source map-frame as the root. The proximity of any map-frame to the source frame is measured as  $\rho$  computed from the compositions of transformations up the tree to the root.

The Dijkstra projection requires, in the worst case,  $\mathcal{O}(N \log N)$  time to complete; however, it need only be recomputed when the current map of the *Atlas* framework changes. (See Section 2.2.5.) The projection may also be lazily computed since the results are not necessary to process current sensor measurements. The map projection is used primarily to determine candidates for map-matching. (See Section 2.2.3.)

## 2.2.2 Genesis

The complexity of each map produced within a map-frame is bounded. The Genesis process creates new local map-frames when entering unexplored regions for which no existing map-frame can explain the sensor measurements. The genesis process adds a new vertex and edge to the *Atlas* graph. Formally, the generation of a new map-frame



$\mathcal{M}_j$  and robot pose  $\mathbf{x}_j$  via genesis is a function of an old map-frame  $\mathcal{M}_i$  and robot pose  $\mathbf{x}_i$ :

$$(\mathcal{M}_j, \mathbf{x}_j) = g(\mathcal{M}_i, \mathbf{x}_i). \quad (2.1)$$

The process of genesis encapsulated by the function  $g$  is broken down as follows:

1. The current robot pose defines the origin of a new frame. Thus, the transformation from the old to the new frame is simply the robot’s pose, expressed in the old frame, at the time the new frame is created.
2. The uncertainty of the transformation is set to the uncertainty of robot pose in the old frame.
3. The robot pose in the new frame is initialized to be coincident with the origin of the new frame.
4. By definition, the uncertainty of the robot pose in the new frame is zero. All of the uncertainty of the robot pose at the time of genesis is captured by the uncertain transformation.

### 2.2.3 Map-Matching

Genesis creates new maps to explain unexplored areas. In cyclic environments the robot will eventually revisit an area that it has already explored. Such loop closure events are automatically discovered and are used to update the connectivity, i.e. add an edge between two existing vertices in the *Atlas* graph. There are four steps to loop closing: candidate selection, map matching, match verification, and graph updating.

First, the robot must determine which of its previously explored maps are possible candidates for closing a loop. The number of candidate maps will be, at most, proportional to the integrated uncertainty around the loop. Candidate maps are determined using map projections as described in Section 2.2.1. While computing the

map projections from the current map-frame, the list of potential map-frames that may possibly overlap the current frame is computed.

Secondly, the map-frames in the potential candidate list are checked to match the current frame. This is achieved with a map-matching module that compares the structure of two maps and returns the probability that the two maps match, as well as the coordinate transformation that best aligns them. (See Section 2.3.2.)

When the map-matching module succeeds in finding a match, the consistency of the alignment must be verified. The verification procedure will be discussed in more detail in the next section. When a match is verified, the *Atlas* graph is updated by simply adding an edge to the graph. The edge contains the alignment transformation and uncertainty returned from the map-matching module. It is important to note that the focus of the *Atlas* framework does not immediately shift to the matched map. The new edge in the *Atlas* graph simply creates the potential for a traversal as described in Section 2.2.5.

## 2.2.4 Cycle Verification

False matches due to ambiguous structure in the environment present one of the most significant obstacles to correctly closing large loops. When the environment contains repetitive structure, map-match results are not unique. Furthermore, the prior uncertainty of the mapping robot before closing the loop may be too large to disambiguate the correct match. (See Figure 2-4.) Even when there is only one match visible, it is possible that the correct match simply has not yet been identified; thus, the single match is uncertain.

There are several approaches to deciding when to accept a map-match. For example, one can employ multiple hypotheses to track each possible decision branch as in Austin and Jensfelt [2]. Alternatively, one can use a method that represents multi-modal probability distributions, such as Monte-Carlo localization [15, 54] or

sum of Gaussians models [19]. Another strategy is to temporarily take the maximum likelihood decision, and then detect and fix errors later by rolling back the computation. This technique is used in Thrun *et al.* [55], where falsely matched links can be corrected recursively when large errors are detected.

Each strategy has its advantages and disadvantages. Methods that perform multiple hypothesis tracking or that employ multi-modal probability distributions are exponentially complex, and aggressive methods for reducing the number of active hypotheses (or modes) are required. Rollback methods may be unable to accurately detect the errors, and there is no definite bound on how far to rollback the computation to fix the errors. (The computation is at least proportional to the time to detect an error, and may be proportional to the size of the whole map.) Also, most data structures are not suited to efficient rollback, for example the mean and covariance of a standard Kalman filter.

The approach adopted in this thesis is to defer the decision to accept a map-match until enough information is available for verification. One potential disadvantage to deferring decisions is that enough information for a validation may never be obtained. In some situations, the mapping algorithm will fail to close loops. This occurs, for example, if an existing path is transversed with little overlap in sensor measurements from different passes.

The essence of map-match verification is to defer the validation of map-match edges in the *Atlas* graph until a “small” cycle is formed that is geometrically consistent. When closing a large loop, the prior uncertainty for the matches may be so large that multiple ambiguous matches are possible. (See Figure 2-4.) By waiting for at least one more distinct map-match, the consistency of cycles formed with the first map-match can be verified. These cycles are much smaller than the large loop’s cycle, and thus the error can be bounded around the cycle. If a cycle’s prior uncertainty is smaller than the expected distance between ambiguous matches and the transfor-

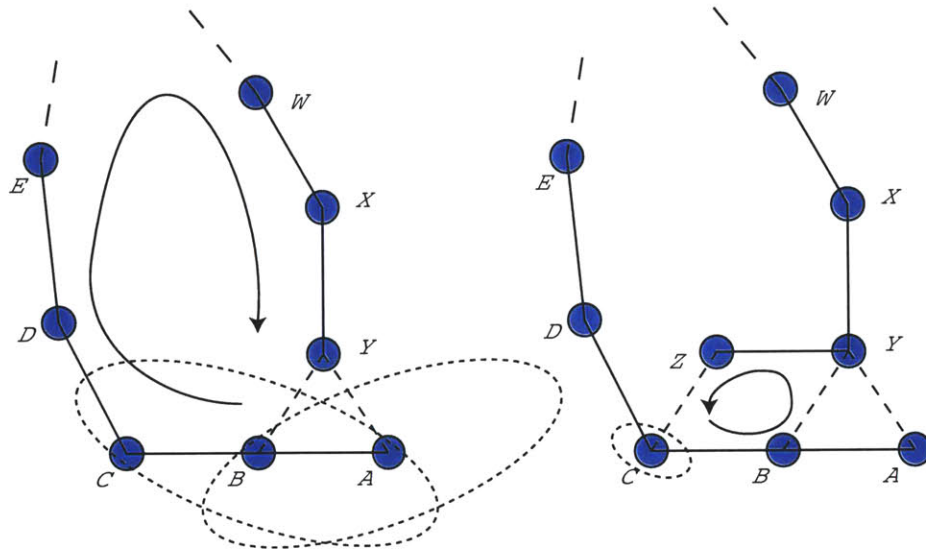


Figure 2-4: On the left is the state of the *Atlas* graph before closing the large loop  $ABCDE \dots WXY$ . There are two potential map-matches,  $YB$  and  $YA$ ; however, the prior uncertainty of the open loop transformations is too large to disambiguate the map-matches. On the right is the state of the *Atlas* graph after mapping node  $Z$  and making the match  $ZC$ . A small cycle  $CBYZC$  is now present with an uncertainty less than the distance between the ambiguities, and both edges  $ZC$  and  $YB$  are validated. The transformations about cycle  $YBAY$  are inconsistent – they agree to a transformation very different from the identity transform; hence edge  $YA$  is not validated.

mations around the cycle are consistent, confidence in correct map correspondence is justified. All non-validated map-match edges involved in the verification cycle are then validated and their effects are propagated in subsequent map projections.

Short cycles containing the current *Atlas* node are quickly discovered in expected constant-time by employing a truncated breadth-first search. A cycle is found when the breadth-first search leads to a node that has been previously visited. The sequence of nodes that make up the cycle is determined by the two distinct paths up the breadth-first search tree from the previously visited node. The breadth-first search is truncated after a maximum depth to ensure constant-time complexity.

The cycle verification step effectively increases the map coverage used to form the match. False positives are then only a problem when the scale of any environment ambiguities is larger than the coverage size of the maps in the verification cycle. Thus the likelihood of false matches is greatly reduced for a small extra computational cost.

### 2.2.5 Traversal with Competing Hypotheses

Since each map-frame covers only a localized portion of the environment, the mapping robot will not be able to match sensor measurements with a map-frame once it leaves the volume covered by the map-frame. The robot will either need to use an adjacent map-frame to explain its observations, or it will have to generate a new map if it cannot find a valid map-frame. To determine if adjacent map-frames can better explain the current sensor measurements, the framework spawns a hypothesis using the transformation in the *Atlas* edge to instantiate the robot in an adjacent map-frame. The hypothesis then processes subsequent sensor measurements to ensure that its map-frame is valid.

Thus, at any given time, there are several competing map-frame hypotheses that attempt to explain the current robot pose and sensor observations. The map-frames that best explain the current sensor measurements will have hypotheses that can

match most of the measurements to the map structure. On the other hand, invalid hypotheses will fail to match sensor measurements to existing structure and will consequently be pruned. The *Atlas* framework assesses the validity of each map-frame’s hypothesis by monitoring a performance metric  $q$  every time step. The performance metric simply reflects the likelihood that the current sensor measurements  $Z$  explain the map  $\mathcal{M}_i$  and current robot pose  $\mathbf{x}_i$ .

$$q_i = P(\mathcal{M}_i, \mathbf{x}_i | Z)$$

The *Atlas* framework maintains a set of several hypotheses that continuously determine the best map-frames to activate. Each map-frame can support only one hypothesis at a time, and the maximum number of total hypotheses  $\mathcal{H}_m$  is fixed so that overall computational requirements remain bounded. If the number of potential hypotheses is greater than  $\mathcal{H}_m$ , then they are instantiated only when existing hypotheses are terminated. In the implementations,  $\mathcal{H}_m$  is set to 5, and this limit is only reached in highly interconnected regions of the *Atlas* graph. The value of the limit has not been shown to have a significant effect on the performance since invalid hypotheses are quickly pruned.

Four types of map-frame hypotheses manage the traversal of focus about the *Atlas* graph by making transitions between adjacent map-frames. These hypotheses are labeled as *juvenile*, *mature*, *dominant* and *retired*. (See Figure 2-5 for a state transition diagram.) Retired hypotheses involve no computation and are simply used to mark inactive maps. The three other types of hypotheses, however, process sensor measurements and are evaluated with the same performance metric.

Mature hypotheses can extend their maps provided they have not reached their map capacity. Additionally, mature hypotheses spawn juvenile hypotheses in their

Table 2.1: *Atlas* hypothesis types

<b>Juvenile</b>	Initial hypotheses instantiated from neighboring map-frames. Used to test for feasible traversals.
<b>Mature</b>	Main hypotheses that are allowed to extend their maps with new features.
<b>Dominant</b>	The hypothesis with the best performance metric. This hypothesis is the focus of the framework since it serves as the root of map projections and is also used for all map-matches.
<b>Retired</b>	Hypotheses that are not actively processing sensor measurements. Mature hypotheses that fail to perform are retired.

adjacent map-frames to test the feasibility of transitioning to a neighboring map. The feasibility is based on how well the juvenile maps explain their sensor measurements without allowing the juvenile hypotheses to extend their maps.

A juvenile hypothesis can “mature” when after a short probationary period its performance metric  $q$  becomes greater than all mature hypotheses. If at the end of this probationary period a juvenile hypothesis’s quality does not warrant promotion it is simply deleted. The probationary period must be long enough to give the juvenile a chance to perform. The period should not be so long, however, such that computational resources are wasted on doomed hypotheses and the chance to instantiate better hypotheses is missed. Typically, a probationary period of 5 – 10 measurement steps is adequate.

The mature hypothesis with the best performance metric is considered the dominant hypothesis. The dominant hypothesis is used for publishing current robot pose and local features to clients of the *Atlas* framework. In other words, it is the output of the framework. Mature hypotheses that fail to perform well are saved and “retired”. A retired hypothesis may be reactivated at a later time as a juvenile.

If there is only one mature but failing hypothesis, then it is necessary to create a new hypothesis to explain current sensor data. Genesis (Section 2.2.2), the process

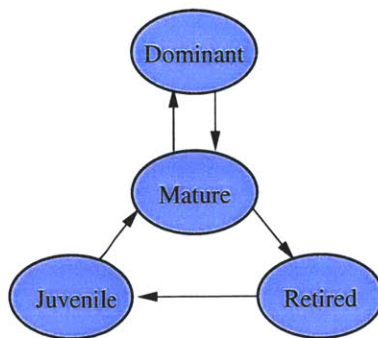


Figure 2-5: *Atlas* hypothesis state transition diagram. The dominant hypothesis is used to provide an instantaneous output from the algorithm. It is simply the most successful of all the mature hypotheses. Mature hypotheses are able to extend maps and spawn juvenile hypotheses in adjacent map-frames. When a mature hypothesis fails to describe the robot’s environment adequately (the robot has moved away, for example), it is retired. The hypothesis can be reinstated as a juvenile at some time in the future by an adjacent mature hypothesis. Juveniles are not allowed to modify their maps. If, after a probationary period, a juvenile’s map is failing to explain sensor data, then it is deleted. However, a successful juvenile is promoted to mature status.

by which new map-frames are created, instantiates a new mature hypothesis. New map-frames are needed when none of the existing hypotheses can adequately explain the sensor measurements, such as when the robot moves into an unexplored region. The new hypothesis is mature and not juvenile since it needs to immediately begin mapping new structure; juvenile hypotheses are used solely to evaluate a transitions to existing map-frames.

## 2.3 Atlas Modules

Two types of modules abstract the details for each particular implementation of the *Atlas* framework. These are the local SLAM modules, which process the sensor measurements, and the map-matching modules, which determine alignments between overlapping maps. This thesis describes several implementations using different sensors and map representations, including laser scans, sonar ranggers, and omni-



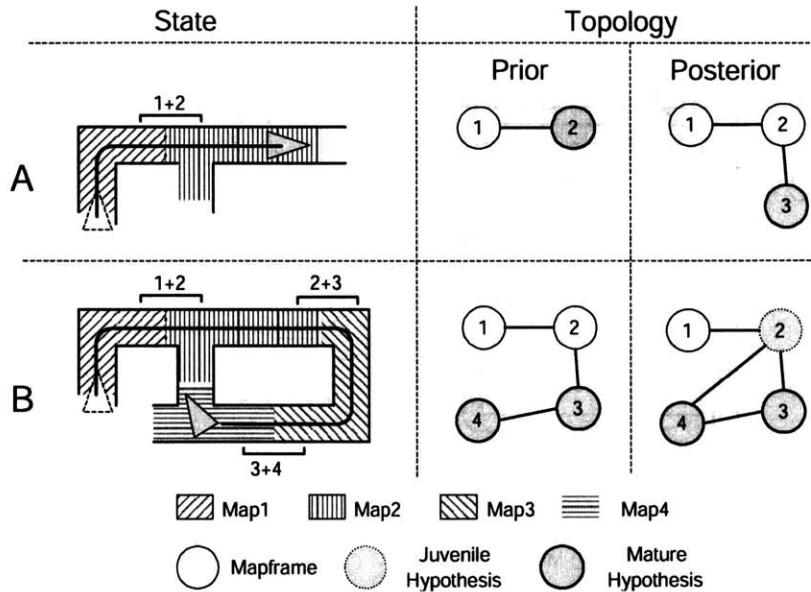


Figure 2-6: The spatial and topological evolution of an *Atlas* graph. This set of diagrams highlights key aspects of the *Atlas* framework. The environment depicted is fictional and the robot path is construed to be illustrative. The right hand column of this diagram uses shading to indicate map-frames that successfully represent the local region. Note that the maps do, in fact, overlap despite what shading suggests. This intersection is denoted by labeled brackets on the spatial diagrams. Note also that the extent of a map-frame is not defined or bounded by the area covered but by the performance metric.

**A Genesis:** *Atlas* has built two maps (1,2). Map 1 no longer explains the surroundings and is inactive. Map 2 is at capacity so genesis of Map 3 occurs. An edge is built between Maps 2 and 3. Map 3 immediately becomes the dominant hypothesis.

**B Map-Matching:** Mature hypotheses are present in both Maps 3 and 4. The Dijkstra projection suggests that Map 4 may be “close” to Map 2. The map-matching algorithm confirms this conjecture and a new link is created between Maps 4 and 2. This is loop closure. Map 2 is now adjacent to a mature Map 4, and so shortly after the edge creation a juvenile hypothesis is attached to Map 2.

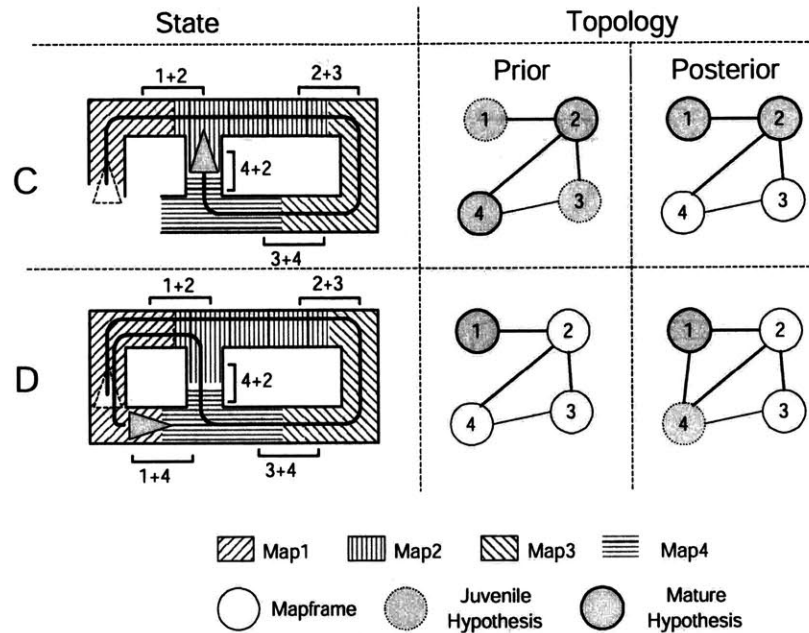


Figure 2-7: Continuation of Figure 2-6

**C Hypothesis Cull:** The robot has recently traversed into Map 2 (from Map 4) which has also become dominant. Juvenile hypotheses have been installed in the adjacent Map 1 and 3. A short time later the juvenile hypothesis in Map 3 has been terminated, since it cannot adequately explain the central corridor. The previously mature hypothesis in Map 4 has also been culled for the same reason. The juvenile hypothesis in Map 1, however, has been promoted to mature status. At this point the estimate of the transformation between Maps 1 and 4 can be updated using an observation constructed from the fact that the vehicle is simultaneously in Maps 1 and 4.

**D Loop closure:** Initially only one mature hypothesis exists (in Map 1). Unlike in case **A** genesis is not imminent (due to low vehicle uncertainty and the fact that Map 1 is not full). Instead the map-matching algorithm conjectures that Maps 1 and 4 may be adjacent. The map-matching algorithm confirms this and another new link is created. This completes the topological and spatial description of the environment.

directional video, which will be described in Chapters 3, 4, 5, and 6.

### 2.3.1 Local SLAM

*Atlas* abstracts its local mapping and navigation processing so that many different SLAM methods may be incorporated under a single framework. The framework needs to know only the module’s estimate of the current robot pose with its uncertainty, and the value of the map’s performance metric for processing the current sensor measurements. The framework does not need to know anything about the sensor measurements or the actual map representation employed; rather, it is only concerned with coordinate frames and performance metrics.

The local SLAM module must limit the complexity of its map representation such that the sensor processing time can be bounded. Local map complexity (if feature-based) is typically bounded by limiting the number of elements which are inserted into the map. After the map’s capacity has been reached, the map can still be used to process further sensor measurements; it is only restricted from growing further.

### 2.3.2 Map-Matching

*Atlas*’s map-matching module finds correspondences between two maps and returns the coordinate transform that best aligns them. The module’s design depends primarily on the type of map representation from the local SLAM module; however, most implementations follow a general form.

The map-matching process can be described as a search for a coordinate transformation that aligns two overlapping map-frames and a quantitative way to assess the resulting alignment. The uncertainty from the prior estimate for the map-frame alignment transformation (as computed by the map projection), may be very large – too large, in fact, to be able to rely on the simple strategy of nearest-neighbor feature gating for data association. Thus, the map-matching module needs to pursue

a method that is robust to large initial errors in the transformation.

In general terms, map-matching comprises two steps:

1. Determining the probability  $m_{ij}$  that the two maps  $\mathcal{M}_i$  and  $\mathcal{M}_j$  match by identifying common structure.

$$m_{ij} = P(\mathcal{M}_i \cap \mathcal{M}_j)$$

2. Producing the alignment transformation  $T_i^j$  and its uncertainty  $\Sigma_{ij}$  between maps  $\mathcal{M}_i$  and  $\mathcal{M}_j$ .

$$\{T_i^j, \Sigma_{ij}\} = P(T_i^j | \mathcal{M}_i \cap \mathcal{M}_j)$$

The exact form of  $\mathcal{M}_i \cap \mathcal{M}_j$  used for determining common structure is not dictated by the *Atlas* framework, but is left as a function to be defined in a particular implementation. For the feature-based SLAM results in this thesis, the operation  $\mathcal{M}_i \cap \mathcal{M}_j$  is defined as the search for correspondence between features in  $\mathcal{M}_i$  and  $\mathcal{M}_j$ , and  $P(\mathcal{M}_i \cap \mathcal{M}_j) = \frac{\|\mathcal{M}_i \cap \mathcal{M}_j\|}{\min(\|\mathcal{M}_i\|, \|\mathcal{M}_j\|)}$ . If a small number of features match for two maps, then the probability of a successful match is low.

Repetitive structure in the environment may cause false positive matches to be discovered in map-matching. The degree of repetition can be partially assessed by map-matching a map with itself. Only map structure elements that match uniquely within a map should be used to evaluate a match to another map.

## 2.4 Global Map Frame Optimization

We are often motivated to provide a single global map of the robot’s environment. For example, in Section 3.5 we compare an estimated map with an architectural drawing. This “globalized” representation is a result of a post-processing procedure to find a

global projection of each map-frame. In other words, we wish to find the position and orientation of each map-frame with respect to a single frame. We choose to reference all maps to the first map-frame created (frame 0).

The Dijkstra projection does this when using map-frame 0 as the source; however it uses only a minimal subset of the edges in the graph. We wish to find an optimal projection that incorporates all the edges.

When there are loops in the graph, there will be a disparity  $\nu_{i,j}$  between the transformation  $T_i^j$  stored in the *Atlas* graph edge and the transformation derived from the global poses of each frame ( $T_0^i$  and  $T_0^j$  respectively).

$$\nu_{ij} = T_i^j \oplus T_j^0 \oplus T_0^i \quad (2.2)$$

We seek to find the global arrangement  $\mathcal{T}^*$  of all  $N$  frames  $\mathcal{T} = \{T_0^1 \dots T_0^N\}$  that minimizes this error over all edges. This can be posed as a non-linear least squares optimization problem:

$$\mathcal{T}^* = \arg \min_{\mathcal{T}} \sum_{ij} \|\nu_{ij}\|^2 \quad (2.3)$$

We use the Dijkstra projection to compute the initial global arrangement, and the optimization typically converges in less than 5 iterations using the Matlab optimization toolbox.

## 2.5 Summary and Running Time Analysis

Figure 2-8 summarizes the processing performed during each cycle of the algorithm. Step 1, local map iteration, runs in constant-time, because the complexity of each local map and the number of non-retired hypotheses are bounded. Step 2, the management of hypothesis state transitions, is bounded, because it is assumed that each local map is connected to a bounded number of other local maps (bounded degree assumption).

**Atlas Algorithm:**

1. **Local SLAM Iteration:** execution of one iteration of the local SLAM module (e.g., feature-based or scan-match) in each non-retired map-frame using the new sensor measurements.
2. **Hypothesis State Transitions:** creation, promotion, and demotion of hypotheses.
3. **Map Projection Iteration:** if the dominant map changes in Step 2, restart the shortest path computation; otherwise, iterate one step of the shortest path computation and update the map-match candidate list.
4. **Map-Matching:** execute map-matching between the current dominant map and the next map-match candidate; perform cycle verification if potential match found; add valid edge.

Figure 2-8: Summary of the computation performed for each iteration of the *Atlas* algorithm.

The computational cost to run map projection to completion is  $\mathcal{O}(n \log n)$  when using Dijkstra’s shortest path algorithm and  $\mathcal{O}(n)$  when using breadth-first search (where  $n$  is the number of map-frames). In Step 3, this computation is amortized over time, expending  $\mathcal{O}(1)$  time per iteration. Furthermore, since completing the projection is not critical, there is no increase in computational complexity. New potential maps are added to the candidate list as the map projection is processed. Finally in Step 4, map-matching occurs in constant-time, because the size of the maps and the number of maps being matched are bounded.

Together the modularity and performance properties of the *Atlas* framework allow a variety of SLAM implementations to run efficaciously in large-scale environments. The following four chapters will present in detail different implementations within the *Atlas* framework.

# Chapter 3

## Atlas with 2D Laser Lines

### 3.1 Introduction

Many indoor environments are composed of large, flat surfaces such as walls and doors. These features typically appear as straight lines in laser scans, and these lines represent the prominent features a localizing robot must extract when building a map of an indoor environment. Typically, the line features are estimated from multiple range measurements which reduces the effects of sensor noise, leading to more precise navigation. The appearance of straight lines in laser scanning provides an additional robustness to non-stationary objects in the scene. These features, which should not be mapped, tend not to appear as straight lines and can thus be excluded from the representation of the environment.

The use of the *Atlas* framework for localization and mapping is demonstrated with a simple robot in a standard, indoor environment consisting of long, cyclic corridors with doors and walls. The implementation involved the creation of a feature-based map derived from line segments in laser scans of this environment. To accomplish this task, a SICK PLS scanner is mounted horizontally on a B21 robot. This laser scanner uses a spinning mirror to make 180 range measurements in a 180° field of

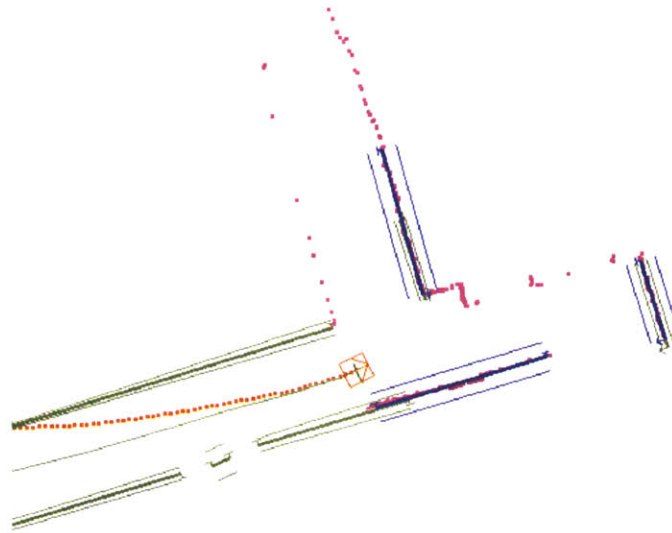


Figure 3-1: Typical Laser Scan with estimated lines and covariances

view with roughly a 10-centimeter range accuracy. (See Figure 3-1 for a typical laser scan.) The B21 robot is a cylindrical robot about 3.5 feet high with a diameter of 21 inches. It is primarily used as a testbed in indoor environments because its wheels cannot handle rough terrain.

Despite the advantages of scanned lines, certain challenges and problems emerge when relying solely upon scanned lines when attempting SLAM. Not every indoor environment has enough planar features from which to fully localize the robot; which typically occurs in long corridors where only the two parallel lines from each side are observed, and the robot cannot measure its location with respect to the long dimension of the corridor. Other sensors, such as sonar, can help in these situations because sonar can observe point features from imperfections in the wall. These additional observations would allow the robot to measure the motion along the corridor. (See Chapter 4.)



## 3.2 Line Feature Estimation

Line segments in 2D have four degrees of freedom (DOF). Two DOFs parameterize the infinite extent line, and the remaining two parameterize the end points of the segment on the line. It is difficult to estimate the exact position of the endpoints of lines in laser scans because the end points are often occluded and have minimal support from range measurements. Therefore, the endpoints are unreliable to use for localization. The geometric line on which the segment lies, however, is a good feature to localize from. Re-observations of a line can measure the translation error in the direction perpendicular to the line as well as the heading error of the robot. Consequently, when there is a set of multiple non-parallel lines, the robot can be accurately localized.

The line features in this *Atlas* SLAM module implementation are parameterized in polar form by  $\rho$  and  $\phi$ . The perpendicular distance of the line from the coordinate origin is  $\rho$ , whereas the angle of the normal to the line is  $\phi$ . A point  $(x, y)$  is on the line  $L(\rho, \phi)$  if the residual function  $h_{\text{line}}(\rho, \phi, x, y)$  equals zero.

$$h_{\text{line}}(\rho, \phi, x, y) = x \cos(\phi) + y \sin(\phi) - \rho \quad (3.1)$$

The endpoints are parameterized as the signed distances of the endpoints from the point on the line that is closest to the origin. (See Figure 3-2.)

A line  $L(\rho, \phi)$  can be fitted to a list of points  $\{x_i, y_i\}$ , by minimizing the squared distance of each point from the line. It is important to note that the residual function  $h_{\text{line}}$  in equation 3.1 computes the distance of the point from the line.

$$L(\rho, \phi) = \arg \min_{\rho, \phi} \sum_i h_{\text{line}}(\rho, \phi, x_i, y_i)^2 \quad (3.2)$$

Since the residual function is non-linear in the line parameters, a least squares solution

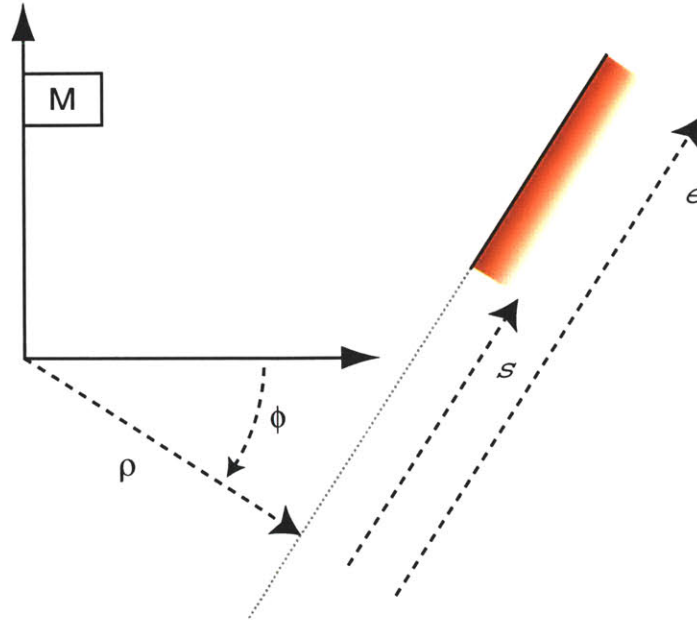


Figure 3-2: The line is parameterized by  $\rho$  and  $\phi$ , which describe the perpendicular distance of the line from the origin and the angle of the normal. The endpoints ( $s, e$ ) are parameterized by their signed distance from the line's closest approach to the origin.

cannot be used. There is, however, a closed-form solution for the above minimization. This solution is found by finding the centroid of the points and the angle of the axis of minimum inertia. Hence, the first and second moments of the points are initially computed.

$$\begin{aligned}
 x_c &= \frac{1}{N} \sum_i x_i \\
 y_c &= \frac{1}{N} \sum_i y_i \\
 s_{xx} &= \sum_i (x_i - x_c)^2 \\
 s_{xy} &= \sum_i (x_i - x_c)(y_i - y_c) \\
 s_{yy} &= \sum_i (y_i - y_c)^2
 \end{aligned}$$

Then the line parameters are determined as:

$$\begin{aligned}\phi &= \frac{1}{2} \arctan \left( \frac{2s_{xy}}{s_{xx} - s_{yy}} \right) - \frac{\pi}{2} \\ \rho &= x_c \cos(\phi) + y_c \sin(\phi)\end{aligned}$$

The  $\frac{\pi}{2}$  term transforms the angle of the axis of minimum inertia into the angle of the line normal.

The covariance of the line parameters can be estimated from the Jacobian  $\mathbf{H}$  of the residual equation 3.1.

$$\begin{aligned}\mathbf{H}_i &= \frac{\partial h_{\text{line}}(x_i, y_i, \rho, \phi)}{\partial \rho, \phi} \\ &= \begin{bmatrix} -1 & y_i \cos(\phi) - x_i \sin(\phi) \end{bmatrix}\end{aligned}$$

Assuming that the line residual of each point is independently identically distributed Gaussian noise with a covariance of  $\sigma$ , then the covariance  $\Sigma_{\rho\phi}$  of the line parameters can be computed.

$$\Sigma_{\rho\phi} = \sigma \left( \sum_i \mathbf{H}_i^T \mathbf{H}_i \right)^{-1} \quad (3.3)$$

The covariance can be used to draw 1-sigma error bounds on line segments. Unlike the elliptical covariance contour of a point, the error contour for a line is depicted as a hyperbola. The hyperbola can be parameterized as a perpendicular deviation to either side of the line.

$$d(\lambda) = \pm \sqrt{\lambda^2 c_{\phi\phi} - 2\lambda c_{\rho\phi} + c_{\rho\rho}}$$

where  $\lambda$  parameterizes the signed distance along the line from the point closest to the origin, and  $c_{\rho\rho}$ ,  $c_{\rho\phi}$ , and  $c_{\phi\phi}$  are the components of the covariance matrix  $\Sigma_{\rho\phi}$ . These

error bounds are also depicted as hyperbolas about the line segments in Figure 3-1.

### 3.2.1 Line Splitting and Fitting

Given a single laser scan, a low-level algorithm must segment the range measurements into line segments and estimate the line parameters and corresponding uncertainties of each segment. This task is accomplished by recursively fitting and splitting the range measurements on a contour formed by the scan until the segment adequately fits a line or is too short.

There are two basic thresholds used in this process. The first is the tolerance for the maximum distance a point can be from a segmented line,  $D_{\max}$ . This distance should be about 2 standard deviations of the range measurement noise so that the measurement noise does not result in over-splitting of line segments. The second is the minimum number of points required on each line segment,  $N_{\min}$ . When there are too few points on a line, the fitted line parameters are not robust enough with the sensor noise on each range measurement, and hence the points are not used.

The split and fit algorithm starts out by treating the sequence of range measurements  $r_i$  from the corresponding angles  $\theta_i$  as a contour  $\mathcal{C}_{1,N}$  in space with a simple polar to Cartesian coordinate transform.

$$\begin{aligned}(x_i, y_i) &= (r_i \cos(\theta_i), r_i \sin(\theta_i)) \\ \mathcal{C}_{1,N} &= \{x_i, y_i\} \quad i \in 1 \cdots N\end{aligned}$$

where  $\theta_i$  is the scan angle for each range measurement  $r_i$ .

The line  $L$  formed by the endpoints of the contour  $(x_1, y_1)$  and  $(x_N, y_N)$  and the distance  $d_i$  of every point on the contour  $\mathcal{C}_{1,N}$  from the line  $L$  is computed. If the maximum distance  $d_j$  is less than  $D_{\max}$ , and  $N$  is greater than or equal to  $N_{\min}$ , a line segment is found. The parameters of this line segment are then estimated using all

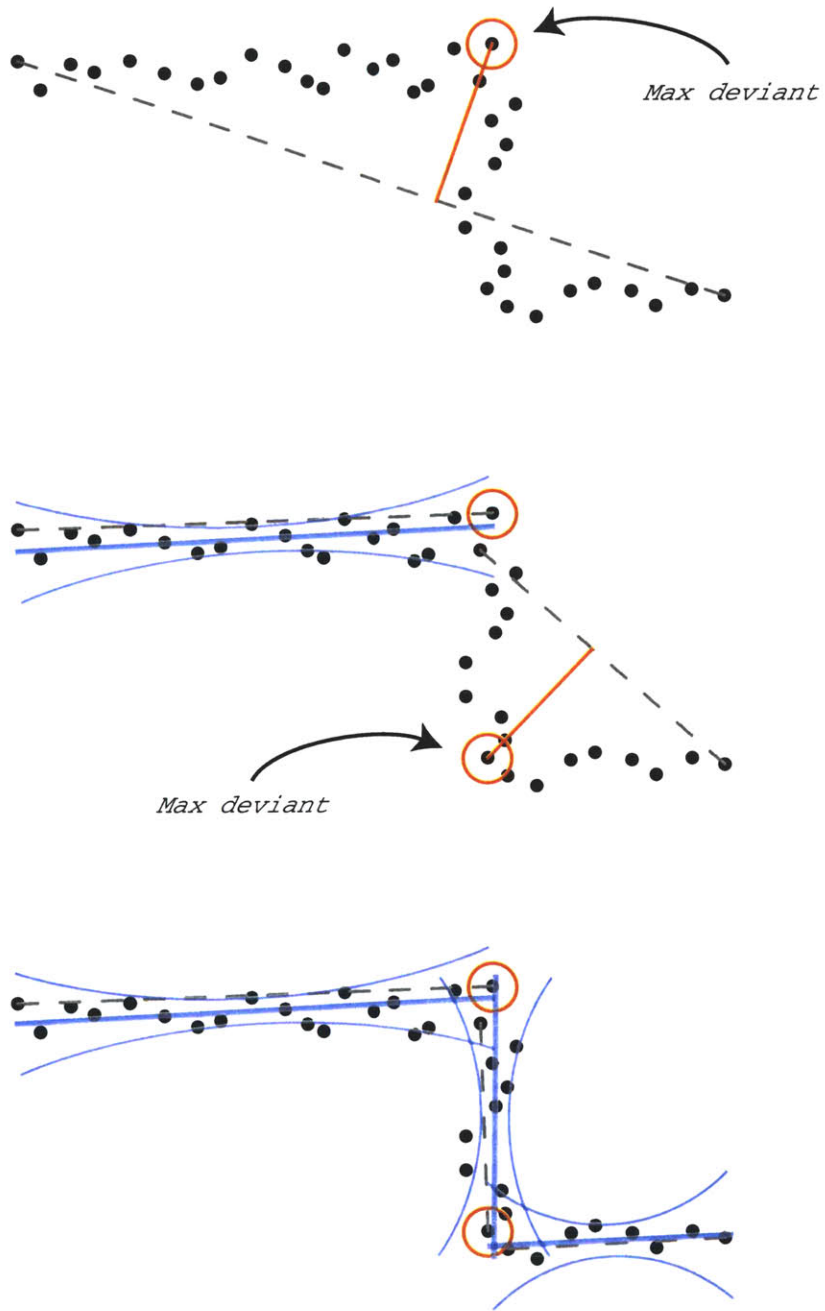


Figure 3-3: Three steps of the line splitting process are depicted from top to bottom. The line between the first and last points in the segment is used to find the max deviant. The max deviant is the best place to split the set of points if they do not accurately fit on a line. The line parameters and covariance are estimated from sets of points with no deviant larger than  $D_{\max}$ .

the measurements in the contour by minimizing the squared distance of each point from the line as in equation 3.2. The covariance of each line segment is also computed using equation 3.3, as this will be necessary for incorporating measurements into the SLAM Kalman Filter. (See Section 3.3.)

If the length of the segment is less than  $N_{\min}$ , the segment is discarded. Otherwise, when the max deviant  $d_j$  is larger than  $D_{\max}$ , the contour is split into two pieces at the max deviant. The procedure recurses on the sub-contours  $\mathcal{C}_{1,j}$  and  $\mathcal{C}_{j+1,N}$  until all the line segments have been found. (See Figure 3-3.)

### 3.3 Landmark Kalman Filter

The laser line local SLAM module is based upon an Extended Kalman Filter (EKF). The basics of the EKF in SLAM are described in further detail in [17].

The three main models that compose the SLAM module are the sensor model, the map representation model, and the robot model. The sensor model is represented by the line features extracted from the laser scans. Similarly, the map representation is simply a collection of line segments parameterized relative to the map-frame's coordinate origin. Finally, the robot model contains the position and orientation of the robot relative to the coordinate origin, as well as the dynamics that describe how the robot moves.

#### 3.3.1 Odometry Model

The B21 robot uses a synchro-drive with four wheels that can turn while holding the robot in place. The software on the robot internally integrates the odometer encoders on the wheels and reports the position and angle of the robot relative to where the robot was at the last reset. The reported pose of the robot is corrupted by integrated errors due to wheel slippage and skidding, and thus is dependent on the amount of

turning and the speed of the robot. It is unnecessary to model all the physics of the 4-wheeled system; instead, it is simpler to model the robot's drive mechanism with a two-wheeled odometry model.

The two-wheeled odometry model consists of two wheels at a fixed separation on a single axle. Each wheel has an odometer that measures its turning motion. The robot's frame of reference is set midway between the two wheels, with the  $x$ -axis running along the axle towards the right wheel, and the  $y$ -axis pointing forward. The odometers are polled periodically (at about 5 Hz on the B21 robot), and the differential motion  $\Delta \mathbf{x}_v = [\Delta x \ \Delta y \ \Delta \theta]^T$  can be computed from the left and right measurements  $\Delta L$  and  $\Delta R$ :

$$\begin{bmatrix} \Delta x \\ \Delta y \\ \Delta \theta \end{bmatrix} = \begin{bmatrix} 0 \\ \frac{s_R \Delta R + s_L \Delta L}{2} \\ \frac{s_R \Delta R - s_L \Delta L}{B} \end{bmatrix} \quad (3.4)$$

where  $s_R$  and  $s_L$  are the right and left wheel scale factors in meters per click,  $B$  is the length of the wheel baseline, and  $\Delta \theta$  is in units of radians.

The current robot pose  $\mathbf{x}_v$  is maintained in the Kalman state vector.

$$\mathbf{x}_v = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$$

At every time step, the current robot pose is propagated by the function  $\mathbf{x}_v(i) = \mathbf{f}_v(\mathbf{x}_v(i-1), \Delta \mathbf{x}_v(i))$ .

$$\mathbf{f}_v(\mathbf{x}_v(t), \Delta \mathbf{x}_v(t + \Delta t)) = \mathbf{x}_v(t) \oplus \Delta \mathbf{x}_v(t + \Delta t) \quad (3.5)$$

The propagation function uses the motion  $\Delta \mathbf{x}_v(i)$  computed from the odometry mea-

surements in the body frame of the robot and composes the transformation with the previous pose of the robot  $\mathbf{x}_v(i-1)$  to obtain the predicted pose  $\mathbf{x}_v(i)$ .

The Extended Kalman filter also requires the Jacobian  $\mathbf{J}_{\mathbf{x}_v}$  of the propagation function  $\mathbf{f}_v(\mathbf{x}_v, \Delta\mathbf{x}_v)$  with respect to the state  $\mathbf{x}_v$ .

$$\mathbf{J}_{\mathbf{x}_v} = \frac{\partial \mathbf{f}_v}{\partial \mathbf{x}_v} = \mathbf{J}_1(\mathbf{x}_v, \Delta\mathbf{x}_v) \quad (3.6)$$

Where  $\mathbf{J}_1(\mathbf{a}, \mathbf{b})$  is the Jacobian of  $\mathbf{a} \oplus \mathbf{b}$  with respect to  $\mathbf{a}$ . This Jacobian is used to transform the state error covariance at each time step.

The propagation noise source is modeled to arise from uncertainty in the wheel baseline and the wheel scale factors, with a slip factor perpendicular to the wheels. This correctly models the fact that there is no error when the robot is stationary, and that the error increases when moving faster or when turning.

$$\begin{bmatrix} \Delta x \\ \Delta y \\ \Delta \theta \end{bmatrix} = \begin{bmatrix} \delta \xi \left( \frac{s_R(1+\delta s_R)\Delta R + s_L(1+\delta s_L)\Delta L}{2} \right) \\ \frac{s_R(1+\delta s_R)\Delta R + s_L(1+\delta s_L)\Delta L}{2} \\ \frac{s_R(1+\delta s_R)\Delta R - s_L(1+\delta s_L)\Delta L}{B+\delta B} \end{bmatrix} \quad (3.7)$$

The noise variables  $\delta \xi$ ,  $\delta s_R$ ,  $\delta s_L$  and  $\delta B$  are modeled as zero mean Gaussian distributed random variables with standard deviations of  $\sigma_\xi$ ,  $\sigma_{s_R}$ ,  $\sigma_{s_L}$  and  $\sigma_B$ , respectively.

Since the noise model is non-linear, at each time step the odometry model is linearized. The Kalman Filter process noise  $\mathbf{Q}_v$  is computed by transforming the independent noise covariance on the slip, scale factors and wheel baseline errors with the Jacobians ( $\mathbf{J}_{\text{odo}}$ ) of Equation 3.7 with respect to noise variables and the Jacobian



( $\mathbf{J}_{\text{motion}}$ ) of Equation 3.5 with respect to the motion.

$$\mathbf{J}_{\text{odo}} = \begin{bmatrix} \Delta y & 0 & 0 & 0 \\ 0 & \frac{s_R \Delta R}{2} & \frac{s_L \Delta L}{2} & 0 \\ 0 & \frac{s_R \Delta R}{B} & -\frac{s_L \Delta L}{B} & -\frac{\Delta \theta}{B} \end{bmatrix}$$

$$\mathbf{J}_{\text{motion}} = \mathbf{J}_2(\mathbf{x}_v, \Delta \mathbf{x}_v)$$

$$\mathbf{Q}_v = \mathbf{J}_{\text{motion}} \mathbf{J}_{\text{odo}} \begin{bmatrix} \sigma \xi \\ \sigma s_R \\ \sigma s_L \\ \sigma B \end{bmatrix} \mathbf{J}_{\text{odo}}^T \mathbf{J}_{\text{motion}}^T$$

Where  $\mathbf{J}_2(\mathbf{a}, \mathbf{b})$  is the Jacobian of  $\mathbf{a} \oplus \mathbf{b}$  with respect to  $\mathbf{b}$ .

The B21 robot does not directly report the right and left wheel encoder measurements  $\Delta R$  and  $\Delta L$ . However, they can be easily computed by inverting Equation 3.4.

$$\Delta R = (\Delta y + \Delta \theta \frac{B}{2}) / s_R$$

$$\Delta L = (\Delta y - \Delta \theta \frac{B}{2}) / s_L$$

The estimated odometer values are subsequently substituted into the odometry noise model to compute the process noise matrix  $\mathbf{Q}_v$ .

### 3.3.2 Advanced Odometry Model

The odometry model in the previous section is not always sufficient to model the errors. The model in Equation 3.7 does not account for biases that cause the robot to drift significantly to the left or right for longer periods of time. The robot will often veer to one side because one wheel gets less traction than the other. A simple way

to model this phenomenon is to have a bias variable  $\lambda$  that makes one wheel's scale factor larger than the other:

$$s_R \rightarrow s_R(1 + \lambda) \quad (3.8)$$

$$s_L \rightarrow s_L(1 - \lambda) \quad (3.9)$$

The bias factor  $\lambda$  changes slowly as the robot moves over different surfaces and thus cannot be calibrated ahead of time. Instead the robot state in the Kalman filter is extended with this bias parameter, and it is estimated online.

The estimated bias factor is employed to correct the reported odometer motion by plugging Equations 3.8 and 3.9 into the odometry model, Equation 3.4, and simplifying.

$$\begin{bmatrix} \Delta x \\ \Delta y \\ \Delta \theta \end{bmatrix} = \begin{bmatrix} 0 \\ \frac{s_R \Delta R + s_L \Delta L}{2} + \lambda \frac{s_R \Delta R - s_L \Delta L}{2} \\ \frac{s_R \Delta R - s_L \Delta L}{B} + \lambda \frac{s_R \Delta R + s_L \Delta L}{B} \end{bmatrix}$$

To model the fact the the odometry bias can change slowly over time, the bias treated as a first-order Markov random variable with a time constant  $\tau_\lambda$  and standard deviation  $\sigma_\lambda$ . The Markov model induces the propagation:

$$\lambda(t + \delta t) = \lambda(t)e^{(\delta t/\tau_\lambda)} + \delta\lambda$$

Where  $\delta\lambda$  is zero mean Gaussian distributed noise with standard deviation of  $\sigma_\lambda$ .

The addition of the odometry scale bias to the Kalman state vector also adds

elements to the state propagation Jacobian in Equation 3.6.

$$\frac{\partial \mathbf{f}_v}{\partial \lambda} = \begin{bmatrix} -\frac{s_R \Delta R - s_L \Delta L}{2} \sin(\theta) \\ \frac{s_R \Delta R - s_L \Delta L}{2} \cos(\theta) \\ \frac{s_R \Delta R + s_L \Delta L}{B} \end{bmatrix}$$

### 3.3.3 Measurement Models and Processing Summary

For each new laser scan the local SLAM module makes several computations. First the laser segmenter processes the current scan to extract the visible line segments as described in Section 3.2.1. Secondly the Kalman filter is propagated to the current time step using the odometry models depicted in Section 3.3.1 and 3.3.2. Next the current view of laser lines is predicted from the Kalman state, and the measured lines are matched to the mapped lines. The matched lines are used to update the Kalman filter to simultaneously correct for the growth of localization errors from propagating the odometry model and reduce the uncertainty of the mapped lines. Any measured lines which were not matched to the map are tracked as preliminary lines. If any preliminary lines have been tracked for enough time steps they are initialized as new features into the map. The final step is to report the performance metric to the *Atlas* framework. The performance metric is based on the number of matched lines and the number of newly visible lines.

#### Line predicting

After the propagation step of the Kalman filter and after the lines have been extracted from the laser scan, the current view of mapped lines must be represented with respect to the current robot pose. Each mapped line  $\mathbf{x}_{L_i} = [\rho \ \phi]^T$  is transformed by the

function  $\mathbf{h}_L(\mathbf{x}_v, \mathbf{x}_{L_i})$  to the predicted line  $L_{v_i} = [\rho_v \ \phi_v]^T$ .

$$L_{v_i} = \mathbf{h}_L(\mathbf{x}_v, \mathbf{x}_{L_i}) \quad (3.10)$$

$$\rho_v = \rho - x \cos(\phi) - y \sin(\phi) \quad (3.11)$$

$$\phi_v = \phi - \theta \quad (3.12)$$

The Jacobian of the prediction function  $\nabla \mathbf{h}_L$  is used to extract the covariance of the predicted line from the Kalman filter state covariance  $\mathbf{P}$ .

$$\begin{aligned} \nabla \mathbf{h}_L &= \begin{bmatrix} \mathbf{H}_v & \cdots & \mathbf{H}_{L_i} & \cdots \end{bmatrix} \\ &= \begin{bmatrix} -\cos(\phi) & -\sin(\phi) & 0 & \cdots & 1 & x \sin(\phi) - y \cos(\phi) & \cdots \\ 0 & 0 & -1 & \cdots & 0 & 1 & \cdots \end{bmatrix} \end{aligned} \quad (3.13)$$

where  $\mathbf{H}_v$  is the derivative of the prediction function with respect to the current robot pose, and  $\mathbf{H}_{L_i}$  is the derivative with respect to the  $i$ th mapped line. Note that all the other matrix elements of the Jacobian are zero. Thus the predicted line covariance  $\Sigma_{L_{v_i}}$  is simply formed as follows:

$$\Sigma_{L_{v_i}} = \nabla \mathbf{h}_L \mathbf{P} \nabla \mathbf{h}_L^T \quad (3.14)$$

## Line matching

Line matching is straightforward since the map has been projected into the current robot view. Nearest-neighbor gating is used to determine the best matches of extracted lines to map lines. It is not a problem if more than one extracted line matches a map line since the mapped lines may be longer. The converse, however, is not permissible because the information from the measured lines cannot be used twice. The metric for determining nearness of lines is based on the Mahalanobis

distance  $d_z$  of the line parameters:

$$d_z = \begin{bmatrix} (\rho_z - \rho_v) & (\phi_z - \phi_v) \end{bmatrix} (\Sigma_{L_z} + \Sigma_{L_v})^{-1} \begin{bmatrix} (\rho_z - \rho_v) \\ (\phi_z - \phi_v) \end{bmatrix}$$

where the subscript  $z$  marks the measured line extracted from the current view, and  $v$  marks the predicted line and covariance as in Equations 3.10 and 3.14.

The lines' endpoints are also checked to make sure that the lines sufficiently overlap. The endpoints are parameterized by the signed distance along the line where 0 is the point on the line closest to the origin of the map-frame. The overlap  $o$  and the coverage  $c$  between two lines  $a$  and  $b$  with endpoints  $(s_a, e_a)$  and  $(s_b, e_b)$  are initially computed as follows:

$$\begin{aligned} o &= \min(e_a, e_b) - \max(s_a, s_b) \\ c &= \max(e_a, e_b) - \min(s_a, s_b) \end{aligned}$$

where  $s_a < e_a$  and  $s_b < e_b$ . It makes sense to compare the overlap  $o$  with the coverage  $c$  of the two lines such that the overlap fraction  $o_p$  can be used instead of a metric distance.

$$o_p = \frac{o}{c}$$

The percentage overlap  $o_p$  is simpler to threshold than the plain overlap  $o$  since it does not depend on the units used to represent the endpoint distances.

Both a near threshold  $\sigma_{\text{near}}$  and a far threshold  $\sigma_{\text{far}}$  are used to judge the Mahalanobis distance  $d_z$  of the matching. The threshold values are based on the chi-squared distribution with two degrees of freedom. Thus to ensure a valid match with a probability of 0.999, the threshold  $\sigma_{\text{near}}$  is set to 13.8. In this implementation, the threshold

$\sigma_{\text{far}}$  is set to 36, which corresponds to a statistical distance 6 standard deviations away.

Only measured lines that are marked as “far” from every mapped line are considered new lines in the view. This is necessary to limit the clutter that would otherwise be present due to the measurement noise and a tight near threshold. This strategy also partially prevents multiple lines from being mapped into the Kalman state by the same physical line.

Usually every mapped line must be compared with every measured line to find the matches. The simple pseudocode for the algorithm to match the set of measured lines  $L_z$  with mapped lines projected into the current view  $L_v$  follows:

```
MATCH-LINES-SLOW( $L_z, L_v$ )
1   $m \leftarrow \{\}$ 
2  for  $i \leftarrow 1$  to length [ $L_z$ ]
3  do for  $j \leftarrow 1$  to length [ $L_v$ ]
4      do if MATCH( $L_z[i], L_v[j]$ )
5          then  $m \leftarrow \{m, \{i, j\}\}$ 
6  return  $m$ 
```

The process of line matching can be sped up by sorting the lines. The lines are sorted increasingly by their distance from the origin of the coordinate frame. Since the lines are sorted, when a unmatched mapped line is further from the origin than a measured line, no other subsequent mapped line will match the measured one. Likewise, the loop to match the next measured line can be sped up, because the loop may start with the first mapped line that was close to the previously measured line.

```

MATCH-LINES-FAST( $L_z, L_v$ )
1   $m \leftarrow \{\}$ 
2  SORT( $L_z$ )
3  SORT( $L_v$ )
4   $j_0 \leftarrow 1$ 
5  for  $i \leftarrow 1$  to length [ $L_z$ ]
6  do for  $j \leftarrow j_0$  to length [ $L_v$ ]
7      do if LESS-THAN( $L_z[i], L_v[j]$ )
8          then break
9          if GREATER-THAN( $L_z[i], L_v[j]$ )
10             then  $j_0 \leftarrow j$ 
11             if MATCH( $L_z[i], L_v[j]$ )
12                 then  $m \leftarrow \{m, \{i, j\}\}$ 
13  return  $m$ 

```

The comparison for determining whether a line is greater or less than another must take into account the uncertainty of the line. An assumption is made on the maximum allowed uncertainty of the lines'  $\rho$  parameters. Then only lines whose  $\rho$  difference is greater than the maximum allowed uncertainty  $\Delta\rho_{\max}$  are considered either greater than or less than the other. Otherwise, the lines are neither greater or less than with respect to each other.

```

LESS-THAN( $L_a, L_b$ )
1   $\Delta\rho \leftarrow (\rho_a - \rho_b)$ 
2  if  $\Delta\rho < -\Delta\rho_{\max}$ 
3      then return true
4      else return false

```

```

GREATER-THAN( $L_a, L_b$ )
1   $\Delta\rho \leftarrow (\rho_a - \rho_b)$ 
2  if  $\Delta\rho > \Delta\rho_{\max}$ 
3      then return true
4      else return false

```

This procedure decreases the running time complexity of the simple matching algorithm from  $\mathcal{O}(NM)$  to that of  $\mathcal{O}(N \log N + M \log M)$ , where  $N$  and  $M$  are the number of mapped lines and measured lines, respectively. The running time complexity of the inner loop is  $\mathcal{O}(1)$ , since only a constant number of map lines will be neither greater than or less than a measured line. Thus the matching takes  $\mathcal{O}(M)$  time, and consequently the total running time is dominated by the initial sorting of the lines.

It is important to note that since the capacity of the local map is limited in the *Atlas* framework, the number of features to match does not grow without bound. Hence the matching procedure is considered to take  $\mathcal{O}(1)$  time, regardless of which procedure is implemented. Nevertheless, the fast line match procedure improves the running time constants.

### Line updating

The measurement model relates the matched map line with the measured line via the current robot pose. The parameter residual of the predicted line, via Equation 3.10, with the measured line is used to update the Kalman state. The prediction Jacobian, Equation 3.13 is used to form the Kalman gain matrix and to update the state covariance. Each matched line is updated separately since the uncertainties of measurements are independent from one another, which reduces the size of the matrices used in the update.



Only the line parameters, and not the endpoints, are used to update the Kalman filter state and covariance. The endpoints of the lines are not reliable in updating the robot pose. The endpoints are often formed by occlusions and their errors are poorly modeled by the Gaussian assumptions of a Kalman filter. However, the measured endpoints are used to increase the extent of the mapped lines. This is necessary since line overlap is used during matching.

### **Line tracking**

A set of preliminary lines  $L_p$  is kept in the current view to track new lines that were not matched. The new lines are those measured lines classified as “far” from every mapped line. The preliminary lines are continually propagated into the current view; thus the robot’s motion uncertainty must be integrated into the lines’ uncertainty. The propagation function is identical to Equation 3.10, except that the robot’s motion  $\Delta \mathbf{x}_v$  is used in place of the robot’s pose.

The preliminary lines are then matched with the new lines. The same function to match the predicted map lines as described above is used here. New preliminary line tracks are started from the leftover lines, if any, that are “far” from all the other lines.

### **Line mapping**

When a line has been tracked for several time steps, the algorithm is assured that the line is not a spurious line extracted from a noisy scan. Consequently, a preliminary line that has been tracked for a minimum number of time steps is mapped as a new feature, with corresponding Kalman filter state. The mapping process uses a function  $\mathbf{g}()$  that maps the measured line  $L_z$  via the current robot pose  $\mathbf{x}_v$  to the new feature

state  $L$ .

$$\begin{aligned} L &= \mathbf{g}(\mathbf{x}_v, L_z) \\ \rho &= \rho_z + x \cos(\phi_z + \theta) + y \sin(\phi_z + \theta) \\ \phi &= \phi_z + \theta \end{aligned}$$

The Jacobians of the mapping function with respect to the Kalman state  $\nabla \mathbf{g}_x$  and with respect to the measurement  $\nabla \mathbf{g}_z$  are used to extend the Kalman filter covariance.

$$\begin{aligned} \nabla \mathbf{g}_x &= \begin{bmatrix} \cos(\phi) & \sin(\phi) & -x \sin(\phi) + y \cos(\phi) & \cdots \\ 0 & 0 & 1 & \cdots \end{bmatrix} \\ \nabla \mathbf{g}_z &= \begin{bmatrix} 1 & -x \sin(\phi) + y \cos(\phi) \\ 0 & 1 \end{bmatrix} \end{aligned}$$

where all elements in  $\nabla \mathbf{g}_x$  corresponding to states other than the current robot pose are zero.

Only the most recent measurement of the line is mapped into the new line state, since it is not corrupted by the propagation noise of the robot's motion. New lines can be mapped into the state only if the map's capacity has not been reached.

### Performance metric

The final step in each SLAM iteration is to report the performance metric. The performance metric  $q$  indicates how well the current local SLAM module is performing and is used by the *Atlas* framework to decide which of several map hypotheses to use. The metric ranges between the values of 0 and 1, which indicate the worst and best performances, respectively.

There are three parts to the performance metric, which describe the hypothesis' ability to explain the current sensor measurements ( $q_{\text{meas}}$ ), its ability to explain the

current robot pose ( $q_{\text{robot}}$ ), and how well conditioned the current robot pose is ( $q_{\text{cond}}$ ). These three metrics all range between 0 and 1, and the composite metric is their product.

$$q = q_{\text{meas}} \cdot q_{\text{robot}} \cdot q_{\text{cond}}$$

The first part of the metric,  $q_{\text{meas}}$ , is composed by comparing the probability density of the residuals of the matched features to the maximum possible probability density of the measurements.

$$q_{\text{meas}} = \frac{\sum_i p(L_{z_i} - L_{v_i}; \mathbf{h}_{L_i} \mathbf{P} \mathbf{h}_{L_i}^T + \mathbf{R}_i)}{\sum_i p(0; \mathbf{R}_i)} \quad (3.15)$$

where  $p(x; \Sigma)$  is the  $m$ -dimensional Gaussian probability density function:

$$p(x; \Sigma) = \frac{1}{(2\pi)^{\frac{m}{2}} \det(\Sigma)^{\frac{1}{2}}} \exp \left\{ -\frac{1}{2} x^T \Sigma^{-1} x \right\}$$

The numerator of Equation 3.15 is the sum of the Kalman filter innovation probability densities  $P(\mathbf{x}, Z)$ , and the denominator is the sum of the measurement model densities  $P(Z | \mathbf{x})$  evaluated at their modes. Measured lines with no matches to feature states have infinite residuals, thus they contribute zero to the sum in the numerator. Interestingly, this expression reduces to  $P(\mathbf{x})$  which describes the probability density of the Kalman filter state, as opposed to  $P(\mathbf{x} | Z)$ , which is the entity that the Kalman filter maintains with the state vector and covariance matrix.

Note that  $q_{\text{meas}}$  is not sufficient by itself as the performance metric, since it fails to describe how certain the Kalman filter is of the current robot pose. For example, even when lines are mapped for very uncertain robot poses, the predicted lines can still have a small uncertainty and residual when matching measured lines.

The Kalman filter, however, does maintain a measure of how well the robot is

localized, which is used to form the second component,  $q_{\text{robot}}$ . The trick is expressing the measure of robot uncertainty in the covariance of the Kalman filter as a probability between 0 and 1. A measure of the typical performance expected for the robot  $\mathbf{P}_{xx}^*$  is employed to compare with the current uncertainty of the robot  $\mathbf{P}_{xx}$ .

$$\begin{aligned}
 q_{\text{robot}} &= \frac{p(0; \mathbf{P}_{xx})}{p(0; \mathbf{P}_{xx}) + p(0; \mathbf{P}_{xx}^*)} \\
 &= \frac{1}{1 + \sqrt{\frac{\det(\mathbf{P}_{xx})}{\det(\mathbf{P}_{xx}^*)}}}
 \end{aligned} \tag{3.16}$$

Thus when the robot location is very uncertain, the determinant of its covariance increases and the metric goes to zero. Likewise when the robot location is very certain, its covariance is small and the metric tend towards one.

The third component of the performance metric,  $q_{\text{cond}}$ , measures how well the robot's position covariance is conditioned. Measuring the condition of the robots position is necessary since partially observable features are used to update the robot. When there is only one line to measure, or when all measured lines are parallel, then the components of the robot's pose uncertainty in the direction of the lines will grow without bound. This leads to highly eccentric covariance ellipses, which is undesirable. The condition number of the covariance is the ratio of the maximum to the minimum eigenvalue. The lengths of the symmetrical axes of the covariance ellipse are proportional to the square roots of the eigenvalues. Thus the following metric:

$$q_{\text{cond}} = \frac{\text{mineig}(\mathbf{P}_p)}{\text{maxeig}(\mathbf{P}_p)}$$

indicates the eccentricity of the robot's positional covariance, where  $\mathbf{P}_p$  is the Kalman filter covariance corresponding to the positional components of the robot's pose. The condition of the robot's orientation is not necessary, since in this model it is

1-dimensional and fully observable from a single line. When the covariance of the position is well-conditioned, the eigenvalues have about the same magnitude and the metric  $q_{\text{cond}}$  tends to one. Otherwise, when the covariance becomes highly elliptical, the metric will tend to zero.

All three components of the performance metric are necessary to indicate how well the current map is performing. The three sub-metrics indicate different qualities that must all be good for the composite metric to be good. These qualities are that the robot must effectively explain the current sensor measurements, the robot must be certain of its pose, and the positional certainty must be well shaped.

### 3.3.4 Robot Relocalization

When instantiating a juvenile hypothesis in a neighboring map frame, it is necessary to do so without allowing the information from the parent map to make the juvenile map over-confident. This is important since the *Atlas* framework does not maintain any cross-covariances between map-frames. All map-frames are to be independent from each other. Therefore when the new robot position is initialized into the juvenile map-frame, it must do so without prior information, which means that the initial robot pose has an infinite variance.

Since it is difficult to represent an infinite variance in the standard form of the Kalman filter, it is simpler to view the relocalization of the robot as mapping a new robot “feature” from the first observed line features. This process is identical to mapping new line features from the robot pose. The only issue is that the data association of measured lines to the mapped lines requires the robot pose. The strategy employed in this implementation is to take the robot pose from the parent map, project it into the juvenile map with a reasonable (but not necessarily consistent) covariance, and use the pose for the initial data association only. Then the robot pose is reinitialized when enough lines have been found from which the robot can be fully observed. The

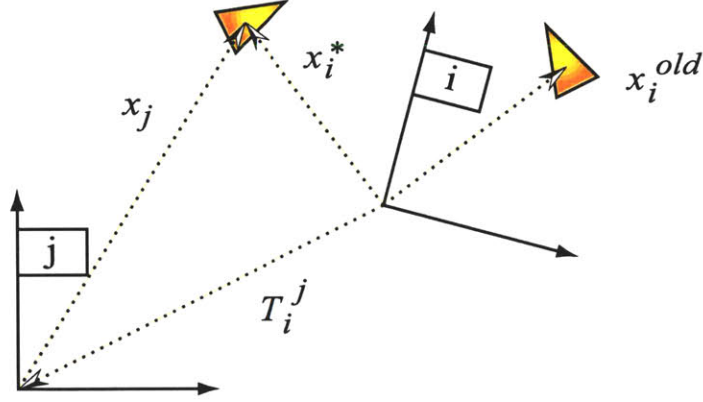


Figure 3-4: Seeding the robot position for a juvenile hypothesis in frame  $i$  using the current pose in the adjacent map-frame  $j$ . The retired hypothesis attached to frame  $i$  has the robot location at  $\mathbf{x}_i^{old}$ . The hypothesis is rejuvenated to have the robot pose of  $\mathbf{x}_i^*$ .

details of this approach follow.

When creating a juvenile hypothesis in a retired map-frame  $\mathcal{M}_i$  we reinitialize its robot pose  $\mathbf{x}_i$  using the robot pose  $\mathbf{x}_j$  from an adjacent map-frame  $\mathcal{M}_j$ . (See Figure 3-4.) First the hypothesis is seeded with a robot pose  $\mathbf{x}_j^*$  projected into frame  $i$ :

$$\begin{aligned}\mathbf{x}_i^* &= T_i^j \oplus \mathbf{x}_j \\ \Sigma_{\mathbf{x}_i}^* &= J_1(T_i^j, \mathbf{x}_j) \Sigma_{\mathbf{x}_j} J_1(T_i^j, \mathbf{x}_j)^T + \\ &\quad J_2(T_i^j, \mathbf{x}_j) \Sigma_{\mathbf{x}_j} J_2(T_i^j, \mathbf{x}_j)^T\end{aligned}$$

where  $J_1(\cdot, \cdot)$  and  $J_2(\cdot, \cdot)$  are the Jacobians of the transformation composition operators [50].

The hypothesis now enters a bootstrapping phase, in which a consistent initialization of the vehicle into the juvenile hypothesis is sought. Sensor measurements, interpreted with the seeded robot pose,  $\mathbf{x}_i^*$ , are accumulated. This continues until enough measurements,  $Z$ , have been collected to solve explicitly for the robot pose

independently of  $\mathbf{x}_i^*$ ; this function is  $w$ . This approach conserves the statistical independence of map-frames.

$$(\mathcal{M}_i, \mathbf{x}_i^{\text{new}}) = w(\mathcal{M}_i, Z)$$

If an explicit solution to  $w$  cannot be computed because of lack of explained sensor measurements, then the hypothesis is invalid and terminated. Otherwise a tenable juvenile hypothesis exists.

### 3.4 Map Matching

The Map Matching module is utilized to match corresponding structure and produce the coordinate transformation between two map-frames. The *Atlas* framework uses the transformation to hypothesize loop closures in the *Atlas* graph. The coordinate transformation between two maps is easy to produce, given corresponding structure. The challenge is to discover the corresponding structure between two maps.

The map matching technique employed in this implementation uses signature strings to aid in the correspondence process. The map signatures are composed from canonically sorted, transformationally invariant elements of the mapped line segments. A variant of the longest common substring algorithm is used to efficiently match signature strings.

Each pair of matching signatures between two maps produces a potential transformation which may align the maps. The signature match is then scored by how many lines are brought into alignment. The alignment is evaluated by transforming the lines in one map such that the lines are in the second map’s coordinate frame, then counting the nearest neighbor corresponding lines that fall within their respective Mahalanobis distances. The signature pair that produces the most correspondences

is deemed the best pair, and the transformation is re-estimated directly from the correspondences.

Map matching can be treated as a search through the transformation space that maximizes the alignment of the two maps. The signature strings are simply utilized to discover sample transformations in the transformation space to test for alignment. Once the best correspondence between the two maps has been determined, the transformation can be optimized to minimize the alignment error between corresponding map features.

### 3.4.1 Line Signature Strings

The method builds a signature string for each line in the map. The elements of the signature string consist of transformationally invariant metrics from the signature's line to all other lines in the map. There are two types of such metrics, depending on whether the lines are parallel or not. The metric using parallel elements is the distance between the midpoints of the two line segments. Non-parallel lines have a metric that consists of the angle between the two lines and the distance along the line of intersection. (See Figure 3-5.)

The distances of the intersections along the line are not invariant to translations. This adds some complication to the matching, since first a reasonable offset must be determined. Several offsets are considered by comparing signature elements with matching angles and clustering the differences between the intersection distances. The string match is repeated using the offset computed from every cluster.

The elements of each signature are sorted into a canonical order which defines the sequence or string to be matched. First all the parallel elements are sorted by their distance from the line, then the orthogonal elements are sorted by their intersection distance. When two signature elements of the same type are compared, their Mahalanobis distance is used to determine whether they match.



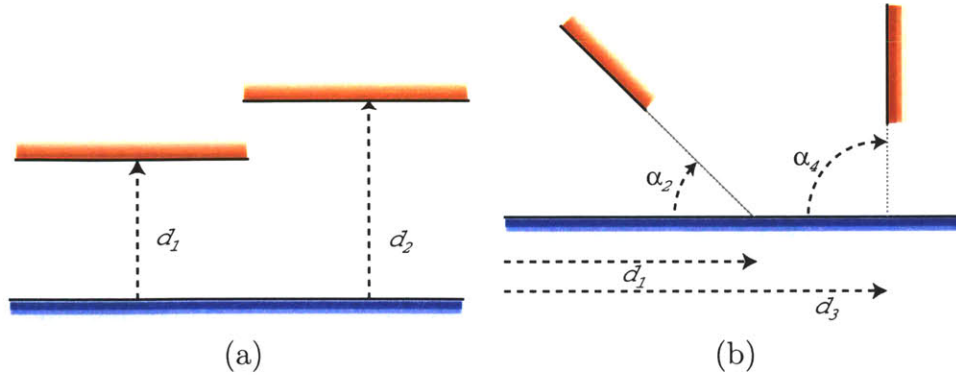


Figure 3-5: Line Signature Elements (a) Parallel Signature elements use the distance between the lines as the metric. (b) The non-parallel elements use the angle and the distance of the intersection along the line as the metric.

### 3.4.2 Repetitive Structure

Repetitive structure in the environment can lead to false map matches. The signature strings of lines in similar configurations, such as those occurring in multiple equally spaced corridors, will be nearly identical. Furthermore, since the transformations of the multiple false matches are likely to be fairly close to that of the correct match, the cycle verification procedure described in Section 2.2.4 is unlikely to detect the error.

By matching a map with itself, the repetitive structure can be identified. The ambiguous signatures are subsequently not used to match with other maps. This strategy keeps the focus of the map matching on the unique aspects of each map-frame and reduces the chance of false map matches due to repetitive local structure. Ambiguities with scales larger than the coverage of the map frame may still cause false matches; however, these false positives are mitigated by the *Atlas* cycle verification procedure.

### 3.4.3 Alignment Optimization

The correspondences discovered after matching signature strings are used to compute the coordinate transformation that best aligns the maps. Each pair of matched lines

provides two constraints on the transformation:

$$\begin{aligned}x \cos(\phi_a) + y \sin(\phi_a) &= \rho_a - \rho_b \\ \theta &= \phi_a - \phi_b\end{aligned}$$

Where  $x$ ,  $y$ , and  $\theta$  parameterize the transformation, and the matched line pair is  $\{L(\rho_a, \phi_a), L(\rho_b, \phi_b)\}$ .

The transformation parameters can be determined by a weighted least squares solution using the constraints from all line correspondences found. The constraints can be represented as a linear system:  $\mathbf{H}\mathbf{x} = \mathbf{z}$ , where  $\mathbf{x}$  is the vector of transformation parameters, and each constraint provides two rows of  $\mathbf{H}$  and  $\mathbf{z}$ .

The covariance matrix  $\Sigma_z$ , whose inverse defines the weights for the least squares solution, is determined from the Kalman filter covariances of the features in both maps:

$$\Sigma_z = \nabla_{\mathbf{a}\mathbf{z}}\Sigma_{\mathbf{a}}\nabla_{\mathbf{a}\mathbf{z}}^T + \nabla_{\mathbf{b}\mathbf{z}}\Sigma_{\mathbf{b}}\nabla_{\mathbf{b}\mathbf{z}}^T$$

where  $\Sigma_a$  and  $\Sigma_b$  are the corresponding sub-matrices of the matched features from their respective Kalman filters.

The least-squares solution and covariance for the transformation parameters are subsequently:

$$\begin{aligned}\mathbf{x} &= (\mathbf{H}^T \Sigma_z^{-1} \mathbf{H})^{-1} \mathbf{H}^T \Sigma_z^{-1} \mathbf{z} \\ \Sigma_x &= (\mathbf{H}^T \Sigma_z^{-1} \mathbf{H})^{-1}\end{aligned}$$

### 3.4.4 Summary

The Map-Matching procedure for feature-based map representations is summarized as follows:

1. A signature for both maps is constructed which is an ordered list of elements describing properties of the map that are invariant to translation and rotation of its coordinate frame. A comparison operator is defined over two signatures, yielding a set of correspondences between elements in each list.
2. Each map is matched with itself to identify repetitive structure. Any elements that correspond to other elements in the same map are removed from the map's signature. This dramatically reduces the likelihood of false map matches due to repetitive structure, by focusing on the unique elements of each local environment.
3. The signatures of both maps are now compared. Each element to element correspondence defines a potential alignment transformation from  $\mathcal{M}_i$  to  $\mathcal{M}_j$ .
4. Each potential alignment transformation is applied to  $\mathcal{M}_i$ . The validity of each transformation is evaluated by counting the number  $\eta$  of feature pairs it brings into alignment with nearest-neighbor gating.
5. The correspondences from the best (largest  $\eta$ ) potential transformation with  $\eta > \eta_{\min}$  are used to refine the transformation and its covariance. Each correspondence defines a constraint on the transformation. The combined set of  $\eta$  constraints are solved in weighted least-squares sense using the covariances of the feature estimates within each map to form the weights. This process also yields the covariance of the transformation. The parameter  $\eta_{\min}$  is set to 4 in this implementation.

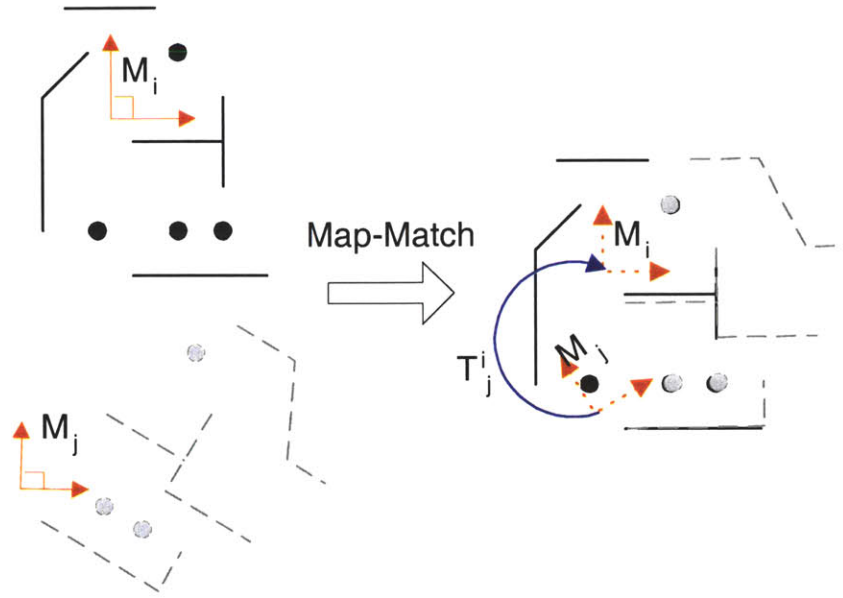


Figure 3-6: Map-Matching as a search for a transformation between maps that maximally aligns common mapped features. Here two maps  $i$  and  $j$  share features, and a good map match can be found between them. Note how only a subset of features are matched.

### 3.5 Experimental Results

The experiments used to evaluate the laser line implementation of the *Atlas* framework arise from two major data sets. The first data set challenges the framework with a large scale task to navigate and map the entire third floor corridor network surrounding MIT's Killian Court, a cluster of 12 interconnected buildings at the center of the MIT campus. The second data set illustrates the long duration performance when repeatedly traversing a relatively small area in the MIT Physics' office floor. For both experiments, *Atlas* adjacency matrix, map time history, and the final optimized map are depicted. Also the intra-map Kalman filter residual and inter-map optimization residual statistics are reported.

### 3.5.1 MIT's Killian Court

The Killian Court data set is just over two hours long, throughout which the robot travels a distance of 1,453 meters. Figure 3-7(a) shows the hand-drawn topological path of the vehicle superimposed on an architectural drawing of the Killian court area. The route contains nested loops of various sizes and topologies, starting and ending in the elevator of Lobby 7. Figure 3-7(b) shows the dead-reckoned path resulting from simply integrating the odometry data. The dead-reckoning path clearly reveals systematic biases in the odometry.

The severities of the odometry error make this data set challenging. The advanced odometry modeling is necessary to improve the navigation performance in the presence of extreme odometry errors; however, the addition of extra model parameters makes the Kalman filter difficult to tune. Figure 3-9(a) depicts the odometry bias  $\lambda$  (from Equations 3.8 and 3.9) as estimated by the local maps' Kalman Filters.

Further analysis of the values for  $\lambda$  that have been estimated on-line has revealed that  $\lambda$  has a strong heading dependence that is reminiscent of the heading-dependent compass biases that one sees with autonomous underwater vehicle navigation data. This dependence can be visualized by plotting the estimated  $\lambda$  value vs. the original uncorrected odometry heading. (See Figure 3-9(b).) Accordingly, it is possible to compute a one-time calibration for  $\lambda$  as a function of heading for the b21 synchro-drive mechanism. Dramatically improved dead-reckoning estimates are obtained when this heading-dependent calibration is applied to the b21 odometry data, as illustrated by comparing Figure 3-10(a) with Figure 3-10(b).

In this section, results are presented both with and without the correction of the systematic odometry bias errors. All subsequent results in the thesis utilized the corrected odometry.

## Results with systematic bias errors

Figure 3-8(a) shows the resulting maps under a Dijkstra projections with the first map as root, where as Figure 3-8(b) shows the result of applying the global optimized map projection as described in Section 2.4. A total of 75 map-frames were built, each containing a maximum of 15 mapped line segments.

## Results after removal of systematic bias errors

Figure 3-11(a) shows the resulting maps under a Dijkstra projections with the first map as root, where as Figure 3-11(b) shows the result of applying the global optimized map projection as described in Section 2.4. A total of 84 map-frames were built, each containing a maximum of 15 mapped line segments.

Figure 3-12(a) plots the numerical label of the dominant map-frame with time. During map-frame genesis, a counter is incremented and the newly created map is labeled with its value. As new ground is covered, the value of the dominant map ID increases. When the robot returns, however, to a previously mapped area, the dominant map ID decreases when a loop closure traversal is successful. For example, approximately 47 minutes into the experiment the robot returned to the area first mapped. Similarly, after an hour and a half, the vehicle returned to a region mapped 50 minutes earlier, and at the end of the run, the robot returns to its starting area.

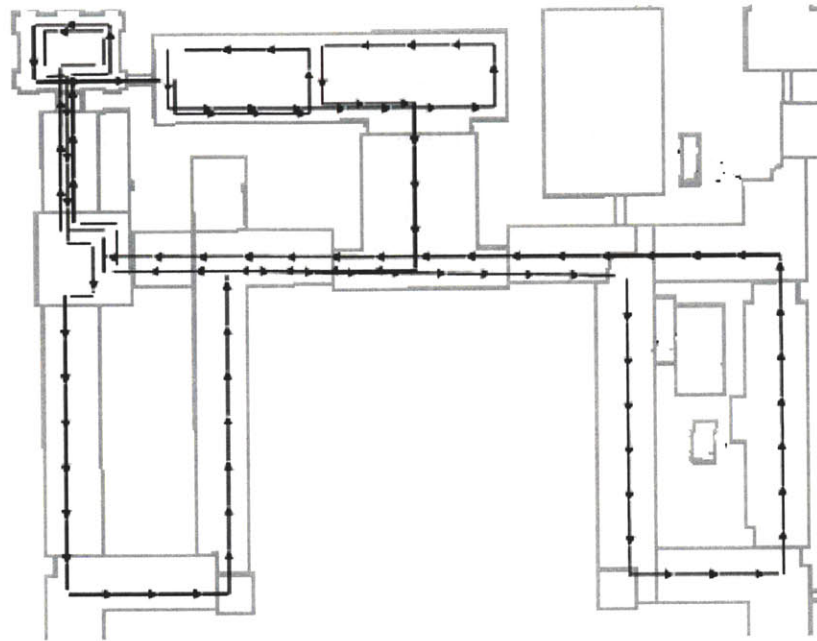
The histograms of the Kalman filter line update residuals (Figure 3-13) show the performance of the local maps. The residuals are from the difference between the line measurements and the predicted measurement computed from the Kalman filter robot state and associated map line. The lines residuals when taken together have a mean squared error of  $(6.2\text{cm})^2$  and  $(2.3^\circ)^2$  on the  $\rho$  and  $\phi$  line parameters, respectively. The residuals are in agreement with the accuracy of the laser scanner which has about a  $10\text{cm}$  standard deviation in the range measurements.

The global map performance can be characterized by the residuals from the global

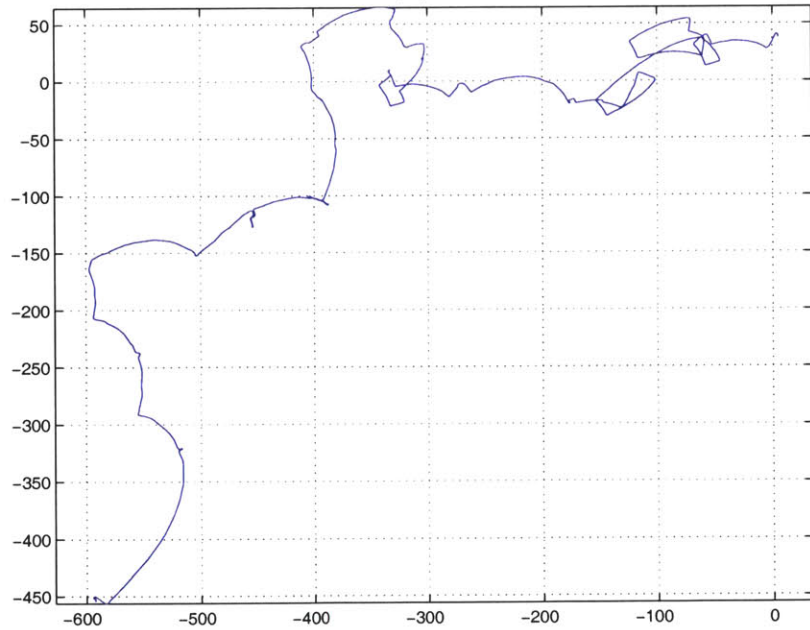
optimized map projection. These residuals are the differences between the projected map origins and the corresponding *Atlas* edge transformations after the final optimization iteration. Figure 3-14 shows the histograms of the residuals where the standard deviation of the errors are 14.0cm, 13.0cm, and 1.2°, for the  $x$ ,  $y$ , and  $\theta$  parameters, respectively.

Figure 3-15 shows the instantaneous sum of kernel and user time for the *Atlas* process as well as its smoothed value. Note that as more features are mapped, and more map-frames are created, the mean processor load stays nearly constant.

The experiment demonstrates the ability of *Atlas* to map a large network of campus corridors with laser lines while closing multiple, nested loops, in roughly constant time performance per measurement.



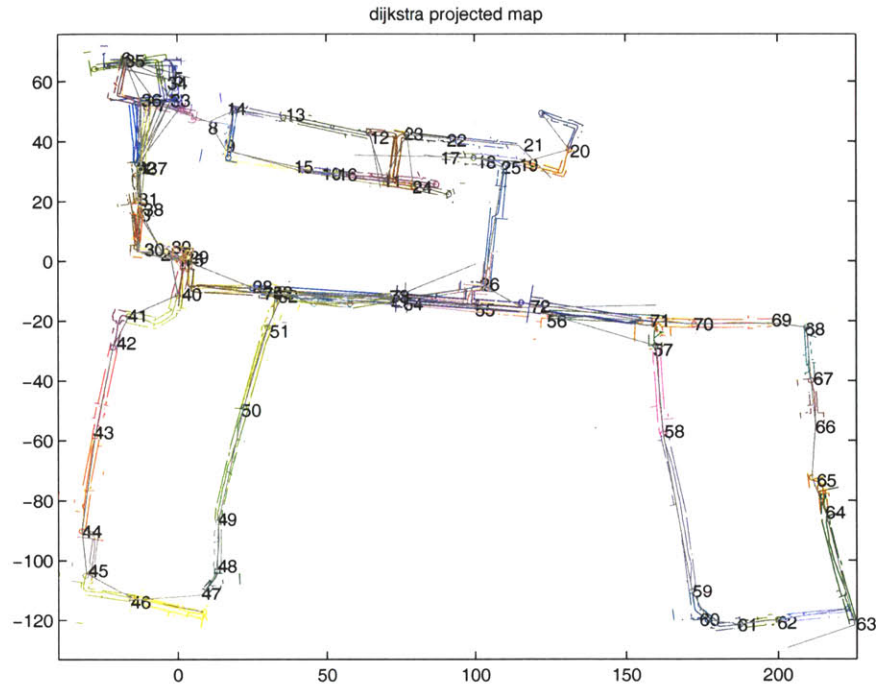
(a)



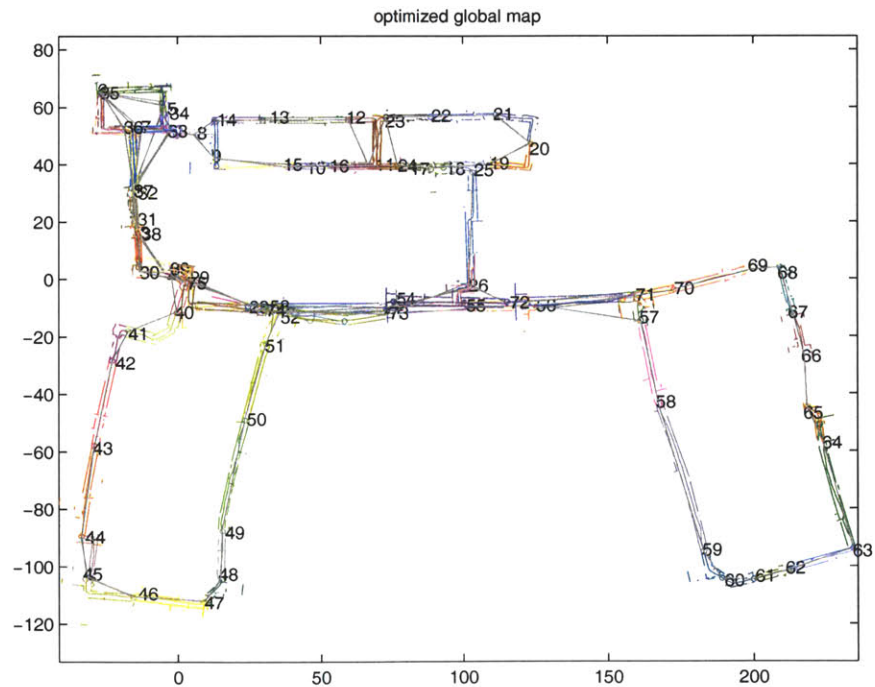
(b)

Figure 3-7: (a) The manually drawn topology of the driven route overlaid on an architectural drawing of part of the MIT campus (Killian Court). The principle east-west passage, known as the “infinite corridor”, is approximately 225 meters long. (b) The trajectory derived from uncorrected odometry alone.



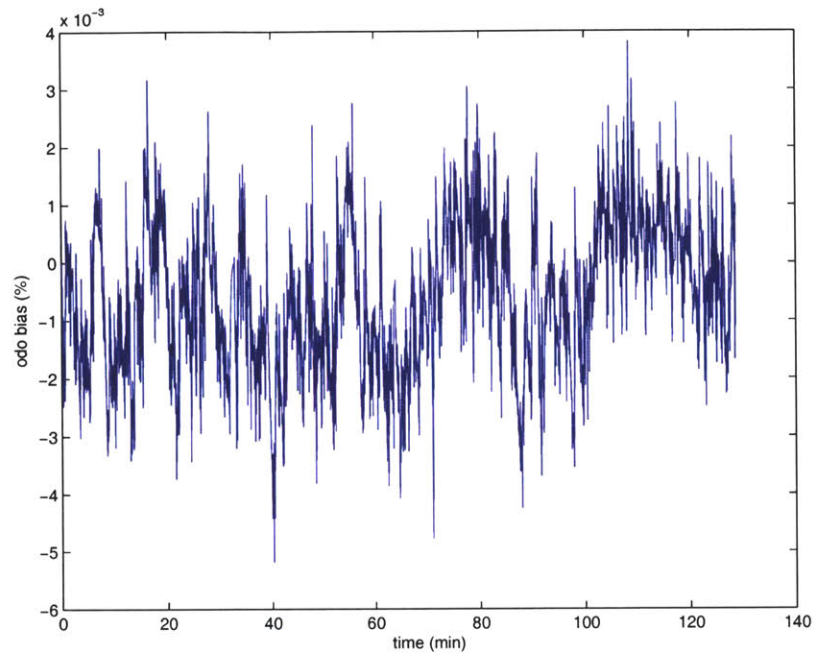


(a)

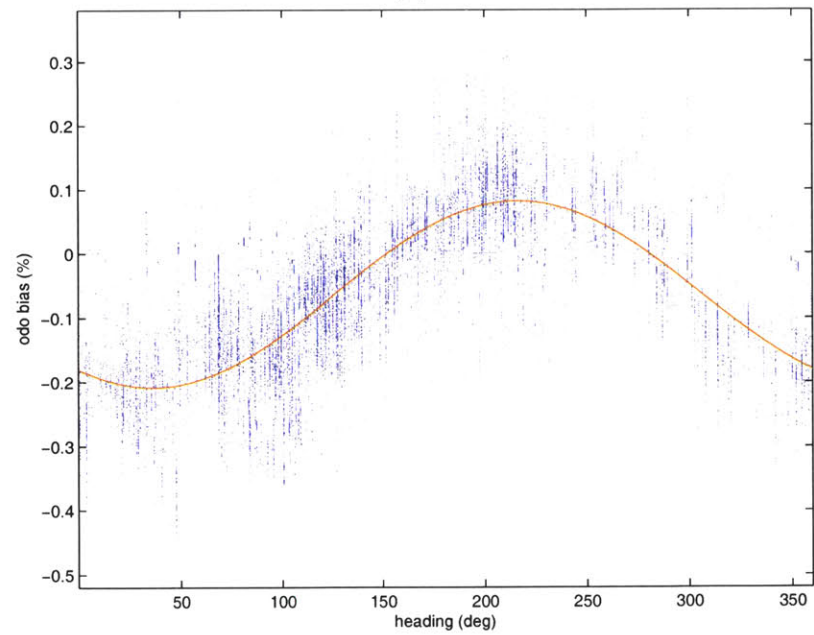


(b)

Figure 3-8: (a) Dijkstra projection and (b) global optimized map projection for processing of laser data on the Killian Court data set using uncorrected odometry. Each local map is drawn in a different color and labeled with the map id next to its coordinate origin. The valid *Atlas* edges are drawn in black, whereas unverified edges are drawn in magenta.

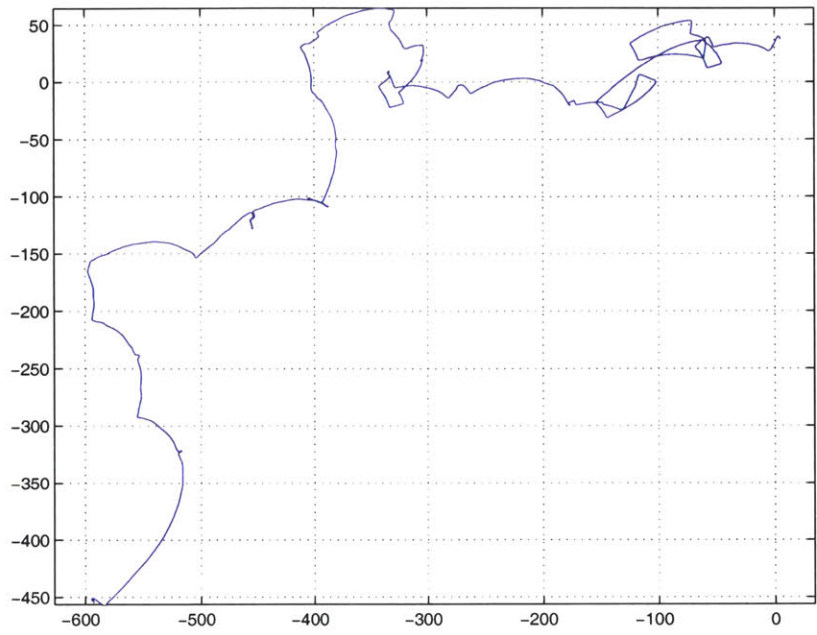


(a)

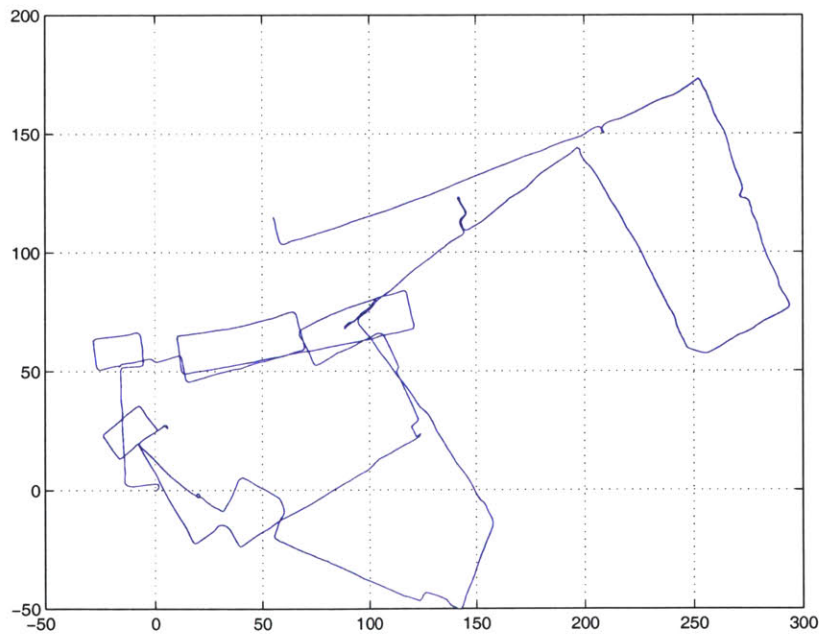


(b)

Figure 3-9: The estimated odometry bias parameter  $\lambda$  as a function of (a) time and (b) uncorrected heading on the b21 robot. A sinusoidal function is fitted to the values, shown in red. Once the systematic dependence is factored out from the odometry data, the corrected path shown in Figure 3-10(b) is produced.

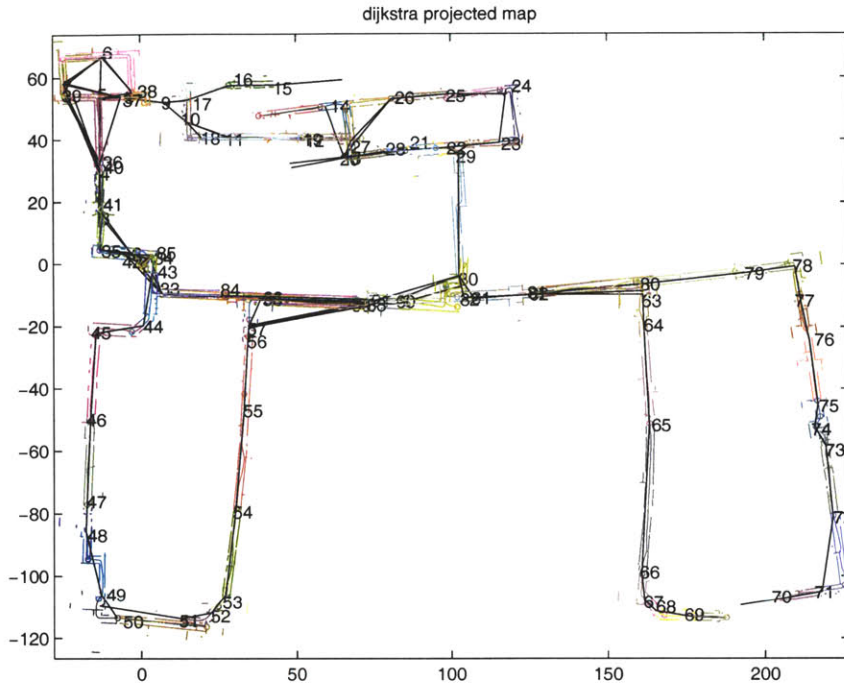


(a)

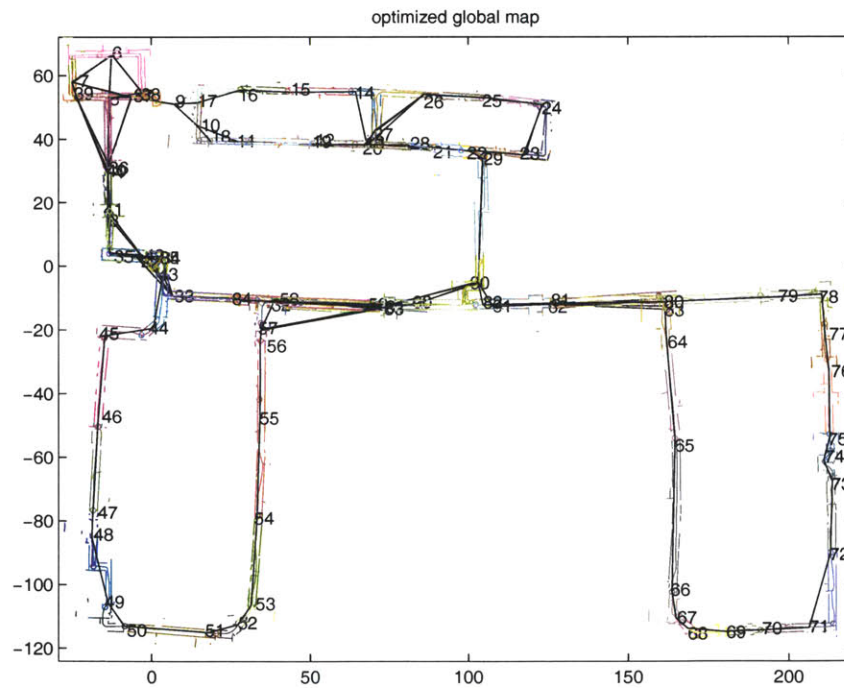


(b)

Figure 3-10: (a) The original uncorrected odometry trajectory for the Killian Court data set. (b) The updated odometry trajectory when compensating for the systematic heading dependency on the bias parameter  $\lambda$ .

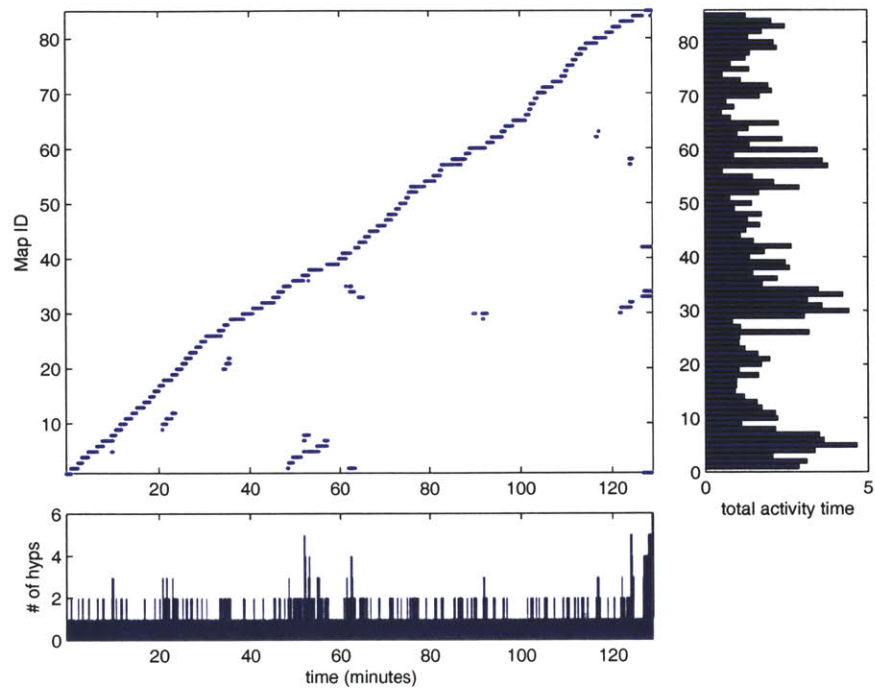


(a)

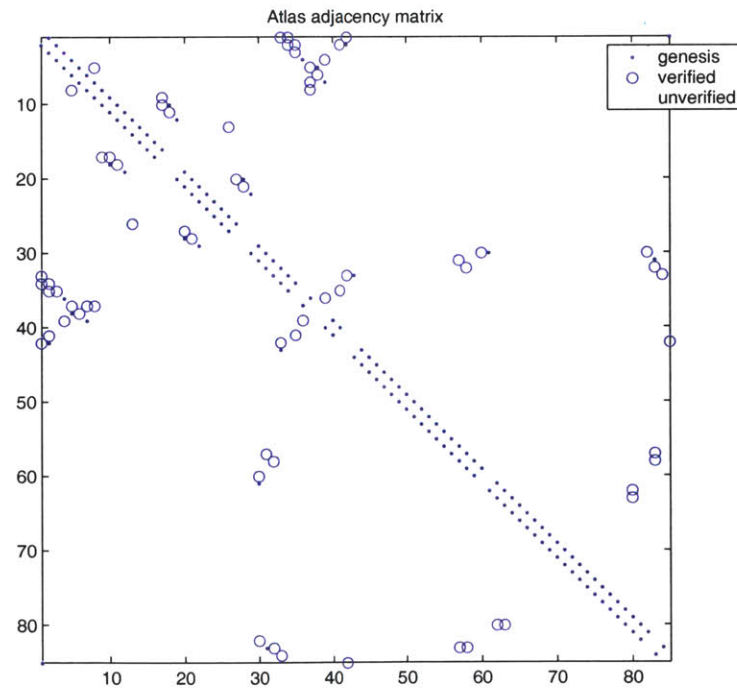


(b)

Figure 3-11: (a) Dijkstra projection and (b) global optimized map projection for processing of laser data on the Killian Court data set using corrected odometry. Each local map is drawn in a different color and labeled with the map id next to its coordinate origin. The valid *Atlas* edges are drawn in black, whereas unverified edges are drawn in magenta.



(a)



(b)

Figure 3-12: (a) Map ID vs. time, total activity vs. map ID, and the number of active hypotheses vs. time for the laser feature-based SLAM processing. (b) *Atlas* adjacency matrix. Dots indicate genesis edges, circles indicate verified edges, and crosses indicate unverified map-match edges.

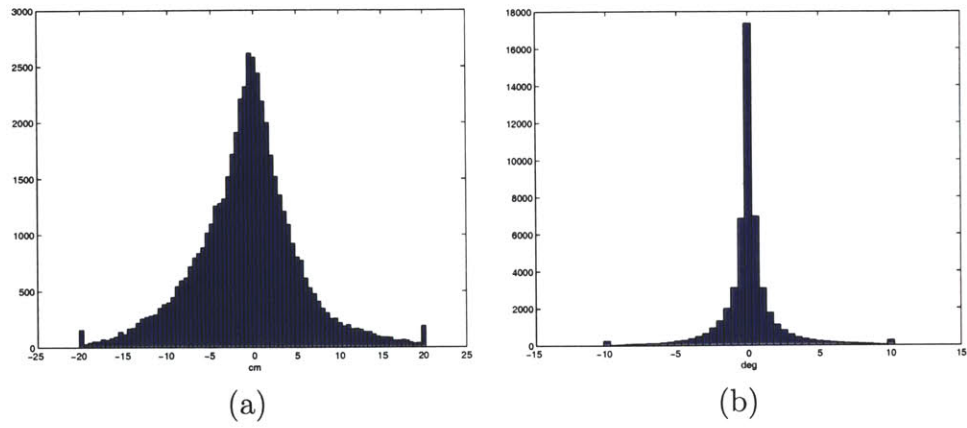


Figure 3-13: The histograms of the Kalman filter residuals for line updates on (a) rho and (b) phi parameters.

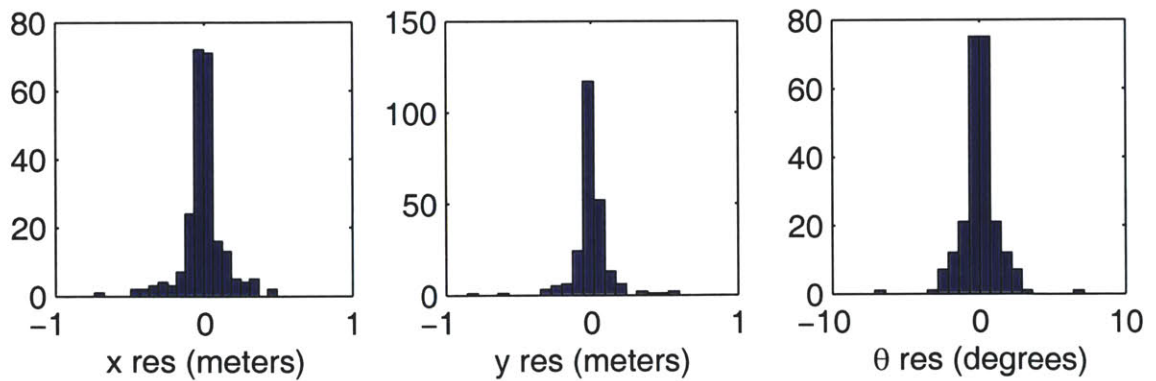


Figure 3-14: The histograms of the Global map projection optimization residuals for the  $x$ ,  $y$ , and  $\theta$  parameters of each map-frame transformation.

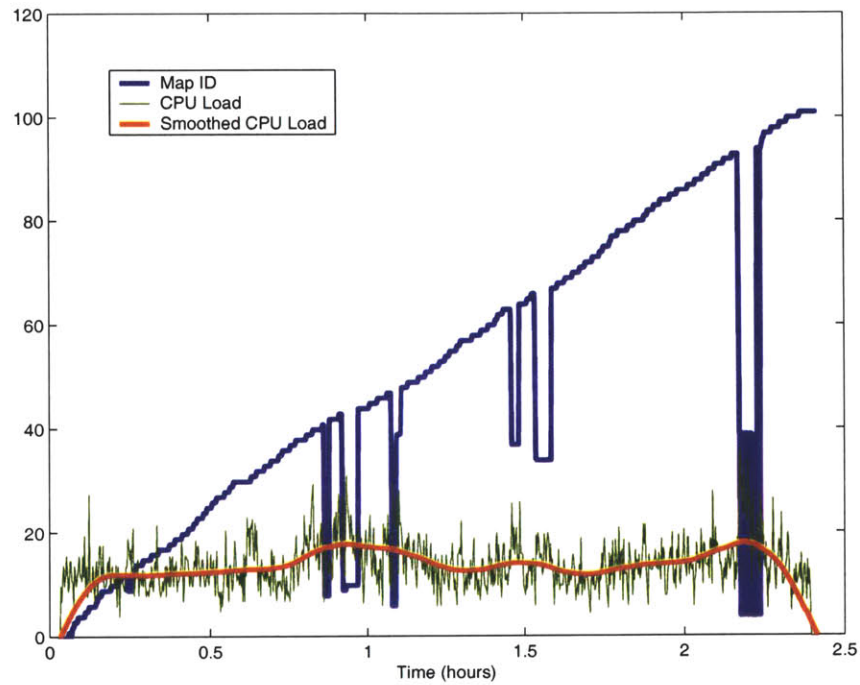


Figure 3-15: Processor load and current map ID for laser data run. The general linear increase in current map ID is indicative of the mapping of new areas. The occasional “fall-back” to a map with a lower ID represents successful loop closing —the re-use of an existing map. (The processor load is plotted in arbitrary units.)

### 3.5.2 Ten loops in a small-scale environment

The second experiment illustrates a situation in which many loops are repeatedly performed in a relatively small-scale environment. The robot executed multiple “figure eight” maneuvers within the environment depicted in Figure 3-16(a). Each corridor was traversed several times in both directions. The total path driven was 690 meters taking 45 minutes to complete. Figure 3-16(b) is the “dead-reckoned” path using odometry data.

Figure 3-17 shows the *Atlas* output for this environment with laser feature-based local mapping under the Dijkstra and optimized map projections. A total of twenty six map-frames were created. Figure 3-18(a) shows the adjacency matrix of the *Atlas* graph. Figure 3-18(b) plots the ID of the dominant map vs. time for this experiment, as well as the number of active hypotheses vs. time and the total amount of time spent in each map-frame.

The ideal performance would be indicated by no more maps being generated after the area is mapped; however, with the laser line implementation, relocation into previous maps is not always possible and new maps are generated. Relocation mainly fails in the long corridors where a previous map cannot be robustly reinitialized using only two parallel wall measurements. The new maps, however, are quickly matched to previous maps and relocation occurs in other maps.

The histograms of the Kalman filter line update residuals (Figure 3-19) show the performance of the local maps. The lines residuals have a mean squared error of  $(5.3\text{cm})^2$  and  $(1.9^\circ)^2$  on the  $\rho$  and  $\phi$  line parameters, respectively. The global map performance can be characterized by the residuals from the global optimized map projection. Figure 3-20 shows the histograms of the residual of the *Atlas* edge transformations with respect to the projected pose of each map-frame. The standard deviation of the errors are  $10.9\text{cm}$ ,  $9.3\text{cm}$ , and  $0.9^\circ$ , for the  $x$ ,  $y$ , and  $\theta$  parameters, respectively.





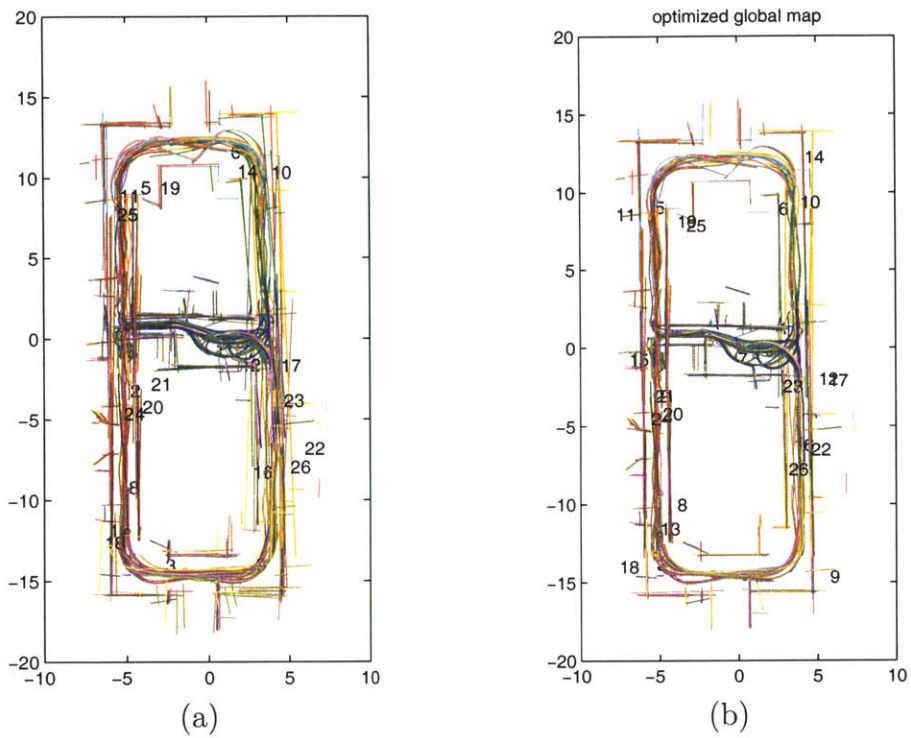
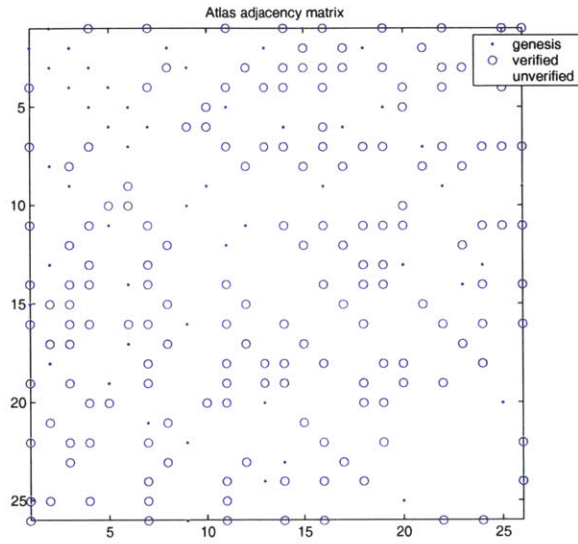
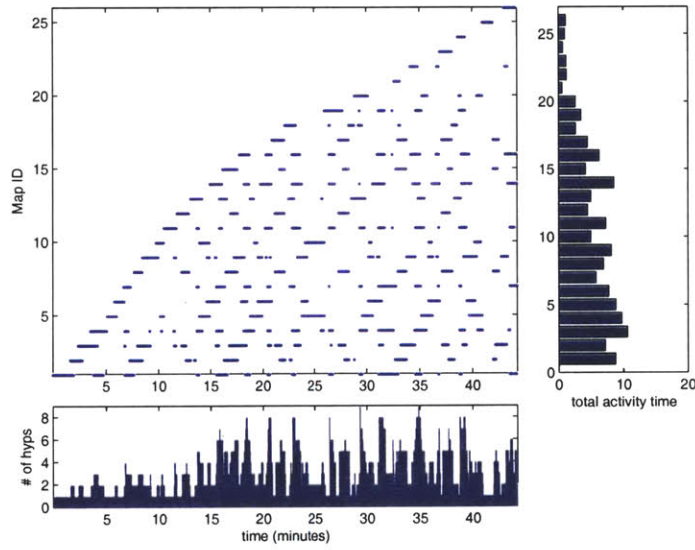


Figure 3-17: (a) Dijkstra projection and (b) global optimized map projection for processing of laser data on the Ten Loops data set. Each local map is drawn in a different color and labeled with the map id next to its coordinate origin. Many of the maps have their origins in the hallways because these are the locations where robot relocation is difficult and new maps are generated.



(a)



(b)

Figure 3-18: (a) *Atlas* Adjacency matrix. Dots indicate genesis edges, circles indicate verified edges, and crosses indicate unverified map-match edges. (b) Map-frame genesis and activity. The generation of new map-frames drops off as the area becomes fully mapped and maps are re-used; however, new maps continue to be generated when transitioning from areas that have poor relocalization characteristics, such as the long hallways. The lower figure shows how the average number of active hypotheses remains nearly constant after the area has been mapped.

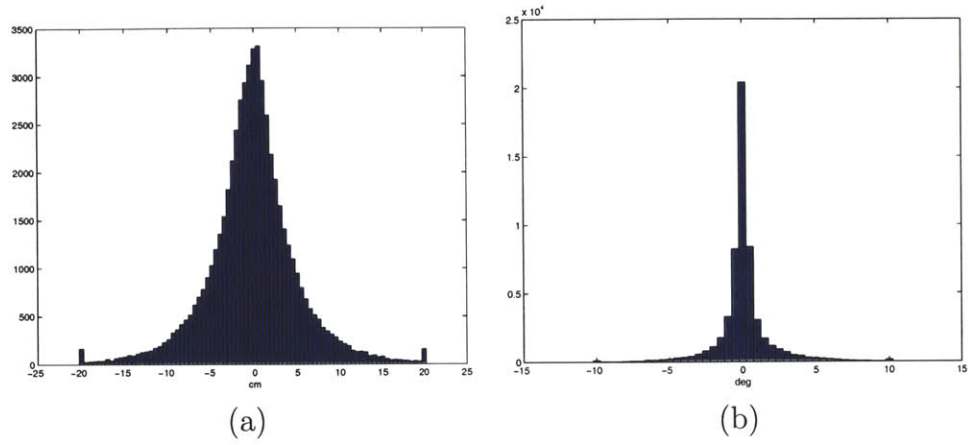


Figure 3-19: The histograms of the Kalman filter residuals for line updates on (a) rho and (b) phi parameters.

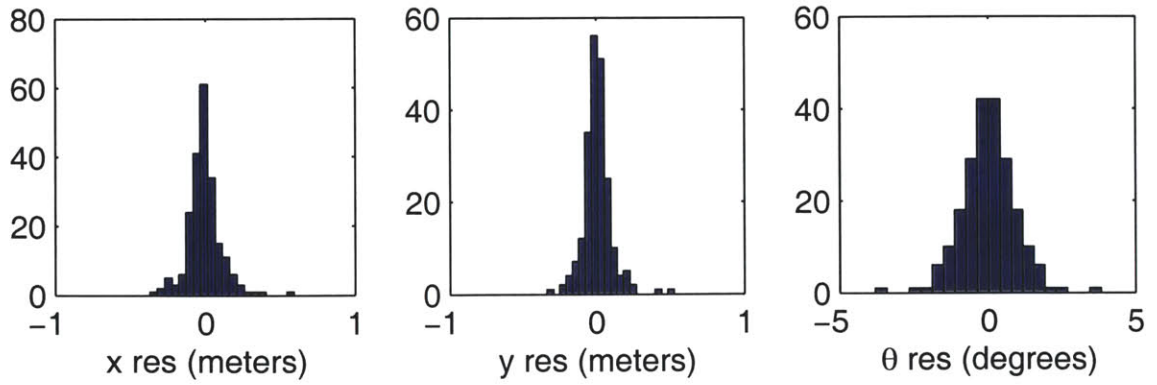


Figure 3-20: The histograms of the Global map projection optimization residuals for the  $x$ ,  $y$ , and  $\theta$  parameters of each map-frame transformation.

# Chapter 4

## Atlas with Wide Angle Sonar

### 4.1 Introduction

The second major implementation of the *Atlas* framework uses wide angle monaural ultra-sonic ranging sensors. For simplicity, these sensors are referred to as sonars. The B21 robot used in this implementation has a ring of 24 sonars. Each sonar sensor sends an ultra-sonic ping with a beam width of about 30 degrees, measuring the time it takes to receive the first echo which is converted (using the speed of sound) to a distance measure. These sonars are partially observable sensors since they measure only the range to the nearest surface in the beam and not the surface's exact bearing within the beam. The sonar measurements can be treated as regions of constant depth from which the phenomena causing the echo is somewhere on the region. This assumes, however, that the echos are from single bounces of the sound ping and not multi-path echos from several surfaces. The first echo received is the shortest and hence typically not due to multipath.

## 4.2 Sonar Features

There are two major types of echos: spectral echos and diffuse echos. Specular echos come from the direct reflection of the sonar ping along the surface normal, and diffuse echos come from the scattering of the sound from a corner or imperfection in the surface. In the indoor environments used to test this implementation, specular echos come from walls and doors whereas diffuse echos come from door moldings, corners and cracks in the walls.

As in Chapter 3, the environment is represented in 2D (the heights of observed objects are ignored). This 2D assumption transforms the regions of constant depth into arcs of constant depth, and dramatically simplifies the models and computation required to process them.

The indoor environment led to the decision to use two types of features to model the objects from which echos arise. The walls and doors are modeled as 2D line segments which primarily consist of spectral echos whose arcs of constant depth are all cotangent. In contrast, corners, surface imperfections and door moldings are modeled by 2D points in which arc of constant depth all intersect. (See Figure 4-1.)

This model is not complete, since there are objects which fit neither model. For example, cylinders look almost like points but have a constant range bias due to their radius. Curved surfaces may look almost like lines, except that the surface normal's direction changes along the curve. However, the environments for which the results are presented do not contain many such objects, and the assumption is that mismodelling a few features does not cause significant problems.

### 4.2.1 Point Features

Point features in the scene result in echos whose arcs of constant depth all intersect at the point.

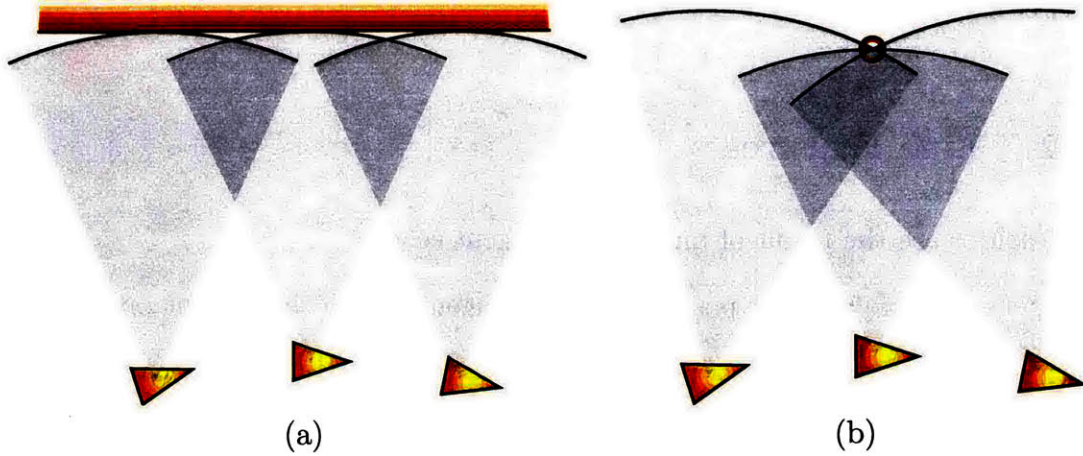


Figure 4-1: There are two major type of features used to model the sonar echos. (a) Line features where all the echos' arcs of constant depth are cotangent. (b) Point features in which all arcs intersect.

The point of intersection of two sonar echos can be determined by first intersecting the circles of the two arcs, then determining which of the two possible solutions (if any) lie within the beam width of the sonar.

The intersection point  $(x, y)$  of two circles at points  $(x_1, y_1)$  and  $(x_2, y_2)$  with radiuses of  $r_1$  and  $r_2$  is solved as follows. First compute the determinant  $det$

$$det = \sqrt{((r_1 + r_2)^2 - d^2)(d^2 - (r_2 - r_1)^2)} \quad (4.1)$$

where  $d^2$  is  $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$ , or just the squared distance between the centers of the two circles. The intersection points are determined using the two possible signs of the determinant  $det$  in the following formulas:

$$x = \frac{1}{2} \left( \frac{-(y_1 - y_2)(\pm det) - (r_2^2 - r_1^2)(x_2 - x_1)^2}{d^2} + (x_1 + x_2) \right) \quad (4.2)$$

$$y = \frac{1}{2} \left( \frac{+(x_1 - x_2)(\pm det) - (r_2^2 - r_1^2)(y_2 - y_1)^2}{d^2} + (y_1 + y_2) \right) \quad (4.3)$$

If  $det$  is real then the circles intersect in two places. If the determinant is zero then the circles are tangent and intersect in one place. Finally, if the determinant is imaginary,

the circles do not intersect.

### 4.2.2 Line Features

Line features are the result of multiple cotangent echos.

In general there are four possible lines cotangent to two circles, but only the lines that are on the same side with respect to the circles are of interest to sonar processing. Similarly to Section 4.2.1, first a determinant  $det$  is computed which indicates whether the two circles have cotangent lines or not.

$$det = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 - (r_2 - r_1)^2} \quad (4.4)$$

The cotangent lines, parameterized by  $\rho$  and  $\phi$ , are then:

$$\rho = \frac{(x_1 y_2 - x_2 y_1)(\pm det) - (y_2 - y_1)(r_1 y_2 - r_2 y_1) - (x_2 - x_1)(r_1 x_2 - r_2 x_1)}{(x_2 - x_1)^2 + (y_2 - y_1)^2} \quad (4.5)$$

$$\phi = \arctan \left( \frac{-(x_2 - x_1)(\pm det) - (y_2 - y_1)(r_2 - r_1)}{(y_2 - y_1)(\pm det) - (x_2 - x_1)(r_2 - r_1)} \right) \quad (4.6)$$

When the determinant  $det$  is imaginary, there are no cotangent lines. This occurs when one circle lies completely inside the other.

## 4.3 RANSAC Data Association

Because both line and point features have two degrees of freedom, and each measurement has only one DOF, these features cannot be initialized from a single position. Multiple echoes (at least two) must be grouped and matched from various positions so that all the DOFs of the features become observable. Thus a mechanism is required to discover which echoes belong together from multiple vantage points and to determine which type of model best describes the groups of echos. This mechanism



is the sonar data association and segmentation engine, or simply referred to as “the engine” in this chapter.

The sonar data association and segmentation engine includes an efficient random sample and consensus (RANSAC) based method [6]. The basic idea is to pick a random sample of two sonar echos from a window of different measurement positions, check if the samples fit a line or a point, and then count how many of the remaining echos are in agreement with the line or point.

Others have used Hough transform approaches which are similar but much less efficient. The Hough transform approaches grid the feature parameter space, then discretizes each echo into a grid, accumulating votes for which grid cells (i.e. feature parameters) could have generated the echo. The grid cells with the most votes indicate the features that best explain the data. The RANSAC approach is similar since each random sample determines a point in the feature parameter space that is then voted on by the remaining echos; however, the entire parameter space need not be represented.

### **4.3.1 Adjacency preprocessing**

The basic implementation of the RANSAC strategy is not necessarily efficient, since it requires that every echo be checked when scoring a random sample. This is inefficient because most of the echos do not match one another. Therefore the engine makes use of some preprocessing to roughly determine which echos may possibly match, consequently speeding up the process for finding a valid random sample and scoring the relevant echos for consensus.

The engine builds a graph with all the sonar echos from a short sequence of robot positions as vertices. The edges of the graph correspond to echo pairs which may match. In fact, there are two graphs; one for discovering point matches and another for line matches. As each new round of sonar measurements becomes available, the engine adds new nodes and edges to the graphs. Likewise, when old positions leave

the window of active positions, the old vertices and their corresponding edges are removed from the graphs. The graphs utilize an adjacency list sorted by the echo's id number to speed up these incremental changes.

The tests for determining which echos may match, and hence which edges exist in the graphs, take advantage of the bounds of the sonar beam. When considering point features, two echos are compared by first checking the distance between the center of each arc of constant depth. See Figure 4-2. If the distance is greater than the average width of the two beams at the range point then the pair is ignored. Subsequently if the determinant computed as in Equation 4.1 is imaginary, then the echos also cannot match.

Echos are also checked for matching as line features. First the echos are checked to determine whether they come from the same general direction. The angle between the center of the two sonar beams must be less than the beam width's angle before the echos can be considered to originate from the same line. The echos are further checked by looking at the difference of their centers projected on an approximate guess to the line's normal. The approximate normal is simply computed as the average direction of the two echos. If the projected difference is less than a threshold computed from the beam width angle and expected sensor measurement noise, then the echo pair is considered for further processing. As a final test, the determinant from Equation 4.4 is computed, and must be real-valued, for the two echos to be considered a possible line match. (See Figure 4-3.)

The employment of graphs greatly increases the performance of the RANSAC trials. Valid pairs of echos are more likely to be picked because only adjacent pairs are considered for samples. Furthermore only the echos in the union of the sample pair's adjacency must be tested to score the pair.

The sonar data association engine processes sonar echos to extract groups of echos that originate from possible points or lines in the environment. For each navigation

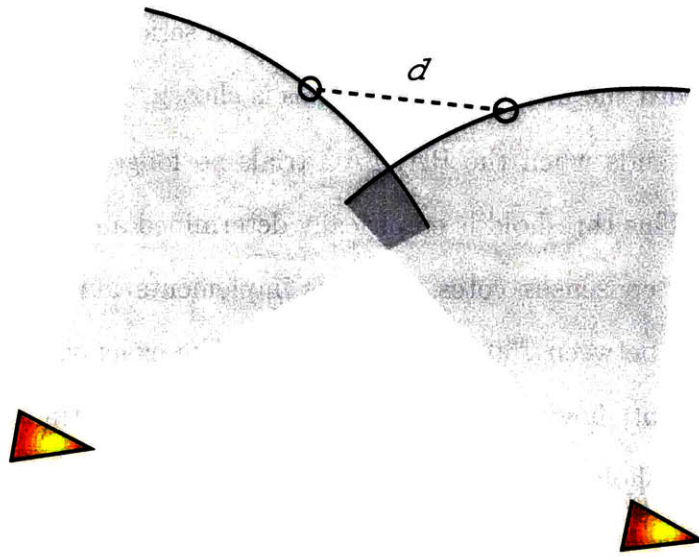


Figure 4-2: Sonar echoes are considered to match at a point only when the distance between the centers of their arcs is less than the average width of the arc.

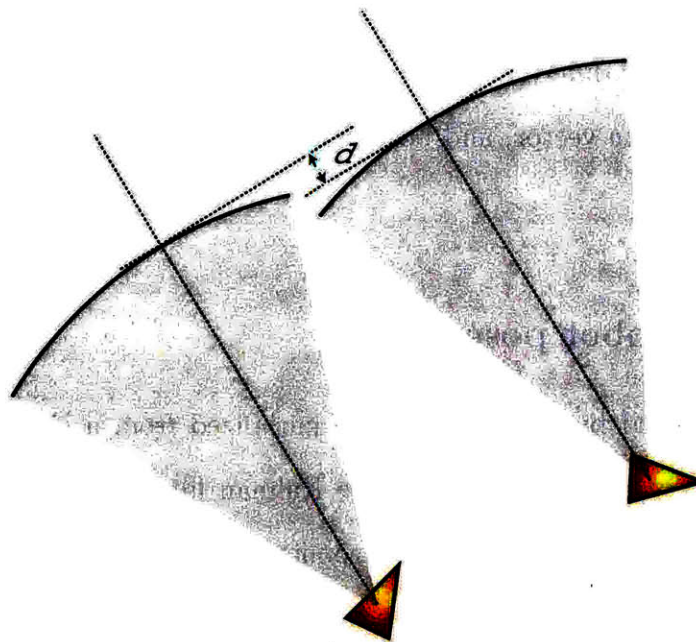


Figure 4-3: Sonar echoes are considered to match on a line only if the angle between the beam directions is less than the beam width, and the projected range difference is less than a threshold.

time step, the engine extracts points and lines one at a time, marking the corresponding echos as used. Each line or point is the result of a series of RANSAC trials where the line or point with the most consensus echos is chosen.

The extraction ends when the RANSAC trials no longer find consensus greater than a threshold. This threshold is empirically determined and typically ranges from between 10 and 40 consensus votes. (In this implementation, the total number of active echos ranges between 750 and 1500 echos.) The exact value of the threshold is not very significant; however, it may slightly increase the time spent processing the echos if the threshold is too low. Conversely, if the threshold is set too high, the engine may not be able to detect valid features.

## 4.4 Multiple Vantage-Point Kalman Filter

As with the laser lines implementation in Chapter 3, the sonar *Atlas* implementation uses an Extended Kalman filter to maintain the local navigation and map state. There are, however, a few key differences. The sonar implementation maintains several past robot poses in the state vector, and both points and lines are mapped as features of the environment.

### 4.4.1 Saved robot poses

Since features mapped by sonar cannot be initialized from a single vantage point, multiple robot poses are maintained in the Kalman filter state. The Kalman filter maintains the current robot pose as the robot moves within the environment, and periodically the Kalman state is augmented with a copy of the current pose that will not be propagated in subsequent time steps. These augmented states behave like dropped vantage-points from which multiple views of sonar echoes are used to initialize the mapped features.

A significant baseline between vantage-points is needed to properly initialize the features from sonar echos. Hence it doesn't make sense to save too many dropped states which are too close together. A new saved robot pose is created only when the robot travels a minimum distance (0.4 meters in this implementation) from the previously dropped pose. Likewise, it doesn't make sense to have an ever-increasing bank of saved robot poses. Firstly, the bounded-complexity assumption for the *Atlas* local SLAM module would be violated. Secondly, vantage-points are less likely to share measurements from corresponding features as the distance between them increases.

These problems are remedied by removing the corresponding rows and columns of the oldest saved robot pose from the Kalman state and covariance matrix. The complexity of the filter is bounded by limiting the maximum number of saved states. When the limit has been reached, then the oldest robot pose is discarded before saving a new one.

It is important to note that this implementation of the Kalman filter is similar to that of a fixed-lag Kalman smoother [25]. Both Kalman filter versions have a bank of states corresponding to the main state at past time steps. The difference lies in the Kalman propagation step. In the fixed-lag smoother, all the states are propagated each time step; the extra states are simply delayed from the current state. In the multiple vantage point Kalman filter, only the current state is propagated; the extra states' positions in the state vector are simply shifted as new states are dropped and old ones deleted. The later approach has the advantages that the dropped states need not be propagated at every time step; neither do they need to be created at regular time intervals. The placement of dropped states can be linked to distance traveled instead of time spent, which is more appropriate for ensuring adequate geometry when initializing new features.

## 4.4.2 Feature updates

Updates to the saved robot poses in the Kalman filter are transferred to the corresponding poses in the sonar data association engine. Since the engine maintains a much denser collection of robot poses, the corrections for poses without a corresponding Kalman filter state are interpolated.

Measured lines and points discovered with the data association engine are used to match mapped lines and points from the Kalman filter state. In fact, the procedure for matching lines is identical to that presented in Section 3.3.3, and the point matching procedure is very similar. When a match between a measured line or point and a mapped line or point is found, the Kalman filter is updated.

The corresponding sonar echos are used to update the Kalman filter. The measured lines are simply used to match with the mapped lines and determine the association of echos to their respective features. Using the echos directly is more preferable than using the measured line features, because it is easier and more accurate to model the noise. The covariances of the measured line and points from data association engine are approximated from the nonlinearities of their initialization functions. Using the echos directly avoids an extra linearization step.

Feature updates are processed by forming the residual for each measured echo. The echos can be predicted from the corresponding vantage-point and mapped features. The predicted echo distance for point features uses the distance between the point feature and the sonar sensor's position, transferred into map coordinates.

$$\begin{aligned}d_p &= \sqrt{\Delta x^2 + \Delta y^2} \\ \Delta x &= (\cos \theta_r x_s - \sin \theta_r y_s + x_r) - x_p \\ \Delta y &= (\sin \theta_r x_s - \cos \theta_r y_s + y_r) - y_p\end{aligned}$$

where  $(x_s, y_s)$  is the location of the sonar sensor in the robot's frame,  $(x_r, y_r, \theta_r)$  is the

pose of the corresponding vantage-point,  $(x_p, y_p)$  are the coordinates of the mapped point, and  $d_p$  is the predicted point echo distance.

The Jacobian of the measurement prediction function can be easily computed with the chain rule, since the sensor pose is composed with the robot pose as:

$$\begin{aligned}\nabla_{\mathbf{r}} \mathbf{d}_p &= \begin{bmatrix} \frac{\Delta x}{d} & \frac{\Delta y}{d} & 0 \end{bmatrix} \cdot \mathbf{J}_1(\mathbf{x}_r, \mathbf{x}_s) \\ \nabla_{\mathbf{p}} \mathbf{d}_p &= \begin{bmatrix} -\frac{\Delta x}{d} & -\frac{\Delta y}{d} \end{bmatrix}\end{aligned}$$

where  $\mathbf{x}_s$  is the pose of the sonar sensor with respect to the robot frame, and  $\mathbf{x}$  is the pose of the robot within the map frame.

When the corresponding feature is a line feature, then the echo distance  $d_l$  is predicted by computing the distance of the sonar sensor from the mapped line as follows:

$$\begin{aligned}d_l &= \rho_l - \cos \phi_l x_{rs} - \sin \phi_l y_{rs} \\ x_{rs} &= \cos \theta_r x_s - \sin \theta_r y_s + x_r \\ y_{rs} &= \sin \theta_r x_s + \cos \theta_r y_s + y_r\end{aligned}$$

Likewise its Jacobians are computed:

$$\begin{aligned}\nabla_{\mathbf{r}} \mathbf{d}_l &= \begin{bmatrix} -\cos \phi_l & -\sin \phi_l & 0 \end{bmatrix} \cdot \mathbf{J}_1(\mathbf{x}_r, \mathbf{x}_s) \\ \nabla_{\mathbf{l}} \mathbf{d}_l &= \begin{bmatrix} 1 & (\sin \phi_l x_{rs} - \cos \phi_l y_{rs}) \end{bmatrix}\end{aligned}$$

The predicted echo distances could be used for data association with the measured echos; however, it is more robust to use the measured features for matching with mapped features. There are many spurious echos that are formed from moving objects, multi-path, or cross-talk between sensors. By grouping echos into sets

corresponding to lines or points, most spurious echos are disregarded; the remaining, likely reasonable, echos are used to update the Kalman filter.

### 4.4.3 Feature Initialization

New features are initialized into the map maintained by the Kalman filter from measured features that do not correspond to any existing map feature. Multiple vantage-points are needed to initialize each new feature.

Only echos with corresponding saved vantage-points can be used to initialize the feature in the Kalman filter. The echos from oldest and newest vantage-points are used with Equations 4.2,4.3 or Equations 4.5,4.6 to initialize the feature's state. Subsequently the remaining echos with corresponding vantage-points are used to update the filter in the same manner as described above.

The Jacobians of the feature initialization function could be derived by directly differentiating the function; however, it is easier to convert the Jacobians of the measurement prediction functions. For example, the feature initialization function  $\mathbf{g}(\cdot)$  and measurement function  $\mathbf{h}(\cdot)$  are related as follows:

$$\mathbf{y} = \mathbf{g}(\mathbf{x}, \mathbf{z})$$

$$\mathbf{z} = \mathbf{h}(\mathbf{x}, \mathbf{y})$$

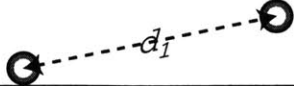
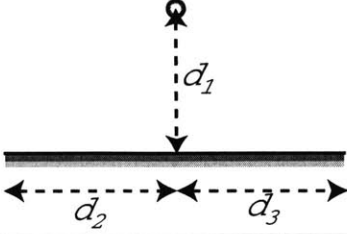
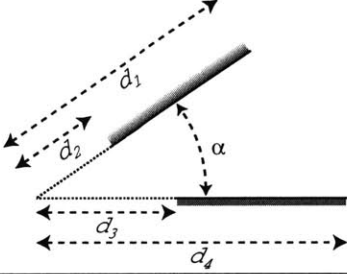
where  $\mathbf{x}$  is the robot state,  $\mathbf{y}$  is the new feature state, and  $\mathbf{z}$  is the vector of initial measurements. The Jacobians of the initialization function  $\nabla \mathbf{g}$  are related to the Jacobians of the measurement functions  $\nabla \mathbf{h}$ .

$$\nabla_{\mathbf{x}} \mathbf{g} = -(\nabla_{\mathbf{y}} \mathbf{h})^{-1} \nabla_{\mathbf{x}} \mathbf{h}$$

$$\nabla_{\mathbf{z}} \mathbf{g} = (\nabla_{\mathbf{y}} \mathbf{h})^{-1}$$



Table 4.1: Defining map signature elements between pairings of line and point features.

Pairing	Geometry	Element
Point-Point		$[d_1]$
Point-Line		$\begin{bmatrix} d_1 \\ d_2 \\ d_3 \end{bmatrix}$
Line-Line		$\begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ d_4 \\ \alpha \end{bmatrix}$

When exactly two echos are used to initialize a point or line feature, then the Jacobian  $\mathbf{H}_y$  is square. When there is a nonzero baseline between the vantage-points, then this Jacobian will also be invertible.

#### 4.4.4 Performance Metric

The *Atlas* performance metric is computed exactly in the same manner as with the laser lines implementation. Please see Section 3.3.3 for details.

## 4.5 Map Matching and Signatures

The map matching module for the sonar implementation is nearly identical to that from the laser lines implementation. The main difference is in the design of the

signatures.

In this implementation, the maps consist of 2D point and line features. The elements used in creating a map signature are pairings of non-parallel lines, point-line pairs and point-point pairs drawn from the map. For each pairing, the signature element consists of distances and/or angles independent of the map-frame's orientation and location. Table 4.1 defines the transformationally invariant metrics used for the three species of pairings.

The number of signature elements to compare when matching maps with  $n$  features is  $O(n^2)$  which may lead to  $O(n^4)$  matches that must be performed. However, the number of matches that need to be tested can be reduced to  $O(n^2)$  by sorting the signature elements into a canonical order, which then reduces the total computational burden to  $O(n^2 \log n)$ .

Each pair of matching signatures defines a potential transformation which aligns the two map frames. Subsequently each matching signature pair is scored by applying the alignment transformation and counting the number of map features that correspond. The correspondences from the highest-scoring signature pair are used to optimize the alignment transformation between the map frames and to determine its covariance.

Unfortunately, since the map contains point features, the alignment optimization cannot be solved with a linear system of constraints. Instead the optimization is solved with an iterated linearized solution. The constraints from each feature pair are linearized about the previous solution, and a correction is solved for. In practice, two iterations are sufficient for convergence, since the initial transformation from the signature pair is close to the optimal solution.

## 4.6 Experimental Results: Killian Court

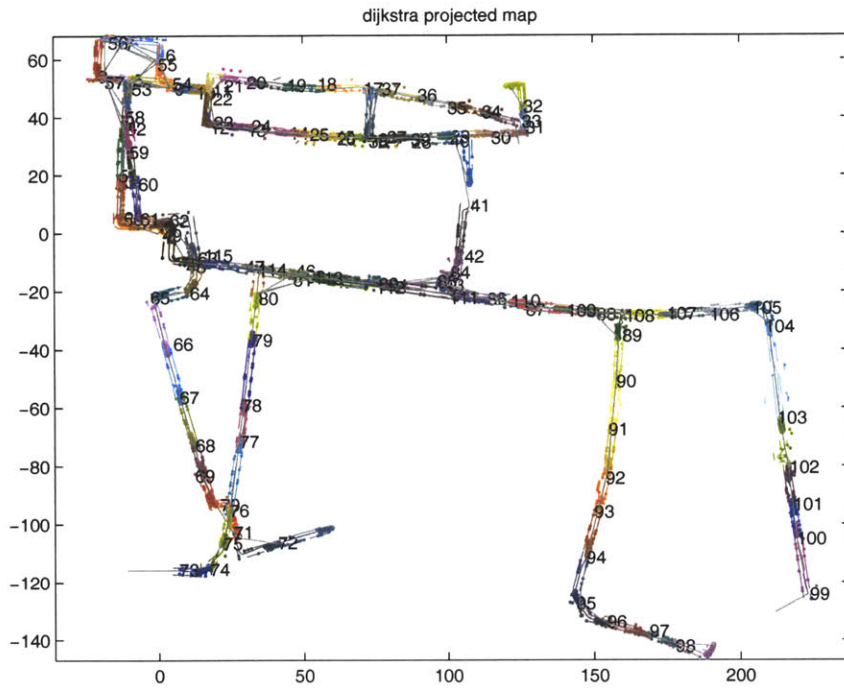
The same experiment as in Section 3.5.1 is used to evaluate the sonar *Atlas* implementation. The robot's local mapping module was configured to save 10 poses each 0.5 meters apart. The capacity was set such that a maximum of 20 features could be mapped in each map-frame.

Figure 4-4(a) shows the resulting maps under a Dijkstra projections with the first map as root, where as Figure 4-4(b) shows the result of applying the global optimized map projection. A total of 115 maps were built, which is more than the laser implementation used because the sonar maps tended to be a bit smaller.

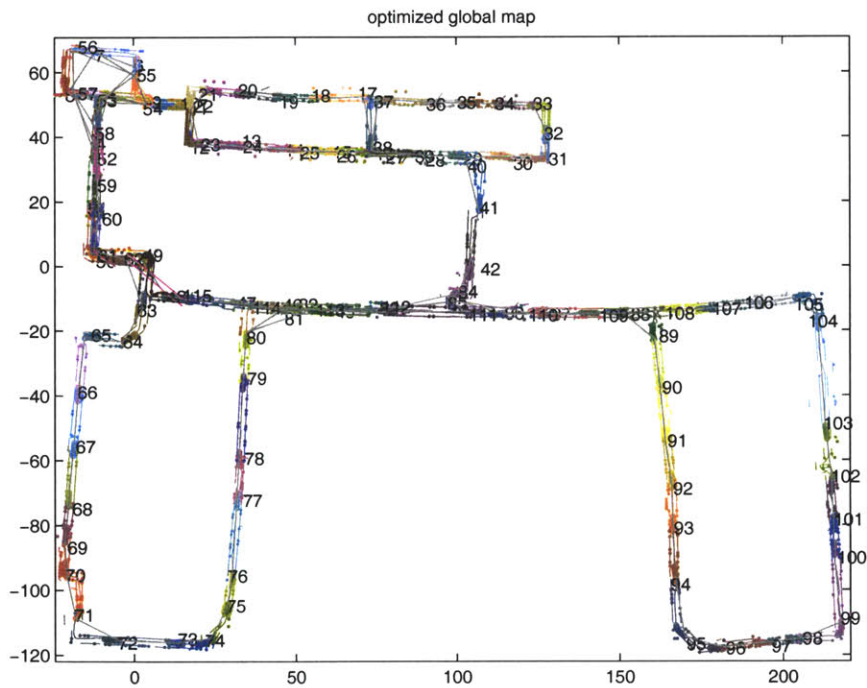
The adjacency matrix is displayed in Figure 4-5(b) and indicates the loops that have been closed. Since relocation has not been implemented when using sonar alone, all genesis map edges are sequential.

The histograms of the Kalman filter update residuals (Figure 4-6) show the performance of the local maps. The residuals are from the difference between the measured range measurements and the predicted range measurement computed from the Kalman filter robot state and associated map point or line. The residuals when taken together have a mean squared error of  $(2.7cm)^2$  and  $(1.3cm)^2$  on the point and line features, respectively. The residuals on the echos when measuring point features is large than that when measuring line features because there are more modeling errors on the points. The fact that the shape of the line residual histogram is not symmetric is due to some corners in the environment incorrectly being classified as lines; however, there are not enough misclassifications to severely affect performance.

Figure 4-7 shows the histograms of the residuals of the differences between the optimized projected map origins and the corresponding *Atlas* edge transformations. The standard deviation of the errors are  $13.8cm$ ,  $10.6cm$ , and  $1.3^\circ$ , for the  $x$ ,  $y$ , and  $\theta$  parameters, respectively.

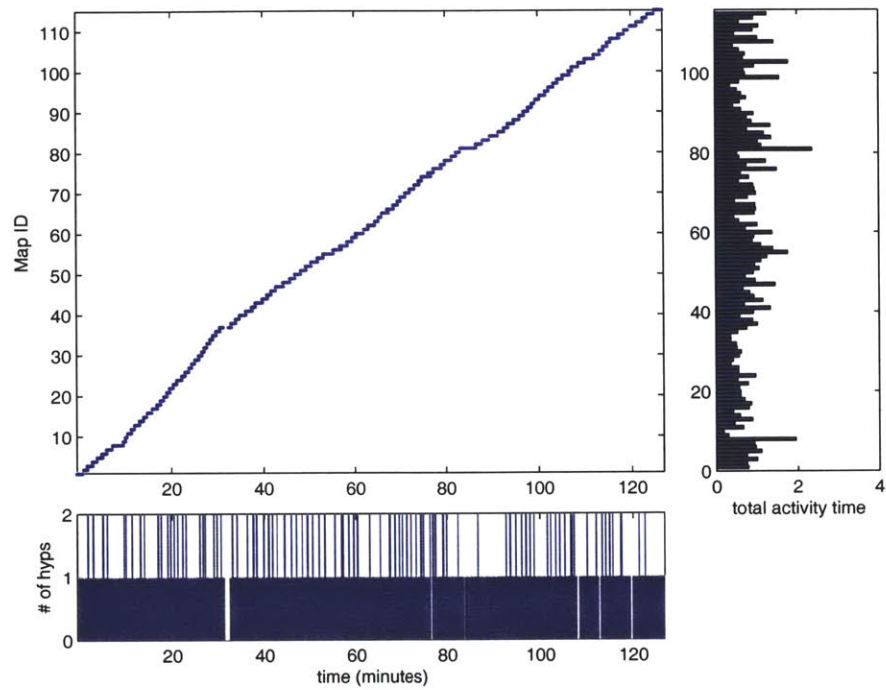


(a)

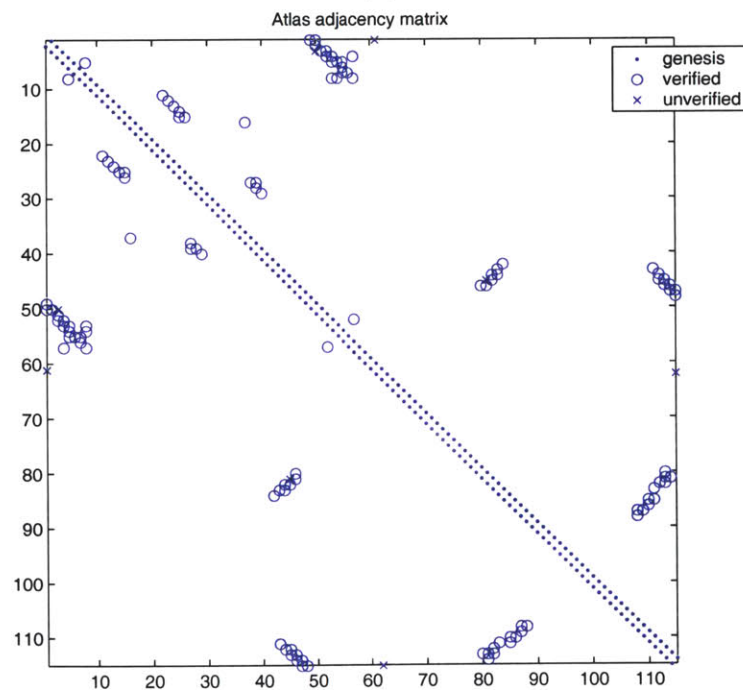


(b)

Figure 4-4: (a) Dijkstra projection and (b) global optimized map projection for processing of sonar data on the Killian Court data set. Each local map is drawn in a different color and labeled with the map id next to its coordinate origin. The valid *Atlas* edges are drawn in black whereas unverified edges are drawn in magenta.



(a)



(b)

Figure 4-5: (a) Map ID vs. time, total activity vs. map ID, and the number of active hypotheses vs. time for the sonar feature-based SLAM processing. (b) *Atlas* adjacency matrix. Dots indicate genesis edges, circles indicate verified edges, and crosses indicate unverified map-match edges.

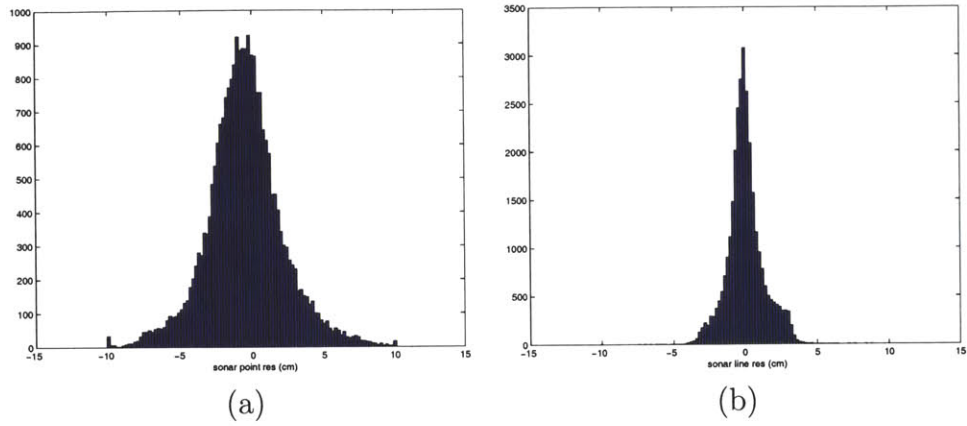


Figure 4-6: The histograms of the Kalman filter measurement residuals for (a) point updates and (b) line updates.

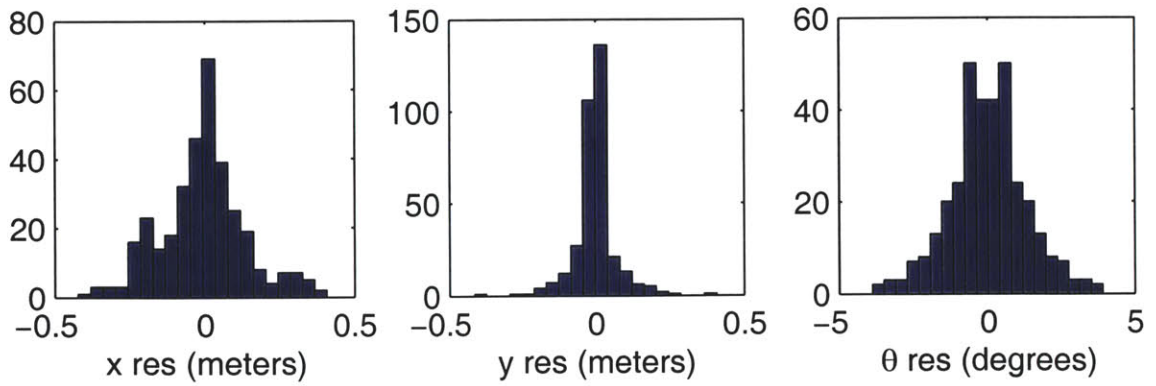


Figure 4-7: The histograms of the Global map projection optimization residuals for the  $x$ ,  $y$ , and  $\theta$  parameters of each map-frame transformation.

# Chapter 5

## Atlas with 2D Laser Scan-matching

### 5.1 Introduction

Scan-matching is another local navigation module for *Atlas*. It uses the laser scan points directly instead of extracting features such as lines from the scans. Scan-matching has been a popular approach to SLAM with dense laser scanner data [37, 27, 53].

In this chapter, a novel implementation that combines scan-matching with a linear Gaussian state estimation formulation [48] is presented. The key to the method is to use past vehicle poses as elements in the SLAM state vector [34], and using scan matching to formulate measurements that are a function of two different vehicle poses. This provides an effective means to obtain uncertainty estimates for SLAM with scan-matching, an issue that has been identified as problematic in previous research with scan matching [60].

The map representation is a collection of laser scans. Each laser scan is associated with a saved robot pose. The map state vector is the concatenation of the current robot pose and all saved robot poses. The joint probability of all the poses is maintained with a single multivariate Gaussian probability density function. Scans are

matched with the Iterative Closest Point (ICP) algorithm [5]. The ICP algorithm produces the relative transformation between two scans, its uncertainty, and a score of how much the scans overlap.

## 5.2 Iterative Closest Point Scan-matching

ICP is a simple algorithm used to align two clouds of points with unknown correspondences. The algorithm proceeds in two steps. In the first step, point correspondences are found by matching each point from one scan to its closest point in the other scan. The second step then finds a coordinate transformation that minimizes the error between the matched point correspondences. These two steps are repeated until convergence is achieved or a maximum number of iterations has occurred.

The pseudocode for the ICP scan matching algorithm follows:

```

ICP-SCAN-MATCH(scana, scanb, Tab)
1  for i ← 1 to maxiter
2  do μab ← FIND-CLOSEST-POINTS(scana, scanb, Tab)
3     Tab ← UPDATE-TRANSFORM(μab, scana, scanb, Tab)
4  Σab ← COMPUTE-COVARIANCE(μab, scana, scanb, Tab)
5  return {Tab, Σab}

```

where scan<sub>a</sub>, and scan<sub>b</sub> are the laser scans, T<sub>a</sub><sup>b</sup> is the relative coordinate transform between the scans' centers, μ<sub>ab</sub> is the correspondence between the points of the two scans, and Σ<sub>ab</sub> is the covariance of the final alignment.

### 5.2.1 Normals

The ICP algorithm can be improved when surface normals for each scan point are available. The normals are used in the first step to limit matches between points with



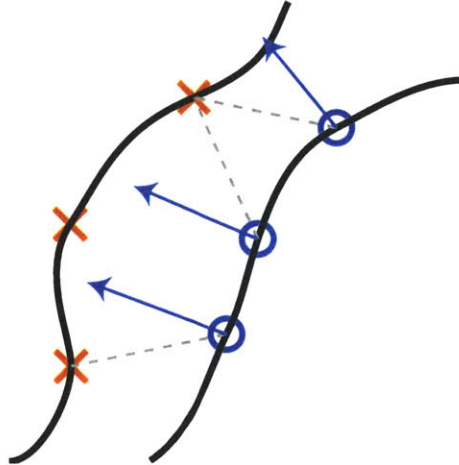


Figure 5-1: Iterative Closest Point. The main assumption is that the points are sampled from a surface. Consequently after determining correspondences between closest point, the “force” on each point is directed along the surface normal.

normals that are not pointing in the same general direction. In the second step, the error of the point match is only considered in the direction of the surface normal. This alleviates the issue when a surface is not sampled at exactly the same points in the two scans. (See Figure 5-1.)

The normals for each scan point are approximated using the assumption that points are sampled from a continuous surface. Given the sequential points  $A$ ,  $B$ , and  $C$ , the normal for point  $B$  is taken as the average of the normal for lines  $\overline{AB}$  and  $\overline{BC}$ . (See Figure 5-2.) The continuous surface assumption does not hold at occlusion boundaries or at step edges in the scan. Therefore if the distance between points  $A$  and  $B$  or between  $B$  and  $C$  is too great, only the shorter line is used to determine normal. When both distances are too large, i.e. when a thin pipe is measured, the normal is simply set to point towards the origin of the scan.

The quantization and sensor noise present in the SICK PLS scanner makes the computation of normals troublesome. The scanner has approximately  $7cm$  standard deviation measurement noise, and the readings are quantized to  $5cm$ . Since adjacent scan points are separated by a distance that varies from about  $2cm$  to  $30cm$ , the noise has a significant effect ( $13^\circ$  to  $74^\circ$ ) on the normal directions.

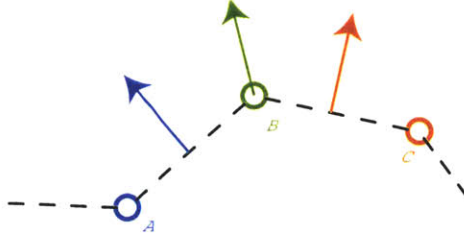


Figure 5-2: The normal for point  $B$  is computed from the average of the normals of the lines  $\overline{AB}$  and  $\overline{BC}$ .

To mitigate this problem, the ranges of each scan are prefiltered to remove noise and smooth the resulting normal directions. The ranges are treated as a 1D sequence filtered by a 24 tap FIR low-pass filter with a cutoff frequency of about  $0.2\pi$ . Since the filter is so long, care must be taken not to smooth away valid range discontinuities arising from, for example, object boundaries. Therefore, the filter is modified by a mask that zeros out the taps corresponding to ranges that are more than a thresholded difference from the center tap. Consequently, when smoothing a particular range measurement, only the nearby ranges that are most likely to be from the same object are used as support in the filter. The edge preserving effects of this modified filter are crucial to computing reliable scan point surface normals.

## 5.2.2 Transformation Update

In each ICP iteration, after the scan point correspondences have been computed, the prior alignment transformation  $T_a^b$  is updated. The update should minimize the alignment error given the current correspondences. The formula for the alignment error uses the difference between each corresponding scan point in the direction of the normals:

$$E_{\text{align}} = \sum_{(a,b) \in \mu_{ab}} (n_a^T (p_a - T_a^b p_b))^2 \quad (5.1)$$

where  $p_a$  and  $p_b$  are the scan point vectors,  $n_a$  is the normal of point  $a$ ,  $T_a^b$  is the alignment transform, and  $E_{\text{align}}$  is the total alignment error.

The objective is to determine the transformation that minimizes the alignment error. There is no linear solution for the optimal transformation because the rotation component of the transformation introduces a nonlinearity. Instead the error function is linearized about an initial guess for the transformation, and an optimal linear correction is solved for. The error Equation 5.1 can be written in vector form:

$$E_{\text{align}} = \mathbf{h}(T_a^b)^T \mathbf{h}(T_a^b) \quad (5.2)$$

where each row of the vector function  $\mathbf{h}()$  is a term from the summation in Equation 5.1. The linearized error can then be expressed with a first order Taylor expansion:

$$\mathbf{h}(T_a^b) \approx \mathbf{h}(T_{a0}^b) + \mathbf{H} \begin{bmatrix} dx \\ dy \\ d\theta \end{bmatrix} \quad (5.3)$$

where  $\mathbf{H}$  is the Jacobian of  $\mathbf{h}$  with respect to  $x, y$ , and  $\theta$ .

Differentiating Equation 5.2 with respect to the unknowns ( $x, y$ , and  $\theta$ ), using the approximation in Equation 5.3, and setting the expression to zero yields the linear

system that is solved for the optimal linear solution.

$$0 = 2\mathbf{H}^T \left( \mathbf{H} \begin{bmatrix} dx \\ dy \\ d\theta \end{bmatrix} + \mathbf{h}(T_{a0}^b) \right) \quad (5.4)$$

$$\begin{bmatrix} dx \\ dy \\ d\theta \end{bmatrix} = -(\mathbf{H}^T \mathbf{H})^{-1} \mathbf{H}^T \mathbf{h}(T_{a0}^b) \quad (5.5)$$

where each row  $H_i$  of  $\mathbf{H}$  is

$$H_i = \begin{bmatrix} -n_x & -n_y & (n_x(p_{bx} \sin \theta + p_{by} \cos \theta) + n_y(-p_{bx} \cos \theta + p_{by} \sin \theta)) \end{bmatrix}$$

The updates are added to the transformation parameters, and subsequently the linearization procedure may be iterated a few times to ensure convergence.

$$\begin{bmatrix} x \\ y \\ \theta \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \\ \theta_0 \end{bmatrix} + \begin{bmatrix} dx \\ dy \\ d\theta \end{bmatrix}$$

### Robust outlier weighting

To mitigate the effect of outliers in the data (from non-overlapping regions of the scans, or moving objects) the error of the matches is modified by a Lorentzian which smoothly down-weights errors when they become too large. The Lorentzian weighting is equivalent to assuming a Cauchy (instead of a Gaussian) error distribution.

The alignment error equation is modified by the Lorentzian  $\rho(\cdot)$ .

$$\begin{aligned} \rho(x) &= \log(\bar{r}^2 + x^2) \\ E_{\text{robust}} &= \sum \rho(h_i(T_a^b)) \end{aligned}$$

where  $\bar{r}$  defines the soft outlier threshold.

Again the parameters which minimize the error are computed by differentiating the error function and setting the results to zero. The only difference between using Equation 5.6 and Equation 5.2 is that the Lorentzian adds the term  $\frac{1}{(\bar{r}^2+h_i^2)}$  to each row. Each term can be seen as a weight for the constraint from each point correspondence that depends on the initial error. The weights are collected into a diagonal matrix  $\mathbf{W}$  and the weighted least squares solution is computed.

$$\mathbf{W} = \begin{bmatrix} \dots & & 0 \\ & \frac{1}{(\bar{r}^2+h_i(T_{a0}^b)^2)} & \\ 0 & & \dots \end{bmatrix}$$

$$0 = \mathbf{H}^T \mathbf{W} \left( \mathbf{H} \begin{bmatrix} dx \\ dy \\ d\theta \end{bmatrix} + \mathbf{h}(T_{a0}^b) \right)$$

$$\begin{bmatrix} dx \\ dy \\ d\theta \end{bmatrix} = -(\mathbf{H}^T \mathbf{W} \mathbf{H})^{-1} \mathbf{H}^T \mathbf{W} \mathbf{h}(T_{a0}^b)$$

Even though the optimal minimization of this Cauchy error distribution is non-linear and requires multiple iterations for convergence, empirical evaluations have determined that one iteration suffices, since the minimization step is repeated in the iterations of the ICP algorithm anyway.

### Alignment Transformation Covariance

The covariance of the scan-match transformation is determined after the final ICP transformation update step. The final point correspondence is used with the alignment error Equation 5.1 to determine the average point error variance. Subsequently the point error variance is used in conjunction with the Jacobians of the alignment

error with respect to the transformation parameters to form the covariance of the alignment transformation.

The final point correspondence is determined by removing all point pairs from the last ICP iteration that exhibit a large error. The value of threshold for the large error is not critical; in this implementation is is taken to be about 30cm, which is 3 times the standard deviation of the laser scan point noise.

The point match error variance  $\sigma_p^2$  is computed as the sample variance of the terms in the summation of Equation 5.1.

$$\begin{aligned}\sigma_p^2 &= \frac{1}{N-1} \sum_{i=1}^N h_i (T_a^b)^2 \\ &= \frac{1}{N-1} E_{\text{align}}\end{aligned}$$

where  $N$  is the number of point correspondences.

The covariance of the transformation parameters  $[x, y, \theta]^T$  is computed by taking the covariance of the final update (Equation 5.5) using the fact that  $E[\mathbf{h}\mathbf{h}^T] = \sigma_p^2 \mathbf{I}$ .

$$\begin{aligned}\Sigma_{ab} &= E \left[ \begin{bmatrix} dx \\ dy \\ d\theta \end{bmatrix} \begin{bmatrix} dx & dy & d\theta \end{bmatrix} \right] \\ &= (\mathbf{H}^T \mathbf{H})^{-1} \mathbf{H}^T E[\mathbf{h}\mathbf{h}^T] \mathbf{H} (\mathbf{H}^T \mathbf{H})^{-1} \\ &= \sigma_p^2 (\mathbf{H}^T \mathbf{H})^{-1}\end{aligned}$$

The covariance of the transformation update is dependent on the number of corresponding scan points and the average error of each correspondence, as well as the geometry of the scans. For example, the covariance will be ill-conditioned when the normals of the scans points do not span a 2-dimensional space. This is particularly evident in environments such as long hall ways where only two parallel flat walls are

observed.

### 5.3 Pose Snapshot Kalman Filter

As in the Chapters 3 and 4, a Kalman filter is used to maintain the state and covariance of the current robot pose and the map states. But in the scan-match implementation the map representation is simply a collection of robot poses with their respective laser scans. (See Figure 5-3.) Therefore the filter state is comprised solely of robot poses. There are no direct map features maintained in the filter, and no features need to be extracted from the laser scans. Each saved robot pose maintains a “snapshot” comprised of the raw measurements from the corresponding time step, hence the term Pose Snapshot Kalman filter.

As with the Multiple Vantage Point Kalman filter from Chapter 4, only the current robot pose is propagated at each time step, and new saved poses are added to the map after a fixed distance of robot travel. Additionally, a new scan pose is added to the map if none of the previous scans matched more than 50 percent with the current scan. Maps are bounded naturally by limiting the number of saved scan poses (in this implementation to 15).

To update the Kalman filter from the new measurements, the current laser scan is matched (with ICP) in succession to each of the saved scans. If there is significant overlap between the current and any saved scan, the transformation from the ICP algorithm is treated as an observation of the relative pose between the two scans.

At every navigation iteration, the Kalman filter is propagated with the odometry measurements, and the current scan is used to update the filter. Robot propagation with odometry is processed as described in Section 3.3.1. Since updates from scan matches are so rich, it is sufficient to use the simple odometry model. It is even possible to utilize the scan-match filter without odometry measurements in environments

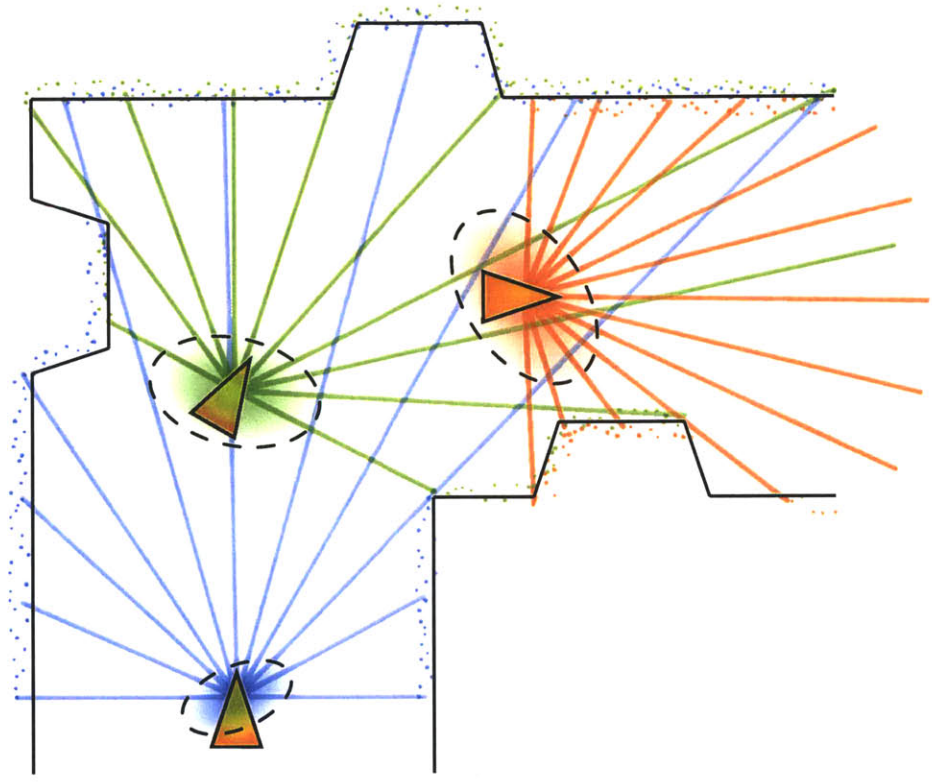


Figure 5-3: The map is represented by saved robot poses and their respective scans. The alignment between the current scan and any suitable saved scan is used to localize the robot.



with no ill-conditioned views. Results are presented in Section 5.5.4 where odometry measurements were not available. In this case, the robot pose is propagated with zero velocity from its previous position, and the propagation uncertainty is increased to account for motion from the maximum velocity of the vehicle.

### 5.3.1 Relocalization

The relocalization process for the scan-match pose snapshot filter is very straightforward. The *Atlas* edge transformation from the parent hypothesis is used to initialize the ICP iterations for the current scan. The pose snapshot with the largest overlap with the current scan is used to initialize the current pose of the robot into the Kalman state as if it were a new feature. Subsequent pose snapshots with significant overlap are updated normally.

### 5.3.2 Performance Metric

After every SLAM iteration the performance metric must be computed for the *Atlas* framework. The performance metric is based on how many scan-points in the current view match to previous scans, as well as the covariance of the current pose.

$$q = \frac{N_{\text{matched}}}{N_{\text{points}}} \cdot \frac{1}{1 + \sqrt{\frac{\det(\mathbf{P}_{xx})}{\det(\mathbf{P}_{xx}^*)}}}$$

where  $N_{\text{matched}}$  is the number of point in the current scan that were matched to previous scans,  $N_{\text{points}}$  is the total number of valid points in the current scan,  $\mathbf{P}_{xx}$  is the sub-block of the Kalman covariance matrix corresponding to the current robot pose, and  $\mathbf{P}_{xx}^*$  is the typical pose covariance for comparison with the current pose covariance. Note that the term in the performance metric for the current robot performance is identical to Equation 3.16.

The parameter  $\det(\mathbf{P}_{xx}^*)$  governs how much pose uncertainty is acceptable in the map hypothesis. Together with the map capacity, the typical robot uncertainty affects the size of the map. When the accepted uncertainty is large, then the robot may continue to use current map for a longer distance than if the parameter restricted the robot uncertainty to be small. This implementation uses the value  $(1^\circ \cdot 25\text{cm} \cdot 25\text{cm})^2$ .

## 5.4 Map Matching

When using scan-matching as the local mapping strategy for *Atlas*, the map matching module also uses ICP. Since there is typically a larger uncertainty in the initial arrangement between the maps, some extra steps are needed to insure that the ICP algorithm converges quickly and to the global minimum.

Firstly, all points from each saved scan are transferred to the base coordinate frame of the map. Then a down-sampled version of the scan is formed with the aid of a grid. The down-sampled scan consists of the average of all the points from the original scans that fall in each grid cell. A modified ICP algorithm subsequently processes the down-sampled scans. The ICP is modified to allow matches, not just between nearest pairs of points, but from all points in the second scan that are within a distance threshold of points in the first scan. These multiple point correspondences enable the algorithm to find a transformation within the “catchment basin” of the global minimum error match.

Once the modified low-resolution ICP match has converged, its output transformation will be slightly biased by the inclusion of multiple correspondences for each point. This transformation is used, however, to initialize a regular ICP match using the full resolution scans. Since the full resolution scan is initialized with a transformation that is a much better guess, it is less likely to get stuck in a false local minimum of the error function.

Unlikely map matches can be quickly detected after the low resolution ICP converges (or doesn't) by thresholding the percentage of overlap between the maps. It is often unnecessary to attempt the computationally intensive full-resolution match.

## 5.5 Experimental Results

The scan-match *Atlas* implementation is evaluated using the same datasets as described in Chapter 3. In addition, results from processing two third-party data sets are also shown. The first third-party data set is from an outdoor experiment, run in Victoria Park, Sydney, courtesy J. Guivant and E. Nebot. The second third-party data set is from an underground coal mine in Pennsylvania, courtesy S. Thrun.

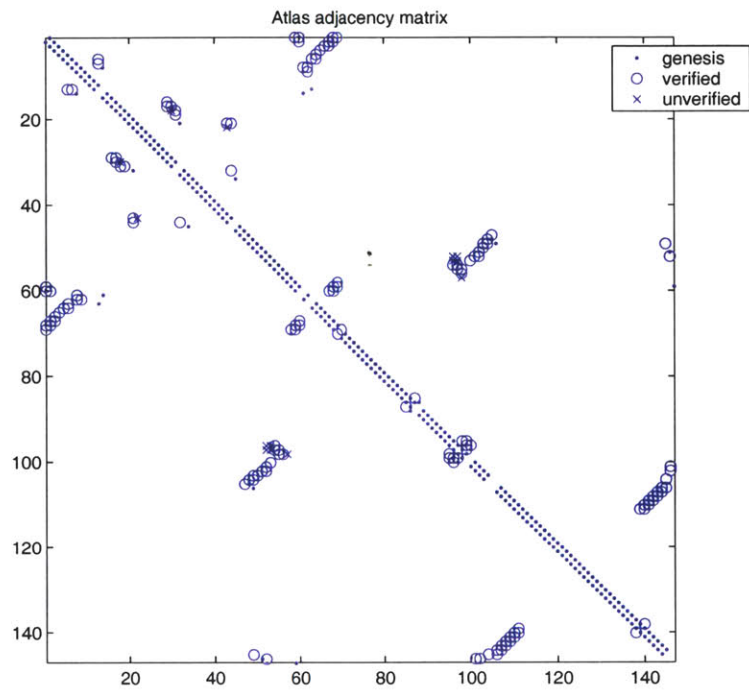
### 5.5.1 Killian Court

Figure 5-4 shows results for the Killian Court experiment (Section 3.5.1, page 85) but using laser scan-matching as the local SLAM module. The maximum number of vehicle poses in a map-frame was set to fifteen. Figure 5-4 shows the global optimized map, Figure 5-5(a) shows the map adjacency matrix, and Figure 5-5(b) shows the active map ID vs. time. The visual quality of the final optimized map is better using scan matching instead of feature-based SLAM, however the scan-matching approach would not be successful with reduced quality data, such as sonar.

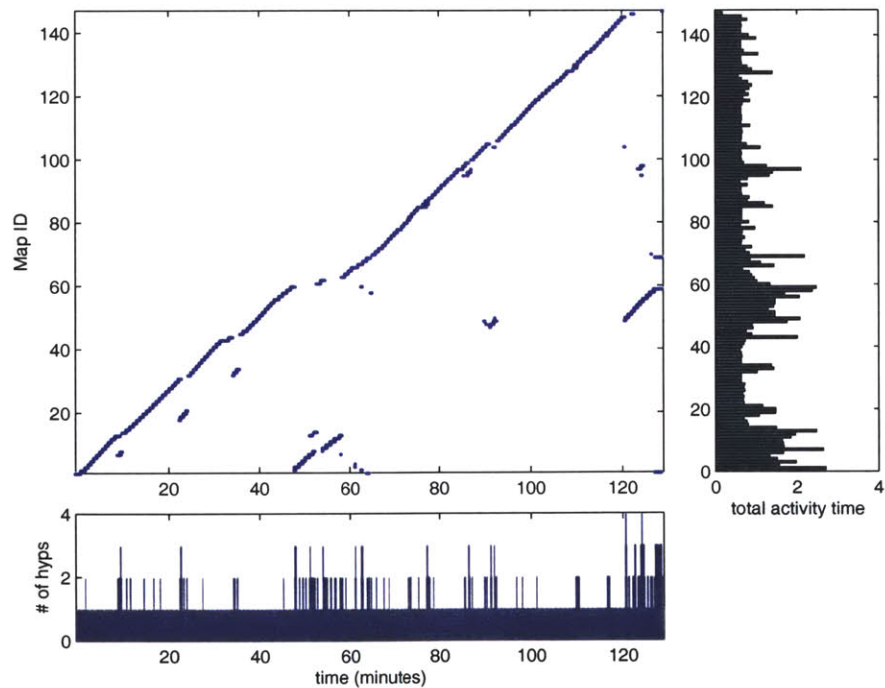
The histograms of the pose update residuals in the scan-match Kalman filter (Figure 5-6) show the performance of the local maps. The standard deviation of the residuals are  $2.2cm$ ,  $3.3cm$ , and  $0.37^\circ$ , for the  $x$ ,  $y$ , and  $\theta$  parameters, respectively.

The global map performance can be characterized by the residuals from the global optimized map projection. Figure 5-7 shows the histograms of the residual of the *Atlas* edge transformations with respect to the projected pose of each map-frame.





(a)



(b)

Figure 5-5: (a) *Atlas* adjacency matrix. Dots indicate genesis edges, circles indicate verified edges, and crosses indicate unverified map-match edges. (b) Map times.

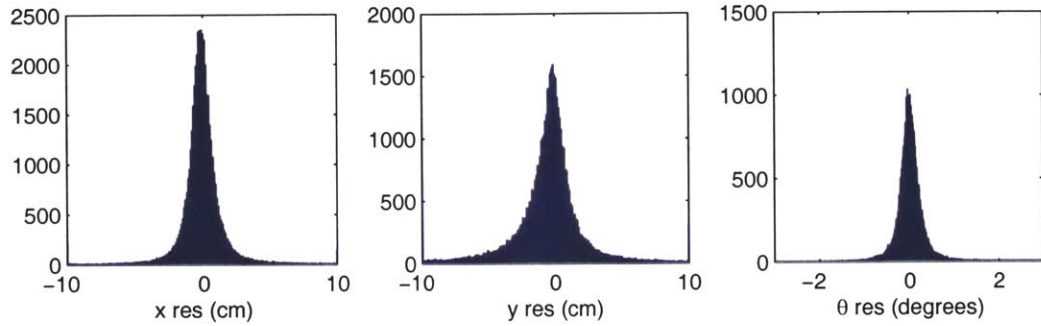


Figure 5-6: The histograms of the Kalman filter residuals for line updates on  $x$ ,  $y$ , and  $\theta$  parameters.

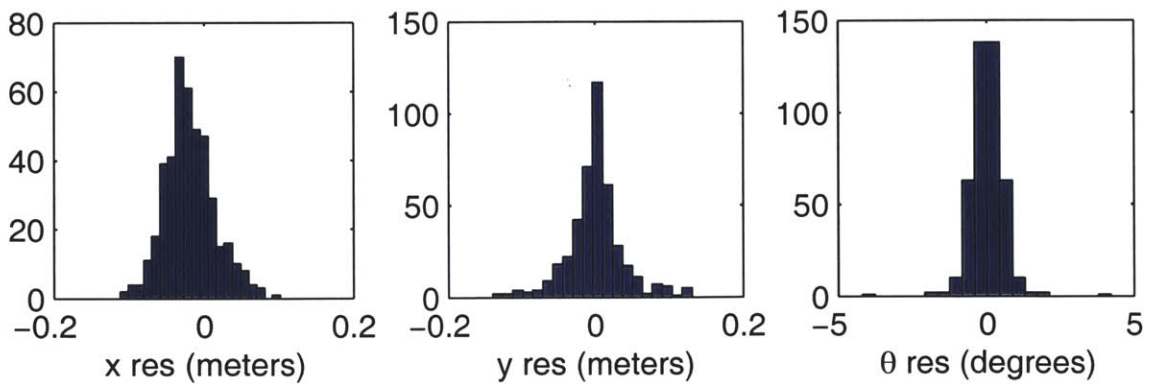


Figure 5-7: The histograms of the Global map projection optimization residuals for the  $x$ ,  $y$ , and  $\theta$  parameters of each map-frame transformation.

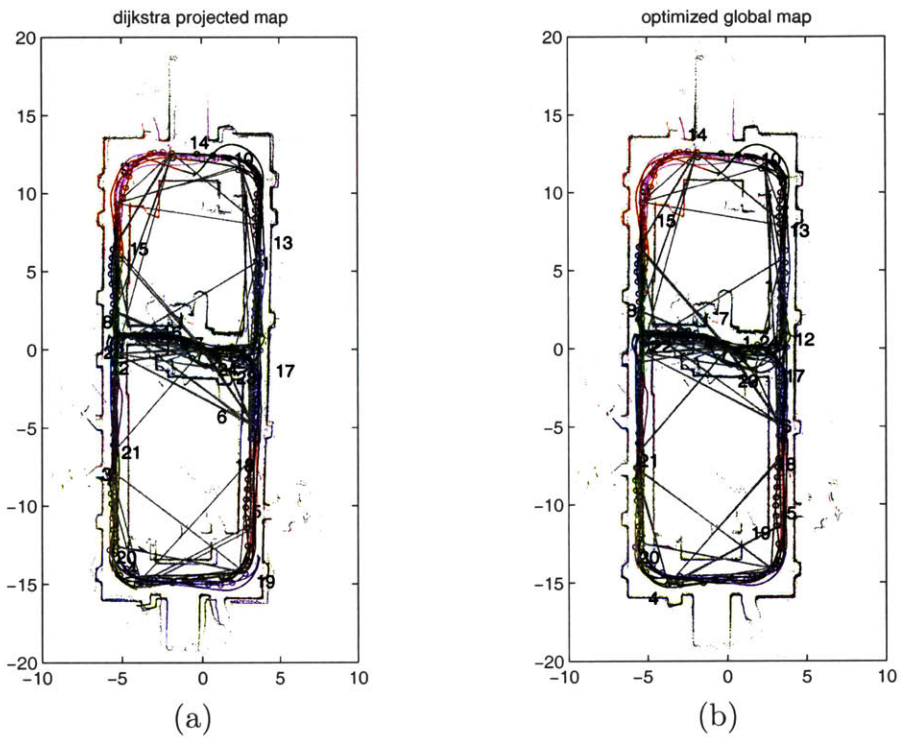


Figure 5-8: (a) Dijkstra projection and (b) global optimized map projection for scan-match processing of laser data on the Ten Loops data set. Each local map is drawn in a different color and labeled with the map id next to its coordinate origin.

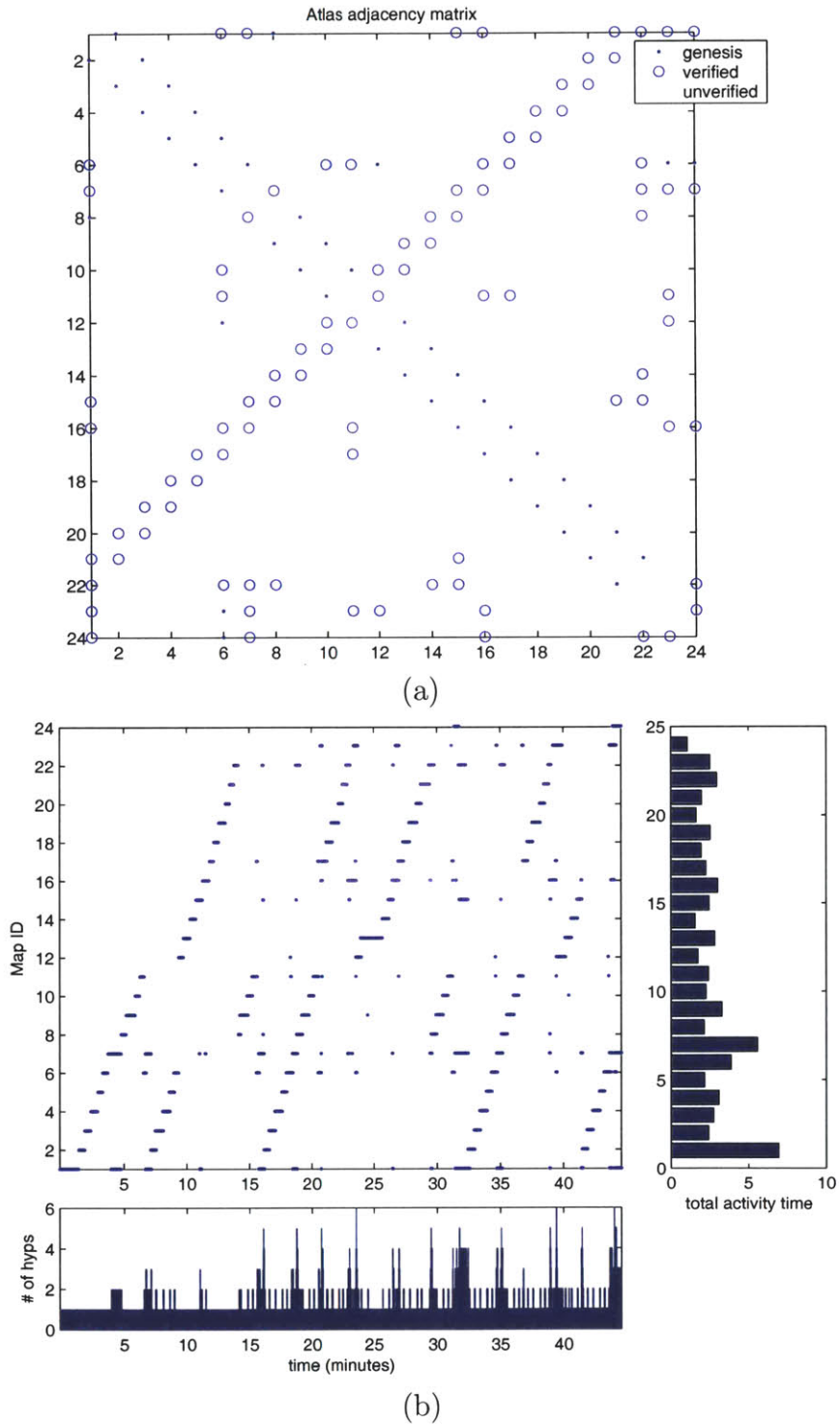


Figure 5-9: (a) *Atlas* adjacency matrix. Dots indicate genesis edges, and circles indicate map matched edges. (b) Map-frame genesis and activity. After every corridor is mapped from both directions, midway through the dataset, the creation of new maps ceases.



### 5.5.3 Victoria Park

Here are the results using scan-matching as the local SLAM module for an outdoor data set, made publicly available by E. Nebot of the University of Sydney. Results for this data set were first published by Guivant and Nebot [26], using the compressed filter, an efficient method for large-scale SLAM based on the Kalman filter. Please refer to Guivant and Nebot [26] for a detailed description of the experimental setup for acquisition of this data. Processing results for the same data set have been published by Liu and Thrun [35], using sparse extended information filters. These researchers have used trees as discrete point features in processing of this data set. In contrast, in this thesis the data is processed using scan-matching as the local mapping module. This data uses a sensor with a much longer range than the indoor scanner used in the Killian Court data set, and mounted on a vehicle with significantly different dynamics. The maximum number of vehicle poses in a map-frame was set to 15. Using algorithm parameters nearly identical to that used to obtain Figure 5-4, the output of *Atlas* is shown in Figure 5-10.

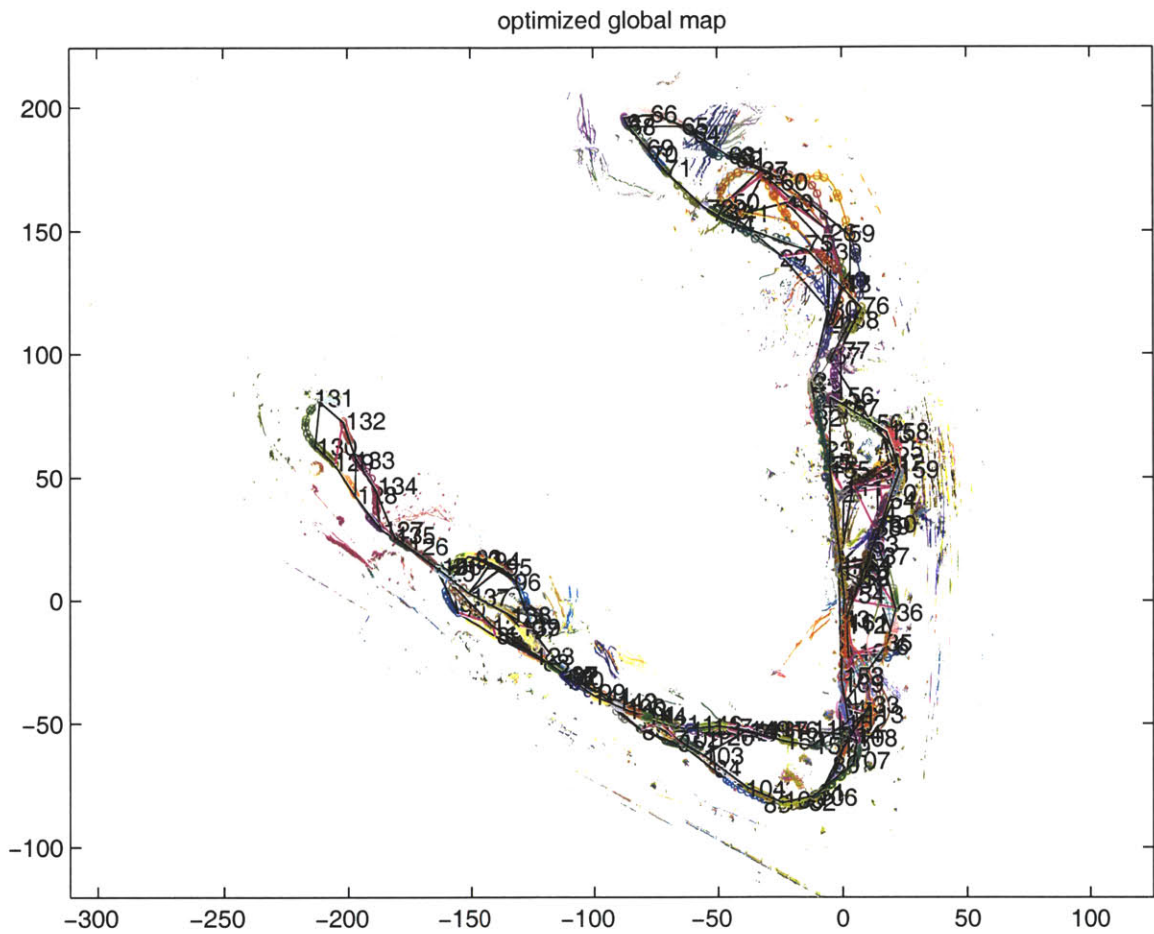
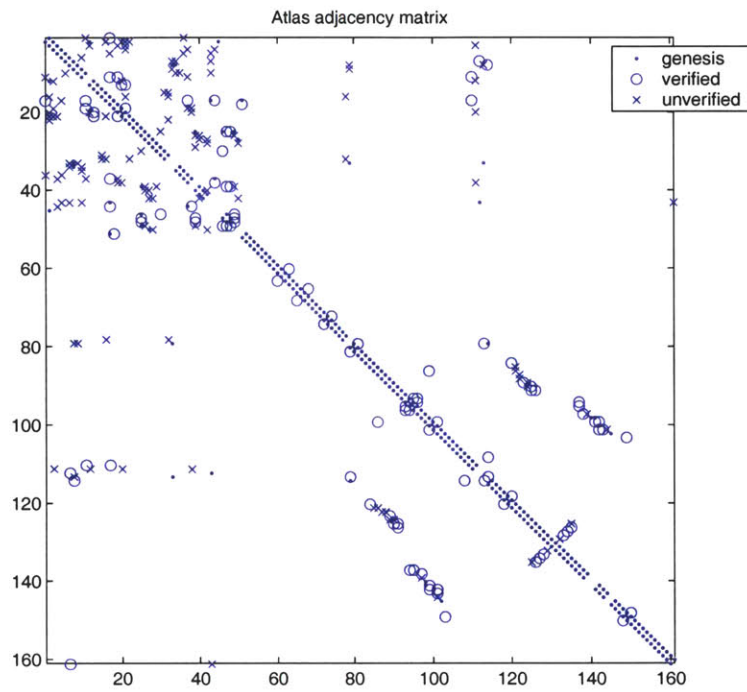
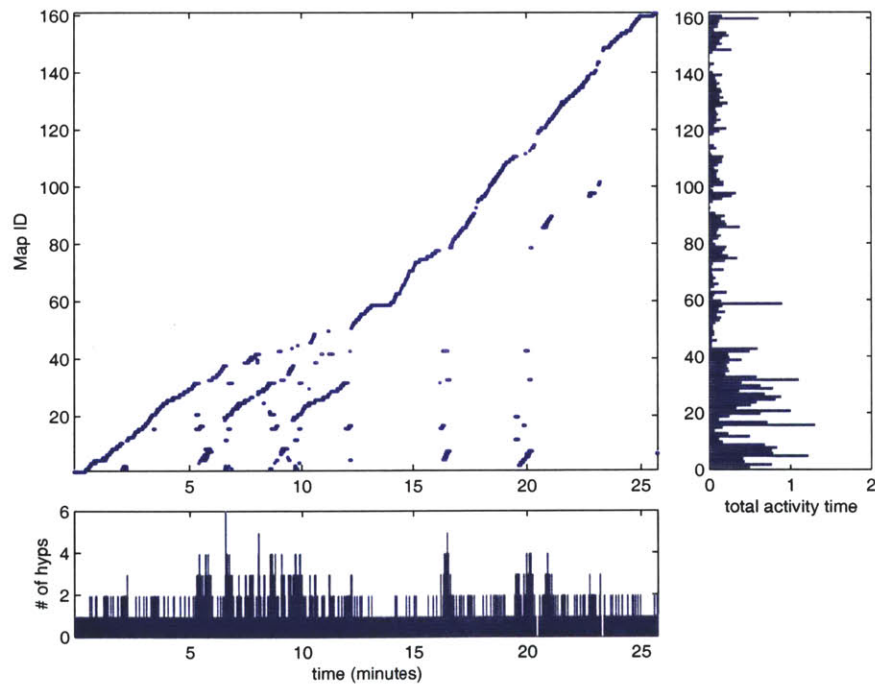


Figure 5-10: Optimized map of Victoria Park data set using *Atlas* Scan-Matching.



(a)



(b)

Figure 5-11: (a) *Atlas* adjacency matrix. Dots indicate genesis edges, circles indicate map matched edges, and crosses indicate unverified edges. (b) Map-frame genesis and activity.

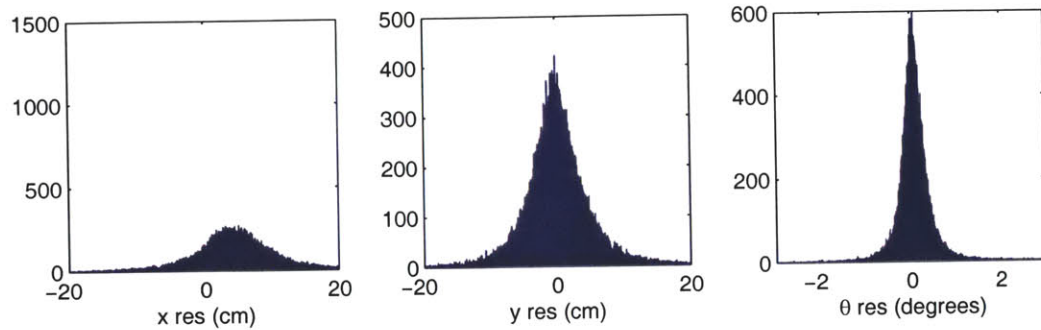


Figure 5-12: The histograms of the Kalman filter residuals for line updates on  $x$ ,  $y$ , and  $\theta$  parameters, in the Victoria Park data set.

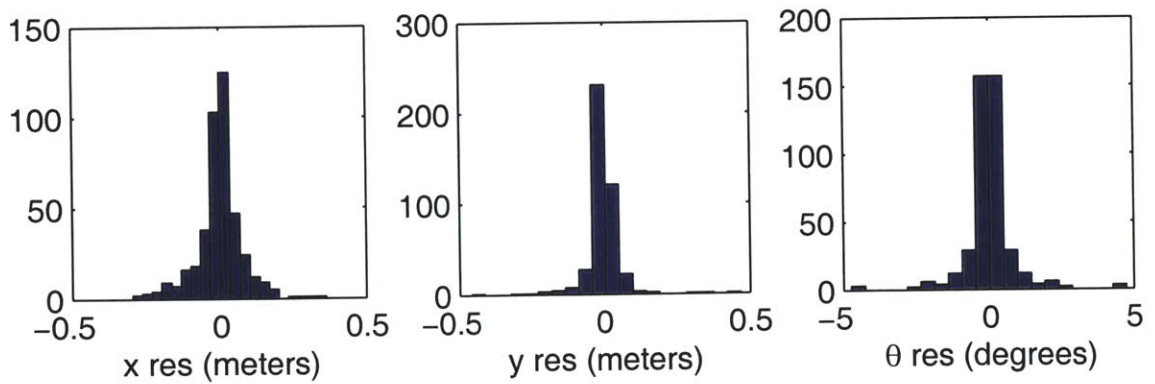


Figure 5-13: The histograms of the Global map projection optimization residuals for the  $x$ ,  $y$ , and  $\theta$  parameters of each map-frame transformation in the Victoria Park data set.

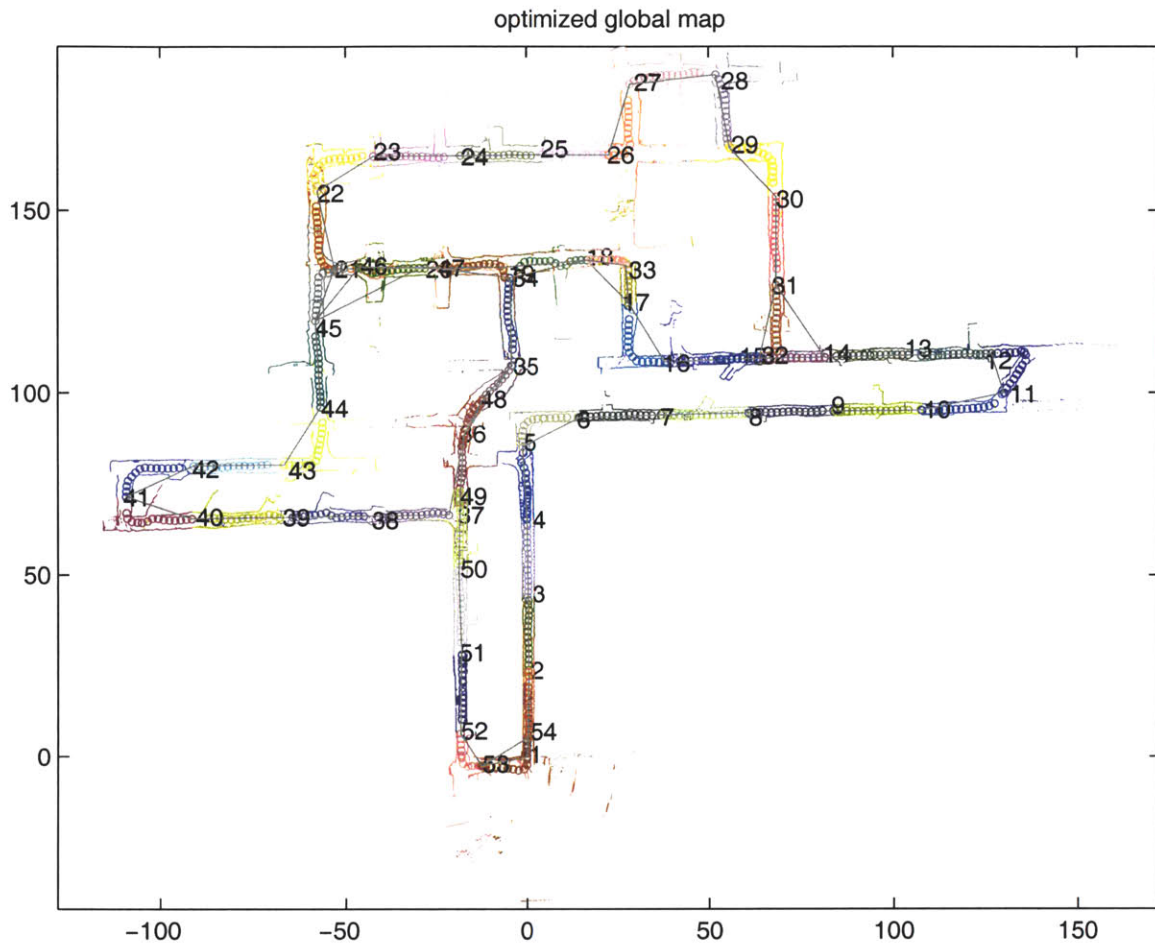


Figure 5-14: Optimized map of Pennsylvania Mine data set using *Atlas* Scan-Matching. (Data made available by S. Thrun (Thrun *et al.* [55]))

### 5.5.4 Pennsylvania Coal Mine

Here are results for an underground mine data set, made available by S. Thrun of Carnegie Mellon University. Processing results for this data set using sparse extended information filters are available in Thrun *et al.* [55]. This data set is more challenging because it consists exclusively of laser scanner data, with no odometry. To cope with the absence of odometry data, a motion model was assumed with the vehicle traveling forward at a velocity of 17.5 cm/sec (with a large corresponding uncertainty). For scan-matching, the maximum number of vehicle poses in a map-frame was fifteen. The output of *Atlas* is shown in Figure 5-14.

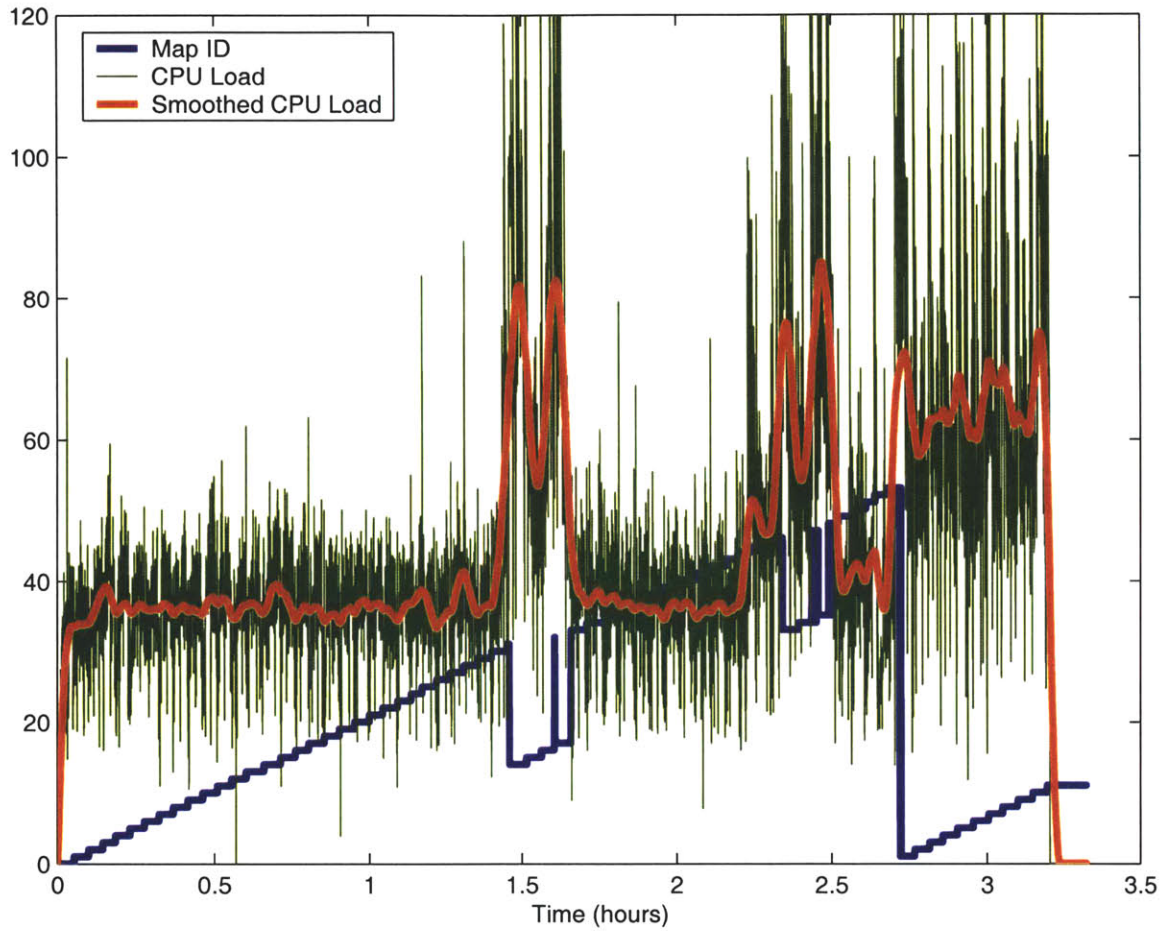
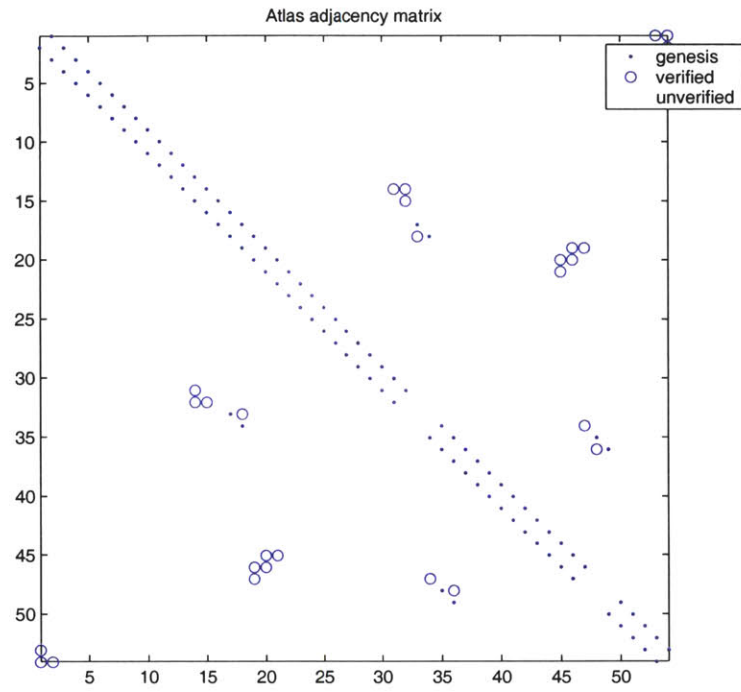
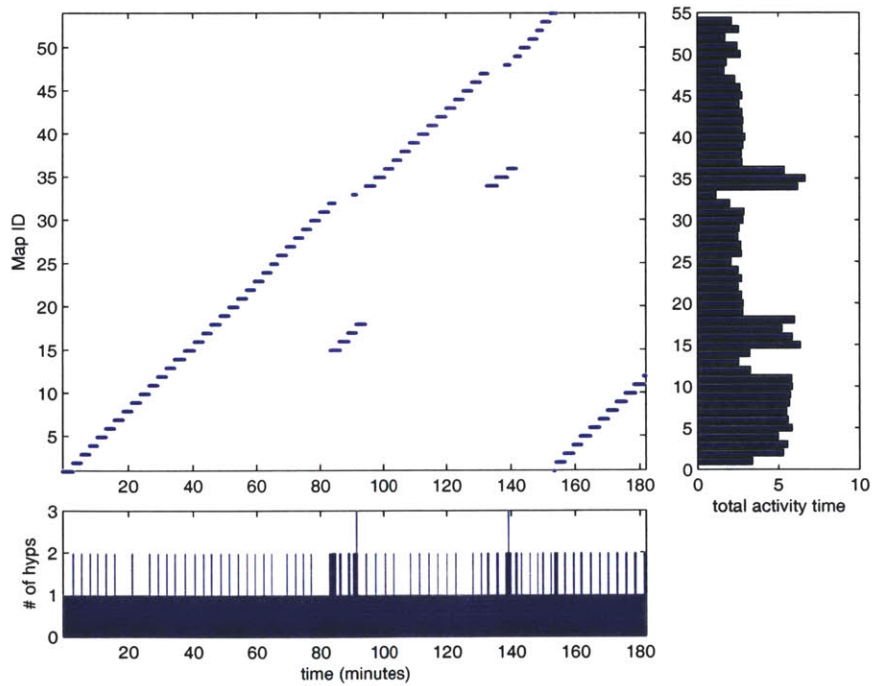


Figure 5-15: Processing time. 58 minutes of CPU time were required to process the 178 minutes of data. The processing rate is proportional to the number of scans matched. There is about a two-fold increase in revisited areas as opposed to new areas since there are saved scans in front of and behind the robot in the former.



(a)



(b)

Figure 5-16: (a) *Atlas* adjacency matrix. Red values indicate genesis edges, green indicates verified edges, and light blue indicates unverified map-match edges. (b) Map times.

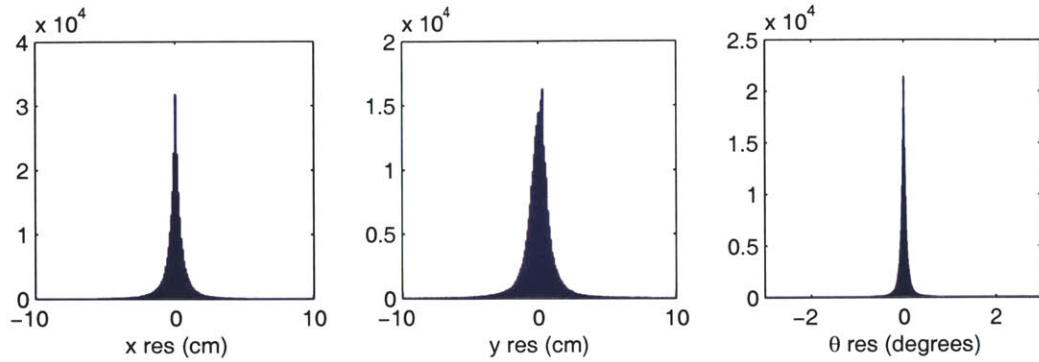


Figure 5-17: The histograms of the Kalman filter residuals for line updates on  $x$ ,  $y$ , and  $\theta$  parameters, in the Coal Mine data set.

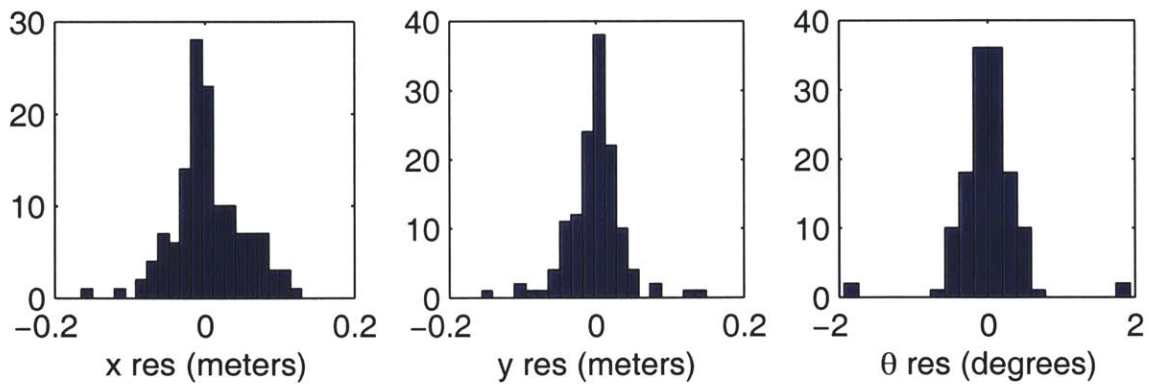


Figure 5-18: The histograms of the Global map projection optimization residuals for the  $x$ ,  $y$ , and  $\theta$  parameters of each map-frame transformation in the Coal Mine data set.



# Chapter 6

## Atlas with Omnidirectional Video

### 6.1 Introduction

The previous chapters have described the Atlas framework and described its successful implementation using laser line segments, sonar, and laser scan-match. The goal of this chapter is to provide a 3D implementation of the *Atlas* framework using computer vision. The beauty of *Atlas* is that it is a general purpose framework in which a variety of strategies can be employed for local mapping and map-matching. The potential benefits of large-scale 3D visual mapping are tremendous, but it is still a very challenging problem because of questions such as the best way to model the environment, perform data association, and handle lighting changes. Most practical vision systems use human input for the correspondences, or limit their scope to short camera excursions where correspondences are easy to determine.

This chapter presents a fairly simple implementation (which may not necessarily be optimal for local mapping) but demonstrates how the *Atlas* framework can be used to enable automated loop closing using computer vision.

The primary mapping sensor used in this implementation is a video camera attached to a parabolic mirror. The mirror provides the camera a 360° by 180° field

of view (FOV), and hence the setup is termed an omniscam, since the camera sees in nearly all directions. Omnidirectional video data offers the advantage of wide field of view imagery. In comparison to an equivalent motion with a smaller field of view, significantly longer tracks are obtained. While resolution is reduced, the better coverage provided by the omnidirectional camera results in more accurate ego-motion estimation and enables matching of local structure estimates to achieve efficient loop closure.

### 6.1.1 *Atlas* requirements

Any implementation of *Atlas* needs to specify several things: (1) local map representation and state estimation method, (2) map-matching technique, and (3) map relocation for hypothesis initialization. As in Chapter 4 when using sonar only, this chapter provides only a partial implementation of *Atlas* in that item (3), map relocation for hypothesis initialization, is not implemented because of the partial observability of the map with a vision sensor. This prevents old maps from being reused to explain current sensor measurements. Methods for extending the *Atlas* framework using local navigation modules with partially observable features to enable reuse of maps without relocation are discussed in Chapter 7.

For this *Atlas* implementation, the local map representation consists of vanishing points (VPs), 3-D line segments, and 3-D points. Correspondence is achieved over short temporal intervals using the KLT tracker for points and a novel 3-D line tracker using dynamic programming based on an ordering constraint. Tracked features serve as input to local mapping using bundle adjustment [29]. The approach is analogous to subsequence concatenation approaches that have appeared in the SFM literature [23, 62], but with much longer sequences than have been previously employed. For example, while Fitzgibbon and Zisserman performed bundle adjustment on local sequences of three images, in this chapter on the order of hundreds of images

are used in each local map. Map matching is performed using iterative closest feature (ICF) matching. Taken together, these three components — feature tracking, local bundle adjustment, and map matching — enable the key contribution of this chapter, the automated closure of large loops.

### 6.1.2 Assumptions

The approach assumes that the scene contains sets of stationary parallel lines. This is a valid assumption in many human-made environments such as indoor offices and hallways, and outdoor urban environments. The technique also assumes that calibration information for the omniscam is available. When there are many sets of parallel lines in the scene and the intrinsic camera calibration is known, then the vanishing points of the lines can be used to estimate the rotation of the camera. A nice feature of VPs is that they are invariant to translation, hence enabling the estimation of rotational errors to be decoupled from the estimation of translational errors.

A critical decision in algorithm development is the choice between batch and recursive methods. An automatic, recursive SFM algorithm must make data association decisions such as “which measurements correspond to previously mapped features,” “which measurements correspond to new features,” and “which measurements are spurious?” The brittleness of most feature detection methods is well-known [36]; erroneous decisions about the origins of measurements can have disastrous consequences. Correspondence errors will lead to filter divergence. This motivates the development of methods that can make delayed decisions, assessing the consistency of data points obtained from multiple vantage points.

Batch algorithms for SFM often use techniques from robust statistics, such as RANSAC, to assess consistency across multiple images in a video sequence, but such a capability is harder to achieve in a recursive SFM implementation. This implementation adopts a similar philosophy, using a technique to delay decisions about the

origins of measurements and to perform consistent initialization of 3-D line and point features using data from multiple vantage points.

The issues of choice of representation and reliable extraction of features have been key issues in vision research [38]. Most SFM algorithms have employed points as features, extracted from images using techniques such as the SUSAN corner detector [49], with random sample consensus (RANSAC) used to determine the correspondence of points across image sequences. Using points as features, various batch SFM algorithms are summarized in Hartley and Zisserman [29] and Faugeras *et al.* [21].

## 6.2 Relationship to Previous Work

The development of SFM algorithms capable of running in real-time has been an important goal in the computer vision research community [29, 21, 57, 51, 14, 8]. Real-time SFM will enable applications such as (1) real-time navigation of mobile robots in unknown environments, (2) real-time capture of 3-D computer models using hand-held cameras, and (3) real-time head tracking in extended environments.

There is increasing interest in the development of structure from motion (SFM) algorithms capable of running in real-time [14, 8]. Real-time SFM will enable applications such as (1) real-time navigation of mobile robots in unknown environments, (2) real-time capture of 3-D computer models using hand-held cameras, and (3) real-time head tracking in extended environments.

The set of choices in the development of an SFM algorithm include:

- state estimation (batch vs. recursive);
- choice of representation (geometric vs. appearance-based);
- choice of features (points, edges, lines, curves, etc.);

- camera geometry (monocular, binocular, trinocular, omnidirectional; affine vs. projective, calibrated vs. uncalibrated cameras);
- representation and manipulation of uncertainty (Kalman filtering, sum of Gaussians, sequential Monte Carlo methods, etc.).

### 6.2.1 Feature Geometry

This implementation follows a geometric approach to computer vision. Projective geometry provides the underpinnings of the methods used to extract structure from the moving camera viewpoints. A comprehensive introduction to this material is provided in well known textbooks by Faugeras [20], Faugeras, Luong and Papadopoulos [21] or Hartley and Zisserman [29].

Appearance based methods provide a potential alternative. The lure of geometric approaches is that geometry is invariant to lighting changes, whereas appearance based approaches are not. Appearance based approaches which take advantage of rich color and texture information over large image areas are better suited for place recognition problems than geometric approaches. The appearance-based approaches can better answer “what” whereas geometry-based approaches better answer “where”. The incorporation of appearance-based techniques would greatly improve the scope for map matching and recognizing loop closures; however, this topic is left for future work as discussed in Chapter 7.

The approach in this chapter relies heavily on vanishing points, thus is suited only to environments with many parallel lines such as urban and indoor scenes. Future *Atlas* implementations can attempt to model more general scenes. Again some preliminary ideas will be presented in Chapter 7.

The local maps are represented with a collection of vanishing points (VPs), 3D line segments, and 3D points. The features are extracted from the edges in each video frame.

## 6.2.2 Chapter Summary

The remainder of this chapter will first go over the geometry of features used to create the local maps. Subsequently, the feature tracking methods, including the novel approach to track line segments associated with a vanishing point, are described. Then the preprocessing Kalman filter, used to track the vanishing points and initially correct the rotational errors, is presented. The camera pose results from the preprocessing Kalman filter are used with the feature tracking results to initialize the 3D structure of the lines and points and to provide a starting state for the bundle adjustment phase of the processing. The bundle adjustment is a batch optimization of the camera path and feature geometry used to minimize the reprojection errors under a robust Cauchy statistic. The results from the bundle adjustment comprise the *Atlas* map-frame. To complete the *Atlas* implementation requirements, the map-matching module based upon iterative closest feature matching is presented.

Experimental results are reported for long-duration omnidirectional video sequences for indoor, outdoor, and mixed indoor/outdoor environments. The approach is demonstrated experimentally with results from several long (1,000+ frames) omnidirectional video sequences.

## 6.3 Single View Geometry

The camera is modeled as a collection of rays which all pass through a single point. This point is the camera's center of projection. The rays are modeled by unit vectors which indicate the direction of the ray from the camera center. However, in projective geometry, the ray vectors can have an arbitrary scale multiplied to them without changing the ray. Thus the ray vectors do not need to be normalized as unit vectors.

The camera's image is modeled as a mapping from ray-vectors  $[r_x r_y r_z]^T$  to pixel coordinates  $(x_i, y_i)$ . Similarly, the pixel coordinates are mapped to unit vectors with

the inverse mapping. The most common image mapping model is the linear projective model where the camera rays  $\mathbf{r}$  are multiplied by a 3x3 upper triangular intrinsic calibration matrix  $\mathbf{A}$  and normalized by the  $z$ -coordinate.

$$\begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = \frac{\mathbf{A}\mathbf{r}}{\hat{\mathbf{z}}^T \mathbf{A}\mathbf{r}}$$

The ray vectors are determined from the pixel coordinates via the inverse mapping.

$$\mathbf{r} \simeq \mathbf{A}^{-1} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$$

where the  $\simeq$  symbol means that the left and right sides are equal up to an undetermined scale-factor.

### 6.3.1 Omnicam projection model

The camera used in this implementation uses a parabolic mirror to obtain a field of view of 180°. This large field of view is advantageous since objects remain in view for longer than when using a standard camera, thus leading to more precise navigation. Unfortunately, the disadvantage of the large field of view is the reduction in angular resolution. This resolution reduction makes object recognition difficult and affects the resolution from which maps can be made. The design tradeoff in this thesis is to sacrifice a bit of map accuracy for precision in navigation.

The ray to pixel mapping model is different since the camera images the reflection of a parabolic mirror. All the rays are aligned with the focus of the parabola and are orthographically projected via a telephoto lens into the camera. (See Figure 6-1). Its image mapping model is non-linear, and depends on the radius of the mirror at its

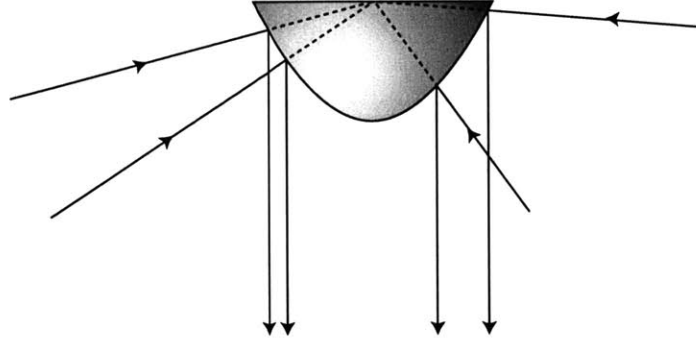


Figure 6-1: The omniscam geometry. The imaged rays all pass through the focus of the paraboloid, but are reflected orthographically into the camera's view

focal point  $\kappa$  and the coordinates of the mirror center  $(m_x, m_y)$  in the image.

$$\begin{bmatrix} x_i \\ y_i \end{bmatrix} = w \begin{bmatrix} r_x \\ r_y \end{bmatrix} + \begin{bmatrix} m_x \\ m_y \end{bmatrix}$$

$$w = \frac{\kappa}{\|\mathbf{r}\| + r_z}$$

Likewise, the inverse mapping identifies the rays for each pixel in the image.

$$\mathbf{r} \simeq \begin{bmatrix} x_i - m_x \\ y_i - m_y \\ \frac{\kappa^2 - (x_i - m_x)^2 - (y_i - m_y)^2}{2\kappa} \end{bmatrix} \quad (6.1)$$

### 6.3.2 Image Points

The camera's pose with respect to the scene's coordinate frame is represented by a 3x3 rotation matrix  $\mathbf{R}$  and a translation vector  $\mathbf{t}$ . The observation of a 3D scene point  $\mathbf{p}$  from the camera view is constructed by forming the ray from the camera center to the point, and rotating it into the camera's coordinate frame.

$$\mathbf{r} \simeq \mathbf{R}^T(\mathbf{p} - \mathbf{t}) \quad (6.2)$$



### 6.3.3 Image Lines

Lines segments in 3D have six DOFs, which can be partitioned into three groups: two for the direction of the line, two for the perpendicular offset of the line from the origin, and two for the locations of the endpoints along the line. The infinite line which contains the segment is then parameterized by the first four degrees of freedom. A 3D line can also be parameterized by any two points that line on the line.

3D lines project to 2D lines when imaged by the camera. The normal of the plane containing the 3D line and the camera projection center is used to parameterize the 2D image line. The line parameters  $\mathbf{l}$  can be determined by taking the cross product of the rays of two points  $\mathbf{r}_1$  and  $\mathbf{r}_2$  that lie on the line.

$$\mathbf{l} \simeq \mathbf{r}_1 \times \mathbf{r}_2 \quad (6.3)$$

When an image ray  $\mathbf{r}$  lies on an image line  $\mathbf{l}$ , the dot product of their parameters is zero.

$$\mathbf{l}^T \mathbf{r} = 0 \quad (6.4)$$

This equation reflects the fact that the image ray lies in the plane defined by the image line.

The ray  $\mathbf{r}$  that lies at the intersection of two image lines  $\mathbf{l}_1$  and  $\mathbf{l}_2$  is formed by taking the cross product of the line parameters.

$$\mathbf{r} \simeq \mathbf{l}_1 \times \mathbf{l}_2 \quad (6.5)$$

This equation is the dual of Equation 6.3. Geometrically, each line's plane contains the intersection ray thus the ray is orthogonal two both plane normals.

### 6.3.4 Vanishing Points

The images of parallel 3D lines also have an intersection point in the image. This point is called the vanishing point or VP. Geometrically, VPs are the common directions of parallel 3-D lines. The parameters of a vanishing point can be estimated from several lines identified to be parallel. Each line  $\mathbf{l}_i$  provides a constraint for the vanishing point  $\mathbf{v}$  using the fact that the dot product of the vanishing point and the line must be zero.

$$\begin{bmatrix} \mathbf{l}_1^T \\ \mathbf{l}_2^T \\ \vdots \\ \mathbf{l}_n^T \end{bmatrix} \mathbf{v} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad (6.6)$$

A non-zero solution to this system of equations is solved by taking the minimum eigenvector of the scatter matrix  $\mathbf{S}_l$  of the line parameters.

$$\mathbf{S}_l = \frac{1}{N} \sum_{i=1}^N \mathbf{l}_i \mathbf{l}_i^T \quad (6.7)$$

$$\mathbf{v} = \text{mineig}(\mathbf{S}_l) \quad (6.8)$$

The inverse of the scatter matrix  $\mathbf{S}_l$  is used to form the uncertainty of the vanishing point parameters. But first the number of degrees of freedom of the vanishing point must be addressed. Since the vanishing point is normalized to be a unit vector, there are only two DOF its parameters. Hence, the variance of the parameters in the direction of the vanishing point is zero. The deviation of the vanishing point is modeled to be entirely in the tangent plane of the unit sphere at the vanishing point. The normal of this plane is parallel to the vanishing point. The plane's coordinate system is represented by two basis vectors which are used to project the covariance of the VP onto the plane.

A consistent method for determining orthonormal basis vectors for the VP tangent plane employs the help of a constant vector  $\mathbf{p}$  that is initialized to be orthogonal to the initial estimate of the VP.

$$\begin{aligned}\mathbf{p} &= \begin{cases} n([100]^T \times \mathbf{v}) & \text{if } \mathbf{v} \cong [001]^T \\ n([001]^T \times \mathbf{v}) & \text{otherwise} \end{cases} \\ \mathbf{v}_x &= n(\mathbf{v} \times \mathbf{p}) \\ \mathbf{v}_y &= n(\mathbf{v} \times \mathbf{v}_x),\end{aligned}$$

where the function  $n(\mathbf{v})$  generates a unit vector in the same direction as  $\mathbf{v}$ :  $n(\mathbf{v}) := \mathbf{v}/\|\mathbf{v}\|$ .

We use the  $2 \times 3$  matrix

$$\mathbf{T}_v := [\mathbf{v}_x \ \mathbf{v}_y]^T \quad (6.9)$$

to project 3-D coordinates into the plane normal to the VP.

Thus the covariance of the vanishing point  $\mathbf{v}$  is computed from its scatter matrix  $\mathbf{S}$  by projecting its inverse through the tangent plane basis vectors  $\mathbf{T}_v$ .

$$\sigma_v^2 = \frac{N}{N-2} \mathbf{v}^T \mathbf{S} \mathbf{v} \quad (6.10)$$

$$\Sigma_v = \sigma_v^2 \mathbf{T}_v^T (\mathbf{T}_v \mathbf{S} \mathbf{T}_v^T)^{-1} \mathbf{T}_v \quad (6.11)$$

where  $\sigma_v^2$  is the sample variance of residuals of Equation 6.6.

### 6.3.5 Line estimation

The duals of Equations 6.6, 6.10, and 6.11 are employed to compute the covariance of the line parameters estimated from a collection of points. In the dual equations,

the role of the line replaces the vanishing point, and the points replace the lines.

$$\begin{bmatrix} \mathbf{p}_1^T \\ \mathbf{p}_2^T \\ \vdots \\ \mathbf{p}_n^T \end{bmatrix} \mathbf{l} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad (6.12)$$

A non-zero solution to this system of equations is solved with a singular value decomposition of the scatter matrix  $\mathbf{S}_p$  of the point ray parameters.

$$\mathbf{S}_p = \frac{1}{N} \sum_{i=1}^N \mathbf{p}_i \mathbf{p}_i^T \quad (6.13)$$

$$\mathbf{l} = \text{mineig}(\mathbf{S}_p) \quad (6.14)$$

Similarly to Equation 6.11, the covariance of the estimated line  $\mathbf{l}$  is computed from its scatter matrix  $\mathbf{S}$  by projecting its inverse through a set tangent plane basis vectors  $\mathbf{T}_1$ . The  $2 \times 3$  tangent plane basis vector matrix is computed by picking any two orthogonal vectors that are orthogonal to the line vector such as two rows of the Householder matrix defined by the line vector.

$$\begin{aligned} \sigma_l^2 &= \frac{N}{N-2} \mathbf{l}^T \mathbf{S} \mathbf{l} \\ \Sigma_l &= \sigma_l^2 \mathbf{T}_1^T (\mathbf{T}_1 \mathbf{S} \mathbf{T}_1^T)^{-1} \mathbf{T}_1 \end{aligned}$$

where  $\sigma_l^2$  is the sample variance of residuals of Equation 6.12. Figure 6-2 shows the projection of a 3-D line and its covariance onto the unit sphere.

### 6.3.6 RANSAC for Vanishing Points

Before the parameter for the vanishing points in each camera view can be estimated, the lines must be grouped into sets belonging to each VP. Clusters of lines with

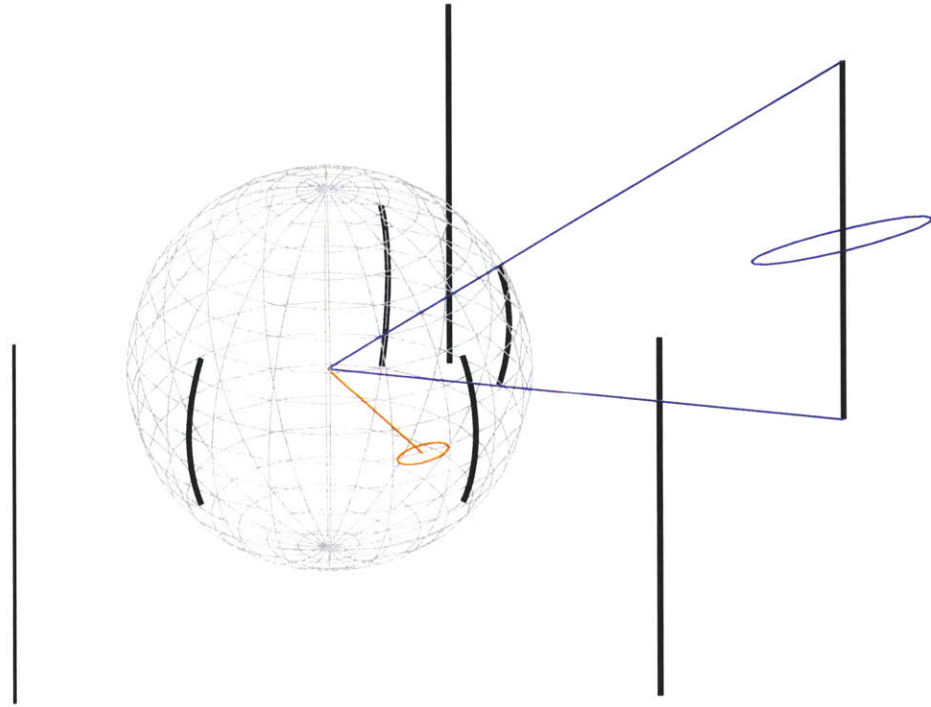


Figure 6-2: Projection of a 3-D line.

a common intersection point are discovered with a random sample and consensus (RANSAC) based method. The details of RANSAC were first introduced by Bolles and Fischler [6]; however, the basic idea is summarized here. In RANSAC, a series of sample models are picked using the minimum number of data points to determine the parameters of a model. Then each sample model is scored by counting the number of remaining data points that agree with the model. The top scoring models and their respective inlier data points determine the segmentation of the data.

The optimal number  $N_{\text{samples}}$  of model samples to pick is computed from the expected percentage of outliers  $P_{\text{outliers}}$  and the desired probability  $P_{\text{desired}}$  of finding

the correct model.

$$N_{\text{samples}} = \frac{\log(1 - P_{\text{desired}})}{\log(1 - (1 - P_{\text{outlier}})^M)}$$

where  $M$  is the number of data points used to determine the sampled models. It is important to note that the number of samples does not depend on the total number of data points.

RANSAC is applied to finding clusters of parallel image lines by identifying the vanishing points as the models and the lines as the data points. The models are picked by sampling random pairs of lines to intersect via Equation 6.5. The potential clusters are scored by counting the number of lines whose residual using Equation 6.4 is less than a predetermined threshold. (To facilitate the use of a metric threshold, both the model point and the line are normalized to unit vectors before applying Equation 6.4, such that the residuals are equal to the cosine of the angle between the vectors.)

### 6.3.7 Expectation Maximization for Vanishing Points

The vanishing point estimates and line association from RANSAC can be refined by an Expectation-Maximization (EM) algorithm in a manner very similar to the work of Antone and Teller [1]. EM is a good tool for solving mixture distribution problems where several models are can be used to fit a data set and the data association is unknown. The process iterates between an Expectation step where, the ownership probability of each data point to the model and the model's mixture probabilities are determined from the current set of model parameters, and a Maximization step where the model parameters are optimized to maximize their likelihood given the data mixture weights.

Each vanishing point model  $\mathbf{v}_j$  models the likelihood of each data line  $\mathbf{l}_i$  as a

Gaussian distribution.

$$P(\mathbf{l}_i | \mathbf{v}_j) = \frac{1}{\sqrt{2\pi\sigma_{ij}^2}} \exp \left\{ -\frac{1}{2} \frac{(\mathbf{l}_i^T \mathbf{v}_j)^2}{(1 - (\mathbf{l}_i^T \mathbf{v}_j)^2)} \frac{1}{\sigma_{ij}^2} \right\}$$

Where the expression  $\frac{(\mathbf{l}_i^T \mathbf{v}_j)^2}{(1 - (\mathbf{l}_i^T \mathbf{v}_j)^2)}$  is the squared distances of the line from the vanishing point in the vanishing point's tangent plane, and  $\sigma_{ij}^2$  is the variance of the line in the direction of the vanishing point plus the vanishing point's variance.

$$\sigma_{ij}^2 = \mathbf{v}_j^T \Sigma_{\mathbf{l}_i} \mathbf{v}_j + \sigma_v^2$$

The vanishing points are modeled with a fixed variance  $\sigma_v^2$  that is proportional to the angular resolution of the camera.

Outlier lines in the data are addressed with the addition of an outlier model an outlier model. The outlier model behaves as a vanishing point with a uniform likelihood. The magnitude of the outlier likelihood is set to be the same as that for a line which is three standard deviations from the VP.

$$P(l_i | v_{\text{outlier}}) = \frac{1}{\sqrt{2\pi\sigma_v^2}} \exp \left\{ -\frac{1}{2} \frac{(3\sigma_v)^2}{\sigma_v^2} \right\}$$

The ownership probabilities  $P(\mathbf{v}_j | \mathbf{l}_i)$  are computed using Bayes' rule.

$$P(\mathbf{v}_j | \mathbf{l}_i) = \frac{P(\mathbf{l}_i | \mathbf{v}_j)P(\mathbf{v}_j)}{\sum_j P(\mathbf{l}_i | \mathbf{v}_j)P(\mathbf{v}_j)}$$

Where  $P(\mathbf{v}_j)$  are the model mixture probabilities, and the sum is over each vanishing point model including the outlier model. These mixture probabilities are recomputed by summing up all the ownership probabilities and renormalizing such that the sum

of all the mixture probabilities equals one.

$$P(\mathbf{v}_j) = \frac{\sum_i P(\mathbf{v}_j | \mathbf{l}_i)}{\sum_i \sum_j P(\mathbf{v}_j | \mathbf{l}_i)}$$

The Maximization step re-estimates each vanishing point from the data lines using the ownership probabilities as weights. The procedure is the same as with Equations 6.7 and 6.8, except that each line vector is weighted by its ownership probability.

$$\mathbf{v}_j = \text{mineig}\left(\sum_i P(\mathbf{v}_j | \mathbf{l}_i)^2 \mathbf{l}_i \mathbf{l}_i^T\right)$$

The EM iterations are continued until convergence which is determined when the change in the model parameters is insignificant. The lines are classified to vanishing points by picking the number of model with the maximum ownership probability.

## 6.4 Multiple View Geometry

### 6.4.1 Point Triangulation

The location of a 3D point can be estimated from two or more observations when the camera poses are known. A linear method solving for the point parameters by minimizing geometric error employs the cross product to avoid explicitly solving for the observation ray scale factors. The constraint from each point's ray observation is formed by taking the cross product of the ray with both sides of Equation 6.2.

$$\mathbf{r} \times \mathbf{r} \simeq \mathbf{r} \times \mathbf{R}^T(\mathbf{p} - \mathbf{t}) = \mathbf{0}$$



A linear system is then constructed by stacking up the constraints from each ray observation.

$$\begin{bmatrix} \vdots \\ [\mathbf{r}_i]_{\times} \mathbf{R}_i^T \\ \vdots \end{bmatrix} \mathbf{p} = \begin{bmatrix} \vdots \\ [\mathbf{r}_i]_{\times} \mathbf{R}_i^T \mathbf{t}_i \\ \vdots \end{bmatrix}$$

where  $[\mathbf{r}]_{\times}$  is the  $3 \times 3$  matrix that forms the cross product when multiplied on the right by a vector. At least two observations are necessary since the cross product matrix is only rank 2, and there are 3 degrees of freedom in the point parameters.

## 6.4.2 Line Triangulation

The triangulation of 3D lines from image line observations at known camera poses is very similar to point triangulation if the vanishing point of the 3D line is known. The task simplifies to the triangulation of points in 2D by projecting everything onto the vanishing point's tangent plane. Each line observation forms a linear constraint on the intersection point of the 3D line with the vanishing point tangent plane.

$$\begin{bmatrix} \vdots \\ (\mathbf{T}_v \mathbf{R}_i \mathbf{l}_i)^T \\ \vdots \end{bmatrix} \mathbf{c} = \begin{bmatrix} \vdots \\ (\mathbf{T}_v \mathbf{R}_i \mathbf{l}_i)^T (\mathbf{T}_v \mathbf{t}_i) \\ \vdots \end{bmatrix} \quad (6.15)$$

where  $\mathbf{c}$  is the 2D vector of the 3D line intersection point in the basis of the tangent plane.

The perpendicular offset of the 3D line from the origin is formed by multiplying the intersection point with the tangent basis vectors as  $\mathbf{T}_v^T \mathbf{c}$ , and the direction of the line is the same as the vanishing point  $\mathbf{v}$ . These two vectors define the triangulated 3D line.

The expected covariance of the line intersection point may be computed from the

inverse of the left-hand side of Equation 6.15.

$$\Sigma_c \simeq \left( \begin{bmatrix} \cdots & (\mathbf{T}_v \mathbf{R}_i \mathbf{l}_i) & \cdots \end{bmatrix} \begin{bmatrix} \vdots \\ (\mathbf{T}_v \mathbf{R}_i \mathbf{l}_i)^T \\ \vdots \end{bmatrix} \right)^{-1}$$

The covariance of the intersection point will be undefined when the left-hand side of Equation 6.15 is ill-conditioned. This happens when the path of the camera centers lies in a plane with the 3D line, such as when the camera travels in the direction of the vanishing point. The system is also ill-conditioned when the 3D line is far from the cameras relative to the length of the path such that the parallax generated by the camera motion is insignificant.

### 6.4.3 Line Segment Matching with Dynamic Programming

Before 3D lines can be triangulated from image lines, the correspondence among image lines from each view must be made. Lines are tracked across consecutive image frames using stochastic nearest-neighbor gating [4] augmented by a novel ordering constraint. When parallel lines are on a single surface, the relative order of the lines is maintained from subsequent views. Though some lines may be occluded by convexities, their order can change only if the lines are on different surfaces and the motion is large with respect to the size of the surfaces. (See Figure 6-3.) Since in practice this is rare, the constraint exploits the fact to aid in tracking lines whose prediction gates overlap. Plain nearest neighbor tracking is not robust enough because many typical scenes have groups of lines that are closer together than the tracking prediction uncertainty.

The lines from the previous view are projected into the current view and then are matched to the newly extracted image lines. After sorting the lines around their respective VPs, a modified version of the standard longest common substring algorithm, which is efficiently solved with dynamic programming [13], is utilized to find

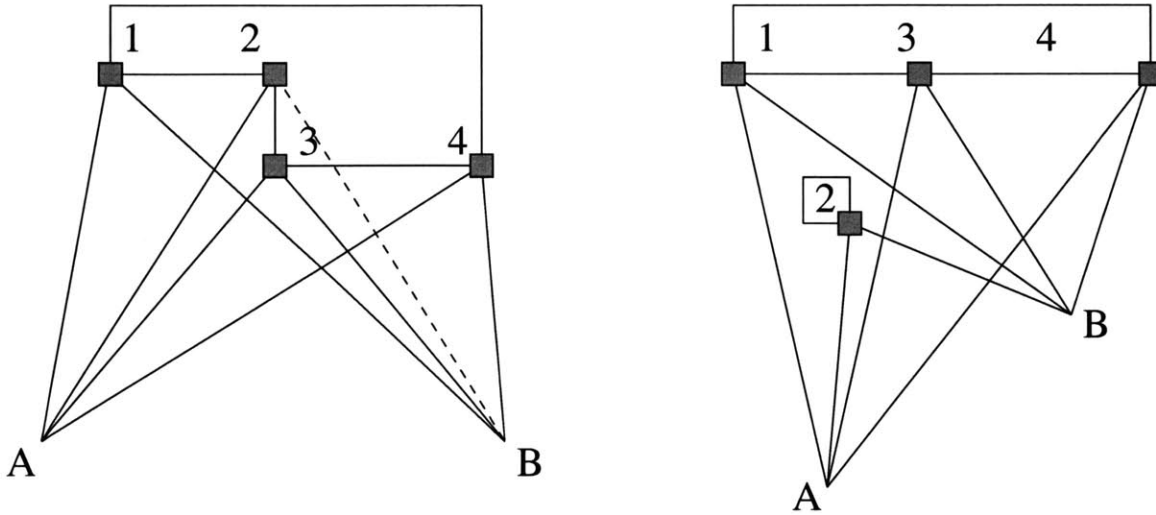


Figure 6-3: The order of the objects cannot change unless they are on different surfaces and the motion is large.

the best match while maintaining the ordering constraint. The VP can be pictured as the north pole of a globe (see Figure 6-2), then all the image line segments run north-south, and they can be lexicographically sorted by the longitude and latitude of their midpoints. The mean image colors to the left and right of line are used to further reduce the chance of false matches.

#### 6.4.4 Bundle Adjustment

Bundle adjustment is used to optimize the pose of the camera and scene structure parameters simultaneously. The procedure forms an high dimensional error vector to minimize with a nonlinear least squares optimizer. The errors to minimize are the reprojection errors of the points, lines, and vanishing points. The methods for performing the nonlinear least square are beyond the scope of this thesis. The reader is referred to Fletcher [24], or Triggs, McLauchlan, Hartley and Fitzgibbon [58], for a more detailed presentation of sparse large scale optimization methods.

$$\epsilon_{r_{ij}} = r_{ij} \times n(\mathbf{R}_j^T(\mathbf{p}_i - \mathbf{t}_j))$$

where  $i$  indexes the points, and  $j$  indexes the camera poses, and  $n(\mathbf{r})$  is a function which normalized vectors to have unit length. The cross product ensures that the observation has only two degrees of freedom and is invariant to the length of the ray vectors.

The line reprojection error is the cross product of the measured image line with the image line between the reprojected endpoints  $L0, L1$  of the 3D line segment.

$$\epsilon_{l_{ij}} = l_{ij} \times n(\mathbf{R}_j^T(L0_i - \mathbf{t}_j) \times \mathbf{R}_j^T(L1_i - \mathbf{t}_j))$$

The error vector is also augmented with the motion model from odometry, which is necessary to determine the otherwise unconstrained global scale-factor. In addition, the odometry error terms help condition the camera poses for view with poor observations of the scene geometry.

$$\epsilon_{d_j} = \begin{bmatrix} \sigma_{d\mathbf{R}} (d\mathbf{R}0_j - d(\mathbf{R}_{j-1}^T \mathbf{R}_j)) \\ \sigma_{dt} (dt0_j - \mathbf{R}_{j-1}^T (\mathbf{t}_j - \mathbf{t}_{j-1})) \end{bmatrix}$$

where  $d\mathbf{R}$  is a 3-vector such that  $R = e^{d\mathbf{R}}$ ,  $d\mathbf{R}0_j$ ,  $dt0_j$ , represent the relative motion from view  $j - 1$  to  $j$  as measured by the odometry, and the scale-factors  $\sigma_{d\mathbf{R}}$  and  $\sigma_{dt}$  weight the odometry errors.

Special attention must be placed on properly weighting the reprojection errors with the odometry errors. This is analogous to choosing process and measurement noise covariances when tuning a Kalman Filter.

Outliers in the data are addressed by applying a Lorentzian weight to each error

term. This is the same procedure as in Section 5.2.2.

$$\epsilon_{\text{total}} = \begin{bmatrix} \rho'(\epsilon_{r_{ij}}) \\ \rho'(\epsilon_{l_{ij}}) \\ \rho'(\epsilon_{d_j}) \end{bmatrix}$$

where  $\rho'(x)$  is the square root of Lorentzian weighting function.

$$\rho'(x) = \frac{2x}{x^2 + \bar{r}^2}$$

where  $\bar{r}$  represents the soft outlier threshold.

The optimization parameters are defined as little updates to the initial camera poses and structure parameters. The optimization routines work best provided with a non-redundant parameterization rather than using extra constraints and redundant parameterizations.

The rotation are modeled by the initial rotation plus a small update  $d\mathbf{R}_j$ . The vanishing point is modeled by the initial vanishing point plus a small deviation in the tangent plane. The 3D line endpoints are modeled by the first end point plus an update in the tangent plane and the second end point from the updated first in the direction of the vanishing point.

$$\begin{aligned} \mathbf{R}_j &= \mathbf{R}0_j e^{d\mathbf{R}_j} \\ \mathbf{t}_j &= \mathbf{t}0_j + d\mathbf{t}_j \\ \mathbf{p}_i &= \mathbf{p}0_i + d\mathbf{p}_i \\ \mathbf{v}_i &= n(\mathbf{v}0_i + \mathbf{T}_v^T d\mathbf{v}_i) \\ \mathbf{L}0_i &= \mathbf{L}00_i + \mathbf{T}_v^T d\mathbf{c}_i \\ \mathbf{L}1_i &= \mathbf{L}0_i + \mathbf{v} \end{aligned} \tag{6.16}$$

This implementation used the function `lsqnonlin.m` for large scale sparse prob-

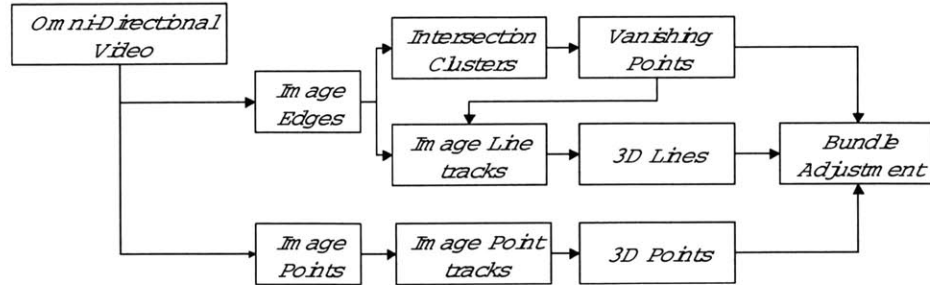


Figure 6-4: Data flow diagram for omnidirectional video processing.

lems in the Matlab optimization toolbox. The `lsqnonlin.m` routine uses the sparsity pattern of the Jacobian matrix to optimized the finite differencing when computing the Jacobian. The covariance of the parameters can be computed from the Hessian of the final optimization iteration, which is returned by the function.

## 6.5 Local Mapping Data flow

The previous sections have setup the important methods and algorithms used in the *Atlas* omniscam mapping module. This section ties together all the pieces by describing the data flow from the video sequence to the local map.

### 6.5.1 Preprocessing

Initially each video frame is preprocessed to extract line segments from edge contours, and estimate vanishing points from the extracted line segments. The edge contours are extracted from the zero crossings of a Laplacian of a Gaussian (LoG) filter on the luminance channel of the image. To compute the LoG image, each video frame

is convolved with a  $5 \times 5$  kernel  $F_{\text{LoG}}$ .

$$F_{\text{LoG}} = \begin{bmatrix} -1 & -3 & -4 & -3 & -1 \\ -3 & 0 & 6 & 0 & -3 \\ -4 & 6 & 20 & 6 & -4 \\ -3 & 0 & 6 & 0 & -3 \\ -1 & -3 & -4 & -3 & -1 \end{bmatrix}$$

The edge contours are determined to sub-pixel accuracy by interpolating detected zero crossings of the LoG image between two pixels. For example, when a zero crossing is detected between pixels  $(i, j)$  and  $(i + 1, j)$  with the LoG values of  $\alpha$  and  $\beta$ , respectively, and  $\alpha \cdot \beta = -1$ , the zero crossing is interpolated to lie at  $(i + \frac{|\alpha|}{|\alpha| + |\beta|}, j)$ .

The line segment extractor transforms the pixels from each edge contour into rays with Equation 6.1 and then recursively splits each contour into straight line segments. The extractor uses the same line splitting procedure as the laser line segmentation described in Section 3.2.1. However, instead of fitting lines to 2D points, the lines are estimated from 3D unit vectors using Equation 6.14.

The mean colors from each side of each extracted line are sampled from the image to aid in tracking correspondences. The sign of the brightness differential across the line is also used to determine the sign of each line parameter. The method employs the convention that the line parameter vector points in the direction of the darker side.

The next stage of preprocessing computes vanishing point directions in each frame using the extracted line segments. As described in Section 6.3.6, a RANSAC-based vanishing point extractor determines the number and initial locations of line intersection clusters in the image. An EM-based algorithm subsequently refines the vanishing point clusters and line classification. (See Section 6.3.7.)

## 6.5.2 Feature Tracking

The *Atlas* omniscam mapping module utilizes an intermediate phase to track feature correspondences from video frame to frame. The three types of features used, points, lines and vanishing points, are tracked using different methods.

A point tracker based on the Kanade-Lucas-Tomasi Feature Tracker [45], extracts image regions of large edge intensity, (by maximizing the minimum eigenvalue of the Hessian of the image derivatives), and tracks the regions in subsequent frames. An image mask prevents the selection of points and tracking on objects fixed within the camera view. The pixel coordinates for each point track are converted to ray vectors using Equation 6.1.

A simple Kalman Filter is used to track the vanishing point estimates from frame to frame. The vanishing point Kalman filter also corrects for the rotational errors in the input odometry measurements. The Kalman filter maintains the current rotation estimate of the camera and estimates of the vanishing points. Vanishing points in the current frame are matched using nearest neighbor gating to the vanishing points in the Kalman state vector. New vanishing points are added to the filter only when a track of intersection clusters from several frames proves to be invariant to the translation of the robot. This procedure prevents non vanishing point intersection clusters from being used.

Lines are tracked from frame to frame using their association with a vanishing point from the vanishing point Kalman filter. The lines from subsequent frames are corresponded using the technique described in Section 6.4.3.

## 6.5.3 Bundle Adjustment

The bundle adjustment is the core of the omniscam local mapping module. The preprocessing steps have extracted point, line, and vanishing point correspondences from the video, as well as corrected the rotational errors in the odometry-based camera



poses. The bundle adjustment will compute the 3D point and line parameters, and refine the camera rotation as well as translation estimates.

A local map-frame consists of a fixed number of sequential video frames to be bundle adjusted in a batch process. The initial estimation of the 3D point and line parameters are computed using the rotationally corrected odometry-based path of the camera and the procedures outlined in Sections 6.4.1 and 6.4.2. The initial structure, camera trajectory, and the correspondences from the preprocessing stages provide the inputs to the bundle adjustment routine.

The coordinate frame freedom in the bundle adjustment is fixed by transforming the results such that the first camera pose in the sequence is at the origin. Subsequent sequences of video frames are grouped for bundle adjustment in separate map-frames. The sequences overlap one another by about 50%. The exact amount of overlap is not a critical design decision; though there is a tradeoff between having a small overlap and little redundancy, and having a large overlap and better chances for many correspondences during later map matching. The *Atlas* genesis edges are formed by relating the camera poses in the overlapped trajectory regions.

## 6.6 Map Matching

The *Atlas* omniscam map match module matches the vanishing points to determine the rotational alignment, then it uses an iterative closest feature (ICF) to determine the translational component of the maps' alignment. The ICF procedure is very similar to the ICP procedure used in the *Atlas* scan-matching map match module from Section 5.4. The main differences are that this version uses both points and lines, there are no surface normals for the points, and transformation updates a 3D translational correction instead of rotation and translation in 2D.

## 6.7 Experimental Results

### 6.7.1 Hardware Setup

The omnidirectional video camera used in the experiments is a Canon Optura Digital Video cam-corder with two mirrors. The parabolic main mirror is mounted to the side of the camera, and a flat secondary mirror folds the optical path such that the cam-corder body doesn't block most of the field of view. The secondary mirror is supported by a thin aluminum arm which obstructs a thin region of the image. See Figure 6-5 for a picture of the camera and mirror setup, and Figure 6-6 for a sample image taken with the "omnicam".

Two calibration marks are placed near the mirror to aid in the calibration of the projection function. The thin aluminum arm of the omnicam vibrates a bit during camera motion such that the image of the mirror shifts a few pixels between frames. The system tracks the dots, computes their centers, and adjusts the mirror center and radius to maintain proper alignment.

The resolution of the camera is determined by the resolution of DV which is  $720 \times 480$ ; hence the maximal radius of the mirror image is just under 480 pixels and the average view arc covered by one pixel is  $0.375^\circ$ . A typical  $30^\circ$  FOV camera has a pixel arc of  $0.0625^\circ$ ; thus it is important to calibrate the mirror center and radius accurately.

The omnicam is mounted on either the B21 robot described in Chapter 3, or on a bicycle instrumented with a wheel odometer and steer column sensor. The audio channel of the DV cam-corder is used to record synchronization pulses from the odometry hardware. The synchronization pulses is a short 10 byte message sampled directly from a RS232 serial waveform at 9600 baud (attenuated to audio voltages). The audio waveform is sampled at 48000 Hz which corresponds to about 5 samples per bit. The 10 bytes consist of the header `0x55`, 4-byte integer Unix time, 4-byte

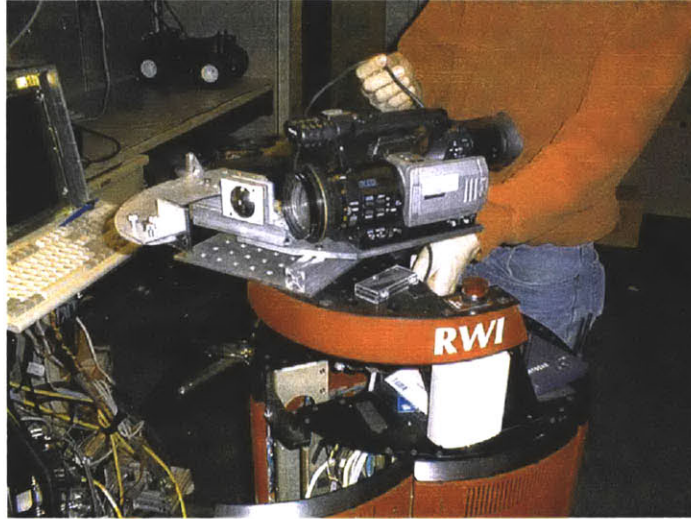


Figure 6-5: The “Omnicam” hardware setup mounted on a b21 robot.

Table 6.1: Image sequences for experiments.

Sequence name	duration	number of frames	total path length
Lounge	3'00"	5,400	29.9 meters
Library	9'42"	17,462	75 meters
Atrium	9'50"	17,684	100 meters
Omnibike	2'58"	5,348	94 meters

integer elapsed navigation time in milliseconds, and a checksum byte. The pulse is repeated at one second intervals.

Experimental results demonstrate the *Atlas* omni-video implementation on three different robot-mounted omnidirectional video sequences and one omnibike sequence, the first and last consisting of 5,000 frames, and the second and third consisting of 17,000 frames each.

### 6.7.2 Lounge Sequence

The first image sequence contains two short loops in a lounge area of our laboratory, with a path length of approximately 30 meters. Figures 6-7 and 6-8 show the re-projected errors for points and 3D lines for local bundle adjustment for one of the submaps (the fifth subsequence). Twenty-six submaps were created. Figure 6-9 shows

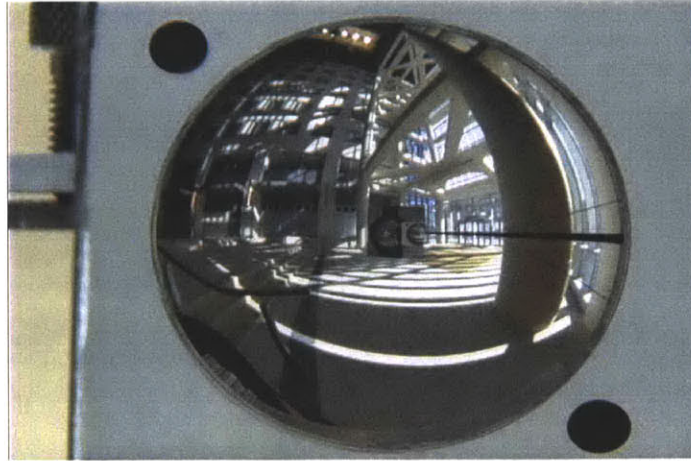


Figure 6-6: A sample omniscam image from the Atrium experiment. The dots to either the side of the mirror are used to track and compensate for motion of the image during jolts which vibrate the mirror arm.

the map match scores, demonstrating successful automated loop closure. The score is simply the number of matched points between maps. The threshold used for a successful match is 200 points. Figures 6-10 and 6-11 show the estimated camera trajectory and scene structure for all maps. For visualization of the scene structure, the algorithm performs a Delaunay triangulation in the view of the camera pose in the middle of the sequence. Delaunay triangles with perimeter greater than a threshold (typically, 1 meter) are not rendered. Figures 6-12 and 6-13 show the estimated camera trajectories. Note that the recovered path closes horizontal and vertical motion components.

### 6.7.3 Library Sequence

In the second experiment, the robot executed four loops around several bookshelves in a library, with a total path length of approximately 75 meters. Thirty-two submaps were created. Figure 6-15 shows the map match scores. The path consisted of one large loop and one smaller loop, and each loop was traversed twice. Figures 6-16 and 6-17 show the estimated camera trajectory and scene structure for all maps.

Figures 6-18 shows the estimated camera trajectories.

#### **6.7.4 Atrium Sequence**

The third experiment was performed on a non-planar path in an atrium between two buildings. This experiment demonstrates closure of a single large, non-planar loop with a total path length of approximately 100 meters. The vehicle traveled down a long ramp at the start of the experiment and returned by traveling up a steeper ramp at the end of the run. Thirty-one submaps were created. Figure 6-21 shows the map match scores. Figures 6-22 and 6-23 show the estimated camera trajectory and scene structure for all maps. Figure 6-24 shows a top view of the estimated camera trajectory, along with the path from dead-reckoning. Figure 6-25 shows a side view of the path, illustrating the motion of the vehicle on the ramps.

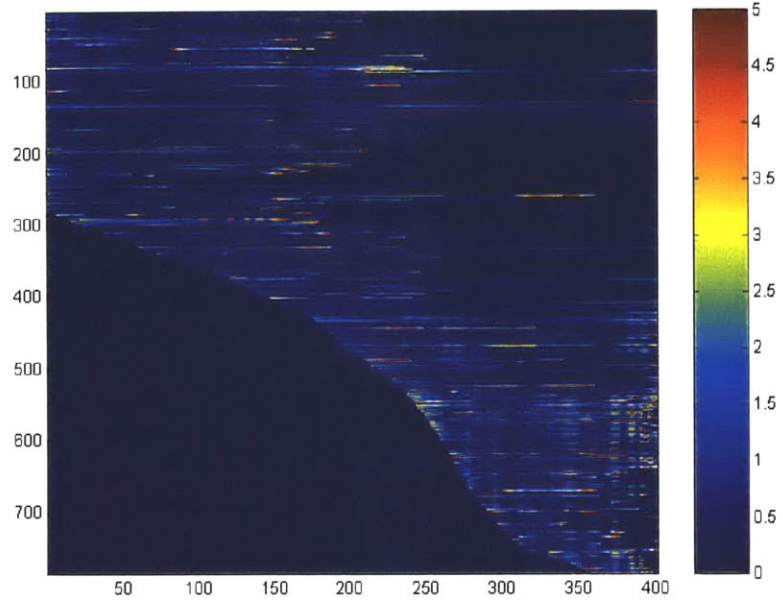


Figure 6-7: Lounge sequence point reprojection errors (in degrees) for the fifth local map. The  $x$ -axis indicates frame number, and the  $y$  axis indicates track ID.

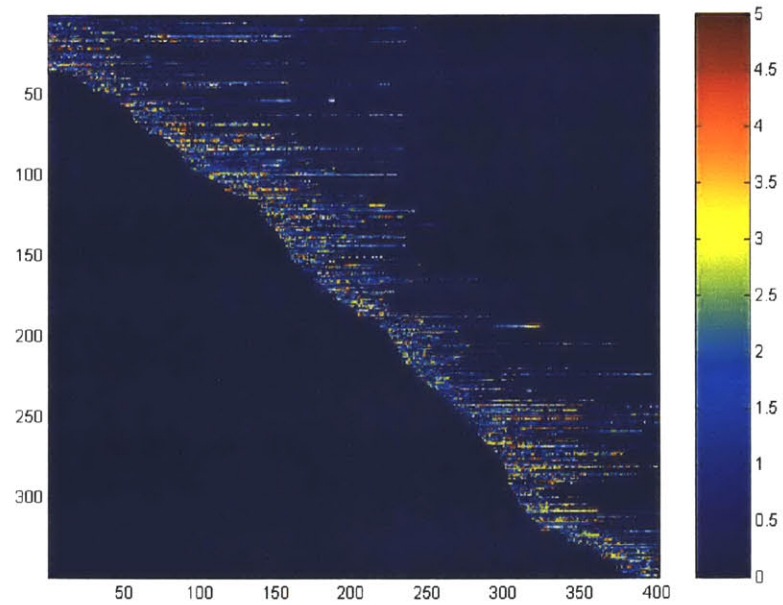


Figure 6-8: Lounge sequence line reprojection errors for the fifth local map. The  $x$ -axis indicates frame number, and the  $y$  axis indicates track ID.

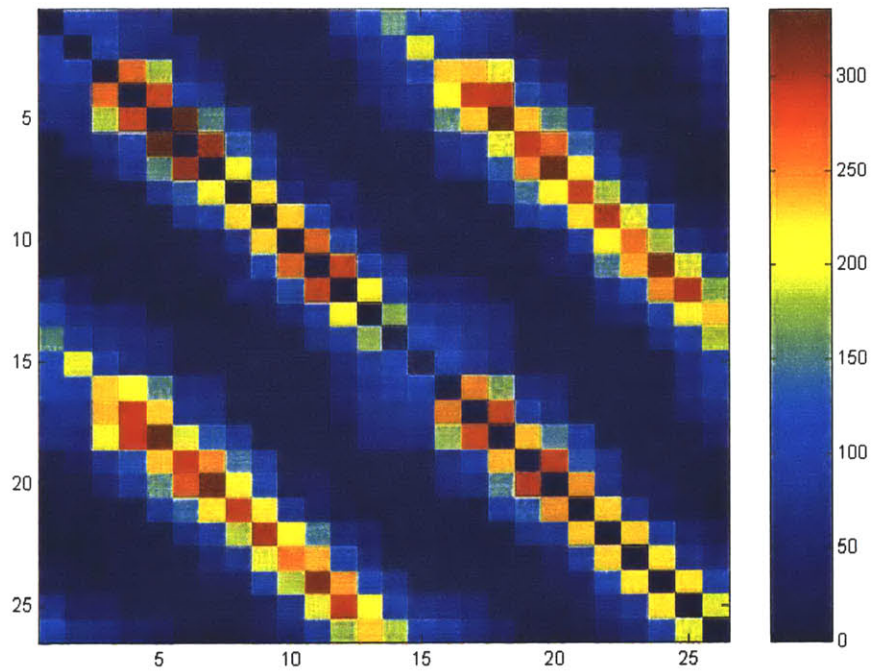


Figure 6-9: Lounge sequence: Map match scores. Values of zero indicate that a given pair of maps were not matched, because they were not estimated to be near enough to one another, based on concatenation of submap connection graph transformations and odometry.

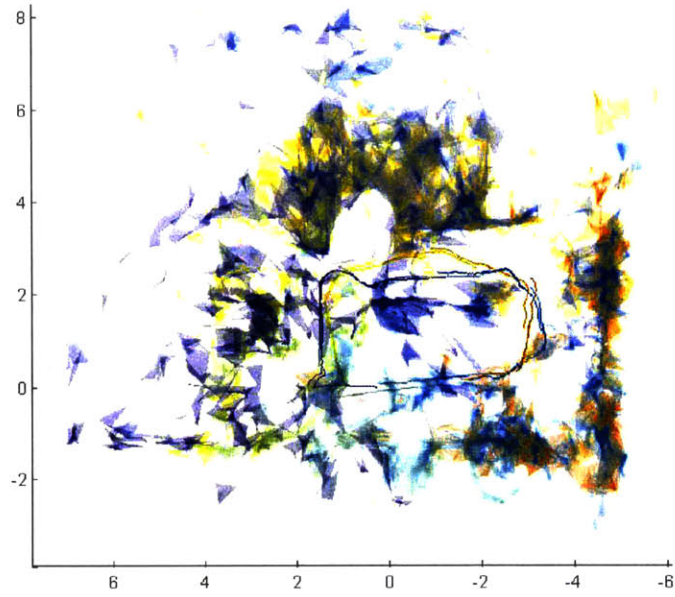


Figure 6-10: Lounge sequence: plan view of all submaps. Colors for Delaunay triangles correspond to structure estimated in different submaps.

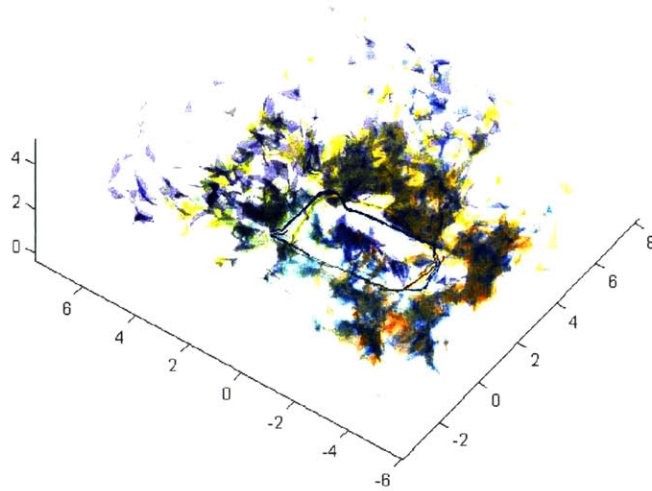


Figure 6-11: Lounge sequence: oblique view.



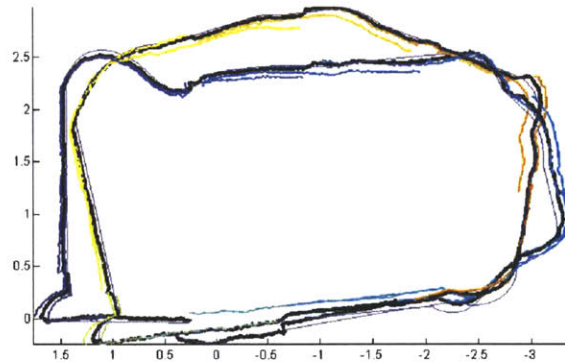


Figure 6-12: Lounge sequence: plan view of total path. Two plots are shown, the solid blue line shows the trajectory estimated only from VPs and odometry. The multi-colored plot shows the camera poses estimated from bundle adjustment and map frame alignment, with a different color for each submap.

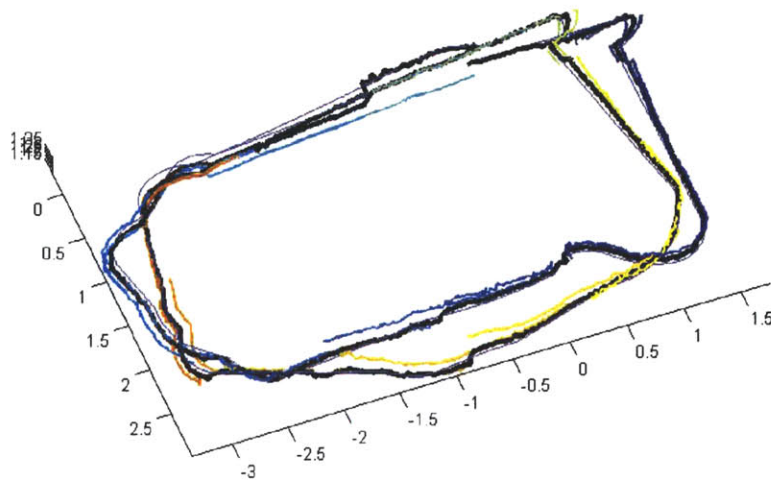
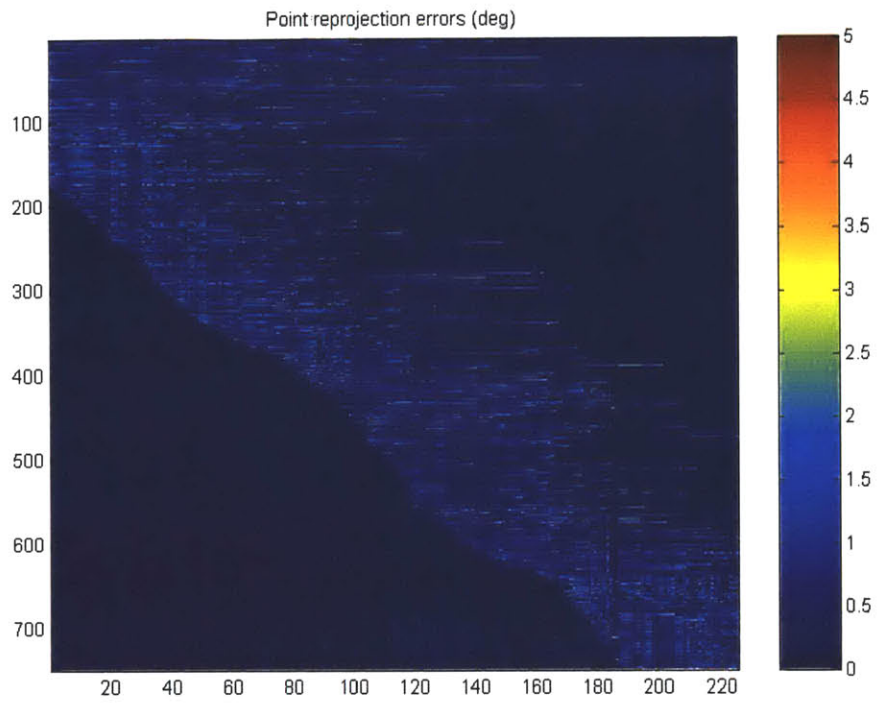
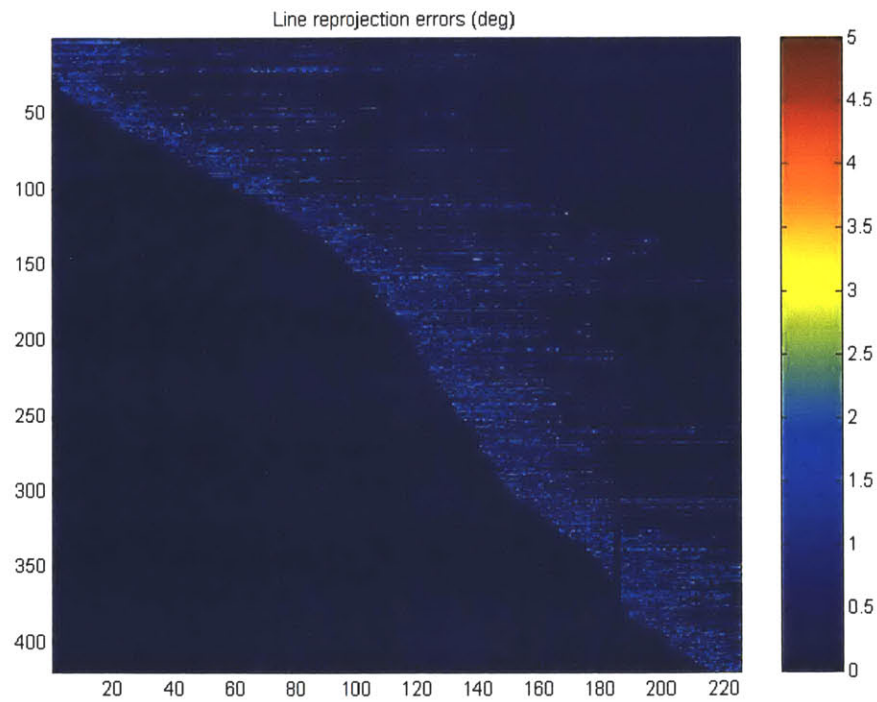


Figure 6-13: Lounge sequence: oblique view of total path.



(a)



(b)

Figure 6-14: Library sequence (a) point and (b) line reprojection error for the fifth local map (in degrees). The  $x$ -axis indicates frame number, and the  $y$  axis indicates track ID.

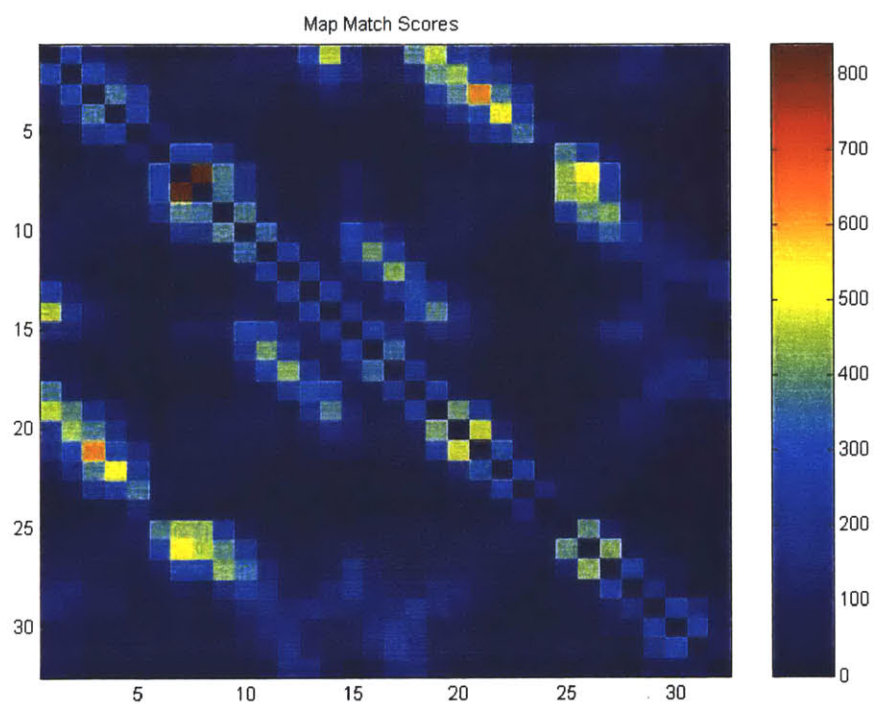


Figure 6-15: Library sequence map match scores.

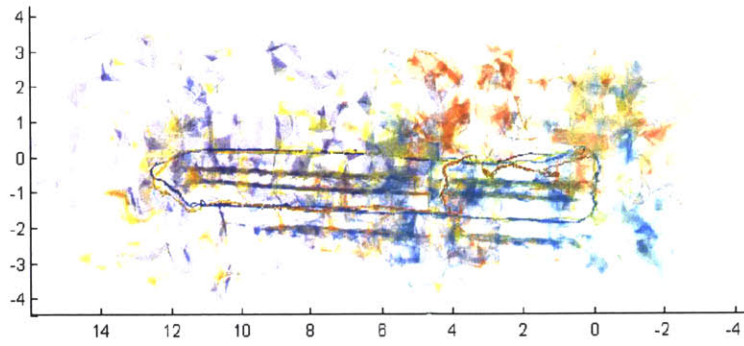


Figure 6-16: Library sequence: plan view of all submaps. Colors for Delaunay triangles correspond to structure estimated in different submaps. The structure of two library bookshelves enclosed by the vehicle trajectory is visible.

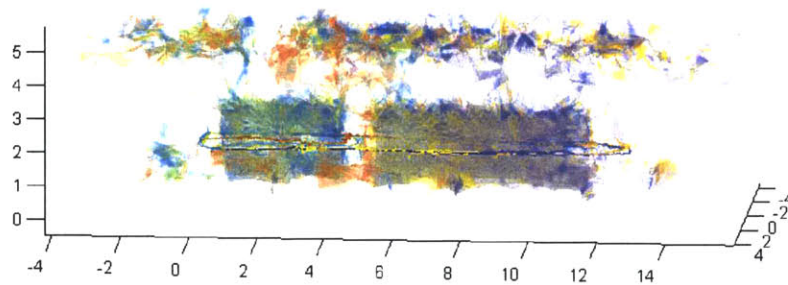


Figure 6-17: Library sequence: oblique view of all submaps.

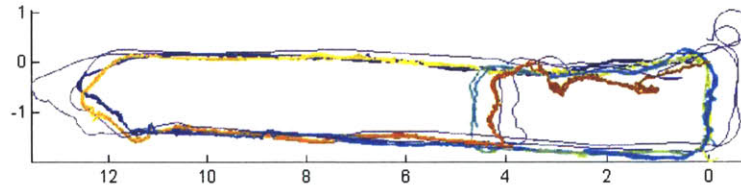


Figure 6-18: Library sequence: plan view of total path. Two plots are shown, the solid blue line shows the trajectory estimated only from VPs and odometry and the multi-colored plot shows the camera poses estimated from bundle adjustment and map frame alignment, with a different color for each submap.

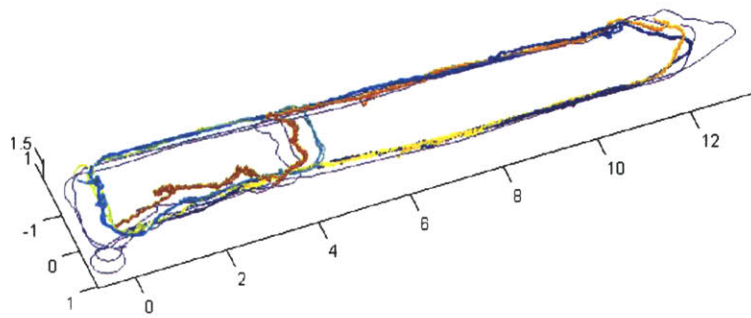
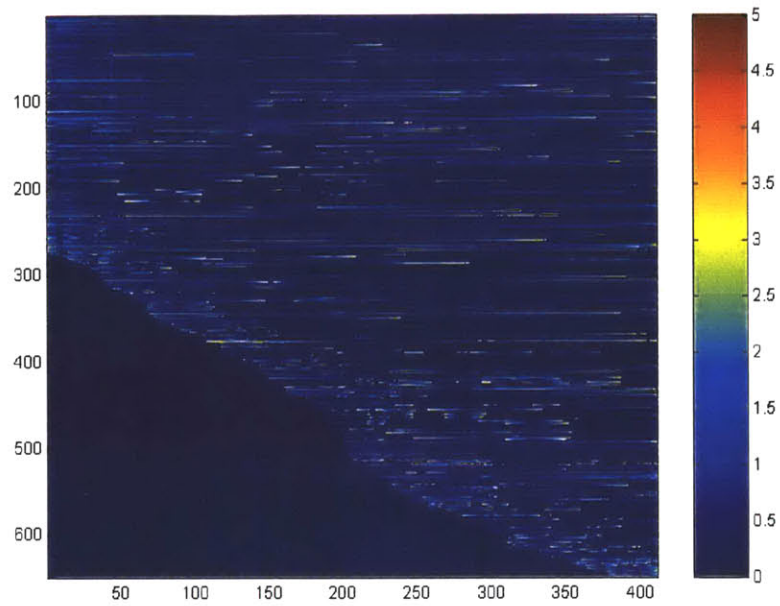
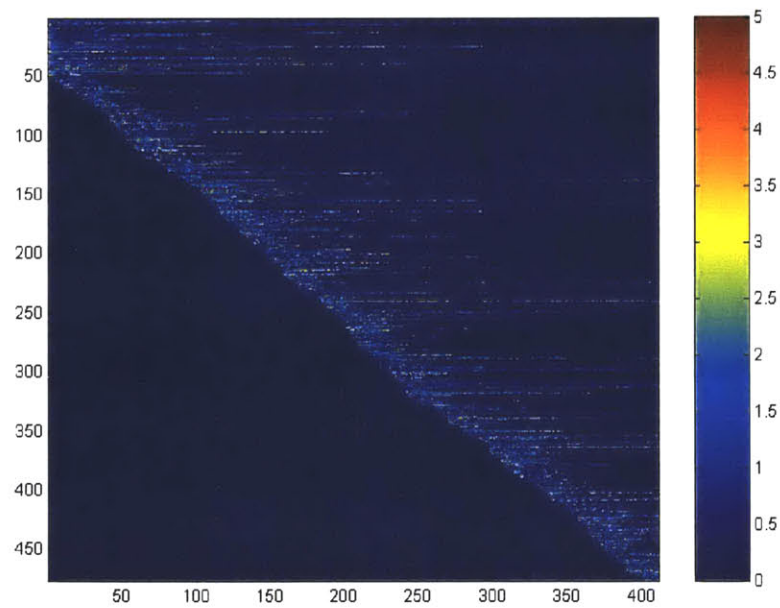


Figure 6-19: Library sequence: oblique view of total path.



(a)



(b)

Figure 6-20: Atrium sequence (a) point and (b) line) reprojection error for the fifth local map (in degrees). The  $x$ -axis indicates frame number, and the  $y$  axis indicates track ID.

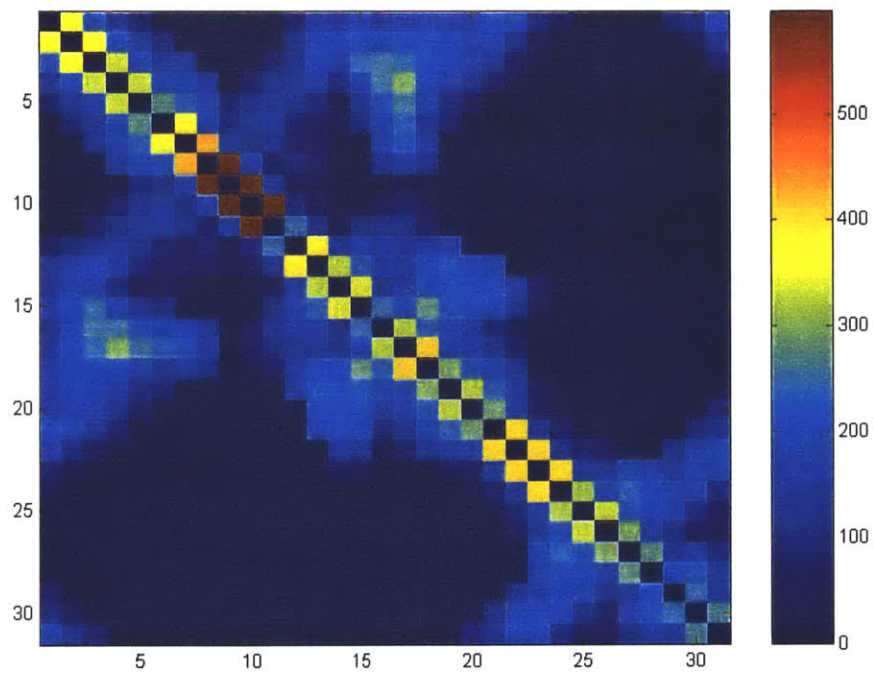


Figure 6-21: Atrium sequence map match scores.

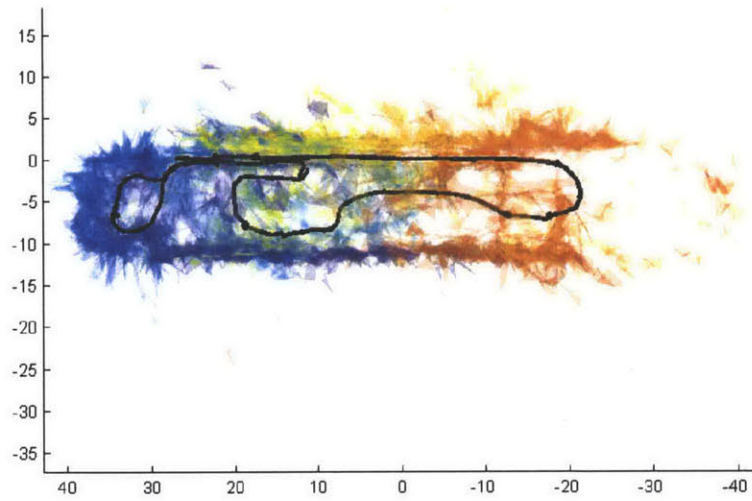


Figure 6-22: Atrium sequence: plan view of all submaps. Colors for Delaunay triangles correspond to structure estimated in different submaps. The estimated camera trajectory after fusion is shown as a heavy black line.

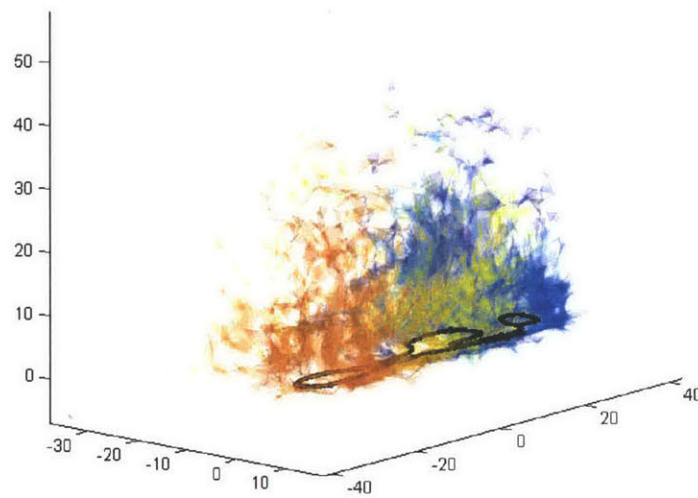


Figure 6-23: Atrium sequence: oblique view of all submaps.



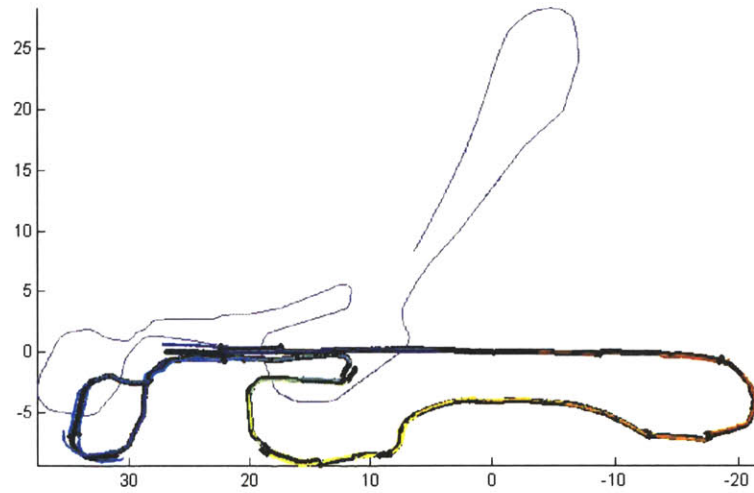


Figure 6-24: Atrium sequence: plan view of total path. Three plots are shown, the solid blue line shows the dead-reckoned camera trajectory estimated from odometry. The multi-colored plot shows the camera poses estimated from bundle adjustment and the heavy black line shows the trajectory estimate after map fusion.

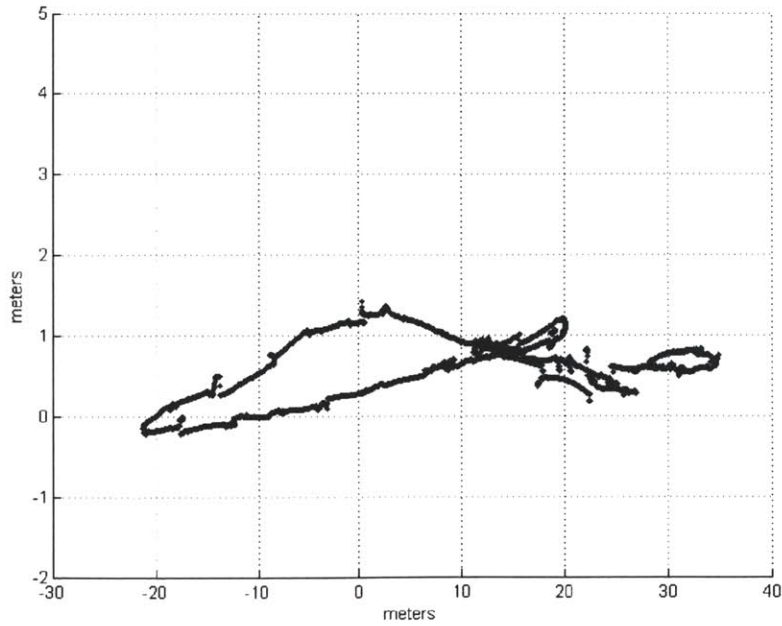


Figure 6-25: Atrium sequence: profile view of total path. (Note the unequal axis scales.)

### 6.7.5 Tech Square with Omnibike

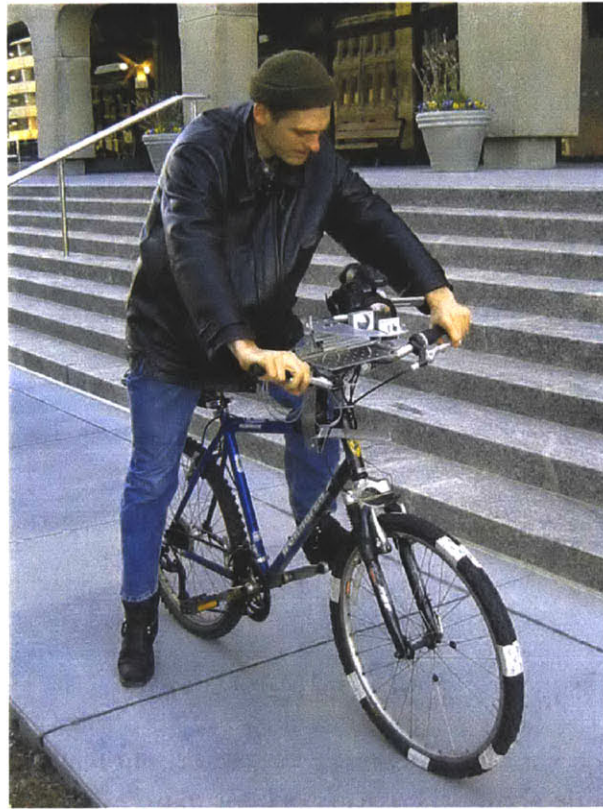


Figure 6-26: Detail view of omnibike mounted on bicycle.

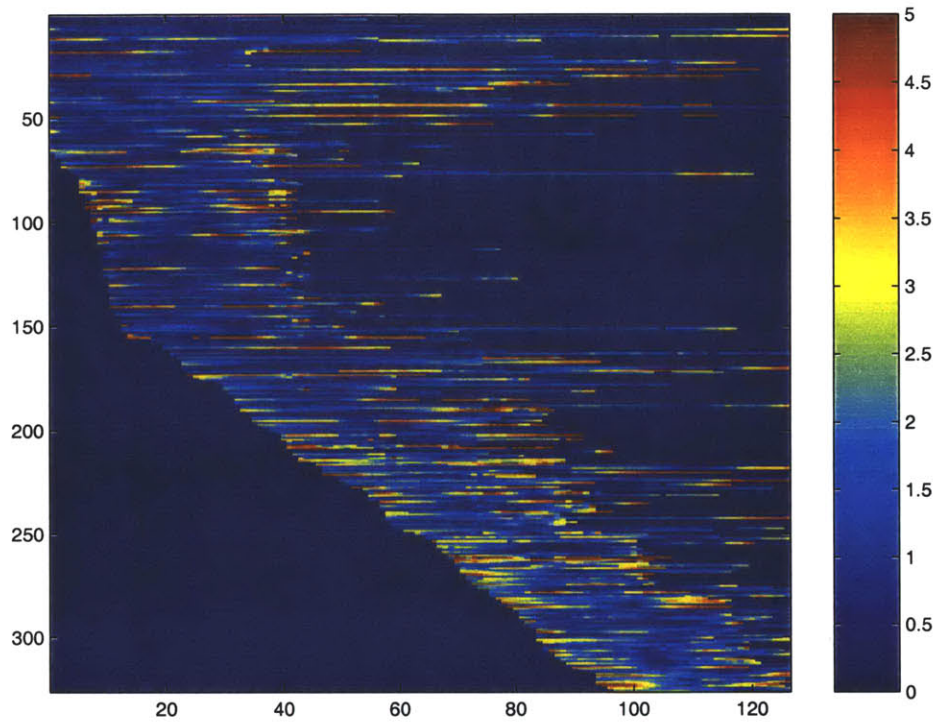


Figure 6-27: Omnibike sequence point reprojection errors (in degrees) for the fifth local map. The  $x$ -axis indicates frame number, and the  $y$  axis indicates track ID.

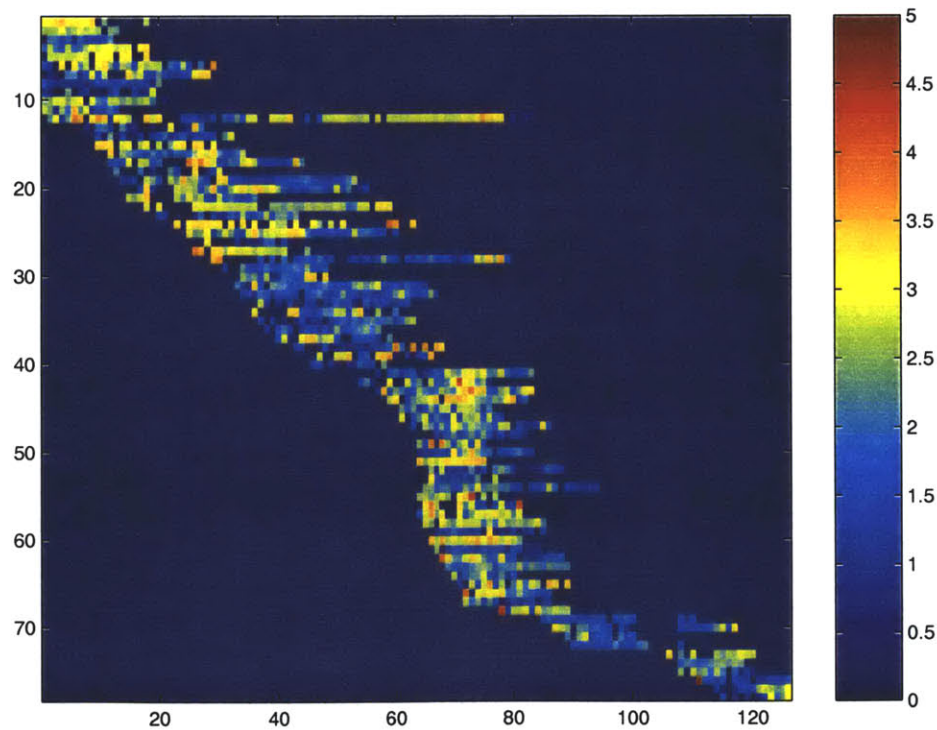


Figure 6-28: Omnibike sequence line reprojection errors for the fifth local map.

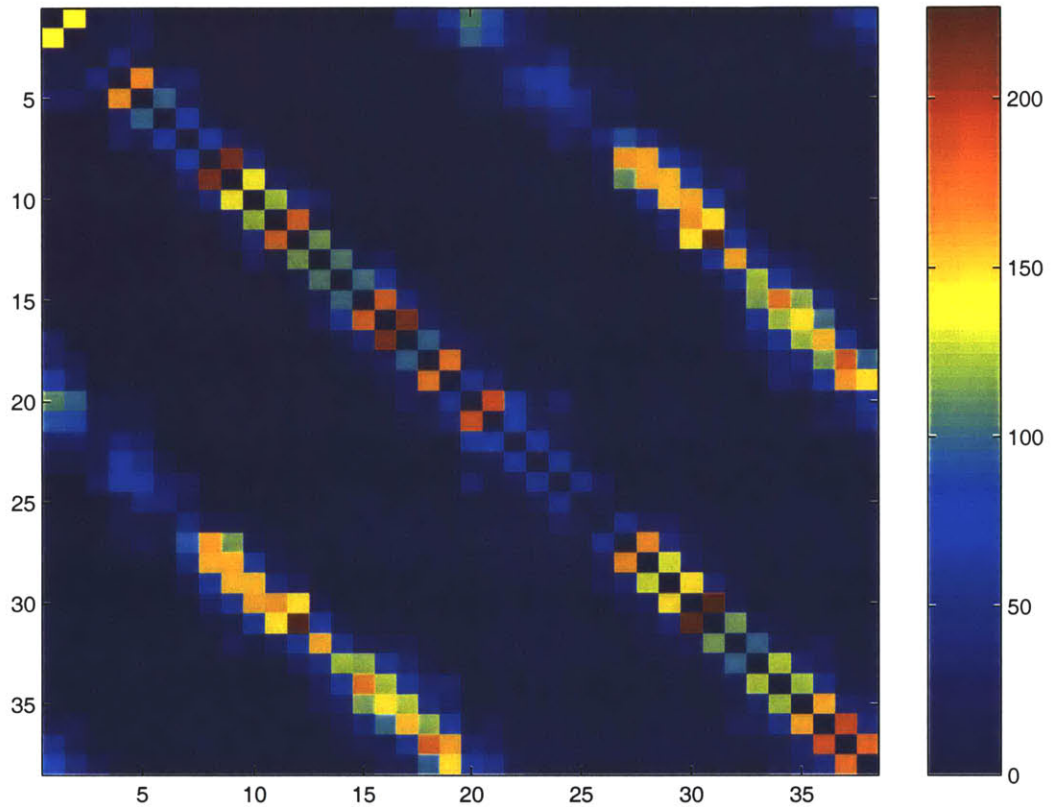


Figure 6-29: Omnibike sequence: Map match scores. Values of zero indicate that a given pair of maps were not matched, because they were not estimated to be near enough to one another, based on concatenation of submap connection graph transformations and odometry.

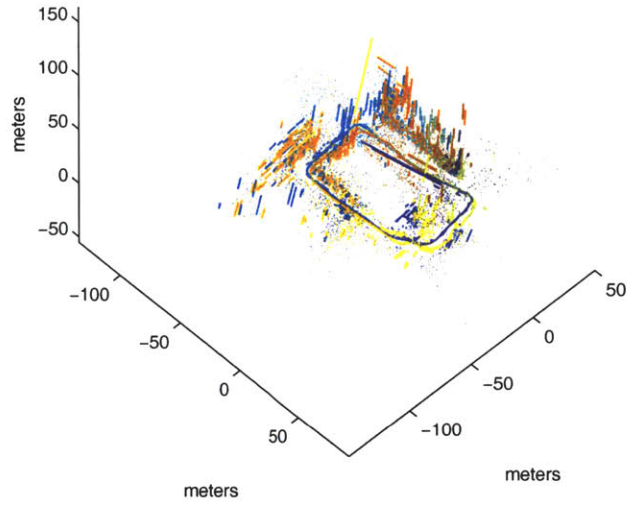


Figure 6-30: Omnibike sequence: oblique view.



Figure 6-31: Omnibike sequence: plan view of all submaps, manually overlaid on an aerial photograph. Colors correspond to structure estimated in different submaps.

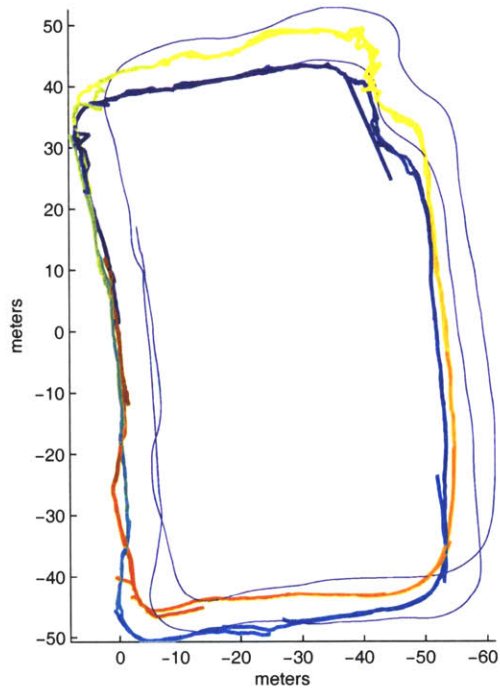


Figure 6-32: Omnibike sequence: plan view of total path. Two plots are shown, the solid blue line shows the trajectory estimated only from VPs and odometry. The multi-colored plot shows the camera poses estimated from bundle adjustment and map frame alignment, with a different color for each submap.

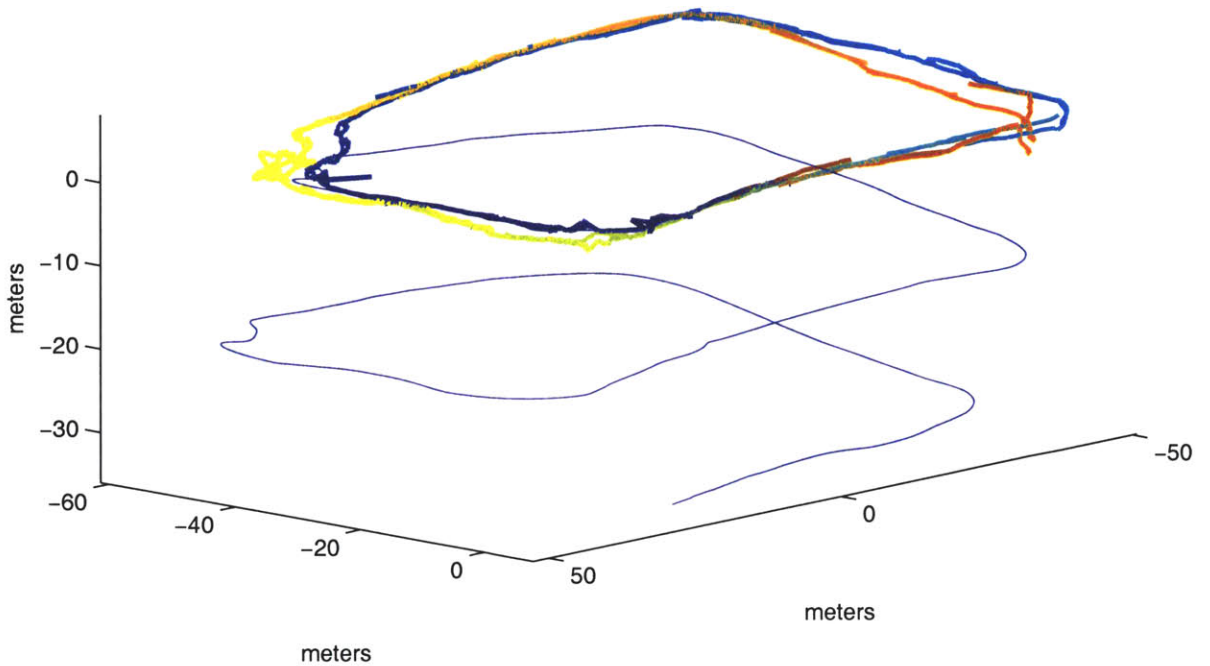


Figure 6-33: Omnibike sequence: oblique view of total path.

# Chapter 7

## Conclusion

This thesis has presented *Atlas*, a general framework for efficient large-scale mapping and navigation. The framework can autonomously handle closing multiple nested loops in large cyclic environments using a variety of sensors and mapping strategies. The performance of the approach has been demonstrated using laser scanner, ultrasonic range and omnidirectional video data. Results have been presented on numerous data sets for feature-based local SLAM (Chapters 3 and 4), scan-matching (Chapter 5), and omni-directional video bundle adjustment (Chapter 6). This chapter describes the failure modes of *Atlas* and makes recommendations for future research.

### 7.1 Summary

The approach combines the advantages of topological and metrical representations. The representation is a graph of coordinate frames. Each vertex in the graph represents a local coordinate frame, and each edge represents the transformation between adjacent local coordinate frames. In each local coordinate frame, the method builds a map that captures the local environment and the current robot pose along with the uncertainties of each. Each map's uncertainties are modeled with respect to its own local frame. Probabilities of entities in relation to arbitrary map-frames are generated

by following a path formed by the edges between adjacent map-frames, using Dijkstra’s shortest path algorithm. Loop-closing is achieved via an efficient map matching algorithm.

Modularity is an important characteristic of the *Atlas* framework, Successful results have been shown for a diverse range of sensor types and operating environments, including building corridors, an underground mine, an outdoor park, and an urban square. A diverse range of local mapping strategies have been utilized, including feature-based Kalman filtering, non-feature-based scan matching, and bundle adjustment.

Another key contribution of *Atlas* is its capability to provide efficient, real-time operation in large environment comprised of multiple nested loops. As described in Section 2.5, the method achieves a growth of complexity of either  $\mathcal{O}(n \log n)$  when using Dijkstra’s shortest path algorithm to select candidates for map-matching, or  $\mathcal{O}(n)$  when breadth-first search is used for this task. Amortization of this computation over the time spent in each submap results in extremely efficient performance. An off-line global alignment step is utilized to generate a single global map for visualization purposes at the end of a mission. This method operates extremely quickly (a few seconds of computation time) for the size of environments considered in this thesis.

### 7.1.1 Implementation Comparison

The Killian Court data set, first described in Section 3.5.1, will be used to compare the *Atlas* implementations from Chapters 3, 4, and 5. The most accurate results are obtained from the global optimized map projection of the scan-match implementation, thus the scan-match optimized robot pose trajectory is used as a reference when comparing the trajectories from other implementations.

The trajectories for each implementation are plotted in Figure 7-1. Figure 7-2 shows the time plots of the global pose errors, and Table 7.1 lists the standard devia-



Table 7.1: Standard deviation of global robot pose errors.

	$x$ (m)	$y$ (m)	$\theta$ (deg)
Scan-match	0.00	0.00	0.00
Laser Lines	3.54	6.43	2.53
Sonar Features	2.60	4.95	3.26
Scan-match Dijkstra	2.35	1.87	2.87
Laser Lines Dijkstra	4.94	8.88	4.28
Sonar Dijkstra	9.73	3.68	7.98
Odometry only	42.57	68.18	17.29

tions of the errors for each parameter. As expected, the global optimized projections have less errors than their respective Dijkstra projections. The discontinuities in Dijkstra projection errors are a result of the loops being split at that point.

### 7.1.2 Connectivity Metric

The topological quality of the resulting *Atlas* graphs from each implementation can be measured by a novel connectivity metric. The metric compares the adjacency of the robot positions from each implementation’s *Atlas* graph with the robot adjacency computed from the best projected trajectory — in this case, the scan-match optimized map projection’s trajectory.

Two robot poses are considered adjacent if their parent map-frames are adjacent and if the translational distance between the poses is less than a threshold. A threshold of 5 meters is used in this thesis. The robot adjacency matrix  $\mathbf{A}$  is defined as:

$$\mathbf{A}(t_i, t_j) = \mathbf{A}_m(m(t_i), m(t_j)) \cap \left( \|\mathbf{X}(t_i) - T_{m(t_i)}^{m(t_j)} \oplus \mathbf{X}(t_j)\| < d_{\text{threshold}} \right)$$

where  $\mathbf{A}_m$  is the map-frame adjacency matrix,  $m(t)$  is the map-frame id at time  $t$ ,  $T_{m_i}^{m_j}$  is the coordinate transform between map  $m_i$  and  $m_j$ , and  $\mathbf{X}(t)$  is the robot pose w.r.t. the map-frame.

The best adjacency  $\mathbf{A}_{\text{best}}$  is computed from the global optimized scan-match tra-



Figure 7-1: A comparison among the robot pose trajectories of the Killian Court data with different implementations and projections.

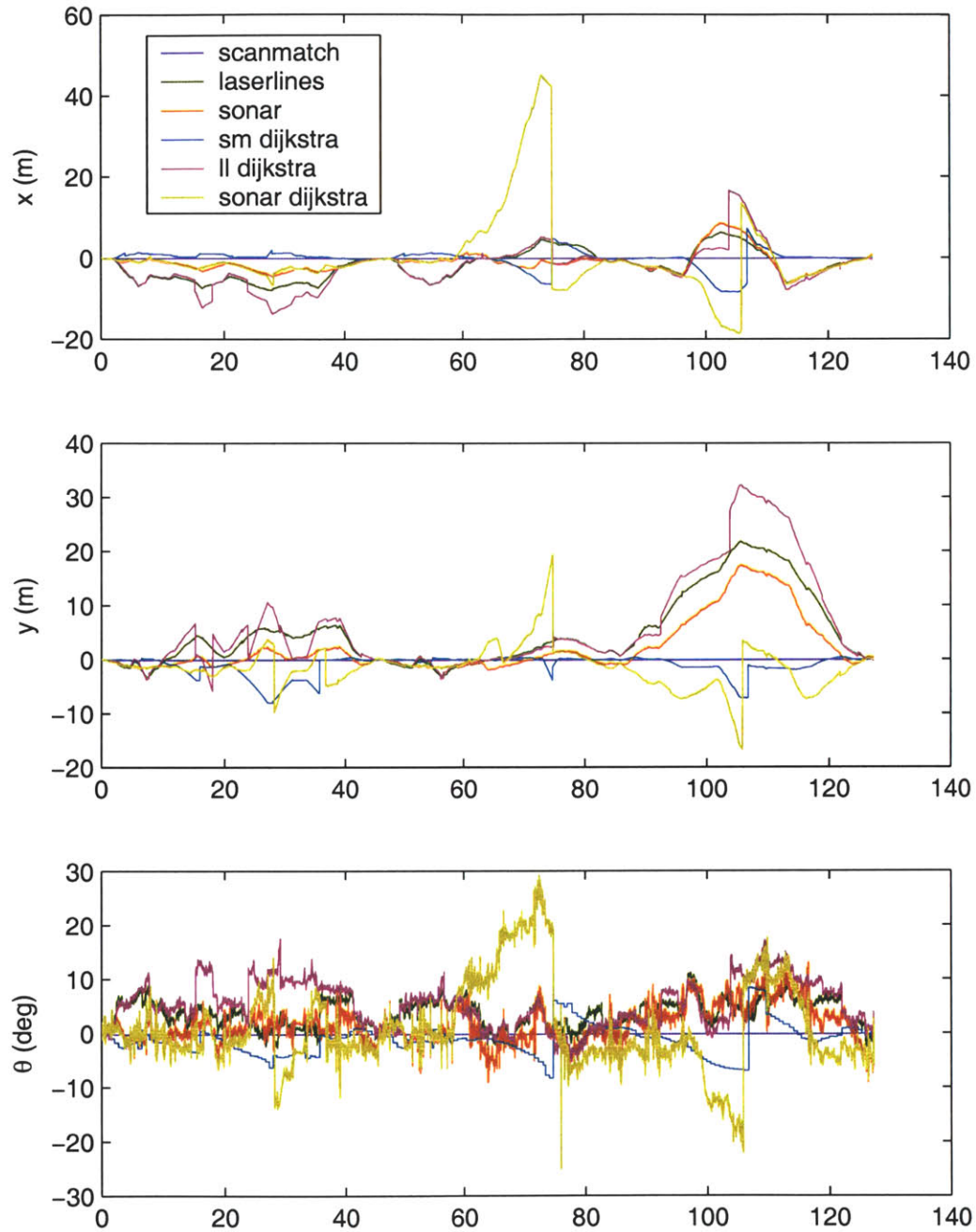


Figure 7-2: The robot pose errors of the Killian Court data set using the scan-matched optimized projection as a reference.

Table 7.2: Topological Connectivity Performance.

Implementation	$C$ using first order neighbors	$C$ using second order neighbors
Scan-match	0.8982	0.9897
Laser Lines	0.7277	0.9461
Sonar Features	0.6664	0.7845

jectory, ignoring its map-frame adjacency, which is displayed in Figure 7-3. To measure the loop closing performance, robot poses which are adjacent in time are not considered.

The connectivity metric is computed as the sum of robot poses that are adjacent in the both the *Atlas* graph and the best projection, divided by the sum of adjacent robots in the best projection.

$$C = \frac{\|\mathbf{A} \cap \mathbf{A}_{\text{best}}\|}{\|\mathbf{A}_{\text{best}}\|}$$

Table 7.2 catalogs the performance for each *Atlas* implementation. None of the implementation have a perfect connectivity score because they do not close all loops immediately. Some time is necessary for the system to recognize and verify a loop closure. When second order map-frame adjacencies are considered, however, the connectivity very closely approaches the optimal for the scan-match and laser lines implementations. The lower connectivity performance for sonar implementation indicates that its map-matching is not as robust.

## 7.2 Failure modes

There are three primary categories of *Atlas* failure modes: missed mapping, erroneous data association, and unsuccessful relocalization.

The first type of failure mode for *Atlas* is missed mapping due to reaching computational limits. There are two cases when the computational limits have been reached: either the open-loop uncertainty is too large such that there are too many

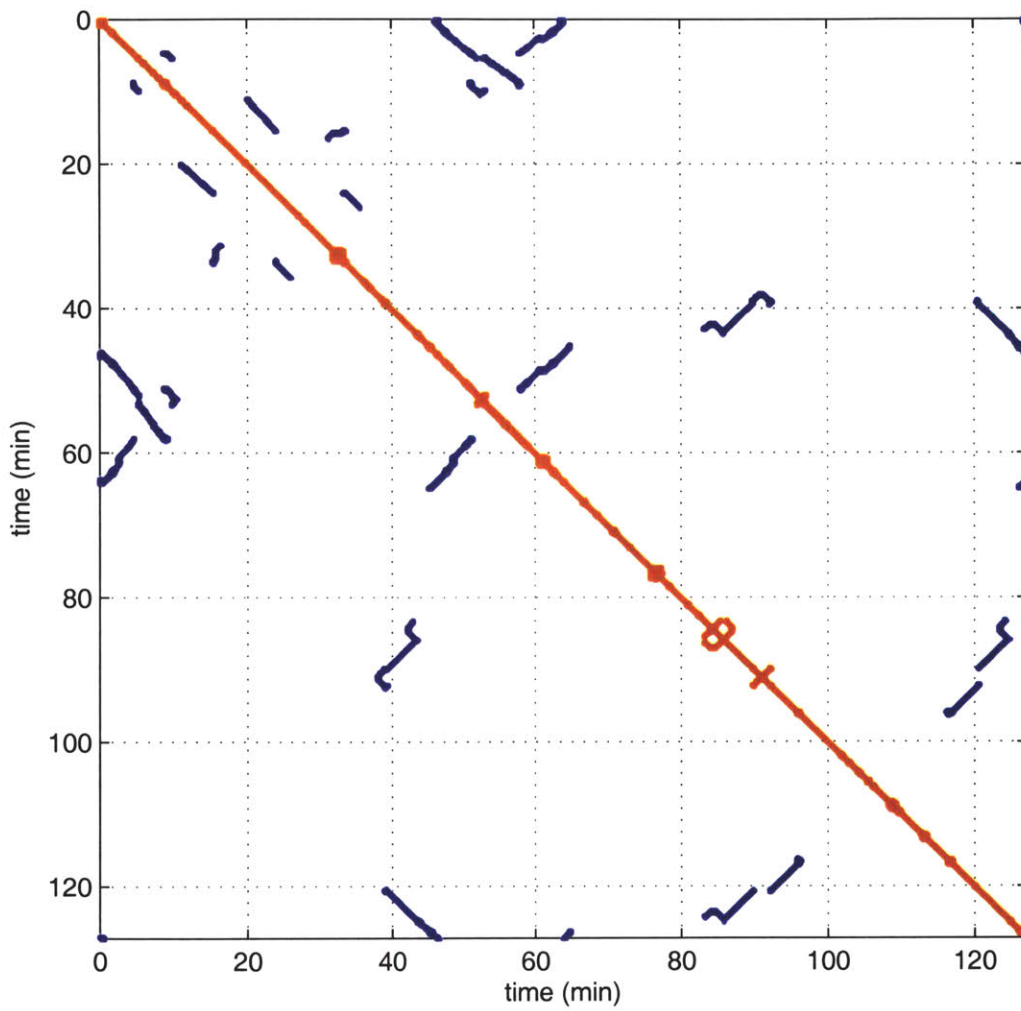


Figure 7-3: The best adjacency of robot poses for the Killian court data set. Robot poses adjacent in time, also indicated in red, are not used when computing the connectivity metric.

map-match candidates to be evaluated, or the length of the open loop is too large such that the appropriate map-match candidates are not placed on the candidate list in time. Since the number of map match candidates is proportional to the open loop uncertainty, when the open-loop uncertainty grows too large, there will be too many map match candidates to evaluate during the time spent in the dominant map. The second case occurs when the dominant map changes before the uncertainty projection computation runs to completion. Since the purpose of the map uncertainty projection computation is to find potential candidates for map-matching, some map-match candidates will be missed. The effect of both these cases is the failure to recognizing loop closure. With amortization and proper planning of the mapping trajectory, the open-loop uncertainty and the loop length are managed to maintain real-time performance while not missing any mapping events.

The second type of failure mode involves incorrect mapping due to data association errors. These errors can occur at multiple levels including low-level data association errors within the local SLAM module, and high-level data association errors when map matching. Local SLAM errors include feature doubling due to missed data associations and incorrect state and inconsistent error estimates due to incorrect data association and poor measurement models. Most local errors will simply make the local maps less accurate, but when the errors are severe enough, they can also affect the operation of the *Atlas* framework. The local errors can prevent map matching from recognizing previously mapped regions, and the inconsistent error estimates can prevent the framework from finding the proper candidates for map matching. High-level errors most likely occur when there are large repetitions of structure in the environment. The cycle verification procedure described in Section 2.2.4 greatly reduces the likelihood of such error by increasing the effective region of map overlap used to assure correct matching; however, there can still be structure repetitions at scales larger than the procedure can handle. A potential mitigation for this failure

mode requires that the region covered by the cycle verification be enlarged with respect to the open loop uncertainty, but then there will be a risk of not maintaining real-time performance.

Finally, the third mode of failure (related to missed mapping) consists of excessive genesis due to the inability to relocalize into previous map-frames. This is especially difficult for partially observable sensor modalities, such as sonar and vision. The implementation of relocalization for sonar and vision is an item for future work (described below). Relocalization failure can also occur with fully observable modalities, such as laser sensing, in environments such as long featureless corridors that do not fully constrain the robot position in all degrees of freedom.

## 7.3 Future Work

The key important issues for future work include relocation with partial observability, on-line map fusion, global state estimation, improved *Atlas* module implementations, multi-robot mapping, and exploration and long-term autonomy.

### 7.3.1 Relocation with partial observability

As described above, relocalization is difficult when using sensors that provide only partial constraints. Whereas laser scanners provide accurate range and angle measurements, sonar only provides accurate range measurements and vision only provides accurate angle measurements. The pose of the robot is not constrained by a single viewpoint. To address this, it is necessary to relocalize a short sequence of robot positions. Challenges in realize this include efficiently finding associations for sensor measurements across maps and consistently exploiting odometry information.

### 7.3.2 On-line map fusion

For many applications, it would be desirable to fuse the structure in overlapping submaps to reduce redundancy and to improve local state estimates by combining measurements. Examples of map fusion in the recent SLAM literature include Tardós *et al.* [50] and Williams *et al.* [61]. Again, one of the key challenges in map fusion is data association. It would be desirable to fuse maps in such a manner as to not increase the capacity of local maps. This will require partial fusion and map clean-up (discarding unnecessary states). An implementation of map fusion will need to address the issue of multiple features which may have used the same measurements.

Currently, with the conservative approach to map matching and loop closure adopted in *Atlas*, the algorithm can generate multiple overlapping submaps for the same region of the environment. For example, in Figure 3-18, the algorithm produced twelve map-frames to cover the area of the “ten loops” experiment. Using a technique such as sequential map joining [50], these twelve map-frames could be combined to form a single map. For larger scale environments, such as the Killian Court data set, the combination of many map-frames into a single map would likely fail due to linearization errors.

Map fusion may be used to side-step the problem of relocalization with partial observability all together. The system would continually generate new maps, use map matching to identify revisited areas, and employ map fusion to combine the new and old maps; thus preventing the graph from growing in density.

### 7.3.3 Global state estimation

This thesis has not addressed the issue of achieving global convergence for repeated traversals of the environment. The estimate of the transformations between maps can be improved each time a link is traversed. For a short time during transition, an estimate of the robot exists in two maps, which can constrain the inter-map co-



ordinate transformation. Maintaining consistency in performing this update is not straightforward because the two robot pose estimates are not independent. Covariance Intersection [59] is a potentially useful technique for fusion state estimates when their correlation is unknown.

For a linear Gaussian SLAM problem with known data association, a submap approach such as *Atlas* will yield state estimation errors that are larger than the full  $\mathcal{O}(n^2)$  solution. However, Newman and Leonard have shown that if the local origins of submaps are defined by map features shared between adjacent submaps, then one can achieve asymptotic convergence to a solution that is effectively the same as the full solution, for situations when the robot can make repeated traversals of the environment [32].

An interesting idea for future work is to employ Markov Chain Monte Carlo state estimation techniques [18] for the concatenation of approximate coordinate transformations in the *Atlas* map projection procedure. Particle filters have the potential to represent the errors of compounding nonlinear coordinate transformation.

### 7.3.4 Improved implementation of *Atlas* modules

For example, the loop-closing performance of *Atlas* using omnidirectional video could be improved by employing appearance as well as geometric information in the map matching and local mapping module. The implementation results described above in Chapter 6 used points, lines, and vanishing points as features for geometric state estimation and map matching. A promising idea to utilize appearance information is to employ the scale invariant feature transformation approach of Se *et al.* [44]. A difficulty in using appearance information involves properly addressing changes in lighting and surface reflections. A goal for future research is to integrate geometric and appearance-based information in variable lighting conditions for better object modeling and more reliable data association.

An additional goal is to achieve realistic three-dimensional visual mapping by expanding the model representation to include surfaces and textures.

### 7.3.5 Multi-robot Mapping

The framework can also be expanded to incorporate strategies for mapping and localization with multiple cooperative robots. Each robot would build and maintain a network of local maps, and when the robots meet, they can add edges to link their networks together. In a cooperative mapping scenario, the issue of reusing measurements must be addressed to correctly account for their correlations. Another issue is effectively manage the communications bandwidth for multiple mobile robots to perform cooperative mapping. The submap graph structure of *Atlas* lends itself to this task because information is localized. Rather than exchanging entire maps, multiple agents can share graph neighborhoods.

### 7.3.6 Exploration and Long-term Autonomy

Many applications exist which would benefit by incorporating the *Atlas* framework to enable efficient scaling in larger environments. Environments of interest include underwater, underground, service applications, space exploration, etc. To achieve long-term, large-scale autonomous mapping and navigation for these applications, it is necessary to couple *Atlas* with a higher-level process for exploration and path planning. In related work, Newman *et al.* [42] have integrated feature-based exploration with SLAM to enable “hands-off” autonomous mapping. It would be desirable to combine autonomous exploration with the *Atlas* framework, to enable autonomous path execution of loop closure.

# Bibliography

- [1] M. Antone and S. Teller. Scalable, extrinsic calibration of omni-directional image networks. *Int. J. Computer Vision*, 49(2/3):143–174, 2002.
- [2] David Austin and Patric Jensfelt. Using multiple gaussian hypotheses to represent probability distributions for mobile robot localization. In *IEEE Intl. Conf. on Robotics and Automation*, 2000.
- [3] T. Bailey and E. Nebot. Localization in large-scale environments. *Robotics and Autonomous Systems*, 37(4):261–281, 2001.
- [4] Y. Bar-Shalom and T. E. Fortmann. *Tracking and Data Association*. Academic Press, 1988.
- [5] P. J. Besl and N. D. McKay. A method for registration of 3-d shapes. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 14:239–256, 1992.
- [6] R.C. Bolles and M.A. Fischler. A RANSAC based approach to model fitting and its application to finding cylinders in range data. In *ijcai*, pages 637–643, Vancouver, Canada, Aug 1981.
- [7] W. Burgard, A.B. Cremers, D. Fox, D. Hachnel, G. Lakemeyer, D. Schulz, W. Steiner, and S. Thrun. Experiences with an interactive museum tour-guide robot. *Artificial Intelligence*, 1999.
- [8] A. Chiuso, P. Favaro, H. Jin, and S. Soatto. 3-d motion and structure from 2-d motion causally integrated over time: Implementation. In *Sixth European Conference on Computer Vision*, 2000.
- [9] K. Chong and L. Kleeman. Large scale sonarray mapping using multiple connected local maps. In *International Conference on Field and Service Robotics*, pages 538–545, ANU, Canberra, Australia, December 1997.
- [10] K. S. Chong and L. Kleeman. Sonar based map building in large indoor environments. Technical Report MECSE-1997-1, Department of Electrical and Computer Systems Engineering, Monash University, 1997.
- [11] Kok Seng Chong and Lindsay Kleeman. Sonar based map building for a mobile robot. In *Proc. IEEE Int. Conf. Robotics and Automation*, pages 1700–1705, April 1997.
- [12] H. Choset and K. Nagatani. Topological simultaneous localization and mapping (slam): toward exact localization without explicit localization. *ieeetra*, 17(2):125–137, 2001.

- [13] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to algorithms*. The MIT Press, 1991.
- [14] A. J. Davison. *Mobile Robot Navigation Using Active Vision*. PhD thesis, University of Oxford, 1998.
- [15] F. Dellaert, D. Fox, and S. Thrun W. Burgard. Monte carlo localization for mobile robots. In *Proc. IEEE Int. Conf. Robotics and Automation*, 1999.
- [16] E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.
- [17] M. W. M. G. Dissanayake, P. Newman, H. F. Durrant-Whyte, S. Clark, and M. Csorba. A solution to the simultaneous localization and map building (slam) problem. *IEEE Transactions on Robotic and Automation*, 17(3):229–241, June 2001.
- [18] A. Doucet, N. de Freitas, and N. Gordon, editors. *Sequential Monte Carlo Methods in Practice*. Springer-Verlag, 2001.
- [19] H. F. Durrant-Whyte, S. Majumder, M. de Battista, and S. Scheduling. A Bayesian algorithm for simultaneous localisation and map building. In R. Jarvis and A. Zelinsky, editors, *Robotics Research: The Tenth International Symposium*, Victoria, Australia, 2001.
- [20] O. Faugeras. *Three-Dimensional Computer Vision: A Geometric Viewpoint*. MIT Press, 1993.
- [21] O. Faugeras, Q-T. Luong, and T. Papadopoulo. *The Geometry of Multiple Images*. MIT Press, 2001.
- [22] H. J. S. Feder, J. J. Leonard, and C. M. Smith. Adaptive mobile robot navigation and mapping. *Int. J. Robotics Research*, 18(7):650–668, July 1999.
- [23] A. W. Fitzgibbon and A. Zisserman. Automatic camera recovery for closed or open image sequences. In *Proceedings of the European Conference on Computer Vision*, pages 311–326. Springer-Verlag, June 1998.
- [24] R. Fletcher. *Practical methods of optimization*. Wiley, 2nd edition, 1987.
- [25] A. C. Gelb. *Applied Optimal Estimation*. The MIT Press, 1973.
- [26] J. Guivant and E. Nebot. Optimization of the simultaneous localization and map building algorithm for real time implementation. *IEEE Transactions on Robotic and Automation*, 17(3):242–257, June 2001.
- [27] J-S. Gutmann and K. Konolige. Incremental mapping of large cyclic environments. In *International Symposium on Computational Intelligence in Robotics and Automation*, 1999.
- [28] D. Hähnel, D. Schulz, and W. Burgard. Map building with mobile robots in populated environments. In *Proc. IEEE Int. Workshop on Intelligent Robots and Systems*, 2002.
- [29] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521623049, 2001.

- [30] B. J. Kuipers. The spatial semantic hierarchy. *Artificial Intelligence*, 119:191–233, 2000.
- [31] J. Leonard and H. Feder. Decoupled stochastic mapping. *IEEE J. Ocean Engineering*, 26(4):561–571, 2001.
- [32] J. Leonard and P. Newman. Consistent, convergent, and constant-time SLAM. In *Int. Joint Conf. Artificial Intelligence*, 2003.
- [33] J. J. Leonard, R. N. Carpenter, and H. J. S. Feder. Stochastic mapping using forward look sonar. *Robotica*, pages 467–480, 2001.
- [34] J. J. Leonard, R. J. Rikoski, P. M. Newman, and M. C. Bosse. Mapping partially observable features from multiple uncertain vantage points. *Int. J. Robotics Research*, 21(10-11):943–976, Oct 2002.
- [35] Y. Liu and S. Thrun. Results for outdoor-SLAM using sparse extended information filters. Submitted for publication, 2002.
- [36] T. Lozano-Pérez. Foreword. In I. J. Cox and G. T. Wilfong, editors, *Autonomous Robot Vehicles*. Springer-Verlag, 1989.
- [37] F. Lu and E. Milios. Globally consistent range scan alignment for environment mapping, 1997.
- [38] D. Marr. *Vision*. New York: W. H. Freeman and Co., 1982.
- [39] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. FastSLAM: A factored solution to the simultaneous localization and mapping problem. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, Edmonton, Canada, 2002. AAAI.
- [40] H. P. Moravec and D. W. Cho. A Bayesian method for certainty grids. In *AAAI Spring Symposium on Robot Navigation*, pages 57–60, 1989.
- [41] E. M. Nebot and H. Durrant-Whyte. A high integrity navigation architecture for outdoor autonomous vehicles. *Robotics and Autonomous Systems*, 26:81–97, 1999.
- [42] P. Newman, M. Bosse, and J. Leonard. Autonomous feature-based exploration. In *Proc. IEEE Int. Conf. Robotics and Automation*, 2003.
- [43] H. Schmidt. GOATS-98 — AUV network sonar concepts for shallow water mine countermeasures. Technical Report SACLANTCEN SR-302, SACLANT Undersea Research Centre, 1998.
- [44] S. Se, D. G. Lowe, and J. Little. Mobile robot localization and mapping with uncertainty using scale-invariant visual landmarks. *Int. J. Robotics Research*, 21(8):735–758, 2002.
- [45] J. Shi and C. Tomasi. Good features to track. In *cvpr*, pages 593–600, 1994.
- [46] A. C. Shultz and W. Adams. Continuous localization using evidence grids. In *Proc. IEEE Int. Conf. Robotics and Automation*, pages 2833–2839, 1998.

- [47] R. Smith, M. Self, and P. Cheeseman. A stochastic map for uncertain spatial relationships. In *4th International Symposium on Robotics Research*. MIT Press, 1987.
- [48] R. Smith, M. Self, and P. Cheeseman. Estimating uncertain spatial relationships in robotics. In I. Cox and G. Wilfong, editors, *Autonomous Robot Vehicles*, pages 167–193. Springer-Verlag, 1990.
- [49] S. M. Smith and J. M. Brady. Susan – a new approach to low level image processing. *Int. J. Computer Vision*, 23(1):45–78, May 1997.
- [50] J.D. Tardós, J. Neira, P.M. Newman, and J.J. Leonard. Robust mapping and localization in indoor environments using sonar data. *Int. J. Robotics Research*, 21(4):311–330, April 2002.
- [51] C. J. Taylor and D. J. Kriegman. Structure and motion from line segments in multiple images. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 17(11):1021–1032, November 1995.
- [52] S. Teller, M. Antone, Z. Bodnar, M. Bosse, S. Coorg, M. Jethwa, and N. Master. Calibrated, registered images of an extended urban area. In *Computer Vision and Pattern Recognition*, volume 1, pages 813–820, Kauai, December 2001.
- [53] S. Thrun. A probabilistic online mapping algorithm for teams of mobile robots. *Int. J. Robotics Research*, 20(5):335–363, May 2001.
- [54] S. Thrun, D. Fox, W. Burgard, and F. Dellaert. Robust monte carlo localization for mobile robots. Technical Report CMU-CS-00-125, Carnegie Mellon University, 2000.
- [55] S. Thrun, D. Hähnel, D. Ferguson, M. Montemerlo, R. Triebel, W. Burgard, C. Baker, Z. Omohundro, S. Thayer, and W. Whittaker. A system for volumetric robotic mapping of abandoned mines. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2003.
- [56] S. Thrun, D. Koller, Z. Ghahramani, H. Durrant-Whyte, and Ng A.Y. Simultaneous mapping and localization with sparse extended information filters. In *Proceedings of the Fifth International Workshop on Algorithmic Foundations of Robotics*, Nice, France, 2002. Forthcoming.
- [57] B. Triggs, A. Zisserman, and R. Szeliski, editors. *Vision algorithms, theory and practice: International Workshop on Vision*. Springer-Verlag, 1999.
- [58] W. Triggs, P. McLauchlan, R. Hartley, and A. Fitzgibbon. Bundle adjustment for structure from motion, 2000.
- [59] J. K. Uhlmann. General Data Fusion for Estimates with Unknown Cross Covariances. In *Proceedings of the SPIE AeroSense Conference, Orlando, Florida*, 1996.
- [60] C.-C. Wang, C. Thorpe, and S. Thrun. Online simultaneous localization and mapping with detection and tracking of moving objects: Theory and results from a ground vehicle in crowded urban areas. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2003.

- [61] S.B Williams, G. Dissanayake, and H. Durrant-Whyte. An efficient approach to the simultaneous localisation and mapping problem. In *Proc. IEEE Int. Conf. Robotics and Automation*, pages 406–411, 2002.
- [62] Z. Zhang and Y. Shan. Incremental motion estimation through local bundle adjustment. Technical Report MSR-TR-01-54, Microsoft Research, 2001.