High Availability Mapping and Localization

by

Xipeng Wang

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Computer Science and Engineering)
in the University of Michigan
2019

Doctoral Committee:

Associate Professor Edwin B. Olson, Chair
Associate Professor Odest Chadwicke Jenkins
Associate Professor Matthew Johnson-Roberson
Assistant Professor Walter S. Lasecki

Xipeng Wang

xipengw@umich.edu

ORCID iD: 0000-0002-1755-2843

Acknowledgments

I would like to thank my advisor Edwin Olson for his guidance and wisdom throughout my graduate studies. I would also like to thank my other committee members (Chad Jenkins, Matthew Johnson-Roberson, Walter Lasecki) for their valuable feedback on this dissertation.

I would like to thank all those who have helped me along the way, especially Professor Yi Lu Murphey who gave me the earliest opportunities of my research career.

Most of all, I would like to thank my family, Junru and Lucas, for their love and support.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

SLAM	Simultaneous Localization and Mapping
AprilSAM	Real-time Smoothing and Mapping
FLAG	Feature-based Localization between Air and Ground
MOSS	Map Optimization for Size and Saliency

ABSTRACT

The problem of robotic mapping and localization is that of constructing a spatial model (*the map*) of an environment and estimating positions of robots inside the map. The solution to the problem is a fundamental requirement to enable mobile robots to operate autonomously in their environments. It is relatively straightforward to solve the problem under certain conditions with perfect sensors and unlimited computational power. The challenge arises when a mobile robot needs mapping and localization available outside the restricted operational envelope that is determined by *scale*, *robustness* and *the amount of prior data needed*. In this thesis, we describe several methods that expand this operational envelope, increasing the *availability* of position estimates to a robot. The idea of "high availability" is common in other system design domains, and we extend that idea here to robot navigation systems that must provide usable data in as a broad range of conditions as possible.

At the heart of mapping and localization is optimization. The computational complexity of the optimization limits the scale of maps that can be built in most robot systems, which limits the *availability* of those methods to relatively small environments. In this thesis, we present a Simultaneous Localization and Mapping (SLAM) algorithm, **AprilSAM**, that can rapidly estimate the maximum likelihood state for factor graph-based large-scale mapping. The algorithm selects between a batch solver and an incremental solver intelligently to balance the speed and the accuracy of the state estimation. As **AprilSAM** takes less running time than state-of-the-art approaches to achieve the same online mapping performance, it achieves high availability.

Performing global localization in a new environment often requires lots of preparations.

This limits the *availability* of algorithms only to familiar environments. Global Positioning System (GPS) sometime could supply the positioning information immediately for outdoor environment. For indoor environment, we propose a factor graph-based localization system, **FLAG**, that provides global positioning based on floor plans. **FLAG** significantly reduces the amount of prior information required to perform indoor localization as it doesn't require sending robots into the environment for mapping.

Pose estimation in a prior map is a common online localization method for robot navigation. The robustness of a localization system directly depends on the quality of the prior map. A high-resolution map provides a detailed representation of the world, but it is susceptible to feature aliasing since a sensor's view is noisy and often ambiguous. Such aliasing leads to poor localization performance which limits the *availability* of algorithms in complex environments. In this thesis, we present a machine learning-based map optimization algorithm, **MOSS**, that learns to adjust maps to support more robust localization. **MOSS** achieves high availability by producing maps that support better localization than the original full map.

CHAPTER 1

Introduction

1.1 Motivation

Autonomous mobile robots are able to avoid obstacles by planning safe paths to reach the desired destination. An accurate map is necessary for efficient planning, and following the path safely requires the robot to be precisely localized. The solution of the mapping and localization problem directly affects basic capabilities of autonomous robots. It is relatively straightforward to solve the problem under certain conditions with perfect sensors and unlimited computational power. The challenge arises when a mobile robot needs mapping and localization available outside the restricted operational envelope. To make the solution available in more conditions, we must first answers: *What are the major factors that restrict the availability of mapping and localization algorithms*?

The first factor is *scale*. Lots of mapping and localization algorithms can work near perfectly on robots like Roomba, in a small controlled environment, but they don't work well on autonomous vehicle that travel hundreds of miles across the country because the problem rapidly becomes difficult when dealing with large numbers of observations. At the heart of mapping and localization is optimization, and the optimization speed determines the scale for real-time mapping operations. The computational cost of early Kalman filter-based [15] grow quadratically. Though graph-based approaches [19] are able to deal with larger number of observations, they still can't scale up easily. This motivates us to develop AprilSAM, a faster optimization algorithm that works for large-scale mapping. It selects between a batch solver and an incremental solver intelligently to offer the trade-off between the speed and the accuracy of the state estimation.

The second factor is *the amount of prior data needed*. Global localization is critical for planning the paths of autonomous robots. Many existing navigation methods perform localization as a pose estimation problem after building a map with the robot [54, 57, 82, 92]. But they require to acquire prior data to build the map, thus the conditions applicable are limited. For example, they don't work well for the application such as search and rescue in which robots may be called upon to operate in a novel environment. GPS could supply the position information when map is not built ahead for outdoor environments. However, its performance significantly deteriorates indoors. This motivates us to develop Feature-based Localization between Air and Ground (FLAG), an indoor localization system that provides global positioning with a floor plan as prior map. It significantly reduces the amount of prior data required for mapping and localization.

The last factor is *robustness*. A high-resolution map provides a detailed representation of the world, but it is susceptible to feature aliasing since a sensor's view is noisy and often ambiguous. Such aliasing leads to poor localization performance. This limits the availability of algorithms for many applications such as autonomous driving which requires reliability. This motivates us to develop Map Optimization for Size and Saliency (MOSS), a machine learning-based map optimization algorithm learns to adjust maps to support more robust localization.

The major challenge of mapping and localization algorithm is to expand the operational envelope that is determined by *scale*, *robustness* and *the amount of prior data needed* so that they can be available in more conditions. In this thesis, we describe several methods that expand this envelope, increasing the *availability* of position estimates to a robot. The idea of "high availability" is common in other system design domains, and we extend that idea here to robot navigation systems that must provide usable data in as broad a range of conditions as possible.

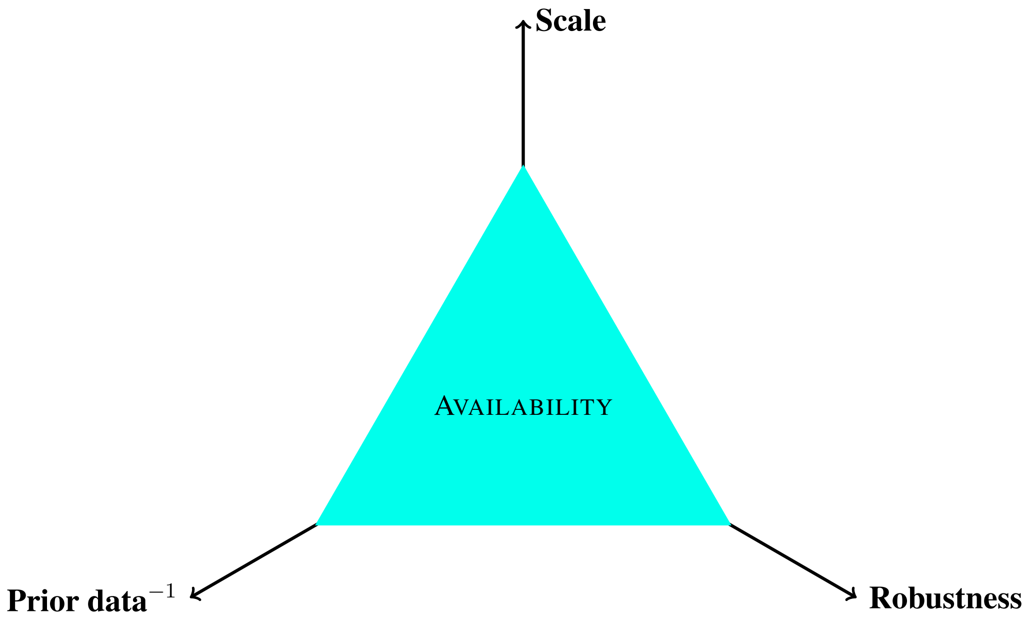


Figure 1.1: A condition envelope for mapping and localization. It is determined by *scale*, *robustness* and *the amount of prior data required*. A mapping and localization algorithm could be applied to more conditions if it is able to work robustly for large-scale places with less required prior data.

1.2 Thesis Contributions

This thesis focuses on developing high availability mapping and localization algorithms that expanding the condition envelope. More specifically, we describe several contributions for increasing mapping speed, improving localization accuracy, and reducing the amount of prior information required for applying the algorithms. These contributions include:

- A SLAM algorithm, Real-time Smoothing and Mapping (AprilSAM), that takes less optimization time than the state-of-the-art approaches to achieve the same mapping performance. (Chapter 3)
- A factor graph-based localization system, FLAG, that provides global positioning in novel indoor environments based on floor plans. (Chapter 4)
- A machine learning-based map optimization method, MOSS, that produces compact maps supporting better localization than the original full map. (Chapter 5)

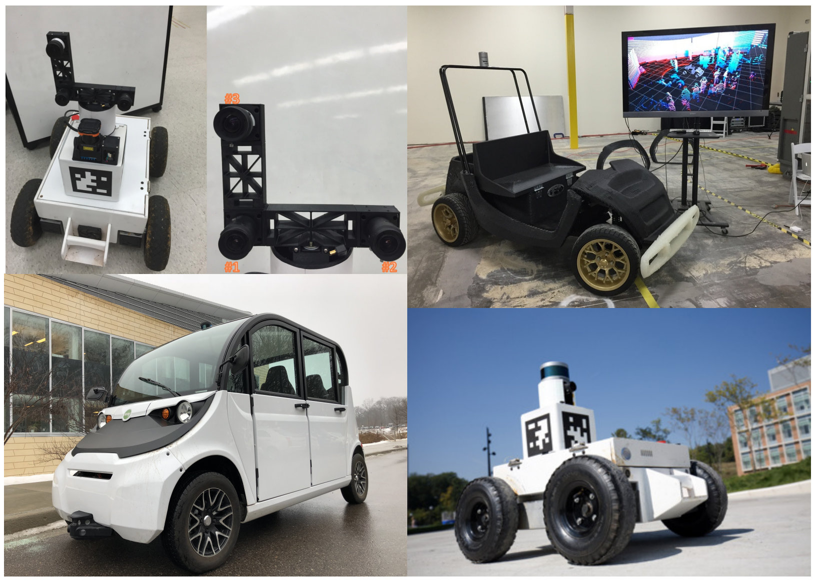


Figure 1.2: Robot applications. mapping and localization is essential for autonomous mobile robots equipped with all kinds of sensors (Lidar, cameras, radars, etc).

1.3 Prior Work

SLAM algorithms are often used to build a map online [53]. It has been extensively studied by many researchers, serving as a central research topic for robotics for decades. Early approaches were largely derived from the EKF (Extended Kalman Filter) [15] and it continues to be used in many applications [22, 50, 103]. The EKF, however, has consistency issues [38] and computational and memory costs that grow quadratically with the state size. The EIF(Extended Information Filter) [88] improves the costs, as do the many variants of SEIF(Sparse Extended Information Filter) [24, 95, 99]. New variants, offering not only computational cost improvements but also interesting cost-versus-quality trade-offs, continue to be developed. The mapping problem has also been approached by means of Rao-Blackwellized particle filters [34, 65, 66].

Recently non-linear optimization [23, 40, 48] gain popularity because these methods offer additional cost/quality trade-offs, and have incorporated ideas and techniques from the machine learning and numerical computing community. The challenge is to extend the state of the art by making these systems faster so that the algorithms are available to use in larger environments with greater numbers of sensor observations. In Chapter 3, this thesis presents a high availability SLAM system, AprilSAM with a fast incremental optimization method for factor graph. Comparing to other state-of-art approaches, AprilSAM takes much less optimiztion time to obtain equivalent performance of accuracy.

Global localization is critical for planning the paths of autonomous robots. Many existing navigation methods perform localization as a pose estimation problem after building a map with the robot [54, 57, 82, 92]. AprilSAM presented in Chapter 3 can be employed when robot is operating in unknown environment to build a map, however, it is not applicable to the application such as search and rescue in which robots may be called upon to operate in a novel environment. GPS could supply this position information when map is not built ahead. However, its performance significantly deteriorates in the surrounding of tall buildings and indoors. Since a new map can't be built immediately, the only way to solve this problem is to re-use existing map resources even though they were not designed for robots at first place.

Early approaches [89, 98] used satellite images. Their key contribution is matching images with strong view point changes by generating virtual affine views. Kim and Walter [46] propose to learn feature matching between ground-level images and satellite image directly using deep neural network. Instead of using satellite imagery, Torii et al. [97] have matched a robot view with Street View panoramas by matching descriptors computed directly on it. However, it only provides topological localization. Agarwal [1] extended the work to be able to compute a full 6DOF metrical localization on the panoramic images. Those methods that match images based on image feature descriptors are not applicable to indoor applications. In Chapter 4, this thesis presents a high availability indoor localization system, FLAG, that provides global positioning with a floor plan as prior map. It is the first concrete step of making mapping and localization available for robots navigating or planning path in a novel indoor environment. The novelty of the method is that it does not require additional map building as floor plans already exist for indoor localization.

FLAG is not applicable to every scenario. The majority of existing global localization systems are matching current observations to prior information inside a pre-built map. Lots of previous research [28, 54, 55] focused on a dedicated map building process using SLAM algorithms. For range-based sensors (e.g. Lidar), a popular map representation is the occupancy grid map, which discretizes the mapped area into cells of equal size. A high-resolution grid map provides a detailed representation of the world, but it is susceptible to feature aliasing since a sensor's view is noisy and often ambiguous. Such aliasing leads to poor localization performance. This limits the availability of algorithms for many applications such as autonomous driving which requires reliability. In Chapter 5, this thesis presents MOSS, a machine learning-based map optimization algorithm learns to adjust maps to support more robust localization. Unlike existing mapping approaches [13, 14, 41] focus primarily on controlling the cost-versus-quality trade-offs of online SLAM, this thesis seeks to optimize a map for achieving better localization performance.

CHAPTER 2

Preliminaries

2.1 Probabilistic Formulation of Mapping and Localization

Mapping can be seen as a process of building a consistent global coordinate system of the world; and localization is to establish correspondence between map coordinate system and the robot's local coordinate system. The goal of mapping and localization is to estimate the both world and robot's states X given all observations Z. According to Bayes law, the maximum likelihood solution is:

$$\begin{aligned} X_{ML} &= \arg\max_x P(X = x|Z = z) \\ &= \arg\max_x P(Z = z|X = x) \end{aligned} \tag{2.1}$$

This could be solved in two different ways. The first is an Bayes filter based method formulated as (2.2). The world and robot's states are modeled as a probabilistic belief $bel(x_t)$ and updated recursively based on $bel(x_{t-1})$ and new observations.

$$\begin{aligned} b\bar{e}l(x_t) &= \int p(x_t|u_t, x_{t-1}) bel(x_{t-1}) dx_{t-1} \\ bel(x_t) &= \eta p(z_t|x_t) b\bar{e}l(x_t) \end{aligned} \tag{2.2}$$

Where $p(x_t|u_t, x_{t-1})$ is the motion model and $p(z_t|x_t)$ is the measurement model.

Another solution is to take entire history into account with the assumption that observations are conditional independent given the states shown in (2.3).

$$X_{ML} = \arg\max_x \prod_{t=t_0}^{T} P(Z = z_t|X = x_t) \tag{2.3}$$

If we assume motion model and measurement model being Gaussian distribution,

$$\begin{aligned} p(x_t|u_t, x_{t-1}) \sim \mathcal{N}(\boldsymbol{x}_t - g(\boldsymbol{x}_{t-1}, \boldsymbol{u}_t)), R) \\ p(z_t|x_t) \sim \mathcal{N}(\boldsymbol{z}_t - h(\boldsymbol{x}_t), Q) \end{aligned} \tag{2.4}$$

we can obtain X_{ML} by solving the the nonlinear least-squares problem

$$X_{ML} = \arg\min_{x} \sum_t (\|\boldsymbol{z}_t - h(\boldsymbol{x}_t))\|^2_Q + \|\boldsymbol{x}_t - g(\boldsymbol{x}_{t-1}, \boldsymbol{u}_t)\|^2_R)^1 \tag{2.5}$$

where $\boldsymbol{z}_t$ is the observation with corresponding information matrix Q, $\boldsymbol{x}_t$ are the states involved in that observation, $h(\boldsymbol{x}_t)$ is the measurement model, and $g(\boldsymbol{x}_{t-1}, \boldsymbol{u}_t)$ is the motion model with corresponding information matrix R.

2.2 Factor Graph Model

Though there are two major formulations of mapping and localization (see in Sec. 2.1), lots of methods exist for solving this problem. In this thesis, we focus on developing algorithms to achieve high availability based on factor graph [20, 58]. This section describes factor graph and then moves on to geometrical and mathematical conventions.

2.2.1 Pose/Feature Graph

In a factor graph there are nodes for unknowns and probability factors defined on them. The graph structure expresses which unknowns are involved in each factor. Mapping and localization problems can be described in terms of a factor graph. The factor graph allows us to specify a joint density as a product of probability factors similar as in (2.3). For pose/feature graph, nodes are robot's poses and landmark positions, and probability factors are constraints among poses and landmarks. Pose is defined as the position and orientation of a robot at a particular point in time. Over time, the continuous trajectory of the robot is discretized into a set of poses. A constraint is represented as an edge in the pose graph. Constraints can express virtually any type of information, including a full rigid-body transformation (Sec. 2.2.3.1), or a range constraint and bearing constraint (Sec. 2.2.3.2), or a position constraint (Sec. 2.2.3.3). In this thesis, we model the uncertainty of constraints as Gaussian distribution. In other word, the probability of factors are represented by Gaussian distribution. This thesis focuses on two-dimensional pose graphs and rigid-body constraints, due to their ubiquity and applicability to many mapping systems.

[1] $\|X\|^2_\Omega = X^T \Omega X$

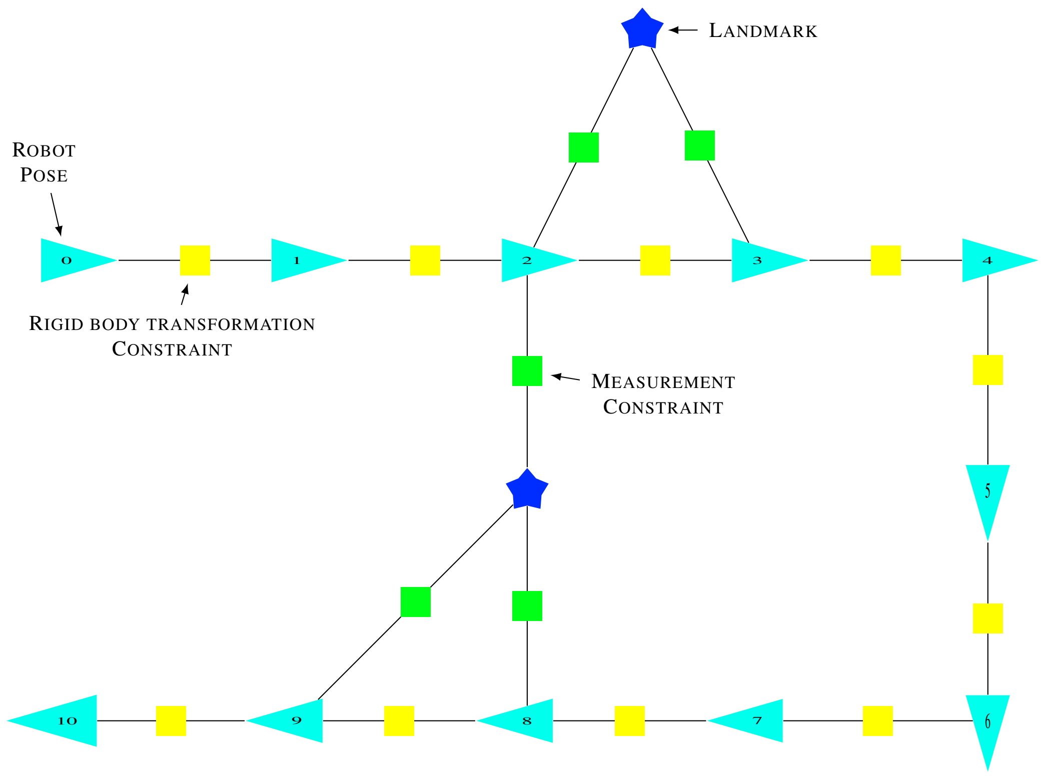


Figure 2.1: Sample pose/feature graph. The robot trajectory is discretized into a series of poses (cyan triangles) from which the robot observes landmarks (blue stars). Both poses and landmarks are nodes in the graph. Graph edges represent constraints between the two nodes, and squares represent the probability factors.

2.2.2 Factor Graph Optimization

2.2.2.1 Chi Square Error

A factor f in the factor graph is a single constraint relates the state nodes x to some observed quantity z. For a particular factor i that relates to state nodes **x** and observation z_i , we write:

$$f_i(\mathbf{x}, z_i) = P_i(z_i|\mathbf{x}) \tag{2.6}$$

If we assume that each factor in the graph are statistically independent, then MAP formulation in (2.1) becomes:

$$\begin{aligned} X_{ML} &= \arg\max_x \prod_i P_i(z_i|x) \\ &= \arg\max_x \prod_i factor_i(x, z_i) \end{aligned} \tag{2.7}$$

When we model the uncertainty of factor i as Gaussian distribution with co-variance as $\Sigma = \Omega^{-1}$, we will get:

$$X_{ML} = \arg\min_x \sum_i \|z_i - f_i(x)\|^2_\Omega \tag{2.8}$$

Note that f in the equation represents the measurement model.

The χ^2 error for the factor graph is defined as:

$$\chi^2 = \sum_i \|z_i - f_i(x)\|^2_\Omega \tag{2.9}$$

The χ^2 error gives us a means to compare different estimates, and it is a good measure in that it is the quantity that the optimization algorithms are explicitly attempting to minimize. However, keep in mind that The χ^2 error is not necessarily a good measure of mapping and localization quality because different state estimation distortions affect the χ^2 error to different degrees [74]. In practice, if we have X_{opt}, we can use metric $\|x - x_{opt}\|$ to evaluate the mapping and localization quality. But we need to compute two different metrics so that positions and rotations are considered separately since state vector having different units (positions in meters and rotations in radians).

2.2.2.2 Linearization

The objective function in (2.8) is not generally linear in the state variables. Optimization methods [4, 62, 94] can be applied here for minimizing the objective function. However, a serviceable approximation [25] can usually be obtained by linearizing the equations around the current state estimate x_0:

$$\begin{aligned} F_i &= f_i(x_0) \\ J_i &= \frac{\partial f_i}{\partial x}|_{x_0} \\ f_i(x) &= F_i + J_i(x - x_0) \end{aligned} \tag{2.10}$$

Then we can write the χ^2 error in terms of this linearization. First we write the residual error of the constraint as $r_i = z_i - f_i(x_0) = Z_i - F_i$. We also let $\delta_x = x - x_0$. These substitutions allow us to rewrite 2.8 as:

$$\chi^2 \approx \sum_i \|J_i\delta_x - r_i)\|^2_\Omega \tag{2.11}$$

If we stack the J_i and r_i matrices, and create a block-diagonal matrix from Ω matrices, we can write:

$$\chi^2 \approx \|J\delta_x - r)\|^2_\Omega \tag{2.12}$$

Where:

$$J = \begin{bmatrix} \cdots & J_1 & \cdots \\ \cdots & J_2 & \cdots \\ & \vdots & \end{bmatrix} \quad r = \begin{bmatrix} r_1 \\ r_2 \\ \vdots \end{bmatrix} \quad \Omega = \begin{bmatrix} \Omega_1 & & \\ & \Omega_2 & \\ & & \ddots \end{bmatrix} \tag{2.13}$$

After linearization, the optimization becomes a linear least square problem, we can differentiate 2.13 with respect to δ_x to find the value of δ_x that minimized the χ^2 error:

$$\begin{aligned} \frac{\partial \chi^2}{\partial \delta_x} &= 2J^TQJ\delta_x - 2JQr = 0 \\ (J^TQJ)\delta_x &= J^TQr \end{aligned} \tag{2.14}$$

The value δ_x represents a change in the state variable that will minimize the χ^2 error. Due to the non-linearity of the constraints, computing δ_x will generally not reduce the error to the global minimum in a single iteration: several iterations may be required in practice.

2.2.2.3 Cholesky Decomposition

There are many ways to solve the linear equation in 2.13: 1) direct methods [96] like inverting, Gaussian Elimination [87], QR factorization [35]; and 2) iterative methods like Jacobi method [84], conjugate gradient descent [51]. Directly inverting J^TQJ is impractical in many cases. In this thesis, we use Cholesky decomposition [64] for solving linear equation in 2.13.

Cholesky decomposition is commonly used to solve the normal equations $A^TAx = A^Tb$ that characterize the least squares solution to the overdetermined linear system. Suppose the goal is to solve linear equation $Ax = b$ and A is positive definite, then A can be factorized as $A = R^TR$, where R is upper triangular matrix. Then $R^TRx = b$ will be solved in two steps: 1) solve the lower triangular system $R^Ty = b$; 2) solve the upper triangular system $R^Tx = y$.

Given a 3×3 matrix A factorized to be R^TR:

$$A = R^TR = \begin{bmatrix} R_{11} & 0 & 0 \\ R_{12} & R_{22} & 0 \\ R_{13} & R_{23} & R_{33} \end{bmatrix} \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ 0 & R_{22} & R_{23} \\ 0 & 0 & R_{33} \end{bmatrix} \tag{2.15}$$

$$= \begin{bmatrix} R_{11}^2 & R_{11}R_{12} & R_{11}R_{13} \\ R_{12}R_{11} & R_{12}^2 + R_{22}^2 & R_{12}R_{13} + R_{22}R_{23} \\ R_{13}R_{11} & R_{13}R_{12} + R_{23}R_{22} & R_{13}^2 + R_{23}^2 + R_{33}^2 \end{bmatrix} \tag{2.16}$$

We obtain the following:

$$R = \begin{bmatrix} \sqrt{A_{11}} & \frac{A_{12}}{R_{11}} & \frac{A_{13}}{R_{11}} \\ 0 & \sqrt{A_{22} - R_{12}^2} & \frac{A_{23} - R_{12}R_{13}}{R_{22}} \\ 0 & 0 & \sqrt{A_{33} - R_{13}^2 - R_{23}^2} \end{bmatrix} \tag{2.17}$$

Therefore we can get following formulae for the entries of R (i is row index and j is the column index). From (2.18) we can see that the whole Cholesky decomposition process is done row by row.

$$\begin{aligned} R_{i,i} &= \sqrt{A_{i,i} - \sum_{k=1}^{i-1} R_{k,i}^2} \\ R_{i,j} &= \frac{1}{R_{i,i}} \left(A_{i,j} - \sum_{k=1}^{i-1} R_{k,i} R_{k,j} \right) \quad j > i. \end{aligned} \tag{2.18}$$

2.2.2.4 Variable Ordering for Cholesky Decomposition

Variable reordering is equivalent to row and column exchanges on a matrix. A good variable reordering [5, 17, 31] helps increase the sparsity structure of the factorized matrix. Different reordering techniques produce different sparsity structure, but all of them try to find a permutation matrix P that minimize fill-in. Unfortunately, computing a reordering which minimizes fill-in is NP-complete [105]. Agarwal and Olson [2] have presented a comprehensive empirical analysis on different variable reordering methods. Note that different reordering techniques do not change the solution but just generate different sparsity patterns. In this section, we will show how variable ordering affects the sparsity structure of the factorized matrix.

Given a 5×5 matrix A:

$$A = \begin{bmatrix} X_{11} & X_{12} & X_{13} & X_{14} & X_{15} \\ X_{12} & X_{22} & X_{23} & 0 & 0 \\ X_{13} & X_{23} & X_{33} & X_{34} & 0 \\ X_{14} & 0 & X_{34} & X_{44} & X_{45} \\ X_{15} & 0 & 0 & X_{45} & X_{55} \end{bmatrix} \tag{2.19}$$

where X_{ij} represents nonzero element at row i and column j.

We can obtain matrix R:

$$R = \begin{bmatrix} X & X & X & X & X \\ & X & X & X & X \\ & & X & X & X \\ & & & X & X \\ & & & & X \end{bmatrix} \tag{2.20}$$

where X represents nonzero element. We permute matrix A using matrix P:

$$P = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{2.21}$$

Then we obtain matrix $A_p = AP$:

$$A_p = \begin{bmatrix} X_{15} & X_{14} & X_{13} & X_{12} & X_{11} \\ 0 & 0 & X_{23} & X_{23} & X_{12} \\ 0 & X_{34} & X_{33} & X_{23} & X_{13} \\ X_{45} & X_{44} & X_{34} & 0 & X_{14} \\ X_{55} & X_{45} & 0 & 0 & X_{15} \end{bmatrix} \tag{2.22}$$

Final R_p will be:

$$R_p = \begin{bmatrix} X & X & & & X \\ & X & X & & X \\ & & X & X & X \\ & & & X & X \\ & & & & X \end{bmatrix} \tag{2.23}$$

Comparing (2.20) with (2.23), we can see that variable ordering does affect the sparsity structure of factorized matrix.

2.2.2.5 Connection Between Factor Graph and Cholesky Decomposition

Fig.2.2 shows an example of pose graph. In the graph, blue triangles are poses and red squares represent rigid body transformation constraint between two poses. We can get the information matrix $I = J^TQJ$ in (2.14):

$$I = \begin{bmatrix} X_{11} & X_{12} & X_{13} & X_{14} & X_{15} \\ X_{12} & X_{22} & X_{23} & 0 & 0 \\ X_{13} & X_{23} & X_{33} & X_{34} & 0 \\ X_{14} & 0 & X_{34} & X_{44} & X_{45} \\ X_{15} & 0 & 0 & X_{45} & X_{55} \end{bmatrix} \tag{2.24}$$

Information matrix I encodes the factor graph structure. Comparing the graph structure with the information matrix, we can see that if there exist factors between two poses, there will be a corresponding nonzero element in the matrix. For example, pose 1 and pose 5 has a factor between them, and there is a nonzero element X_{45} in the matrix I.

As shown in (2.18), the decomposition process is done row by row. Performing Cholesky decomposition is same as eliminating node in the graph one by one following the variable ordering.The variable ordering determines the sparsity of final factorized matrix. The Cholesky factorized matrix R of I with the ordering as x_1, x_2, x_3, x_4, x_5.

$$R = \begin{bmatrix} X & X & X & X & X \\ & X & X & X & X \\ & & X & X & X \\ & & & X & X \\ & & & & X \end{bmatrix} \tag{2.25}$$

When we first eliminate x_1 in the pose graph, we induce several new connections: x_2 and x_4, x_2 and x_5, and x_3 and x_5. These new connections will be reflected in matrix R as nonzeros elements in the corresponding positions - R_{24}, R_{25}, R_{35}.

If we arrange variable ordering as x_5, x_4, x_3, x_2, x_1. When we first eliminate x_5 in the graph, we will not induce any new connections. So R_{13}, R_{14} in final R matrix are all zeros:

$$R = \begin{bmatrix} X & X & & & X \\ & X & X & & X \\ & & X & X & X \\ & & & X & X \\ & & & & X \end{bmatrix} \tag{2.26}$$

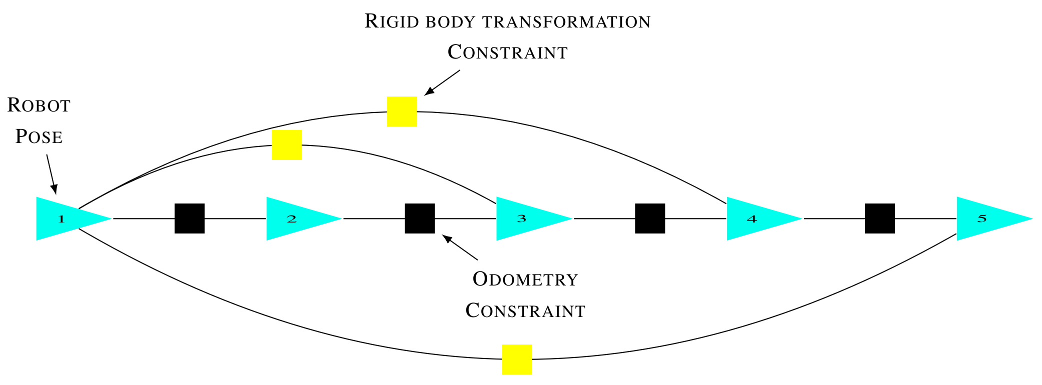


Figure 2.2: A pose graph. Cyan triangles are poses. Squares represent rigid body transformation constraint between two poses. The information matrix of this pose graph is shown in (2.24).

2.2.3 Common Constraint Forms

2.2.3.1 Rigid Body Transformation Constraint

A 2-D rigid body transformation is parameterized by three values, two translations in x and y, and a rotation θ. Those parameters can be used to compute a 3×3 transformation matrix as:

$$T_a = \begin{bmatrix} cos(\theta_a) & -sin(\theta_a) & x_a \\ sin(\theta_A) & cos(\theta_a) & y_a \\ 0 & 0 & 1 \end{bmatrix} \tag{2.27}$$

Suppose we can have a global coordinate frame G, then pose a can be parameterized as $\xi_{ga} = [x_{ga}, y_{ga}, \theta_{ga}]$ which encodes the transformation information from local frame to global frame. Based on sensor readings from IMU and wheel encoder or methods like scan matching [71], we can obtain a measurement of the transformation $\xi_{ab} = [x_{ab}, y_{ab}, \theta_{ab}]^T$ between pose a and pose b.

Connecting back to factor graph based optimization, the $F_{ab} = [\hat{x}_{ab}, \hat{y}_{ab}, \hat{\theta}_{ab}]^T$ will be:

$$\hat{x}_{ab} = cos(\theta_a)(x_b - x_a) + sin(\theta_a)(y_b - y_a) \tag{2.28}$$

$$\hat{y}_{ab} = -sin(\theta_a)(x_b - x_a) + cos(\theta_a)(y_b - y_a) \tag{2.29}$$

$$\hat{\theta}_{ab} = \theta_b - \theta_a \tag{2.30}$$

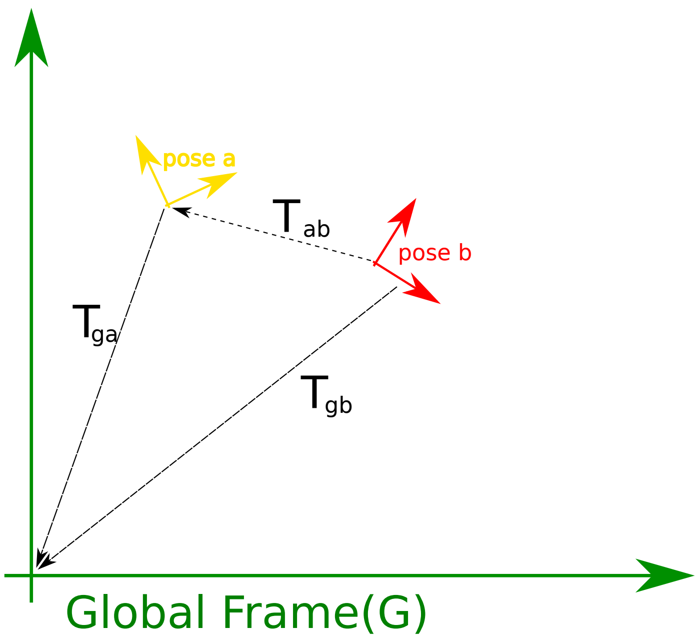


Figure 2.3: Coordinate transformations. Each robot pose represents a local coordinate system which can be related to the global coordinate by a rigid-body transform. Two local coordinate systems can also be related, e.g., $T_{gb} = T_{ga}T_{ab}$. Point P, expressed with respect to coordinate frame B, can be projected to coordinate frame A by $T_{ab}P$ and to the global coordinate frame by either $T_{gb}P$ or $T_{ga}T_{ab}P$.

And r_{ab} will be:

$$r_{ab} = \xi_{ab} - F_{ab} \tag{2.31}$$

The Jacobian matrix is:

$$\begin{aligned} d_x &= x_b - x_a,\ d_y = y_b - y_a \\ c &= cos(\theta_a),\ s = sin(\theta_a) \\ J_{ab} &= \frac{\partial F_{ab}}{\partial(\xi_a, \xi_b)} \\ &= \begin{bmatrix} -c & -s & -s * d_x + c * d_y & c & s & 0 \\ s & -c & -c * d_x - s * d_y & -s & c & 0 \\ 0 & 0 & -1 & 0 & 0 & 1 \end{bmatrix} \end{aligned} \tag{2.32}$$

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

$$\Sigma = \begin{bmatrix} \Sigma_{xx} & \Sigma_{xy} & \Sigma_{x\theta} \\ \Sigma_{yx} & \Sigma_{yy} & \Sigma_{y\theta} \\ \Sigma_{\theta x} & \Sigma_{\theta y} & \Sigma_{\theta\theta} \end{bmatrix} \tag{2.33}$$

2.2.3.2 Distance and Bearing Constraint

Different types of sensors produce different types of geometrical information. Monocular cameras can only measure the bearing to a feature. Various radio-frequency and acoustic sensors are capable of measuring the range between a feature and the robot, but cannot determine the bearing. A lidar, can measure both bearing and distance of features with respect to the robot. Assume the distance and bearing measurement observed from pose a to landmark l are $\gamma_{al} = [r_{al}, \phi_{al}]$. Connecting back to factor graph based optimization, the $F_{al} = [\hat{r}_{al}, \hat{\phi}_{al}]^T$ will be:

$$\hat{r}_{al} = \sqrt{(x_a - x_l)^2 + (y_a - y_l)^2} \tag{2.34}$$

$$\hat{\phi}_{al} = arctan2(y_a - y_l, x_a - x_l) - \theta_a \tag{2.35}$$

And r_{al} will be:

$$r_{al} = \gamma_{al} - F_{al} \tag{2.36}$$

Then Jacobian matrix is:

$$\begin{aligned} d_x &= x_a - x_l \\ d_y &= y_b - y_l \\ \phi_g &= arctan2(d_y, d_x) \\ J_{al} &= \begin{bmatrix} \frac{d_x}{\hat{r}_{al}} & \frac{d_y}{\hat{r}_{al}} & 0 & -\frac{d_x}{\hat{r}_{al}} & -\frac{d_y}{\hat{r}_{al}} \\ -\frac{d_y}{(1+\phi_g^2)d_x^2} & \frac{d_y}{(1+\phi_g^2)d_x^2} & -1 & \frac{1}{(1+\phi_g^2)d_x} & -\frac{1}{(1+\phi_g^2)d_x} \end{bmatrix} \end{aligned} \tag{2.37}$$

The uncertainty of ranging and bearing measurements can be represented by a Gaussian distribution:

$$\Sigma = \begin{bmatrix} \Sigma_{rr} & \Sigma_{r\phi} \\ \Sigma_{\phi r} & \Sigma_{\phi\phi} \end{bmatrix} \tag{2.38}$$

2.2.3.3 Position Constraint

GPS (Global Position System) can provide position information directly. Assume the position measurement observed for pose a is in global coordinate frame $\eta_{ga} = [x_{pos}, y_{pos}]$. Connecting back to factor graph based optimization, the $F_{ga} = [\hat{x}_{pos}, \hat{y}_{pos}]$ will be:

$$\begin{aligned} \hat{x}_{pos} &= x_a \\ \hat{y}_{pos} &= y_b \end{aligned} \tag{2.39}$$

And the r_a will be:

$$r_{ga} = \eta_{ga} - F_{ga} \tag{2.40}$$

Then the Jacobian matrix is:

$$J_{ga} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{2.41}$$

The uncertainty of position measurements can be represented by a Gaussian distribution:

$$\Sigma = \begin{bmatrix} \Sigma_{pos_x pos_x} & \Sigma_{pos_x pos_y} \\ \Sigma_{pos_y pos_x} & \Sigma_{pos_y pos_y} \end{bmatrix} \tag{2.42}$$

CHAPTER 3

AprilSAM: Real-time Smoothing and Mapping

3.1 Introduction

Simultaneous localization and mapping (SLAM) [6, 12, 21] is often used to build a map online. Computational complexity limits the scale of maps that can be built in most robot systems, which limits the *availability* of those methods to relatively small environments. A SLAM solution should be both fast and accurate; speed ultimately determines the size of the environment that the robot can operate in within real-time performance constraints, while numerical accuracy is important for generating high-quality maps and position estimates. In this chapter, we propose a real-time SLAM system, AprilSAM, that uses a new variable reordering algorithm coupled with fast incremental Cholesky factorization to improve system accuracy while maintaining real-time performance.

Smoothing methods formulate SLAM as a nonlinear least squares problem, which they solve to convergence by linearizing at a current estimate. Early smoothing methods (e.g. $\sqrt{\text{SAM}}$ [19]) employ a batch update at every step, making them too slow for real-time usage in large-scale maps. Incremental approaches such as iSAM [42] and iSAM2 [43] try to avoid batch updates, instead solving the least squares problem incrementally.

iSAM [42] uses matrix factorization for incremental updates and periodically performs batch updates. Because errors can accumulate without the linearization point being updated, iSAM's batch update does not guarantee that the nonlinear optimization converges to the optimal solution. Variable ordering has a large effect on performance as well: the computational cost of an incremental update can vary by an order of magnitude depending on the order in which variables are marginalized.

iSAM2 [43] introduces the Bayes tree to achieve incremental variable re-ordering and fluid re-linearization, eliminating the need for periodic batch updates. iSAM2 primarily focuses on improving running time, trading off some solution accuracy. Although iSAM2 elegantly connects matrix factorization with graphical model inference, implementing the

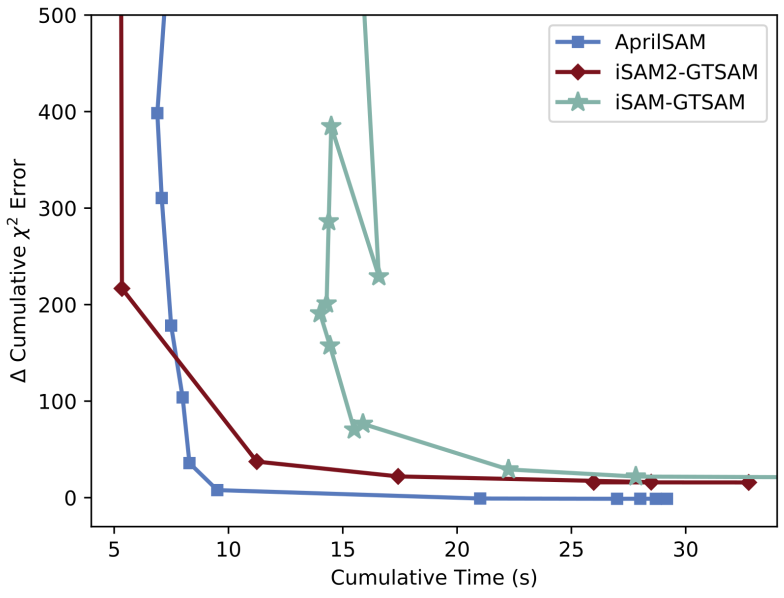


Figure 3.1: Cost-Accuracy trade-off for iSAM, iSAM2 and AprilSAM. All three algorithms have free parameters that trade off accuracy versus computational costs. The plot above shows the performance of these algorithms as the free parameter is varied. AprilSAM achieves the lowest absolute error and does so with generally lower computational times than both iSAM-GTSAM and iSAM2-GTSAM. However, iSAM2-GTSAM is faster for low-accuracy applications. Note that the relationship between the varied parameters and the cumulative time and χ^2 error is non-monotonic.

Bayes tree is complex.

In this chapter, we present a SLAM algorithm based on fast incremental Cholesky factorization, which we call AprilSAM. Many of AprilSAM's core steps can be implemented easily using standard sparse linear systems libraries, which both eases implementation and allows highly optimized libraries to be used. In spite of this simplicity, AprilSAM achieves state-of-the-art performance with highly accurate solutions.

The effectual contributions of AprilSAM are:

- Better absolute error than either iSAM or iSAM2 on benchmark datasets and
- Generally lower error for a given amount of computation time.

The technical contributions enabling these results include:

- A dynamic variable ordering algorithm that lowers the amount of computation expected in subsequent incremental updates,
- An algorithm that provides criterion for selecting between incremental and batch updates, and

- A partial backsolve method that reduces the amount of computation involved in the incremental Cholesky decomposition.

3.2 Related Work

There has been much progress over the past decade to develop online SLAM solutions, much of which has been devoted to speeding up or avoiding altogether the batch updates of smoothing algorithms. We focus our review of related work on these types of approaches. For a more general overview of SLAM research, please refer to a recent survey (e.g. [11, 12]).

Kaess et al. [42] propose iSAM, an incremental smoothing and mapping algorithm that incrementally updates the QR factorization of the smoothing information matrix. To avoid unnecessary fill-in and accumulation of error, iSAM performs periodic variable re-ordering and re-linearization.

AprilSAM builds upon the foundational ideas of iSAM with some novel changes that lead to greater efficiency and higher accuracy. Although incremental updates are effective much of the time, a batch update is often necessary for the system to converge to the optimal solution. iSAM's periodic batch update does not account for accumulated state changes, leading to high-error solutions when a linearization point differs from the optimal solution. In this chapter, we provide an algorithm that adaptively selects between incremental and batch updates (see Sec. 3.4.4). Additionally, iSAM's incremental updates can become inefficient due to poor variable ordering, especially in the case of large loop closures. We propose a variable ordering algorithm that reduces the amount of computation needed for subsequent incremental updates (see Sec. 3.4.3). Finally, the QR factorization employed by iSAM induces more nonzero elements during the factorization process than the Cholesky factorization does, leading to slower computations (see Sec. 3.3). Motivated by this observation, we utilize an incremental Cholesky factorization rather than iSAM's QR factorization.

Kaess et al. [43] evolve the original iSAM to propose iSAM2, which achieves improvements in efficiency through incremental variable re-ordering and fluid re-linearization, eliminating the need for periodic batch steps. iSAM2 introduces the Bayes tree, a novel data structure for sparse nonlinear incremental optimization. Though iSAM2 elegantly connects matrix factorization with graphical model inference through the Bayes tree, its implementation is complex. in this chapter, we present an incremental Cholesky factorization approach that can be implemented easily with standard sparse linear systems libraries, which eases implementation and allows highly optimized libraries to be used. iSAM2's fluid re-linearization and incremental re-ordering are used primarily to improve its running time, trading off some solution accuracy. As we show in Sec. 3.5, AprilSAM yields solutions with lower absolute error as well as generally lower error given equivalent running time.

Polok et al. [81] present an incremental block Cholesky factorization for solving non-linear least square problems. Their algorithm maintains both the smoothing information matrix and the factorization matrix while AprilSAM only keeps the latter one. Polok uses CCOLAMD [17] to order variables, which only considers the most recently accessed variables. However, we observe that computing variable orderings is hardly a bottleneck in this incremental process. Matrix reconstruction and the partial Cholesky decomposition take most of time, and their running time depends on variable ordering. in this chapter, we propose a variable ordering algorithm for batch updates that anticipates possible upcoming loop closures. As we show in Sec. 3.5, this allows for faster incremental updates in subsequent steps.

Ni and Dellaert [69], Ni et al. [70] propose a novel batch algorithm for SLAM problems that distributes the workload in a hierarchical manner. They show how the original SLAM graph can be partitioned recursively into multiple-level submaps using the nested dissection algorithm [93], which leads to a cluster tree. By employing the nested dissection algorithm, their method greatly minimizes the dependencies between two subtrees, and the original SLAM graph can be optimized using a bottom-up inference along the corresponding cluster tree. In this thesis, we do not explicitly apply the nested dissection algorithm to generate the cluster tree structure. Instead, we apply a min-heap based ordering method to allow faster incremental updates for large loop closures. Loop closures increase the connection degrees of nodes in a pose-graph, so they tend to be the separator point in the nested dissection algorithm. Our proposed algorithm also leverages this natural cluster tree structure, which only requires the update of a partial matrix in incremental factorization.

3.3 Problem Statement

In this section, we formulate Pose-SLAM as a least squares problem. Our goal is to estimate the robot states X given rigid transformation observations Z. According to Bayes law, the maximum likelihood solution is:

$$\begin{aligned} X_{ML} &= \arg\max_x P(X = x | Z = z) \\ &= \arg\max_x P(Z = z | X = x) \end{aligned} \tag{3.1}$$

If we assume a Gaussian measurement model, we can obtain X_{ML} by solving the the nonlinear least-squares problem

$$X_{ML} = \arg\min_{x} \sum_{k} \|\boldsymbol{z}_k - h(\boldsymbol{x}_{\boldsymbol{i}k}, \boldsymbol{x}_{\boldsymbol{j}k})\|^2_{\Omega_k} \tag{3.2}$$

where $\boldsymbol{z}_k$ is the k^{th} rigid transformation observation with corresponding information matrix Ω_k, $\boldsymbol{x}_{\boldsymbol{i}k}$ and $\boldsymbol{x}_{\boldsymbol{j}k}$ are the nodes involved in that observation, and $h(\boldsymbol{x}_{\boldsymbol{i}k}, \boldsymbol{x}_{\boldsymbol{j}k})$ is the expected rigid transformation based on the measurement model.

We linearize h as

$$h(\boldsymbol{x}_{\boldsymbol{i}k}, \boldsymbol{x}_{\boldsymbol{j}k}) \approx h(\boldsymbol{\mu}_{\boldsymbol{i}k}, \boldsymbol{\mu}_{\boldsymbol{j}k}) + \begin{bmatrix} J_k^{i_k}(\boldsymbol{\mu}_{\boldsymbol{i}k}) & J_k^{j_k}(\boldsymbol{\mu}_{\boldsymbol{j}k}) \end{bmatrix} \begin{bmatrix} \delta\boldsymbol{x}_{\boldsymbol{i}k} \\ \delta\boldsymbol{x}_{\boldsymbol{j}k} \end{bmatrix} \tag{3.3}$$

where $\boldsymbol{\mu}_{\boldsymbol{i}k}$ is the current estimate for node $\boldsymbol{x}_{\boldsymbol{i}k}$, and

$$J_k^{i_k}(\boldsymbol{\mu}_{\boldsymbol{i}k}) = \frac{\delta h(\boldsymbol{x}_{\boldsymbol{i}k}, \boldsymbol{x}_{\boldsymbol{j}k})}{\delta \boldsymbol{x}_{\boldsymbol{i}k}}|_{\boldsymbol{x}_{\boldsymbol{i}k} = \boldsymbol{\mu}_{\boldsymbol{i}k}} \tag{3.4}$$

with analogous definitions for $\boldsymbol{\mu}_{\boldsymbol{j}k}$ and $J_k^{j_k}(\boldsymbol{\mu}_{\boldsymbol{j}k})$. Applying this linearization to (3.2), we obtain the least squares formulation

$$\delta_X = \arg\min_{\delta_x} \|A\delta_x - b\|^2 \tag{3.5}$$

where $A = \Omega^{\frac{1}{2}} J$ and $b = \Omega^{\frac{1}{2}}(z - h(\mu_i, \mu_j))$.

Many methods exist to solve (3.5), with the most common direct approach being matrix factorization. In iSAM [42], Kaess solves the linear system using QR factorization as

$$QR\delta_x = b \tag{3.6}$$

where $A = QR$. In contrast, we utilize the following Cholesky factorization to solve (3.5) by solving the two triangular matrix equation

$$R^T R\delta_x = b^* \tag{3.7}$$

where $A^* = A^T A = R^T R$ and $b^* = A^T b$.

Both factorization methods (Cholesky [30] and QR [27]) compute the same matrix R. In the GraphSLAM problem, however, QR factorization induces many more non-zero ele-

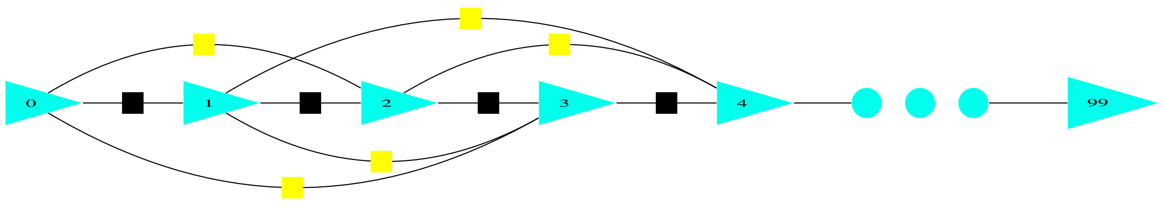


Figure 3.2: A factor graph containing 100 (x, y, θ) nodes, each of which is connected to the three nodes ahead of itself.

ments during this computation than Cholesky factorization does, slowing down the sparse matrix math involved in the computation. GraphSLAM problems have more factors than nodes, making A (used by QR) a tall matrix and $A^T A$ (used by Cholesky) a relatively small square matrix.

We tested both factorization methods on the graph shown in Fig.3.2. This graph has 100 (x, y, θ) nodes, each of which is connected to the three nodes ahead of itself. In Fig.3.3, we show the number of nonzero elements at each process step for QR factorization using the Householder method and for Cholesky factorization. Even though the number of nonzero elements is equal at the end of process, the QR factorization induces more nonzero elements at each step, leading to longer computation time.

Motivated by this observation, we utilize an incremental Cholesky factorization approach instead of the incremental QR factorization of Kaess et al. [42]. QR is expected to produce more accurate results due to its lower condition number, but intuitively, SLAM graphs usually aren't that poorly conditioned. Observations don't vary by 7 orders of magnitude in certainty. The numerical results bear out that stability is not affecting accuracy.

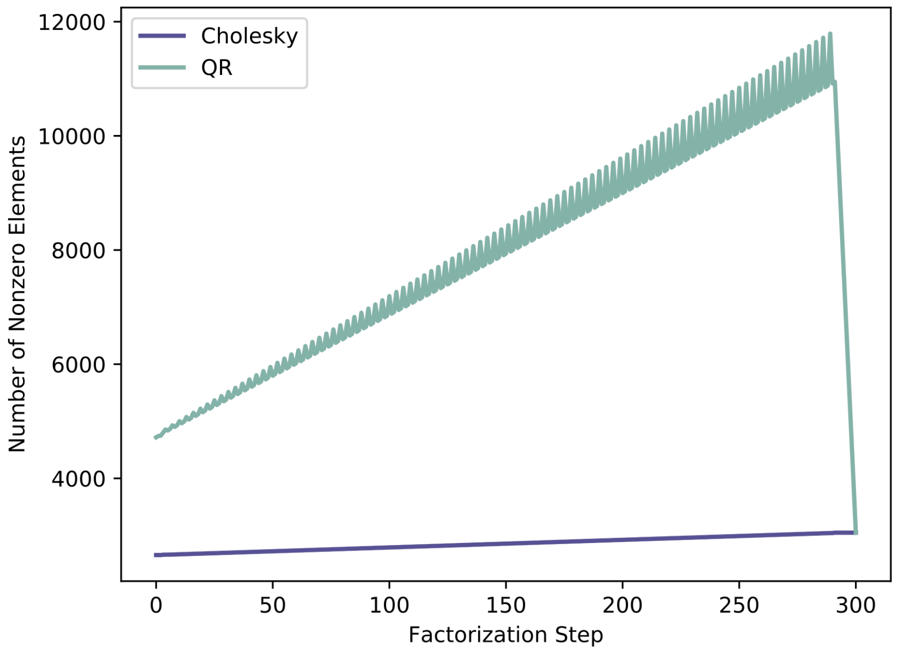


Figure 3.3: The number of nonzero elements at each step of the matrix factorization for the graph shown in Fig.3.2. QR factorization induces more nonzero elements than Cholesky during the factorization process, leading to slower computations.

3.4 Approach

In this section, we present AprilSAM, the incremental SLAM algorithm shown in Alg. 3.1. AprilSAM updates R and y incrementally, a process we describe in Sec. 3.4.1 and Sec. 3.4.2. Each time AprilSAM performs a batch update, we apply a dynamic re-ordering algorithm as described in Sec. 3.4.3. The algorithm AprilSAM uses to select between batch and incremental updates is listed in Sec. 3.4.4.

Moving forward, we make a small notational update for the sake of clarity, dropping the * from A and b. That is, A^* and b^* become A and b, respectively.

Algorithm 3.1 AprilSAM

```
1: function AprilSAM(Graph G, NewFactors f, $R$, $y$, $NL_T$, $\delta_T$, $\Delta_T$)
2:     $G \leftarrow$ AugmentGraph($G$, $f$)
3:     $(R, y) \leftarrow$ IncrementalUpdate(G)
4:     $\delta_x \leftarrow$BackwardSubstitute($R$, $y$)
5:     $L \leftarrow \emptyset$                                  ▷ Set of nodes to be linearized
6:     for all nodes $x_i \in G$ do
7:         $x_i \leftarrow x_i + \delta_{x_i}$
8:         if $\delta_{x_i} \geq \delta_T$ then
9:             $L \leftarrow L \cup x_i$
10:    $n_{lc} \leftarrow$CountPossibleLoopClosureNodes($G$)
11:    if RunTime(incremental) $\times n_{lc} \geq$ runTime(batch) or $|L| \geq NL_T$ or $\|\delta_x\| \geq \Delta_T$
   then
12:        for all nodes $x_i \in G$ do
13:            UpdateLinearizationPoint($x_i$)
14:        $(R, y) \leftarrow$ BatchUpdate($G$)
15:        $\delta_x \leftarrow$BackwardSubstitute($R$, $y$)
16:        for all nodes $x_i \in G$ do
17:            $x_i \leftarrow x_i + \delta_{x_i}$
18: end function
```

3.4.1 Incremental Cholesky Decomposition

Whereas Polok et al. [81] maintains both the information matrix A and the factorization matrix R, our algorithm only requires the latter. Recall the forms of these two matrices:

$$A = \begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{bmatrix} \tag{3.8}$$

$$R = \begin{bmatrix} R_{00} & R_{01} \\ \mathbf{0} & R_{11} \end{bmatrix} \tag{3.9}$$

An intermediate matrix R^{in} of the decomposition process has the form

$$R^{in} = \begin{bmatrix} R_{00} & R_{01} \\ \mathbf{0} & A_{11}^{in} \end{bmatrix} \tag{3.10}$$

where A_{11}^{in} is the filled-in version of submatrix A_{11} resulting from the partial factorization. Given the completed factorization R, we can compute A_{11}^{in} as

$$A_{11}^{in} = R_{11}^T R_{11} \tag{3.11}$$

This in turn allows us to reconstruct the intermediate matrix R^{in}.

Let $\tilde{A}$ be the evolved version of A following the addition of new information. This update only affects a portion A_{11} of the information matrix, with corresponding effects to R_{11} in the factorization. That is, the new information matrix $\tilde{A}$ and its factorization matrix $\tilde{R}$ have the forms:

$$\tilde{A} = \begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} + A_{new} \end{bmatrix} \quad \tilde{R} = \begin{bmatrix} R_{00} & R_{01} \\ \mathbf{0} & \tilde{R}_{11} \end{bmatrix} \tag{3.12}$$

Because $\tilde{A} = \tilde{R}^T \tilde{R}$, we can express the updated portion of the information matrix as

$$\tilde{R}_{11}^T \tilde{R}_{11} = A_{11}^{in} + A_{new} \tag{3.13}$$

Applying (3.11), we can re-write (3.13) in terms of only the previous factorization matrix R and the new information A_{new}. That is,

$$\tilde{R}_{11}^T \tilde{R}_{11} = R_{11}^T R_{11} + A_{new} \tag{3.14}$$

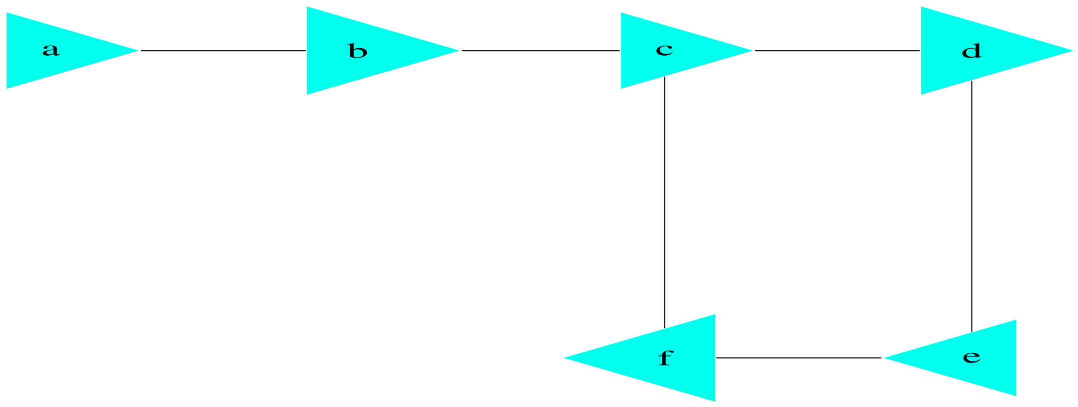
In Fig.3.4, we show an example of a batch update. By ordering nodes c and f at the end of matrix, we separate the graph into small clusters. This cluster structure accelerates the incremental update process.

Fig.3.5 shows an example of adding an odometry edge. Since the new node g is only connected to node f, only columns f and g of R will be changed.

Fig.3.6 shows an example of adding a loop closure. The new node g is connected to nodes f and b. Due to the cluster structure, nodes d and e are separated with loop closure node b, and those columns of R are not changed.

3.4.2 Solving the Triangular Matrix Equations

The Cholesky factorization of (3.7) can be used to solve for δ_x by successive forward and backward substitution. Namely, we solve $R^T y = b$ by forward substitution and use the result to solve $R\delta_x = y$ by backward substitution. As we argued in the previous section, only a portion of R and b change when performing an incremental update. We now show that we do not need to maintain b and solve the first equation every step. Instead, we can update y incrementally and only solve the second equation, $R\delta_x = y$.



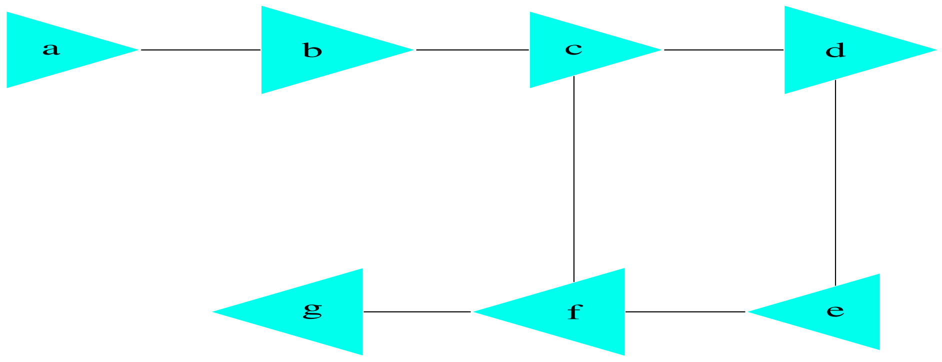
	a	b	d	e	c	f
a	x	x				
b	x	x			x	
d			x	x	x	
e			x	x		x
c		x	x		x	x
f				x	x	x

$A*$

	a	b	d	e	c	f
a	x	x				
b		x			x	
d			x	x	x	
e				x	x	x
c					x	x
f						x

R

Figure 3.4: An example of a batch update with min-heap based ordering. By ordering node c and node f at the end of the matrix, the graph is separated into small clusters. This cluster structure allows for faster incremental updates as shown in Fig.3.5 and Fig.3.6



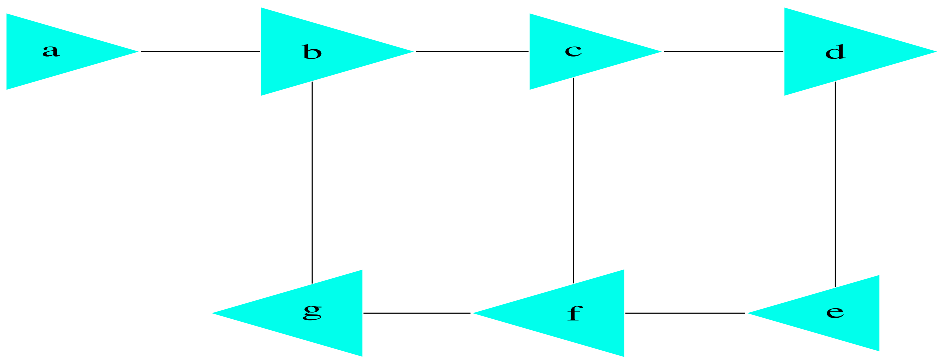
	a	b	d	e	c	f	g
a	x	x					
b		x			x		
d			x	x	x		
e				x		x	
c					x	x	
f						**x**	**x**
g							**x**

$A^{in} + A_{new}$

	a	b	d	e	c	f	g
a	x	x					
b		x			x		
d			x	x	x		
e				x		x	
c					x	x	
f						**x**	**x**
g							**x**

R

Figure 3.5: An example of adding an odometry edge. Since the new node g is only connected to node f, only columns f and g of R will be changed. The red crosses represent the reconstructed portion of intermediate matrix A^{in} plus the new information A_{new}, as detailed in (3.13). The red crosses represent the modified portion of R. No other elements are changed.



	a	b	d	e	c	f	g
a	x	x					
b		**x**			**x**		**x**
d			x	x	x		
e				x		x	
c					**x**	**x**	**x**
f						**x**	**x**
g							**x**

$A^{in} + A_{new}$

	a	b	d	e	c	f	g
a	x	x					
b		**x**			**x**		**x**
d			x	x	x		
e				x		x	
c					**x**	**x**	**x**
f						**x**	**x**
g							**x**

R

Figure 3.6: An example of adding a loop closure. The new node g is connected to nodes f and b. Due to the cluster structure, nodes d and e are separated by loop closure node b, so the corresponding columns of R are not changed.

Recall the forms of R^T, y, and b:

$$\begin{gathered} R^T y = b \\ \begin{bmatrix} R_{00}^T & \mathbf{0} \\ R_{01}^T & R_{11}^T \end{bmatrix} \begin{bmatrix} y_{00} \\ y_{10} \end{bmatrix} = \begin{bmatrix} b_{00} \\ b_{10} \end{bmatrix} \end{gathered} \tag{3.15}$$

When information is added, only a portion of the RHS changes. That is,

$$\tilde{b} = \begin{bmatrix} \tilde{b}_{00} \\ \tilde{b}_{10} \end{bmatrix} = \begin{bmatrix} b_{00} \\ b_{10} + b_{new} \end{bmatrix} \tag{3.16}$$

From (3.15), we obtain the equation for the affected portion of b:

$$b_{10} = R_{01}^T y_{00} + R_{11}^T y_{10} \tag{3.17}$$

Applying (3.16), we write the corresponding equation for the modified portion of $\tilde{b}$ as

$$\begin{aligned} \tilde{b}_{10} &= \tilde{R}_{01}^T \tilde{y}_{00} + \tilde{R}_{11}^T \tilde{y}_{10} \\ b_{10} + b_{new} &= R_{01}^T y_{00} + \tilde{R}_{11}^T \tilde{y}_{10} \end{aligned} \tag{3.18}$$

Finally, substituting the value of b_{10} from (3.17), we obtain an update step that allows us to solve y incrementally:

$$\tilde{R}_{11}^T \tilde{y}_{10} = R_{11}^T y_{10} + b_{new} \tag{3.19}$$

In Fig.3.7, we mark the affected portions of the matrix and vectors as red. Similar to the update of R in the incremental Cholesky decomposition, we only need to maintain $\boldsymbol{y}$ and reconstruct and solve for the affected portion each time. In this case, we do not need to re-calculate parts of $\boldsymbol{y}$ that are not changed.

3.4.3 Min-heap based Ordering

in this chapter, we present a min-heap based node ordering algorithm based on BHAMD [2]. Compared to the original BHAMD algorithm, this ordering has less non-zero fill-ins because it continually checks the scores of a removed node's neighbors and adds them back to correct list. This prevents the case in which a node with large initial score remains in the list even when its score is reduced because of node removal.

In addition, in this min-heap based algorithm, the score of a node not only depends on its connection degree but also its distance to the most recently added node. We can predict which nodes may be included in upcoming loop closures based on connections between

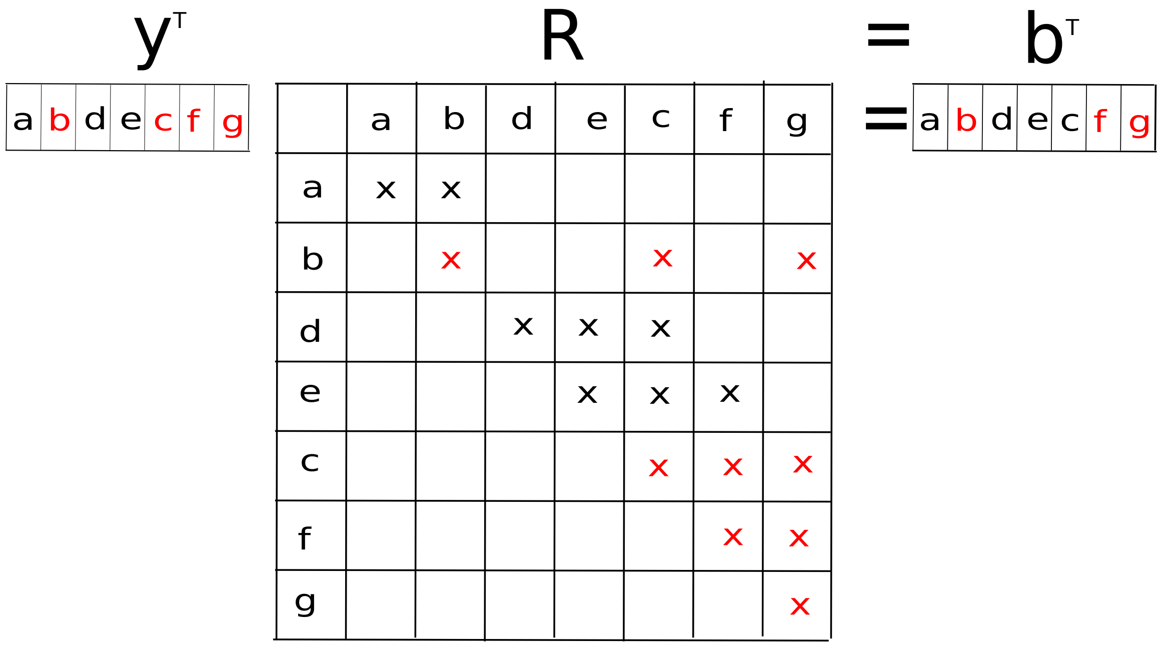


Figure 3.7: Incremental update of $\boldsymbol{y}$. Since only nodes b and f are connected to the newly added node g, in the residual vector $\boldsymbol{b}$, only b, f, and g will change. When we solve this linear system, we only need to solve b, c, f, and g in the $\boldsymbol{y}$ due to the parallel triangular structure. We mark the affected elements in red.

nodes and the radius at which loop closures are considered. Our ordering algorithm will order these nodes at the end of the matrix automatically based on their score.

Alg. 3.2 details the ordering algorithm. We begin by adding lists to the min-heap, where each list is associated with a possible node score. We then add each node to the proper list based on its score.

At each iteration, we extract from the heap the list that is associated with the minimum node score. We consider each node from this list in turn, along with its neighbors. Nodes whose score is no larger than the current minimum score are eliminated. Nodes with larger scores are moved to the list associated with that score.

The score of a node is given by

$$score = \begin{cases} N + \frac{1}{r}, & r < R_T \\ n, & else \end{cases} \tag{3.20}$$

where r is the distance to the most recently added node, R_T is the threshold inside of which loop closures are considered, n is the connection degree of the node, and N is the number of nodes in the graph.

Algorithm 3.2 Min-heap Ordering Procedure

```
1: Create a set of lists where each list is associated with a score and contains nodes having
   same scores.
2: Add all lists into a min heap: MH.
3: while MH is not empty do bestList = getMinList from MH minScore = MH.getMinKey
4:     for each node nd in bestList do
5:         if nd.score $\leq$ minScore then
6:             remove nd
7:             Update nd's score.
8:             for each node nnd in nd's neighbors do
9:                 if nnd.score $\leq$ minScore then
10:                    push it back into the bestList
11:                else
12:                    push it back into the correct list
13:        else
14:            push it back into the correct list
```

3.4.4 Incremental update vs. Batch update

Re-linearization in a batch update can reduce system error, but batch updates are typically more computationally expensive than incremental updates. iSAM attempts to handle this dilemma by performing batch updates at regular intervals. In contrast, AprilSAM performs batch updates when one of three conditions is met (see also in Alg. 3.1):

1. AprilSAM tracks nodes that have changed significantly in a set $L = \{x_i : \delta_{x_i} > \delta_T\}$. If enough nodes have undergone significant changes (i.e. $|L| > NL_T$), AprilSAM performs a batch update.

2. AprilSAM performs a batch update if the norm of the total state changes becomes too large (i.e. $\|\delta_x\| > \Delta_T$). Since the SLAM nonlinear least square problem involves repeatedly solving linear equations, this condition keeps the current solution from diverging too far from the optimal solution.

3. Due to the cost associated with reconstructing $R_{11}^T R_{11}$ in (3.14), incremental updates may actually take more time than batch updates when a large loop closure happens. This is especially true for poor variable orderings. In such a case, a batch update could save time by ordering variables involved in possible loop closures at the end, making subsequent incremental updates much faster. When the estimated time to execute an incremental update (i.e. the running time of the previous incremental update multiplied by the number of nodes involved in possible loop closures) is larger than the running time of the most recent batch update, AprilSAM chooses to perform

a batch update.

3.5 Evaluation

In this section, we evaluate AprilSAM on real-world benchmark datasets. We compare AprilSAM's performance to that of iSAM [42] and iSAM2 [43], which represents the current state-of-the-art. We also evaluate several variants of AprilSAM to gain greater insight into its performance.

3.5.1 Methodology

$\sqrt{\text{SAM}}$-Chol performs a batch update every step using the Cholesky decomposition, providing us with a performance baseline against which to compare the incremental approaches. We introduce here several variants of AprilSAM used in the evaluation to isolate each component of the system:

- *AprilSAM-FullSolve* removes from AprilSAM the mechanism for reducing the computation involved in solving the triangular matrix equations (Sec. 3.4.2), instead solving the full set of equations each time.

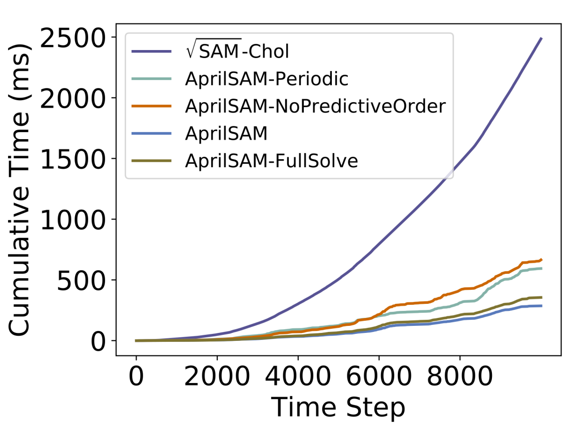
- *AprilSAM-Periodic* removes from AprilSAM the algorithm for selecting between incremental and batch updates (Sec. 3.4.4), instead performing batch updates periodically.

- *AprilSAM-NoPredictiveOrder* is the same as AprilSAM-Periodic, but replaces the variable reordering algorithm (Sec. 3.4.3) with one that simply orders the most recent graph node last and applies BHAMD [2] to the remaining nodes.

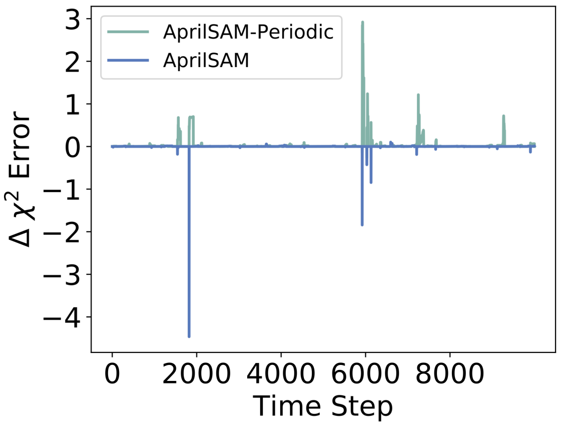
We use the original authors' implementation of iSAM2 in GTSAM [18], which we refer to as iSAM2-GTSAM. We tested the algorithms on two benchmark datasets: W10000 [32] and M3500 [77]. W10000 contains 10000 (x, y, θ) nodes and 64311 (x, y, θ) factors. M3500 contains 3500 nodes and 5453 factors.

3.5.2 Evaluation on W10000 and M3500

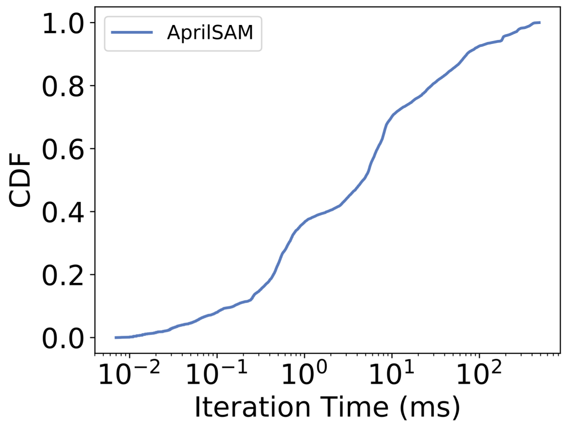
Figs. 3.8 and 3.9 show the performance of the tested algorithms on the W10000 dataset and M3500 dataset, respectively.



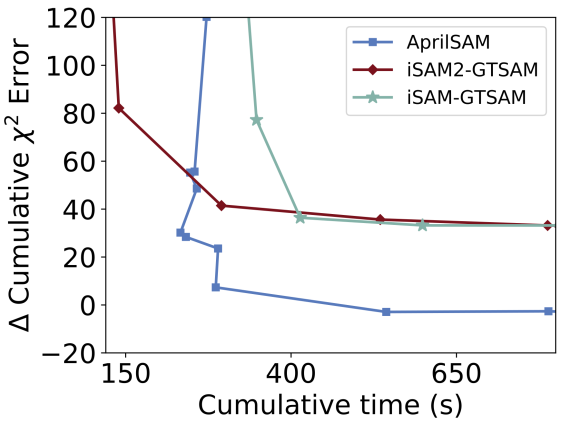
(a)



(b)

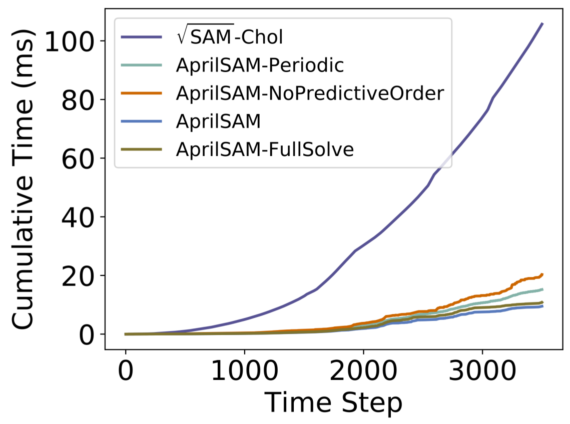


(c)

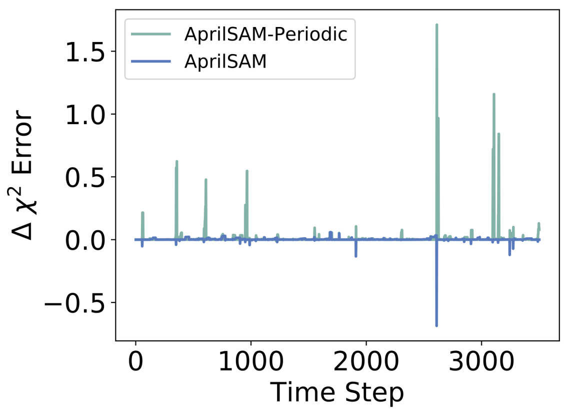


(d)

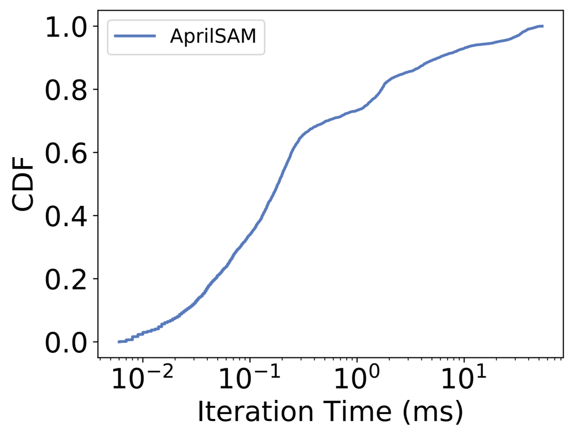
Figure 3.8: Evaluation on W10000 dataset



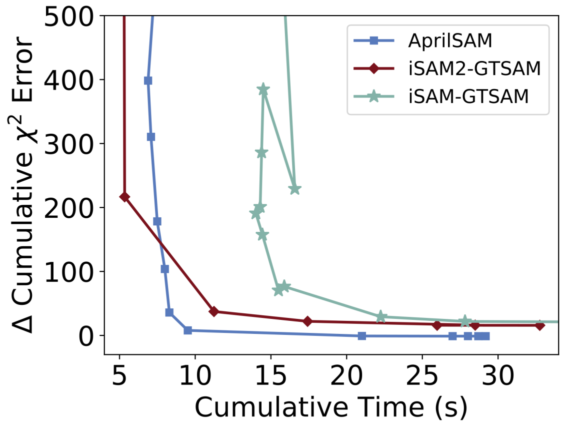
(a)



(b)



(c)



(d)

Figure 3.9: Evaluation on M3500 dataset

Figs. 3.8a and 3.9a show the cumulative running time of the tested algorithms. AprilSAM runs faster than AprilSAM-NoPredictiveOrder, which supports our claim that the min-heap based variable ordering algorithm saves running time.

Figs. 3.8b and 3.9b show the χ^2 error of the algorithms relative to $\sqrt{\text{SAM}}$-Chol, which performs a batch update at each step. Recall the distinction between AprilSAM-Periodic and AprilSAM: AprilSAM uses the criteria listed in Alg. 3.1 to decide between incremental and batch updates, whereas AprilSAM-Periodic performs batch updates periodically. We configured AprilSAM-Periodic to perform a batch update every 100 steps for the W10000 dataset and every 50 steps for the M3500 dataset. AprilSAM has a faster running time and higher accuracy than AprilSAM-Periodic, suggesting that these decision criteria are effective.

Recall that the baseline against which we measure relative accuracy is $\sqrt{\text{SAM}}$, which performs batch updates at each step. The negative error in Figs. 3.8b and 3.9b is a result of

AprilSAM detecting large state changes after an incremental update and then performing a batch update immediately. This situation is similar to performing a batch update twice when calculating χ^2 error based on state, though the linearization point is only changed once right before the batch update.

Figs. 3.8c and 3.9c show the cumulative density function of the distribution of iteration running times. On the W10000 dataset, AprilSAM runs in less than 10 ms on 70 percent of its iterations and less than 100ms on 93 percent. On the M3500 dataset, AprilSAM achieves these same metrics in 93 percent and 100 percent of iterations, respectively.

The incremental algorithms can trade off running time for accuracy, a property which we evaluate in Figs. 3.8d and 3.9d. In the case of iSAM2-GTSAM, we exchange running time for accuracy by varying the value of the parameter β, which determines how many graph nodes are re-linearized [43]. We use $\beta = \{1, 0.9, \ldots, 10^{-1}, 10^{-2} \ldots, 10^{-6}\}$. Though AprilSAM has more free parameters than iSAM2-GTSAM, we only vary the parameter δ_T (see Alg. 3.1), setting its value equal to iSAM2-GTSAM's β. When we examine the cumulative running time and relative cumulative χ^2 square error in Figs. 3.8d and 3.9d, we observe that beyond a certain point, AprilSAM always maintains a lower error given the same amount of running time. In case of iSAM-GTSAM, we use batch update interval as $\{100, 90, \cdots, 10, 5, 1\}$. AprilSAM is better than iSAM-GTSAM in terms of both running time and error. We do not show all points of iSAM-GTSAM and iSAM2-GTSAM because their errors are too large to fit on the plot. In contrast, with the same parameters, AprilSAM's errors are relatively small. This is because its decision to use batch or incremental updates depends not just on state changes, but on the tracked difference in running time between both types of updates.

For small enough parameter settings (which lead to higher running times), both iSAM2-GTSAM and AprilSAM converge to particular values of χ^2 error. AprilSAM significantly outperforms iSAM2-GTSAM in terms of accuracy at this convergence point.

3.6 Summary

In this chapter, we introduce an incremental SLAM algorithm, AprilSAM, that uses a min-heap based variable reordering algorithm coupled with fast incremental Cholesky factorization. This algorithm drives down system error while maintaining real-time performance. We evaluate AprilSAM on real-world data, comparing it with the current state-of-the-art algorithm, iSAM2. We have shown that AprilSAM runs faster and achieves better absolute error than other algorithms. Thus AprilSAM expands the operational envelope in terms of *scale*, increasing the *availability* of position estimates to a robot.

CHAPTER 4

FLAG: Feature-based Localization between Air and Ground

4.1 Introduction

Global localization (i.e. localization in a globally consistent coordinate frame) is critical for planning the paths of autonomous mobile robots, coordinating the actions of multiple agents or creating coherent user interfaces for operators. GPS could supply this positioning information, but GPS does not work well indoors. A robot could also localize using a prior map built with its own sensors, but this requires a previous visit to the site, which limits the *availability* of those methods only to familiar environments. Such a requirement may be unacceptable in many applications (e.g. search-and-rescue) in which a robot must operate in an environment it is visiting for the first time. How can a robot perform global localization in a new indoor environment?

One solution is to leverage existing global map resources from outside of the robot system. For example, a robot can match visual features in its camera view to aerial or satellite imagery to localize globally in an outdoor environment [45, 101]. Satellite imagery is not available for indoor environments, but we can use an analogous resource: floor plans.

In this chapter, we propose and demonstrate FLAG (**G**lobal **L**ocalization in a **F**loor **P**lan), which allows a ground robot to perform global positioning by recognizing landmarks in the floor plan map. In particular, we use a Lidar sensor on the robot to detect large vertical features (e.g. intersections of walls) that are also easy to detect as corners in the floor plans.

FLAG can serve as a replacement for GPS in indoor environments, providing global position updates based on a floor plan map. In between such global fixes, the robot relies on local odometry measurements. In our evaluated system, this odometry is based on wheel encoder and inertial sensors; tracking algorithms such as visual odometry or Lidar odometry could also be applied to improve localization between global corrections. Fig.4.3 further illustrates the role of FLAG in a robot's localization system.

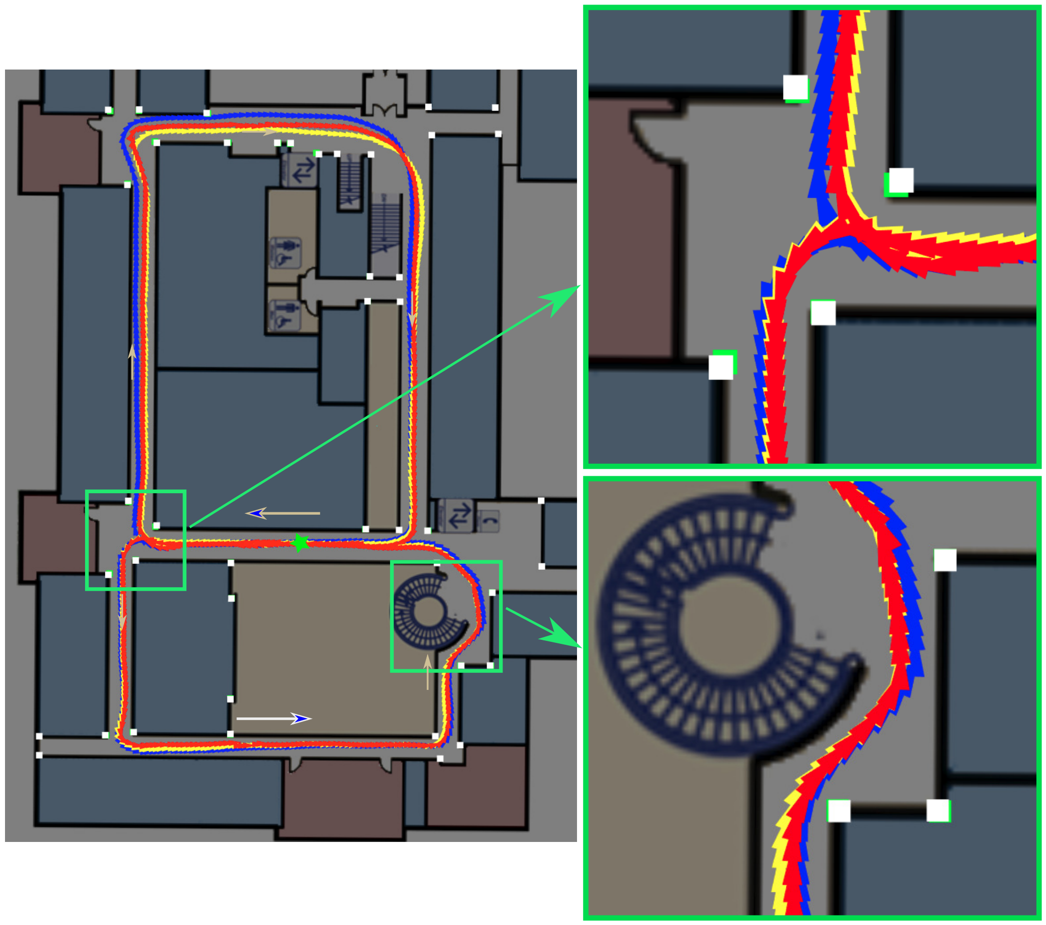


Figure 4.1: Comparison of dead-reckoning (blue), laser scan match-based localization [28] (yellow), and our FLAG method (red) over a 180 m looped path in an indoor environment. Unlike the scan matching method, FLAG uses only a floor plan with labeled corners (white squares) as a prior map, then matches vertical edge features (e.g. wall intersections) observed from the robot's point-of-view to those landmarks. FLAG runs online for reasonable indoor speeds: here, the robot moved at about 2 m/s.

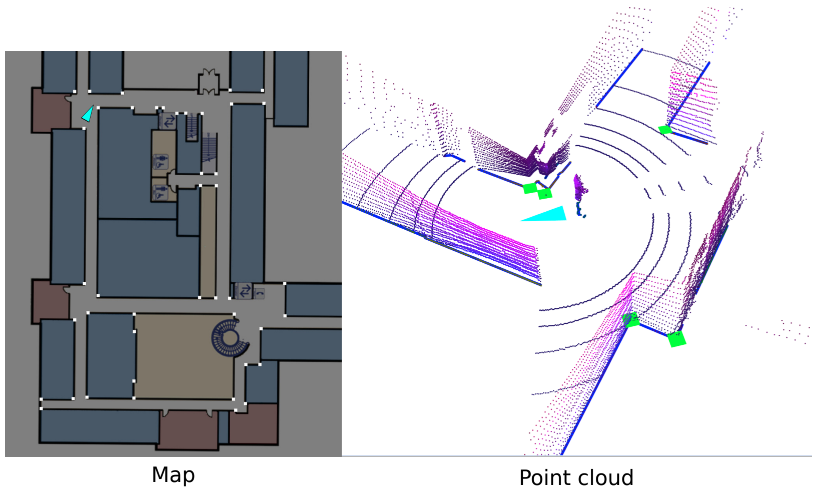


Figure 4.2: FLAG performs global localization in a previously unvisited environment. It uses a floor plan (*left*) as a prior map with labeled corners (white squares) as landmarks. The robot extracts vertical features from a 3D point cloud (*right*) and matches those features (green squares) to the landmarks in the prior map. It adds the robot's discrete poses and the landmark positions to a factor graph to be optimized.

The primary challenge in our approach is data association. The features are descriptorless and the robot's only prior knowledge is the floor plan. The environment may be dynamic and obstacles may obscure the robot's view of landmarks. Furthermore, the locations of landmarks in the floor plan may be imprecise; this might happen, for example, when a low-resolution map image is scaled. The contributions of this chapter target these challenges. These contributions are:

- A system that provides global localization in a floor plan map for a ground robot equipped with a Lidar sensor,
- A factor graph-based localization approach that increases accuracy by modeling uncertainty in the location of labeled landmarks,
- A data association method that increases robustness by using the pairwise measurement consistency checks and max-mixtures error model, and
- Evaluations on real-world and synthetic data demonstrating robust data association and accuracy comparable to laser scanmatching-based localization with a Lidar prior map.

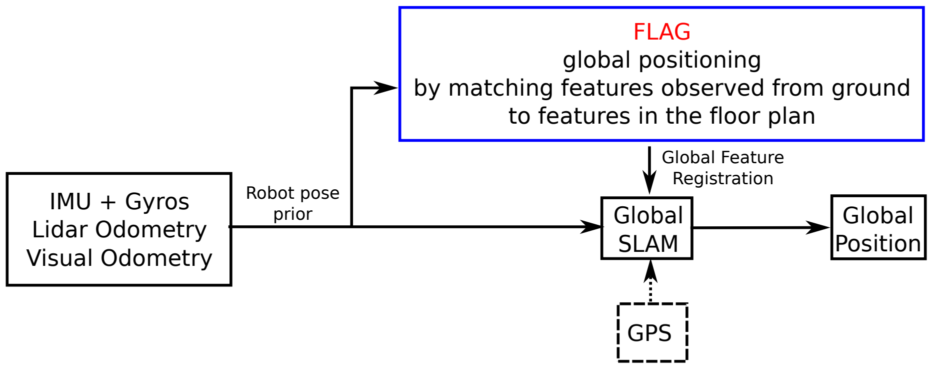


Figure 4.3: Robot localization system architecture showing the role of FLAG as a replacement for GPS.

4.2 Related Work

Our proposed method, FLAG, performs global localization by matching Lidar data to landmarks in a floor plan map. We split our discussion of related work on global localization into two parts, first focusing on Lidar-based approaches and then examining methods that use orthographic imagery.

4.2.1 Global Localization with Lidar-based Sensing

Mobile robots typically produce detailed prior maps offline and then perform global localization by matching observed features with landmarks in the map [12, 60]. For example, Hentschel et al. [36] applied Monte Carlo Localization to a line feature-based reference map. Levinson and Thrun [54], Levinson et al. [55] localize an autonomous vehicle in a pre-built, high-quality road surface map based on the reflectivity of Lidar. Goeddel et al. [28] construct a globally consistent, pose graph-based prior map of the environment and then match 2D laser scans to known locations in the prior map to yield factor potentials in a localization pose graph.

These approaches rely on a prior map generated with the same sensors the robot uses for localization. Typically, this requires the robot to visit and map the environment before operating in it. In contrast, FLAG performs global localization in a previously unvisited environment, using a floor plan image as a prior map and sensing with Lidar.

A few existing methods (e.g. [9, 49, 79, 80]) use aerial images as prior information in graph-based SLAM. These methods seek to use the prior information to make the generated map more globally consistent. In contrast, our paper uses the prior information (i.e. the floor plan map) to perform global localization in real time.

The closest of these methods to our own algorithm is that of Kümmerle et al. [49]. Their method inserts correspondences found between three-dimensional range data and edges detected in aerial images as constraints into a graph-based formulation of the SLAM problem. The matching algorithms they use do not work well in corridors or other less structured environments because the edge features are aliased from the robot's local perspective. In contrast, our method extracts corner features from the overhead image and then uses several techniques to handle data association challenges.

4.2.2 Global Localization in Orthographic Imagery

Other papers (e.g. [7, 45, 56, 98]) have also considered the problem of localizing without a detailed prior map. Instead, they use an existing map resource such as imagery collected by satellite or UAV. Typically, these methods involve comparing ground-level images to a database of georeferenced overhead images.

Viswanathan et al. [98] proposed an algorithm that warps images from a ground robot's omnidirectional camera to project them into a top-down view. These projected images are then matched to a grid of satellite locations using hand-crafted interest point-based features. Kim and Walter [45] used two Convolutional Neural Networks (CNN) to transform ground-level and aerial images into a common feature space. To localize a query ground image, they find the closest georeferenced aerial image in that feature space.

These methods based on visual feature matching tend to fail when faced with significant viewpoint or appearance variations (e.g. matching a query image taken at night to a database image taken during the day) and seasonal changes (e.g. matching a query image with snow to one taken during summer). In addition, visual feature matching will not work in a floor plan map as interest point features (e.g. SURF [8], ORB [83], or those learned by a CNN [45]) tend to be the same for different corners because the pixel values around a corner in a floor plan map are artificial.

Recognizing these drawbacks of visual feature matching, we previously proposed FLAG, which directly matches 3D features (edges) in a stereo view to 2D landmarks (corners) in a satellite image [101]. As most environments have at least some 3D features which are stable over time, this approach is robust to visual appearance changes. However, data association without visual feature signatures is still a challenge, particularly in the presence of dynamic or static obstacles near a landmark. FLAG [101] uses a particle filter to explicitly track multiple hypotheses. In this chapter, we present a new system that can achieve more robust data association by applying two data association techniques: the pairwise consistency check and the max-mixtures error model. The max-mixtures model can directly capture multiple hypotheses that arise from uncertain data associations in the graph-based localization formulation. In addition, using graph-based formulation allows us to model the uncertainty in the location of labeled landmarks. In contrast to pure landmark-based localization, our method refines the positions of landmarks as the robot navigates. In the evaluation, we show that modeling the uncertainty in labeled landmark locations improves global localization accuracy.

4.3 Approach

A key insight with FLAG is to identify features which are co-observable from a low-quality overhead map and from a robot's sensor view. Large vertical edges emerged as a highly-salient, easily-extractable feature that occurs in most indoor environments with enough frequency to be useful but not so much as to confound data association. FLAG extracts these edges from a 3D point cloud and matches them to the known corner features in the floor plan map. It then uses a factor graph-based localization algorithm to solve for the positions of a robot.

4.3.1 Vertical Edges as Global Map Features

A floor plan is an illustrated, overhead view of the environment. Because it is artificial, we cannot extract interest point-based features (e.g. SIFT or SURF) from it that are also visible from the robot's point-of-view. Instead, we use long vertical edges, which occur regularly indoors at wall intersections or other similar permanent structures, as global features. These features have a characteristic appearance from overhead (i.e. corners) and from the ground (i.e. tall vertical lines). By using these features, a robot can localize in a scene it has not previously observed.

In this chapter, we manually label feature locations in the floor plan, though automated extraction would also be feasible. The labeling need not be highly precise: our approach refines the position estimates of the landmarks as the robot navigates.

4.3.2 Detecting Features from the Ground

From the ground, our features appear as large vertical lines. Though the same features may be detectable as corners in a 2D point cloud, such an approach would fail to account for the height of the features. An indoor environment may contain many objects (e.g. furniture) that would appear as corners in planar sensor data but are not part of the permanent structure of the environment. Such transient features do not appear in a floor plan and therefore could not be used by FLAG for localization in a previously unvisited environment.

In this chapter, we detect vertical edge features using a 3D Lidar, though one might also use, for example, a stereo vision sensor [101]. We first project the 3D point cloud into 2.5D (i.e. 2D with height information), then detect corners in that data [78]. We discard any corners less than 1 meter in height as these likely do not correspond to permanent structure visible in the floor plan.

4.3.3 Factor Graph-Based Localization

The positions of the landmarks labeled in the floor plan may be imprecise, whether because of sloppy annotation or lack of resolution in the floor plan image. We therefore treat both the landmark positions and discrete robot poses as variables to be optimized in a factor graph (see Fig.4.4). In this way, the initial landmark position estimates come from the map annotation, and subsequent measurements refine these estimates.

Fig.4.4 illustrates the resulting factor graph representation. We treat the label as an observation of the landmark's location and create a factor potential to represent this measurement. We add factor potentials between robot poses based on odometry measurements. In our evaluation, these come from wheel encoders and inertial sensors, though one could also use tracking-based methods such as visual odometry. Finally, the robot measures ranges to landmarks, which are incorporated as factor potentials between landmarks and robot poses. The factor graph is optimized using AprilSAM [102].

Because the features we use are descriptorless, data association is a primary challenge, especially given that indoor environments may be cluttered with objects that trigger false positive detections. For example, consider the case of a trash can placed near a hallway corner in an office building. The object would not appear in the floor plan, but it may obscure a nearby feature or be mistaken for it. In addition to these types of static obstacles, our system must handle dynamic obstacles, such as a person walking. Filtering out small vertical edges helps with these problems, but we still need a way to improve the robustness of data association.

In contrast to methods which make data association an explicit process prior to inference (e.g. JCBB [68] or IPJC [75]), we move the bulk of data association into the inference process itself using the max-mixtures method [73]. These mixtures are bootstrapped with nearest-neighbor associations.

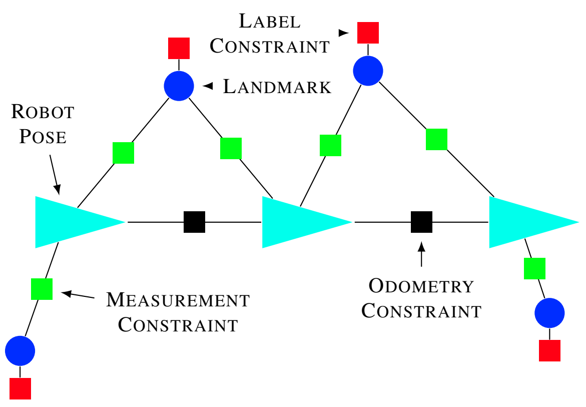


Figure 4.4: Illustration of FLAG factor graph. We discretize the robot's trajectory into a series of poses (cyan triangles) that we add to the graph as nodes. We also add the positions of landmarks (blue circles) as graph nodes. Factors in the graph include odometry constraints (black squares), labeled position constraints of landmarks (red squares), and measurement constraints between pose nodes and landmark nodes (green squares). With this formulation, both the robot poses and landmark positions are updated when the graph is optimized.

4.3.3.1 Pairwise Consistency Check

To reject erroneous data associations caused by dynamic obstacles, we apply a pairwise consistency check (see Fig.4.5). The robot takes multiple measurements of a landmark over the time interval during which the landmark is within view. When combined with the estimate of the robot's pose when the measurement was taken, each observation gives an estimate of the projected position of the landmark. For every pair of these measurements, we compute the distance between the corresponding projected positions of the landmark. If those positions are within a threshold distance, we consider the pair of measurements to be consistent.

We then form an undirected graph in which the measurements are nodes, and we add edges between pairs of nodes corresponding to consistent measurements. Mangelson et al. [61] showed that finding the largest pairwise-consistent set is equivalent to finding the maximum clique in the graph, which is well-studied in graph theory. Olson et al. [76] presented an approximate solution to find a single cluster of well-connected nodes in a graph. In this chapter, we apply a simple variation on this idea. Namely, if there exists a measurement that has enough consistent links to other measurements, we create a factor potential for the measurement and add it into the factor graph. In this way, a moving obstacle will not yield enough consistent measurements to be added into the graph.

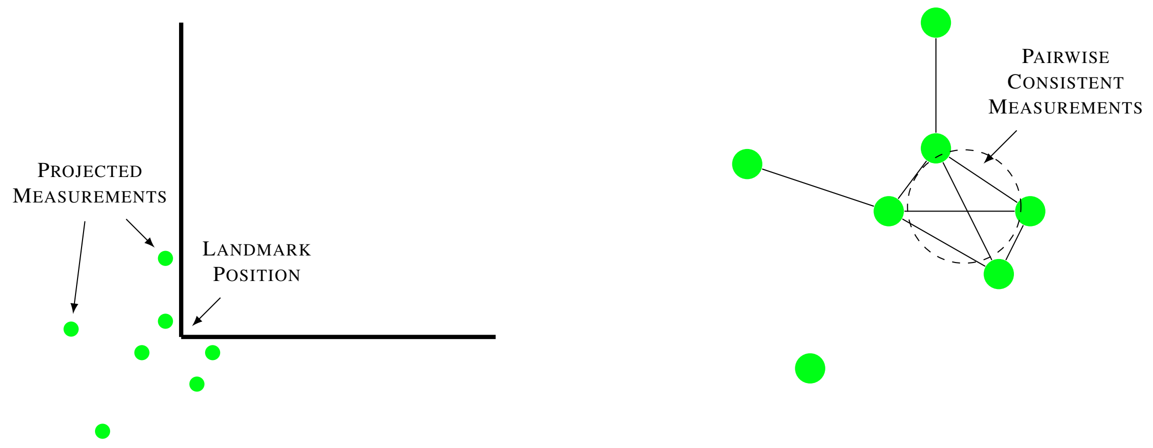


Figure 4.5: Illustration of the pairwise consistency check. The green circles (*left*) are the projected positions of landmarks based on distance and bearing measurements. These measurements form an undirected graph (*right*). Edges exist between nodes if the projected landmark positions from the two measurements are within a threshold distance. A clique in the graph represents a set of internally consistent measurements. We reject measurements that are not part of a sufficiently large clique.

4.3.3.2 Max-Mixture Error Model

To reject erroneous data associations caused by static objects near landmarks, we apply a max-mixtures error model [73]. We used two-component mixtures with one being the nominal data association and the second being a ”null” hypothesis. With the max-mixtures, the localization system can visit all data association decisions to reject outliers as the robot makes new observations.

Consider again the case of a trash can placed near a hallway corner. As the robot approaches the trash can, associating the object to the nearby feature might reduce error compared to an odometry-only solution. However, as the robot continues to observe the object, the repeated low-likelihood measurements resulting from the association cause error to increase. A max-mixtures error model can revisit those previous data associations and apply a null hypothesis to them.

4.4 Experimental Evaluation

We evaluate FLAG in two parts: (1) we test the entire system's performance using real-world data, and (2) we analyze the effect of FLAG's robust data association and factor graph formulation using simulation.

4.4.1 Robot Platform

We performed all experiments with the MAGIC2 robot platform developed by the APRIL Robotics Lab. Relevant sensors onboard the robot include wheel encoders and a fiber optic gyroscope (KVH DSP-1715) for robot odometry and a 3D Lidar (Velodyne VLP-16).

4.4.2 Global Localization

We evaluated our method on real-world indoor datasets to demonstrate its overall performance and to compare it to a state-of-the-art laser scan match-based localization method that uses a Lidar prior map [28, 72].

Our floor plan had a resolution of 0.1 m per pixel. We hand-labeled the initial positions of landmarks at every corner corresponding to an intersection of walls (see white squares in Fig.4.1). We assigned an uncertainty of 10 cm to the factor potentials corresponding to these labeled positions. The odometry measurements came from the fusion of wheel encoders and a fiber optic gyroscope; we assigned x, y, θ uncertainty of 10 cm, 5 cm, and 0.1 degrees per meter traveled. We assigned an uncertainty of 5 cm to the ranging measurements between landmarks and robot poses.

We manually drove the robot at a speed of about 2 m/s in a 180 m loop through the halls of an office building (the BBB at the University of Michigan) for 4 times. We followed a figure-8 pattern; the green star in Fig.4.1 shows the starting point and the tan arrows indicate the path of travel.

Fig.4.1 shows the path of the robot resulting from odometry-based dead-reckoning (blue), a laser scan match-based localization method using a Lidar prior map [28] (yellow), and our proposed FLAG method (red). FLAG shows clear improvements over dead-reckoning, and its performance is similar to that of laser scan matching approach, which we consider an approximate ground-truth. Recall that the laser scan matching approach uses a Lidar prior map generated offline, whereas FLAG uses only a floor plan map. Fig.4.6 shows the average absolute errors of FLAG and dead-reckoning for 4 trips.

Because FLAG adds the positions of landmarks to the factor graph, it continually updates their positions as the robot travels around and makes observations. The white squares in Fig.4.1 are the initial hand-labeled initial positions, and the green squares are the final positions after the whole run.

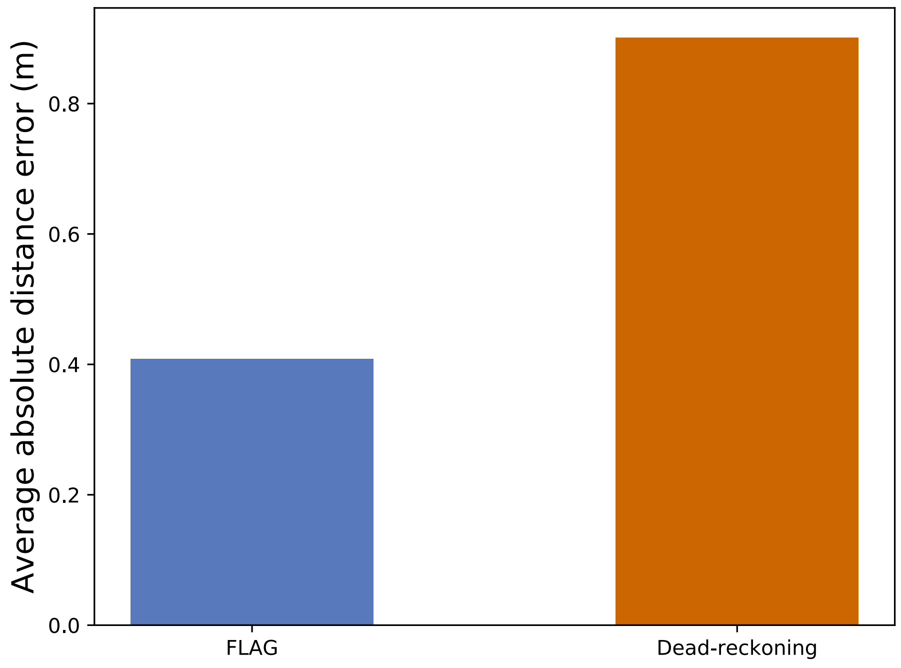


Figure 4.6: Average absolute distance errors of FLAG and dead-reckoning over a 180 m looped path in an indoor environment for 4 different runs (Fig.4.1 shows one of four runs). We use a state-of-the-art laser scan match-based method (with a Lidar prior map) as an approximate ground-truth.

4.4.3 Data Association

To evaluate the performance and reliability of our data association system, we performed an ablation study to show the effects of the pairwise consistency test and max-mixture error model, and to measure the effectiveness of refining the floor plan landmark positions.

4.4.3.1 Pairwise Consistency Check

The primary purpose of the pairwise consistency check in FLAG is to handle dynamic obstacles in the robot's environment. To test the effect of this component, we constructed a simulated scenario in which a moving object passes in a straight line by an existing landmark (see Fig.4.7). We randomly sample 10 measurements from the landmark and 10 measurements from the moving object, both with variance 10 cm. In this experiment, the distance threshold to determine whether measurements are consistent is 10 cm, and the data association threshold is set to be 50 cm. We run 5000 trials for data association with and without pairwise consistency check, incorporating all observations within 50 cm. Fig.4.8 shows the result of this test, namely that the pairwise consistency check results in lower error and lower error variance.

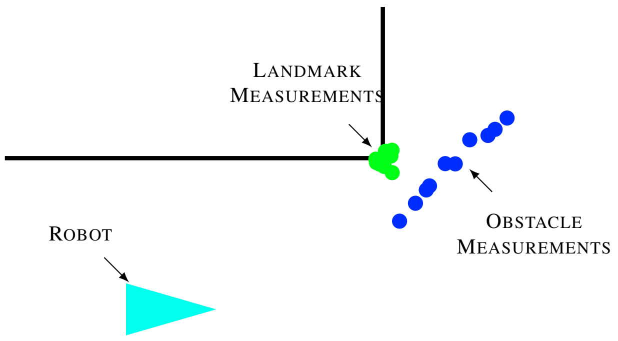


Figure 4.7: Illustration of simulated test of the pairwise consistency check algorithm. Green dots show the 10 measurements of the landmark, which are sampled from its true location with variance 10 cm. The blue dots show observations of a dynamic object moving along a straight line, from which measurements are sampled with variance 1 cm. The results are shown in Fig.4.8.

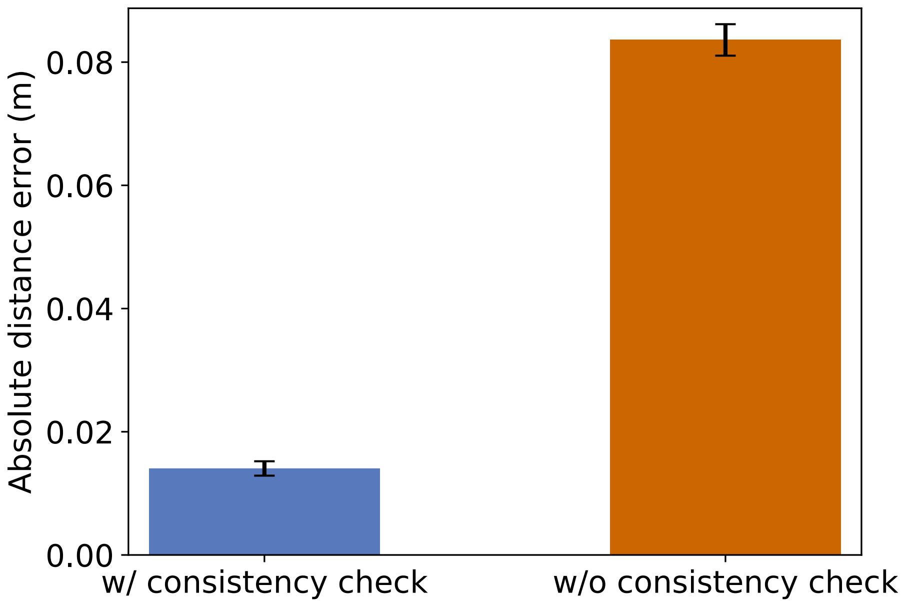


Figure 4.8: Localization error in the presence of dynamic obstacles (see Fig.4.7). Error bars represent three standard error of the mean across the 5000 trials. Using the pairwise consistency check resulted in lower error.

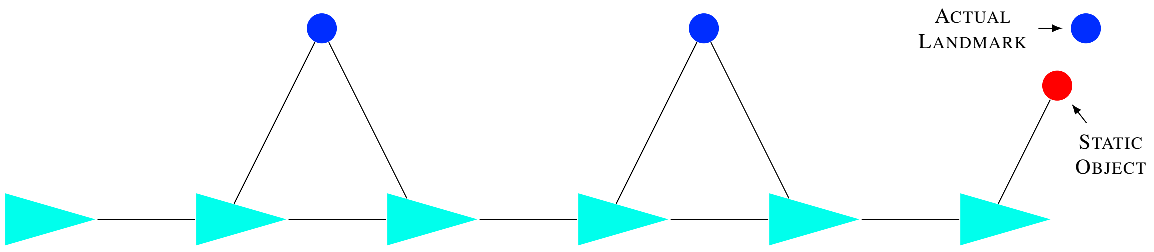


Figure 4.9: Illustration of a simulated scenario testing the max-mixture error model. The rightmost landmark is obstructed by a static object, which the robot observes instead of the landmark. For each trial, we randomly sample the location of the static obstacle within a particular radius of the landmark. We plot the localization error of the last pose in Fig.4.10.

4.4.3.2 Max-mixture Error Model

The primary purpose of FLAG's max-mixture error model is to avoid associating static obstacles with landmarks from the floor plan map. Analogous to the simulation of the previous section, we built a simulated test case in which a robot passes by several landmarks, the last of which is blocked by a static obstacle (see Fig.4.9). We randomly sample the location of this static obstacle within a specified radius of the landmark. As in the previous section, we randomly sample measurements with standard deviation 10 cm.

The robot does not observe the final landmark but rather receives measurements from the nearby static obstacle. We vary the maximum distance of the object from the landmark in the range 10-50 cm and conduct 5000 trials at each radius. We measured the localization of the last node in each trial with and without the max-mixture error model. Fig.4.10 shows the result of this experiment.

Without the max-mixture error model, error increases proportionally to the distance between the obstacle and the landmark. With the max-mixture model, error increases until the distance reaches approximately 20 cm, after which it levels off. Once measurements from the mis-associated obstacle cause error to rise to a certain point, the max-mixture model "turns off" measurements of that landmark by assigning a highly uncertain noise model to them. This mitigates the effect of the erroneous data association.

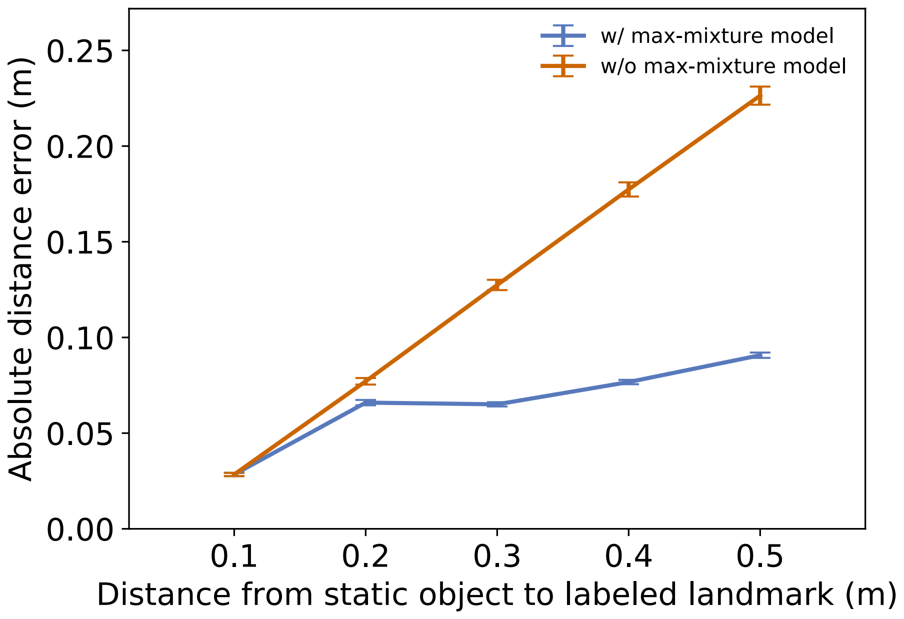


Figure 4.10: Localization error in the presence of a static obstacle. Error bars represent three standard error of the mean across the 5000 trials. As the static object gets further from the true landmark localization, the effect of using the max-mixture model becomes more pronounced.

4.4.3.3 Incorporating Landmark Uncertainties

One source of localization error in our previous approach [101] is not incorporating uncertainties of labeled landmark positions. In this chapter, we model those uncertainties by formulating the localization problem as a factor graph-based SLAM problem that optimizes both robot positions and landmark positions. Fig.4.11 shows the effect of this optimization on the average localization error measured on the real-world dataset of Sec. 4.4.2. We compare the performance of FLAG to that of an otherwise identical method that considers the locations of the landmarks to be static. Incorporating the uncertainty in landmark positions leads to smaller longitudinal, lateral, and heading errors.

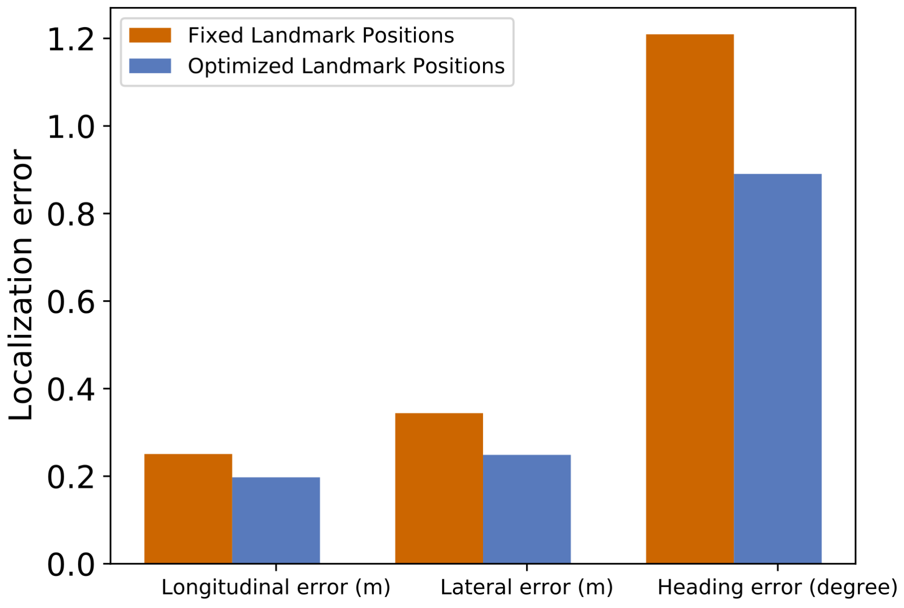


Figure 4.11: Effect of modeling uncertainty in labeled landmark positions on localization performance. Including the landmark positions reduces the longitudinal, lateral, and heading errors.

4.5 Summary

In this chapter, we presented FLAG, a method that enables a ground robot to localize itself globally by identifying landmarks visible in a floor plan map. FLAG is a factor graph-based approach that uses a pairwise measurement consistency check and max-mixtures error model. This results in more robust and accurate localization than the our previous work [101]. We demonstrated that our system has comparable performance to laser scan matching-based localization with a Lidar prior map, showing FLAG expands the operational envelope in terms of *the amount of prior data required*, increasing the *availability* of position estimates to a robot.

CHAPTER 5

MOSS: Map Optimization for Size and Saliency

5.1 Introduction

Pose estimation in a prior map is a common online localization method for robot navigation. A high quality map can improve the robustness of localization in a robot system, which improves the *availability* of algorithms in complex environments. Whereas existing mapping approaches focus primarily on improving the quality of the map in terms of consistency, we seek to optimize a map for subsequent localization performance. To this end, we present a machine learning-based map optimization algorithm to generate compact maps that support robust localization.

For range-based sensors (e.g. LIDAR), a popular map representation is the occupancy grid map, which discretizes the mapped area into cells of equal size [67]. A high-resolution grid map provides a detailed representation of the world but is susceptible to feature aliasing since a sensor's view is noisy and often ambiguous. Such aliasing leads to poor localization performance. Furthermore, the memory consumption of the grid map representation grows quadratically with the range of the sensor. A coarser grid size reduces the memory burden of the grid map but may also harm localization performance. Multi-resolution maps can help with these issues, but it remains a challenge to decompose grid maps with suitable resolutions for each map region.

In this chapter, we use a factor graph-based map representation in which each node stores sparse scans assembled in the sensor's local view, a representation similar to the *keyframe* used in vision-based SLAM [90]. To localize online, we match live scans to the sparse scans in the map. A key advantage of this map representation is that it allows us to select which nodes to include in the map to improve localization performance and reduce memory consumption [86].

Existing methods in factor-graph SLAM (e.g. [14, 39, 41, 47]) have considered node removal as a way to slow the growth rate of the graph. When removing a node from the

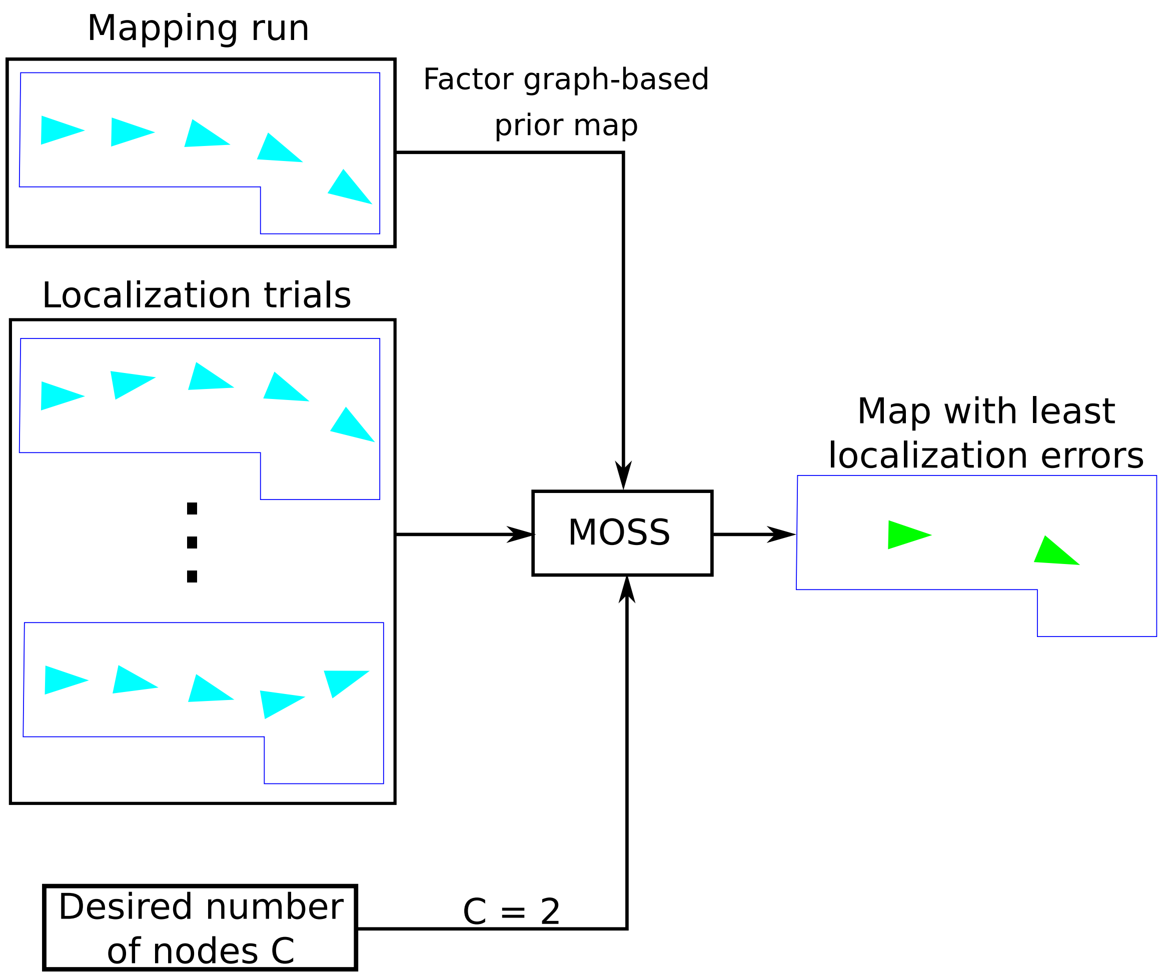

Mapping run
Factor graph-based prior map
Localization trials
Map with least localization errors
MOSS
Desired number of nodes C
C = 2

Figure 5.1: Illustration of MOSS pipeline. MOSS (Alg. 5.1) optimizes a prior map for size and saliency based on the results of localization trials, seeking to eliminate portions of the map prone to feature aliasing.

graph, these methods generate a new set of factors to preserve the measurement information from the sensors. The primary purpose of such node removal is controlling computational cost or memory consumption while building maps online. These methods also treat mapping as a standalone optimization problem, where the optimal map output is the one that best satisfies the observation constraints made during the mapping run.

In this chapter, we optimize the map offline for use in subsequent localization trials. Our algorithm, MOSS (Map Optimization for Size and Saliency), learns distinctive features from an existing prior map generated by factor-graph SLAM algorithms. MOSS eliminates nodes prone to feature aliasing while offline in order to achieve better localization online. We will show experimentally that the optimized maps produced by MOSS lead to better localization performance than the original full map.

The contributions of this chapter include the following:

- A machine learning-based map optimization algorithm producing compact maps that support robust localization, and
- Evaluations on synthetic data demonstrating that MOSS produces maps with better localization performance than either a state-of-the-art node selection method or the original full map.

5.2 Related Work

MOSS draws inspiration from many distinct lines of research in robot mapping and localization, culminating in a key idea: MOSS optimizes a prior map both for size and for future localization performance. We now review these areas of research and discuss their relation to MOSS.

Memory Consumption in Occupancy Grid Mapping

When mapping with laser ranging sensors, one of the most common map representations is the occupancy grid map [67]. However, the memory consumption of basic occupancy grid mapping grows quadratically with the range of the laser sensors. Many researchers have focused on reducing this memory consumption burden [37, 86, 104, 106]. Wurm et al. [104] propose a tree-based representation which offers a lossless compression method that improves the compactness of the representation. Schiotka et al. [86] introduced a memory-efficient map representation based on a constant set of individual scans. Their method incrementally selects scans to add to the map based on the additional information they provide relative to the scans previously selected. They view the full map as the ideal case that gives the best performance when used for localization.

In contrast to this type of approach, we aim to both reduce the size of maps and improve their localization performance. We propose a machine learning-based approach to select a set of nodes in a pose graph that offers better localization performance than the full map.

Node Selection Algorithms in Factor Graph SLAM

Node selection algorithms (e.g. [14, 41, 47]) in factor-graph SLAM are mainly used to control computational cost or memory consumption when building maps online. Johannsson et al. [41] propose an approach that continually uses new measurements to improve the map but doesn't add new nodes for already visited areas. Kretzschmar et al. [47] use expected information gain of observations to select nodes. Carlevaris-Bianco et al. [14] apply marginalization to eliminate nodes and show that they can produce a new set of linearized factors to approximate the true marginalization. When removing a node, these methods create a new set of factors to preserve all measurement information. By comparison, MOSS runs offline to select nodes to include in the map to achieve the best localization performance online. MOSS operates on maps produced via factor graph SLAM and selects a subset of the original nodes to include in the final map, which is a combinatorial optimization problem.

Schaff et al. [85] propose a deep learning-based method to solve a similar combinatorial optimization problem of placing beacons for localization. They formulate the beacon placement as a differentiable neural layer and estimate the current stationary position based on range readings from the beacons.

In this chapter, we address a related but more challenging sequential scenario in which localization performance at previous time steps affects the current localization result. Deep neural network-based methods [44, 100] show success in camera re-localization and visual odometry but not in real-time localization. We therefore use graph optimization [102] to estimate current and historical poses (see Sec. 5.3). We apply machine learning-based approach to select nodes to include in the final map representation.

Learning from Past Experience in Localization

MOSS can be considered a way of learning from past experience to improve localization performance. In the same spirit, Churchill and Newman [16] propose a way to achieve long-term navigation in changing environments by recalling previous experiences in those environments. Maddern et al. [59] proposes a similar approach for laser ranging sensors.

In contrast to MOSS, these methods only *add* information to the map, which increases computational cost and memory consumption for online localization. Additionally, they do not address perceptual aliasing. MOSS seeks to remove aliasing features in the map to improve localization performance.

Back-end Approaches to Feature Aliasing

Max-mixture models [73], sum-mixture models [52] or dynamic covariance scaling [3] can be used in optimization back-ends to deal with outlying observations caused by perceptual aliasing. However, it is still difficult to estimate correct measurement error models in practice. MOSS provides a way to directly remove features that may cause perceptual aliasing. In so doing, it moves outlier rejection to the front-end of online localization.

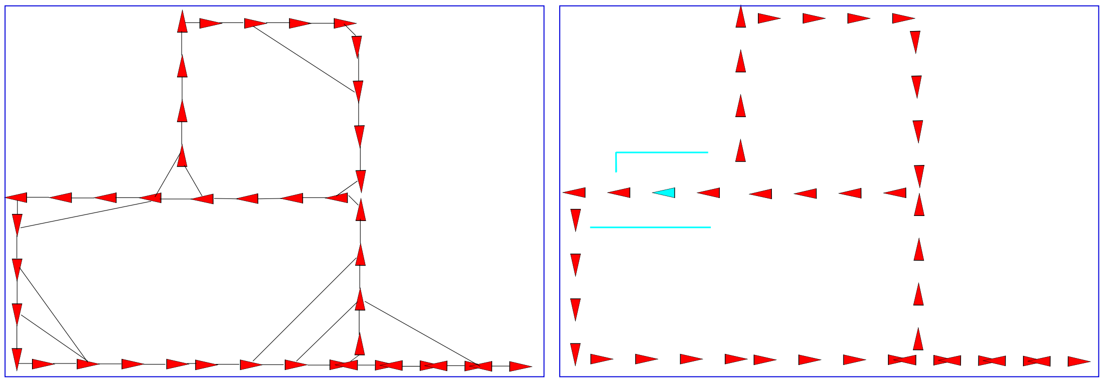


Figure 5.2: *Left*: The pose-graph model. The robot trajectory is discretized into a series of poses (red triangles), which correspond to nodes in the graph. Relative position and orientation constraints between nodes are encoded as graph edges. *Right*: Map product of a pose graph. Once the graph has been optimized, the map is stored as a series of known poses, each with an associated laser scan (cyan lines around cyan triangle).

5.3 Problem Statement

5.3.1 Pose-Graph Map Representation

Fig.5.2 illustrates the pose-graph model [33] we use during mapping. Robot poses correspond to nodes in the graph, and rigid transformation constraints between poses correspond to edges. Inference of the joint probability distribution over the nodes in the graph is equivalent to recovering the maximum a posteriori (MAP) estimate for the robot states X given rigid transformation observations Z. According to Bayes law, the maximum likelihood solution is:

$$\begin{aligned} X_{\mathrm{ML}} &= \arg\max_x P(X = x | Z = z) \\ &= \arg\max_x P(Z = z | X = x) \end{aligned}. \tag{5.1}$$

X_{ML} contains the maximum likelihood solution for each pose in the graph. In storing the map for later use, we include each of these pose estimates along with their associated laser scan measurements. We discard the constraints between poses since the graph will not be optimized further.

5.3.2 Localization

The pose graph map defines a global coordinate frame. The robot maintains an open-loop estimate of its current pose in its own local coordinate frame. The goal of global

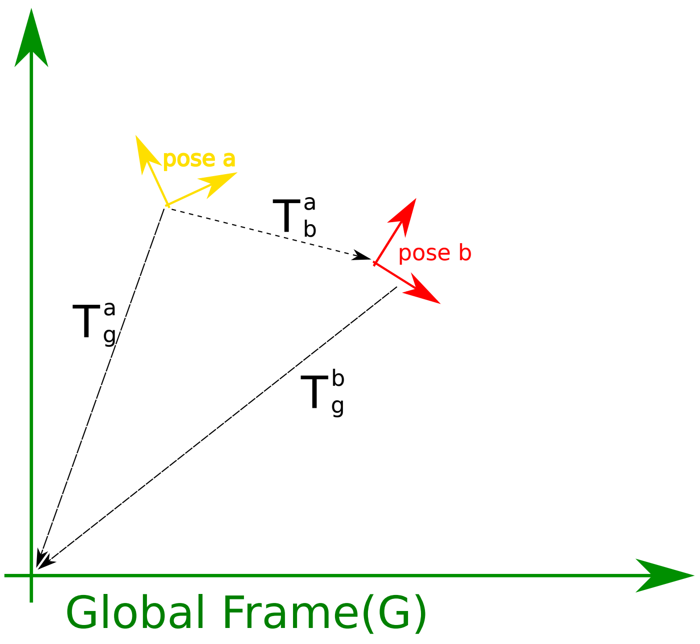


Figure 5.3: Coordinate transformations. Each robot pose represents a local coordinate system that can be related to the global coordinate by a rigid-body transform. Two local coordinate systems can also be related, e.g., $T^a_g = T^a_b T^b_g$.

localization is to compute a transformation from the robot's local coordinate frame to the map's global coordinate frame. We refer to this local-to-global transformation as *L2G*.

We illustrate this coordinate transformation in Fig.5.3. Pose b is a node in the prior map with known *L2G* T^b_g. Pose a is a localization node generate online during the robot's mission. We can obtain the relative transformation between a and b, T^a_b, using laser scan matching. To find the *L2G* estimate of pose a, we compose the relative pose transformation T^a_b with the global transformation T^b_g; that is, $T^a_g = T^a_b T^b_g$.

We formulate online localization as the factor-graph SLAM problem[29] illustrated in Fig.5.4. The nodes of a localization pose graph correspond to poses taken from the robot's trajectory. Odometry measurements become edges between adjacent nodes. Laser scan matching provides factor potentials between robot poses and known locations in the prior map, which are themselves considered fixed. When combined with the global transformation of these prior map locations, the scan matching results give noisy observations of *L2G*. Optimizing the graph yields a maximum likelihood estimate for each pose in the localization graph.

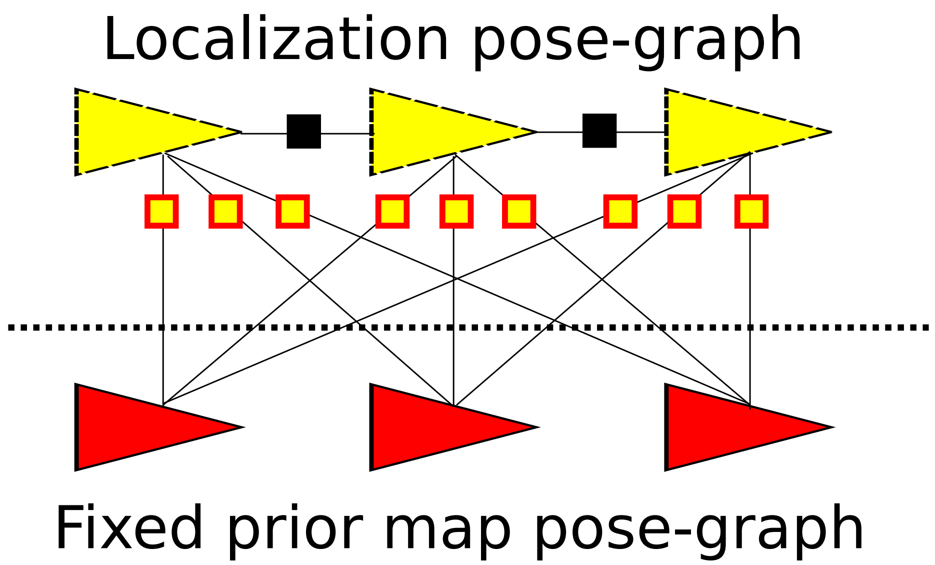


Figure 5.4: A simple example of a factor graph for localization. Yellow triangles are localization nodes and red triangles are map nodes. In addition to standard factors based on inertial measurements (black squares), scan matching factors (yellow squares) are added describing transformations back to a known map (in red). During optimization, the portion of the graph corresponding to the known map is held fixed.

A challenge in reliably localizing in this way is rejecting erroneous observations caused by bad scan matches. Such observations often occur in regions of the map prone to feature aliasing and can lead to catastrophic shifts in the *L2G* estimate. To deal with this issue, we propose to learn a binary function f that selects *L2G* observations to accept or reject. Given such a function, the maximum likelihood solution to (5.1) becomes

$$\begin{aligned} X_{\mathrm{ML}} = \arg\min_{x} \sum_{k} f\left(\boldsymbol{x}_{i_k}, \boldsymbol{y}_{j_k}\right) \left\| \boldsymbol{z}_k^{\mathrm{L2G}} - g\left(\boldsymbol{x}_{i_k}, \boldsymbol{y}_{j_k}\right) \right\|^2_{\Omega_k^{\mathrm{L2G}}} + \\ \sum_{l} \left\| \boldsymbol{z}_l^{\mathrm{ODO}} - h\left(\boldsymbol{x}_{i_l}, \boldsymbol{x}_{j_l}\right) \right\|^2_{\Omega_l^{\mathrm{ODO}}} \end{aligned} . \tag{5.2}$$

$\boldsymbol{z}_k^{\mathrm{L2G}}$ is an *L2G* observation between pose $\boldsymbol{x}_{i_k}$ in the localization pose graph and pose $\boldsymbol{y}_{j_k}$ in the prior map. Ω_k^{L2G} is the information matrix corresponding to observation $\boldsymbol{z}_k^{\mathrm{L2G}}$ and $g\left(\boldsymbol{x}_{i_k}, \boldsymbol{y}_{j_k}\right)$ is the expected *L2G* observation based on the scan matching measurement model. $\boldsymbol{z}_l^{\mathrm{ODO}}$ is a rigid transformation observation between poses $\boldsymbol{x}_{i_l}$ and $\boldsymbol{x}_{j_l}$ of the localization graph based on odometry measurements. Ω_l^{ODO} is the information matrix corresponding to observation $\boldsymbol{z}_l^{\mathrm{ODO}}$, and $g\left(\boldsymbol{x}_{i_l}, \boldsymbol{x}_{j_l}\right)$ is the expected rigid transformation between the poses based on the odometry measurement model.

Our goal is to learn a function f^* that minimizes the error of the solution X_{ML} relative to the ground truth solution X_{GT} over a number of training trials. We define this error as the cumulative Euclidean distance between the poses' translational components. f^* is then given by

$$
\begin{aligned}
f^* &= \arg\min_f \sum_t E(X_{\text{ML}}^{(s)}, X_{\text{GT}}^{(s)}) \\
&= \arg\min_f \sum_s \sum_i \left\| \boldsymbol{x}_i^{(s)}{}_{\text{ML}} - \boldsymbol{x}_i^{(s)}{}_{\text{GT}} \right\|_2 ,
\end{aligned}
\tag{5.3}
$$

where $\boldsymbol{x}_i^{(s)}{}_{\text{ML}}$ and $\boldsymbol{x}_i^{(s)}{}_{\text{GT}}$ are the maximum likelihood solution and ground truth for pose i of training trial s, respectively.

5.4 Approach

Algorithm 5.1 Map Optimization for Size and Saliency

```
1: for $t = 1, \dots, T$ hill climbs do        ▷ Run hill climbs from multiple starting locations
2:     $\boldsymbol{\alpha}_t \sim \mathcal{U}\left(\left\{\boldsymbol{\alpha} \in \mathbb{F}_2^M : w(\boldsymbol{\alpha}) = C\right\}\right)$   ▷ Randomly select $C$ nodes to include in map
3:     for $k = 1, \dots, K$ hill climb steps do
4:         $\boldsymbol{\alpha}' \sim \mathcal{U}\left(\left\{\boldsymbol{\alpha} \in \mathbb{F}_2^M : w(\boldsymbol{\alpha}) = C, d(\boldsymbol{\alpha_t}, \boldsymbol{\alpha}) = 2\right\}\right)$   ▷ Randomly swap node into/out of map
5:         $e(\boldsymbol{\alpha}') = \sum_{s=1}^{S} \left\| \boldsymbol{X}_{\text{ML}}^s(\boldsymbol{\alpha}') - \boldsymbol{X}_{\text{GT}}^s \right\|_2$   ▷ Compute error for map over localization trials (see (5.2))
6:         if $e(\boldsymbol{\alpha}') < e(\boldsymbol{\alpha}_t)$ then
7:             $\boldsymbol{\alpha}_t = \boldsymbol{\alpha}'$
8: return $\arg\min_{\boldsymbol{\alpha}_t} e(\boldsymbol{\alpha}_t)$   ▷ Return best node combination found during all hill climbs
```

We now propose a method to find a function $\hat{f}^*$ that approximates f^*. Our method relies on the intuition that the quality of scan matching results varies among the nodes of the prior map. Some regions of the map have rich geometries suitable for scan matching while others are susceptible to feature aliasing. For example, long corridors in an indoor environment yield similar laser scans throughout their span, leading to frequent erroneous matches. In contrast, the geometry of hallway intersections typically leads to reliable matches. We therefore seek to learn which graph nodes to include in the prior map to support reliable localization.

Let $\boldsymbol{\alpha} \in \mathbb{F}_2^M$ be a binary vector such that $\alpha^{(j)}$ indicates whether to include node $\boldsymbol{y}_j$ in the prior map. We can then define f as

$$
f\left(\boldsymbol{x}_{i_k}, \boldsymbol{y}_{j_k}\right) := \alpha^{(j_k)}. \tag{5.4}
$$

In other words, the function f includes observations in the localization optimization as long as they involve a prior map node for which the corresponding element of $\boldsymbol{\alpha}$ is 1.

Given this definition of f, the task of finding an optimal function f^* is equivalent to the task of finding an optimal indicator vector, $\boldsymbol{\alpha}^*$, which is given by

$$\boldsymbol{\alpha}^* = \arg\min_{\boldsymbol{\alpha}} \sum_t E\left(X_{\text{ML}}^{(t)}, X_{\text{GT}}^{(t)}\right). \tag{5.5}$$

Computing an optimal solution to this combinatorial optimization problem is intractable outside of toy problems. We therefore seek to find an indicator vector $\hat{\boldsymbol{\alpha}}^*$ that approximates $\boldsymbol{\alpha}^*$.

Before presenting our method for computing $\hat{\boldsymbol{\alpha}}^*$, we will first discuss two possible alternative methods that we will use as performance baselines in our evaluation. All methods are parameterized by the desired number of nodes, C, to include in the prior map. By considering multiple values of C, they determine an overall estimate of the optimal indicator vector.

5.4.1 Equidistant Nodes

Given a prior map with M nodes, a straightforward way to reduce the map to C nodes is to space the nodes uniformly throughout the mapped trajectory of the robot. For example, to reduce the map to $C = 1$ nodes, we can include the node at the midpoint of the trajectory. Such an approach leads to the indicator vector given by

$$\alpha^{(i)} = \begin{cases} 1 & i \bmod \frac{M}{C+1} = 0 \\ 0 & \text{o.w.} \end{cases} \tag{5.6}$$

5.4.2 Sparse Scan Matching

Schiotka et al. [86] propose Sparse Scan Matching (SSM), which selects the set of map nodes that maximizes the likelihood of mapping observations given the map trajectory. This observation likelihood depends on how much overlap there is between selected scan set $S_{\boldsymbol{\alpha}}$ and all scans in mapping trajectory.

Because this is a combinatorial optimization problem, SSM does not consider all possible solutions for S_α. Rather, it builds the scan set incrementally, each time selecting the scan that adds the most informative candidate scan given the set of already selected scans. In other words, SSM iteratively selects the next scan that maximizes the observation likelihood of the mapping trajectory.

$$
\begin{aligned}
\boldsymbol{\alpha}^* &= \arg\max_{\boldsymbol{\alpha}} p(z_{1:M}|x_{1:M}, S_{\boldsymbol{\alpha}}) \\
&= \arg\max_{\boldsymbol{\alpha}} \prod_{i=1}^{M} p(z_i|x_i, S_{\boldsymbol{\alpha}}) \\
&= \arg\max_{\boldsymbol{\alpha}} \prod_{i=1}^{M} \sum_{s\in S_{\boldsymbol{\alpha}}} p(z_i|x_i, s)p(s|x_i) \\
&= \arg\max_{\boldsymbol{\alpha}} \prod_{i=1}^{M} \sum_{s\in S_{\boldsymbol{\alpha}}} \{\prod_{q\in z_i} p(q|x_i, s)p(s|x_i)\}
\end{aligned}
\tag{5.7}
$$

Here, z_i is the laser scan observation of pose x_i. $S_{\boldsymbol{\alpha}}$ contains the poses and laser scan observations selected by indicator vector $\boldsymbol{\alpha}$.

$p(s|x_i)$ is approximated with a Dirac distribution:

$$
p(s|x_i) = \begin{cases} 1 & \|x - x_s\|_2 = \min_{s'} \|x - s'\|_2 \\ 0 & \text{o.w.} \end{cases} \tag{5.8}
$$

$p(q|x_i, s)$ is calculated base on a Gaussian distribution:

$$
p(q|x_i, s) \sim \mathcal{N}(\min_{(x_s,q_s)\in s} \|x_s \oplus q_s - x \oplus q\|_2; 0, \sigma) \tag{5.9}
$$

$x \oplus q$ transforms the point p from local frame to global frame.

A greedy strategy is used to approximate (5.7). The node is selected incrementally starting with $\boldsymbol{\alpha} = \mathbf{0}$.

$$
\begin{aligned}
k &= \arg\max_{k\in[1,M],\alpha_k^t \neq 1} p(z_{1:M}|x_{1:M}, S_{\boldsymbol{\alpha}^t, \alpha_k^t := 1}) \\
\alpha_k^{t+1} &= 1
\end{aligned}
\tag{5.10}
$$

5.4.3 MOSS

In contrast to SSM, which maximizes the likelihood of observations only from the initial mapping run, we propose a machine learning-based method that minimizes localization error during offline training trials. We call this method Map Optimization for Size and Saliency (MOSS). Instead of using regression models like deep neural network [91], we choose to create factor graph-based models for the localization. The optimal parameter set $\boldsymbol{\alpha}$ (see in (5.4)) of those models can be learned from offline training trials. However, computing the optimal set with respect to the training trials is intractable. We therefore propose to approximate it using the hill-climbing optimization method shown in Alg. 5.1.

MOSS uses the results of S localization trials in this algorithm to select nodes to include in the map, seeking to solve (5.3). For each of these localization trials, we assume that we have access to a ground truth trajectory. While such a ground truth trajectory is trivially accessible in simulation, a real-world ground truth is more challenging to obtain. However, one might use more expensive sensors or artificial landmarks to generate reference trajectories. MOSS could then use those reference trajectories to optimize maps produced with less expensive sensors.

MOSS consists of T hill climbs that may be run in parallel with one another. Each hill climb begins with a combination of C nodes randomly sampled from the M available map nodes (Line 2). In other words, we randomly sample from the set of all binary vectors of length M and Hamming weight C. The variable to which this indicator vector is assigned, $\boldsymbol{\alpha}_t$, keeps track of the best combination of nodes found during hill climb t.

Each hill climb runs for K steps. At each step, we randomly select a node not currently in the map to replace a node that is in the map (Line 4). In other words, we randomly select a binary vector $\boldsymbol{\alpha}'$ with Hamming weight C whose Hamming distance from $\boldsymbol{\alpha}_t$ is 2. We then compute the RMSE (Root-Mean-Square-Error) of the new candidate map with respect to the S localization trials (Line 5). Computing this involves solving (5.2) with the new node combination $\boldsymbol{\alpha}'$. This yields training error $e\left(\boldsymbol{\alpha}'\right)$. If the new node combination has lower error, we replace $\boldsymbol{\alpha}_t$ with it (Line 7).

Once all hill climbs are complete, we return the best node combination found during all hill climbs (Line 8). This vector $\hat{\boldsymbol{\alpha}}^*$ is MOSS's approximation of an optimal combination $\boldsymbol{\alpha}^*$.

5.5 Evaluation

We will now evaluate MOSS in the two simulated scenarios illustrated in Fig.5.5 and compare its performance to that of the *Equidistant* and *SSM* methods. The first scenario, CORRIDOR, consists of a long hallway with a T-shaped intersection at one end. The geometry of this environment, which commonly occurs in indoor settings, leads to feature aliasing. Therefore, CORRIDOR serves as a useful test of MOSS's ability to remove nodes from portions of the map prone to this problem. The second scenario, SQUARE, occurs in a larger environment with multiple L-shaped walls. The environment's large area and interior walls result in variable observations from the robot's local view as it moves through the environment.

For each scenario, we first manually drive the robot through the environment to produce the map factor graph. We then conduct a number of localization trials in which the robot navigates through the environment autonomously. The robot starts at the same position with a known initial pose in each of these trials. We randomly sample waypoints from around the mapping trajectory; the robot navigates to these waypoints using the Dynamic Window algorithm [26]. To increase the variability of the robot's trajectory, we place static obstacles randomly along the robot's path.

The simulated robot is equipped with a 2-D laser scanner with a $360°$ field of view. This sensor has a 4 meter range and a distance error of $\sigma_d = 2$ cm. The robot's odometry measurements are sampled with variance $\sigma_{xy} = 30$ cm for translation and $\sigma_\theta = 0.1°$ for orientation. We generate a *L2G* factor potential $\boldsymbol{z}_k^{\text{L2G}}$ between pose $\boldsymbol{x}_{ik}$ in the localization pose graph and the nearest pose $\boldsymbol{y}_{j_k}$ in the prior map.

We use the ground truth localization provided by the simulator to evaluate the localization performance of the tested methods. We measure the root-mean-square-error (RMSE), which is the average error measured throughout all localization trials.

We set the parameters of Alg. 5.1 at $T = 10$ hill climbs, each consisting of $K = 1000$.

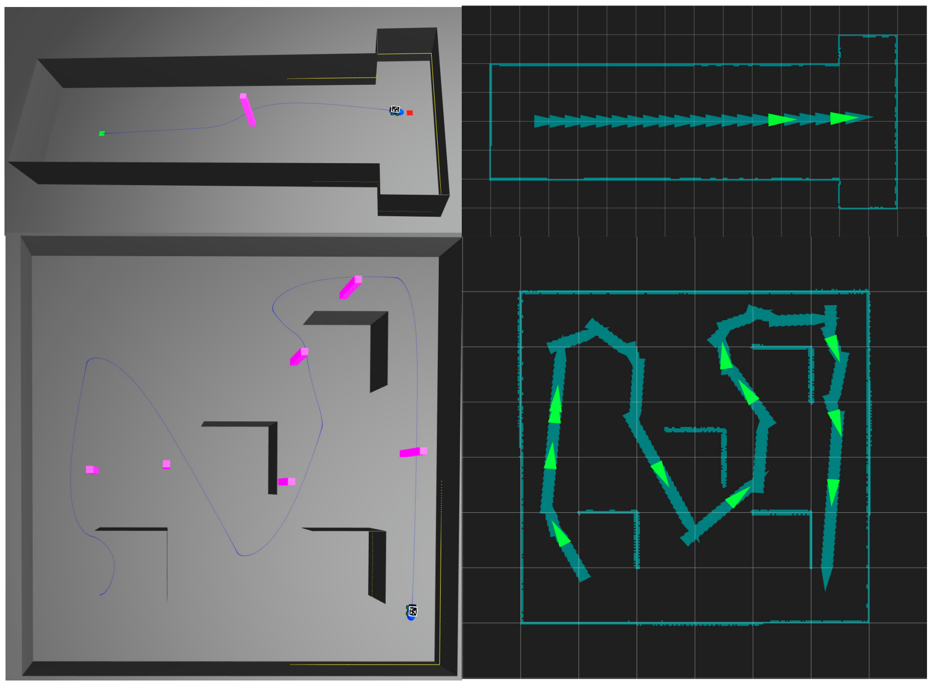


Figure 5.5: *Left*: Example localization trials in two simulated scenarios (CORRIDOR and SQUARE). *Right*: Maps produced by MOSS for both scenarios. Highlighted green triangles indicate nodes selected by MOSS from the full map to include in the map optimized for future localization performance.

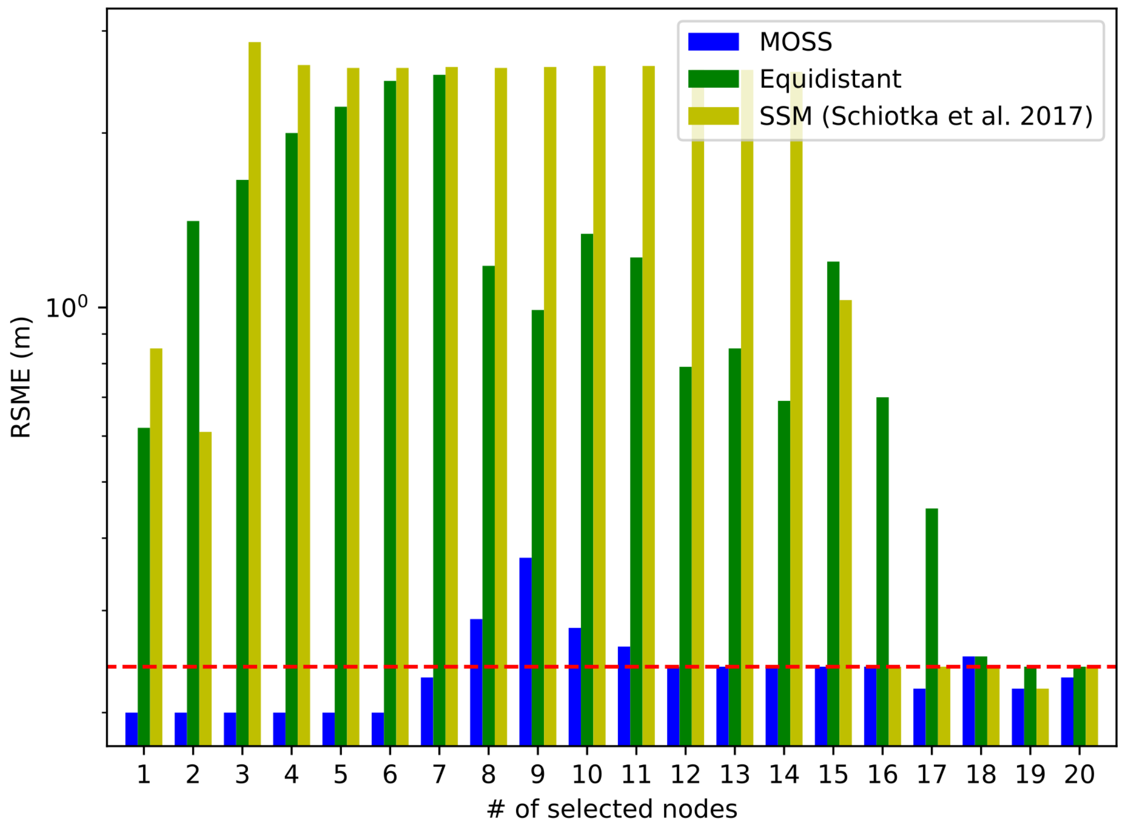


Figure 5.6: Localization test results in CORRIDOR. The red dashed line indicates the localization performance using the original map (i.e. $C = 21$). Overall, MOSS finds map representations with lower error than the competing methods. Furthermore, the best maps produced by MOSS perform better than the original complete prior map.

5.5.1 Corridor

The CORRIDOR scenario covers an area 12 meters in length and 6 meters in width consisting of a hallway and a T-shaped intersection. We conducted a single mapping run, 10 training localization trials, and 10 testing localization trials. The trajectories were 10 to 15 meters in length, and we discretized each of them into 21 poses. We varied the desired number of nodes in the prior map, C, from 1 to 20.

Fig.5.6 shows the localization RMSE for the test trials resulting from using the map generated by each method. The red dashed line indicates the localization performance using the original map. Overall, the best results come from using the map generated by MOSS with $C \leq 6$ nodes. Notably, the localization error using these maps is even lower than the complete prior map (i.e. $C = 21$). This supports our claim that MOSS can eliminate portions of the map that are prone to feature aliasing to achieve better localization performance.

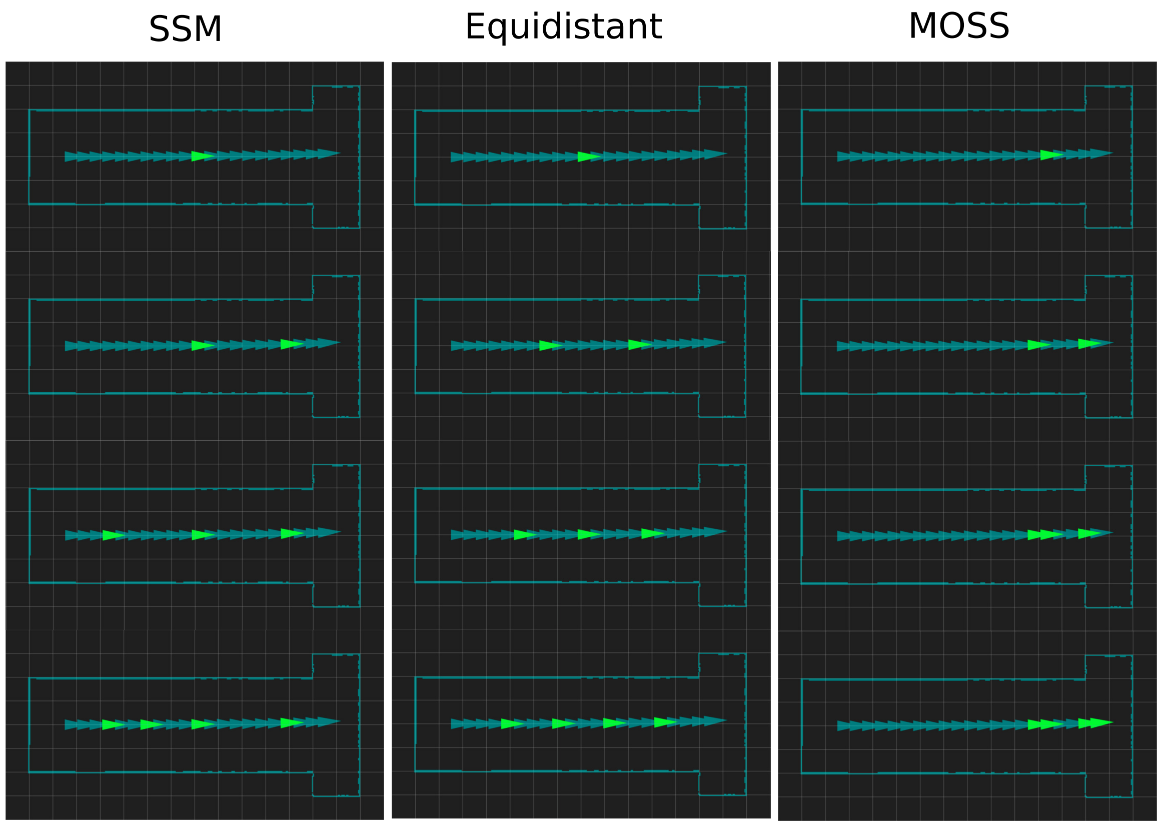


Figure 5.7: CORRIDOR map products of different approaches. Green triangles are the selected nodes. *Equidistant* covers most areas of the mapping environments as nodes are distributed uniformly. *SSM* applies a greedy method to maximize observation likelihood for the mapping trajectory. This does not protect against selecting nodes susceptible to perceptual aliasing during localization, which in turn leads to poor localization performance. Instead of selecting nodes spaced throughout the hallway, MOSS prioritizes nodes with distinctive scan features (i.e. those near the intersection).

To better understand the performance of the methods, consider Fig.5.7, which shows the nodes selected by each method for $C = 1, 2, 3$ and 4 nodes. MOSS selects nodes near the intersection since that region of the map has the most distinctive scan features that support reliable localization. In contrast, the *Equidistant* and *SSM* methods select nodes spaced throughout the hallway. Recall that SSM selects nodes that maximize the likelihood of mapping observations given the trajectory. This does not protect against selecting nodes susceptible to perceptual aliasing during localization.

In the 14-node map selected by SSM, the robot is stuck at the beginning so it never gets chance to match scans to the rest of nodes, while adding one node in 15-node map helps the robot get out the trap. Thus, the error of SSM abruptly falls from 14-node map to 15-node map (See in Fig.5.8).

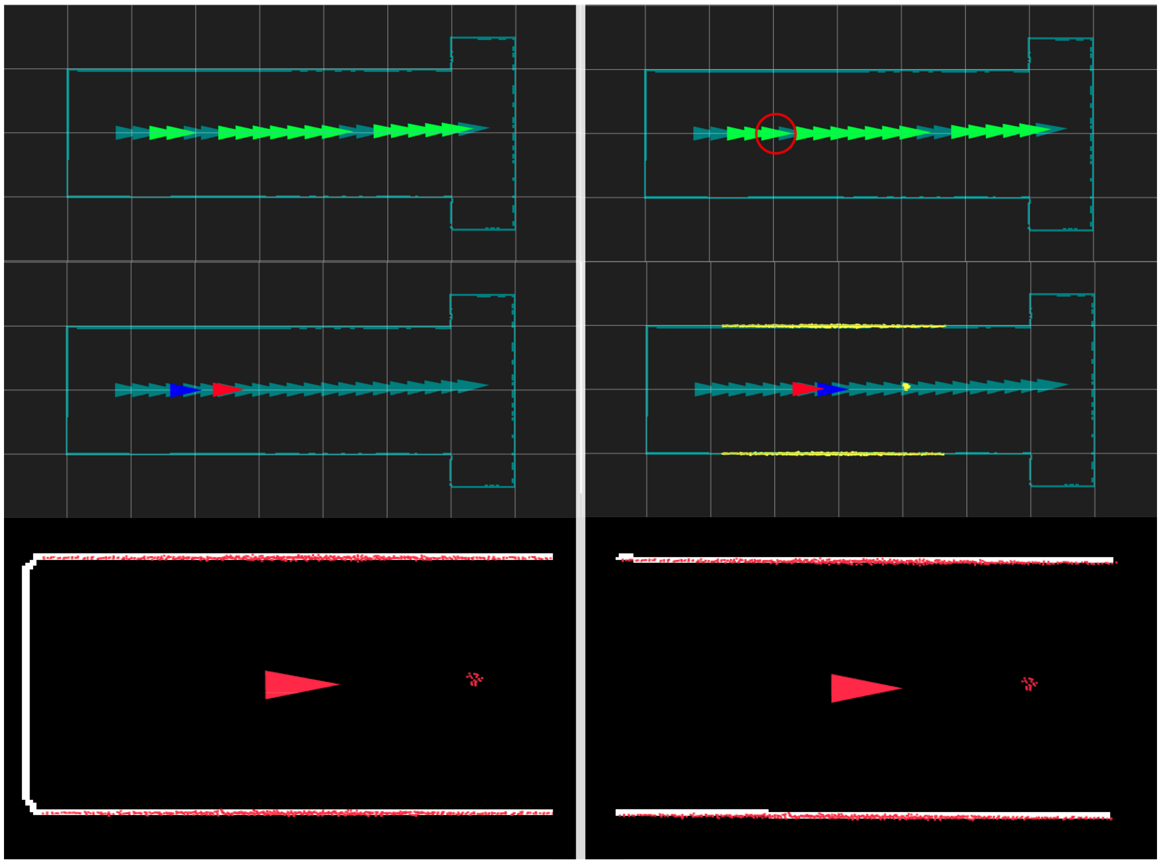


Figure 5.8: *Top left:* 14-node map selected by SSM. *Top right:* 15-node map selected by SSM. New added node is circled. *Middle:* Localization results. The red triangle is the ground truth and the blue triangle is the localized position. In the 14-node map, the robot is stuck at the beginning so it never gets chance to match scans to the rest of nodes, while adding one node in 15-node map helps the robot get out the trap.

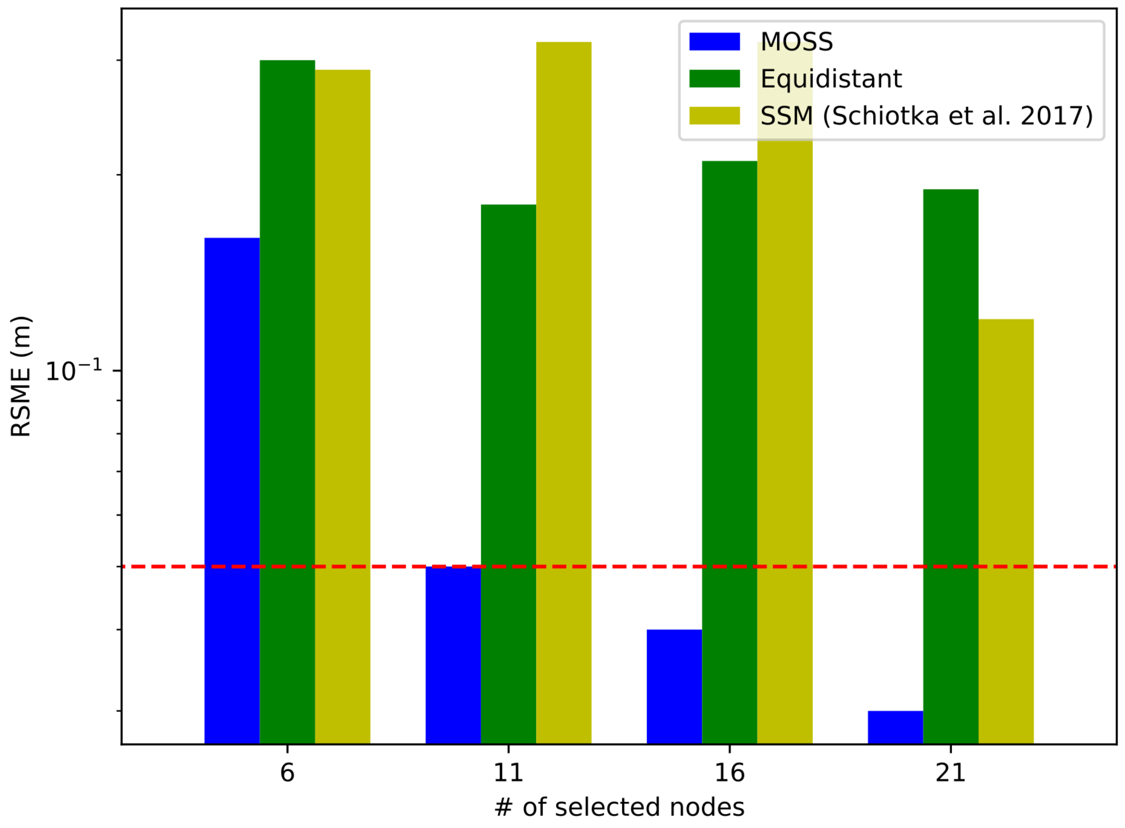


Figure 5.9: Localization test results in SQUARE. The red dashed line is the localization performance using the original map (i.e. $C = 204$). MOSS achieves the best localization results across each tested map size. MOSS achieves localization performance at least as good as that of the original full map with as few as $C = 11$ nodes, a reduction of approximately 95 percent in map size.

5.5.2 Square

The SQUARE scenario consists of a square-shaped environment with 12-meter sides (see Fig.5.5). The limited range of the simulated laser sensor causes the robot's local view to vary significantly for different trajectories, making robust localization challenging. We conducted a single mapping run, 100 training localization trials and 20 testing localization trials. The trajectories were 35 to 50 meters in length, and we discretized each of them into 204 poses. We varied the desired number of map poses as $C = 6, 11, 16,$ and 21.

Fig.5.9 shows the localization RMSE resulting from using the map generated by each method. Again, the red dashed line indicates the localization performance using the original map. The map generated by MOSS with $C = 21$ nodes yielded the best overall performance. MOSS's map with $C = 11$ nodes performed equivalently to the full prior map while reducing the number of nodes in the map by 95 percent.

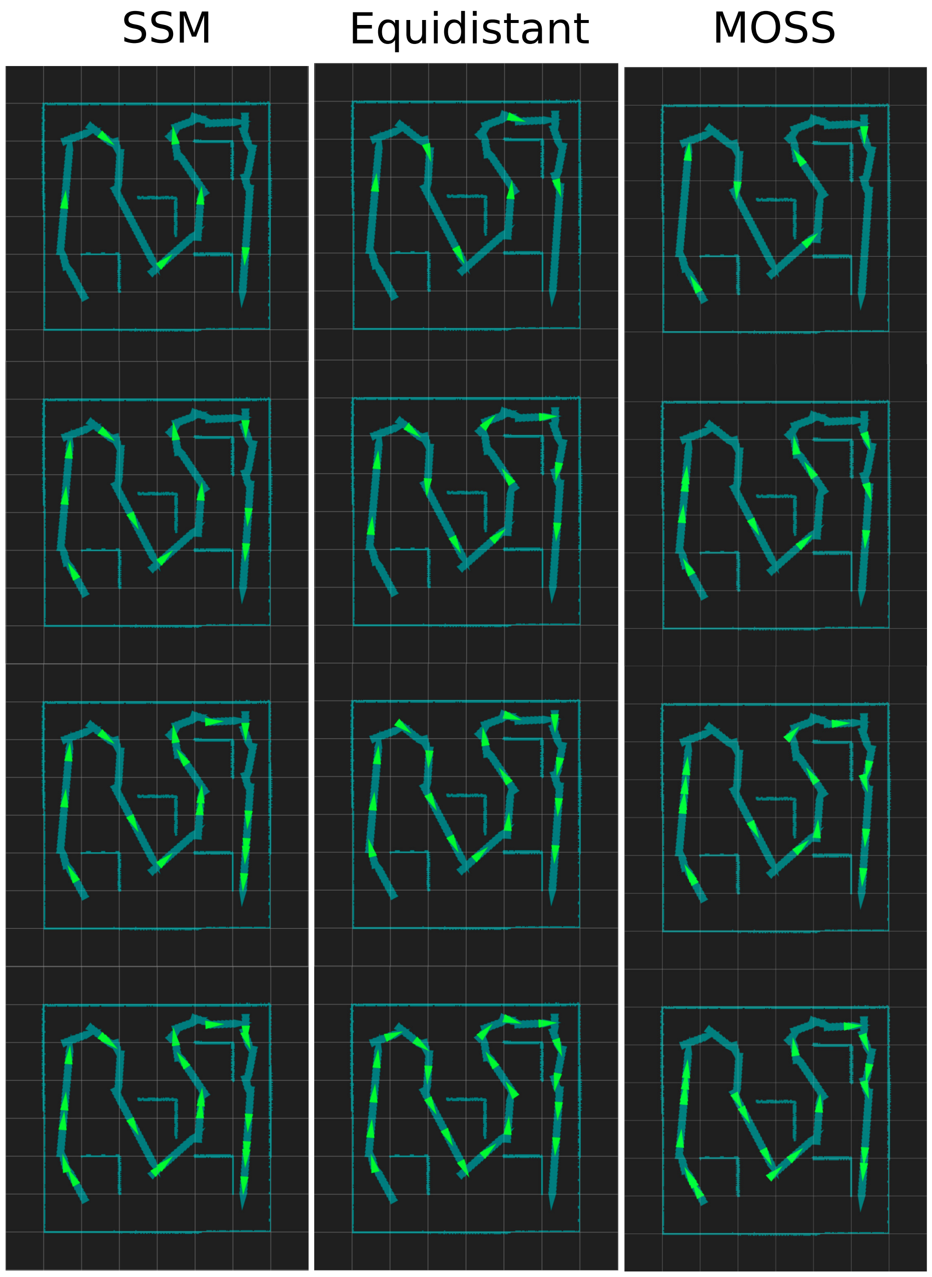


Figure 5.10: SQUARE map products of different approaches. Triangles marked in green are selected nodes in the graph using different algorithms

.

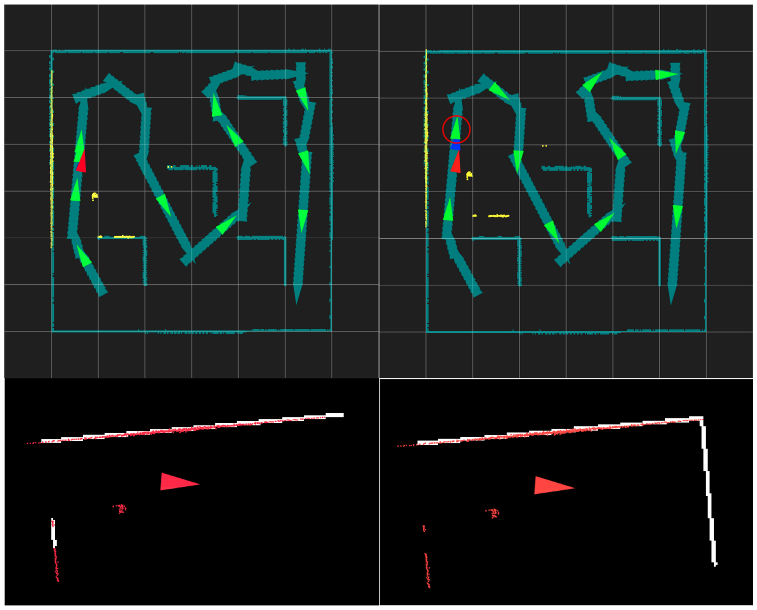


Figure 5.11: An example of Equidistant method's failure. Red triangle is the ground truth position and blue triangle is the localized position. *Left:* Robot is precisely localized in the 11-node map selected by MOSS. Map scans are shown in white and live scans are shown in red. *Right:* Robot is localized in the wrong place because of the aliasing in local view between map scans and live scans. It is pulled closed to the circled map node.

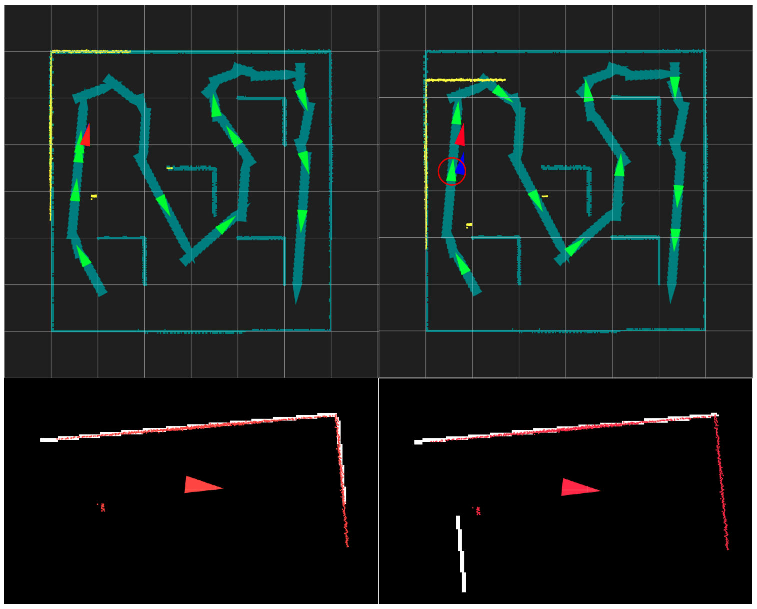


Figure 5.12: An example of SSM method's failure. Red triangle is the ground truth position and blue triangle is the localized position. *Left:* Robot is precisely localized in the 11-node map selected by MOSS. Map scans are shown in white and live scans are shown in red. *Right:* Robot is localized in the wrong place because of the aliasing in local view between map scans and live scans. It is trapped near the circled map node.

5.6 Summary

In this chapter, we presented MOSS, a machine learning-based map optimization approach that produces compact maps supporting robust localization. We first formulated online localization as a factor graph optimization problem. Then we proposed an approach for learning to select the combinations of nodes in the map factor graph that achieves least errors on subsequent localization trials. Our experimental evaluation shows that MOSS achieves better localization results than either an existing state-of-the-art map reduction approach or the original full map. Thus, MOSS expands the operational envelope in terms of *robustness*, increasing the *availability* of position estimates to a robot.

CHAPTER 6

Conclusion

Mapping and localization are fundamental requirements for autonomous robots being able to plan safe paths and reach the desired destination. The challenge arises when a mobile robot needs mapping and localization available outside the operational envelope that is determined by *scale*, *robustness* and *the amount of prior data needed*. In this thesis, we present **high availability** mapping and localization algorithms that focus on expanding this envelope by increasing the mapping speed, improving localization robustness, and reducing the amount of prior data required.

6.1 Contributions

This dissertation included the following contributions.

In Chapter 3, we presented AprilSAM that uses a min-heap based variable reordering algorithm coupled with fast incremental Cholesky factorization. This algorithm drives down system error while maintaining real-time performance for solving large scale mapping problems. We have shown that AprilSAM achieves better absolute error than other state-of-the-art algorithms while running faster to obtain the same mapping performance.

In Chapter 4, we presented FLAG that enables a ground robot to localize itself in a global reference frame by identifying landmarks visible in a floor plan map. We demonstrated that our system has comparable performance to laser scan matching-based localization with a Lidar prior map, showing FLAG's feasibility for real-time global localization in a previously unvisited indoor environment.

In Chapter 5, we presented MOSS that optimizes a map for subsequent localization performance using machine learning-based methods. We have shown that MOSS produces compact maps that support more robust localization than state-of-the-art approaches.

6.2 Discussions

6.2.1 Faster Optimization for Large Scale Mapping

One of major challenge for large-scale mapping is to maintain real-time performance and high quality estimation for optimization when dealing with large number of observations. Despite AprilSAM is the fastest algorithm that gives the most accurate estimates, the running time of it is not bounded even for small environments which are repeatedly explored. It is desirable to at least achieve a persistent mapping solution that scales only in terms of the spatial extent of an environment, and not the duration of the mission. This could be done by applying node removal algorithms [14, 41]. However, as mapping area becomes larger and larger, the algorithm still grows unbounded. Sub-mapping approaches [69, 70] then could be used to solve this problem. It seems obvious to us that the combination of those algorithms will perform the best. However, it is still unclear how to design the three systems that interact with each other to achieve the best. How would the node removal affect the incremental update? How would the map division affect the decision between batch update and incremental update? Exploring this idea is out of the scope of this thesis, but we believe it is a practical approach to solve large scale mapping problems.

6.2.2 Leveraging Existing Map Resource for Localization

Localization in a global reference frame is often done as pose estimation in a global consistent map. In order to achieve good localization performance, very detailed maps are usually built ahead. This can provide the system higher position updates rate compared to the approaches leveraging sparse features in the environment (e.g. FLAG uses corner features that are sparsely distributed in the floor plan map). However, map building is a computationally expensive and time consuming process. If maps do not exist, we can use the algorithms developed in Chapter 3, but if maps exist (even if not designed for robots) and a robot can re-use them, it will allow them to navigate without needing to explore the full environment first. Lots of approaches have been developed in the outdoor environments, but to the best of our knowledge, FLAG is the first system that does not require additional map building by leveraging floor plans in the indoor environments. Although there are still limitations in FLAG, we believe that this technique paves the way towards a new way of localization approach that can be generally applied to indoor environments since architectural floor plan maps exist pretty much for every building.

6.2.3 Map Optimization for Robust Localization

A primary usage of the map for autonomous mobile robots is localization. Most of researches about robotic mapping focus on the consistency of the map product but neglect the use of it afterwards. This causes the problem that the improvements in the robotic mapping didn't bring lots of successes in robotic localization. As more and more robots are deployed into real world, we realize that robust localization is the key for long-term autonomy. Localization failures can be caused by that the live observations don't match the models in the prior map. To address this issue, the maps need to be updated when there is a discrepancy between new observations and models in the map. When dealing with the failures caused by perceptual aliasing, adding more aliased observations wouldn't help but result in more troubles. The novelty of MOSS is that we believe **the less is better** in this case. More specifically, the less features that may lead to wrong data association, the better localization performance we can get. **"The less is better"** may seem counter-intuitive especially in robotic localization that the key insight of Bayes Filter is the probabilistic data fusion. However, it works under the assumption that the error models of the observations are correct. For example, when robot is going through a tunnel, if the system is overconfident about the GPS measurements, it will cause localization failure. Max-mixture model [73] and dynamic covariance scaling [1] are approaches that reject outliers in optimization back-end. But they can only handle to a certain number of outliers. MOSS moves the outlier rejection to the front-end for online localization. It is still a challenging problem to come up with a good metric that defines the quality of features (e.g. laser scans, image features, etc) for localization in the complex world. Semantic information [10, 63] could help the algorithm work better, but it is not clear what semantic information is really helpful in certain environments. For example, semantic label *chair* is not quite helpful for the mapping and localization in a room full of chairs. Therefore, we introduced a data-driven approach that selects features that provide the best localization performance. Essentially, MOSS is computationally figuring out the answers for the most critical question: **will including this feature in map help improve the localization**?

6.3 Future Work

In this section, we present ways in which someone might extend or improve upon the algorithms of this dissertation.

6.3.1 Decision Making in AprilSAM

As presented in Chapter 3, AprilSAM uses the criteria listed in Alg. 3.1 to decide between incremental and batch updates. Choosing between batch update and incremental update is a decision-making problem. In this thesis, we presented several heuristics trying to drive down error and running time for SLAM problem. It would be interesting to design the objective function for making this decision based on whole autonomous system (speed, position uncertainty, planned trajectory, etc). When a vehicle approach an intersection, it is likely there will be a loop closure. If the vehicle stop at the intersection, AprilSAM should take this chance to perform batch update.

6.3.2 Incorporating Graph-based Mapping Techniques into FLAG

In Chapter 4, FLAG performs occasional global position updates directly in a floor plan, and relies upon open-loop odometry maintain local position estimates between global position fixes. However, the data association becomes extremely hard when the robot doesn't see a global feature for a long time, or some structures are not represented in the drawing. In addition to the methods presented in the thesis, incorporating graph-based mapping techniques into FLAG could possibly improve the system performance. The laser-based map can be aligned to the floor plan map and is exploited for relative localization.

6.3.3 Learning Value Function for MOSS

In Chapter 5, the goal of MOSS is to learn a function f^* in (5.3) that minimizes the error of the solution X_{ML} relative to the ground truth solution X_{GT} over a number of training trials. We simplify this problem to find the best combinations of map nodes based on the intuition that the quality of scan matching results varies among the nodes of the prior map. This means function f only takes map node position and map scans as input. We think that incorporating a robot's current position, position uncertainty, robot's live laser scans into function f could further improve the localization performance. For example, if the robot is surrounded by people and live scans look like a clutter of blobs, the system shouldn't trust the scan matching results even map scans seem to be very distinctive features. We think this type of learning can fit into a reinforcement learning framework. Regression function f can be an approximation of the value function which takes a robot's current position, position uncertainty, robot's live laser scans, and graph nodes' positions and laser scans as input. The action is whether (or how much) to trust scan matching results. The expected reward to be maximized is localization accuracy. Once f is learned, it can be applied for online localization to determine whether (or how much) should the system trust the scan matching results.

BIBLIOGRAPHY

[1] Pratik Agarwal. *Robust graph-based localization and mapping*. PhD thesis, University of Freiburg, 2015.

[2] Pratik Agarwal and Edwin Olson. Variable reordering strategies for SLAM. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, October 2012.

[3] Pratik Agarwal, Gian Diego Tipaldi, Luciano Spinello, Cyrill Stachniss, and Wolfram Burgard. Robust map optimization using dynamic covariance scaling. In *2013 IEEE International Conference on Robotics and Automation*. Citeseer, 2013.

[4] Shun-ichi Amari. Backpropagation and stochastic gradient descent method. *Neurocomputing*, 1993.

[5] Patrick R Amestoy, Timothy A Davis, and Iain S Duff. An approximate minimum degree ordering algorithm. *SIAM Journal on Matrix Analysis and Applications*, 1996.

[6] Tim Bailey and Hugh Durrant-Whyte. Simultaneous localization and mapping (slam): Part ii. *IEEE Robotics and Automation Magazine*, 13(3):108–117, 2006.

[7] Mayank Bansal, Harpreet S Sawhney, Hui Cheng, and Kostas Daniilidis. Geo-localization of street views with aerial image databases. In *Proceedings of the 19th ACM international conference on Multimedia*. ACM, 2011.

[8] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *European conference on computer vision*. Springer, 2006.

[9] Federico Boniardi, Tim Caselitz, Rainer Kümmerle, and Wolfram Burgard. Robust lidar-based localization in architectural floor plans. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2017.

[10] Sean L Bowman, Nikolay Atanasov, Kostas Daniilidis, and George J Pappas. Probabilistic data association for semantic slam. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1722–1729. IEEE, 2017.

[11] Guillaume Bresson, Zayed Alsayed, Li Yu, and Sébastien Glaser. Simultaneous localization and mapping: A survey of current trends in autonomous driving. *IEEE Transactions on Intelligent Vehicles*, 2(3):194–220, 2017.

[12] Cesar Cadena, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, José Neira, Ian Reid, and John J Leonard. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on Robotics*, 32(6):1309–1332, 2016.

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

[14] Nicholas Carlevaris-Bianco, Michael Kaess, and Ryan M Eustice. Generic node removal for factor-graph SLAM. *IEEE Transactions on Robotics*, 30(6):1371–1385, 2014.

[15] P Cheeseman, R Smith, and M Self. A stochastic map for uncertain spatial relationships. In *4th International Symposium on Robotic Research*, pages 467–474, 1987.

[16] Winston Churchill and Paul Newman. Experience-based navigation for long-term localisation. *International Journal of Robotics Research*, 32(14):1645–1661, 2013.

[17] Timothy A Davis, John R Gilbert, Stefan I Larimore, and Esmond G Ng. A column approximate minimum degree ordering algorithm. *ACM Transactions on Mathematical Software (TOMS)*, 2004.

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

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

[20] Frank Dellaert, Michael Kaess, et al. Factor graphs for robot perception. *Foundations and Trends® in Robotics*, 2017.

[21] Hugh Durrant-Whyte and Tim Bailey. Simultaneous localization and mapping: part I. *IEEE robotics & automation magazine*, 13(2):99–110, 2006.

[22] Hugh Durrant-Whyte, Somajyoti Majumder, Sebastian Thrun, Marc De Battista, and Steve Scheding. A bayesian algorithm for simultaneous localisation and map building. In *Robotics Research*. Springer, 2003.

[23] Felix Endres, Jürgen Hess, Nikolas Engelhard, Jürgen Sturm, Daniel Cremers, and Wolfram Burgard. An evaluation of the rgb-d slam system. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 2012.

[24] Ryan M Eustice, Hanumant Singh, and John J Leonard. Exactly sparse delayed-state filters for view-based slam. *IEEE Transactions on Robotics*, 2006.

[25] F Dan Foresee and Martin T Hagan. Gauss-newton approximation to bayesian learning. In *Proceedings of International Conference on Neural Networks (ICNN'97)*. IEEE, 1997.

[26] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics and Automation Magazine*, 4(1):23–33, 1997.

[27] Walter Gander. Algorithms for the QR decomposition. *Res. Rep*, 80(02):1251–1268, 1980.

[28] Robert Goeddel, Carl Kershaw, Jacopo Serafin, and Edwin Olson. FLAT2D: Fast localization from approximate transformation into 2D. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, October 2016.

[29] Robert Goeddel, Carl Kershaw, Jacopo Serafin, and Edwin Olson. FLAT2D: Fast localization from approximate transformation into 2D. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, October 2016.

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

[31] Laura Grigori, Erik G Boman, Simplice Donfack, and Timothy A Davis. Hypergraph-based unsymmetric nested dissection ordering for sparse lu factorization. *SIAM Journal on Scientific Computing*, 2010.

[32] Giorgio Grisetti, Cyrill Stachniss, Slawomir Grzonka, and Wolfram Burgard. A tree parameterization for efficiently computing maximum likelihood maps using gradient descent. In *Robotics: Science and Systems*, pages 27–30, 2007.

[33] Giorgio Grisetti, Rainer Kummerle, Cyrill Stachniss, and Wolfram Burgard. A tutorial on graph-based $slam$. *IEEE Intelligent Transportation Systems Magazine*, 2010.

[34] Giorgio Grisettiyz, Cyrill Stachniss, and Wolfram Burgard. Improving grid-based slam with rao-blackwellized particle filters by adaptive proposals and selective resampling. In *Proceedings of the 2005 IEEE international conference on robotics and automation*, pages 2432–2437. IEEE, 2005.

[35] Ming Gu and Stanley C Eisenstat. Efficient algorithms for computing a strong rank-revealing qr factorization. *SIAM Journal on Scientific Computing*, 1996.

[36] Matthias Hentschel, Oliver Wulf, and Bernardo Wagner. A gps and laser-based localization for urban and non-urban outdoor environments. In *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*. IEEE, 2008.

[37] Martial Herbert, Claude Caillas, Eric Krotkov, In-So Kweon, and Takeo Kanade. Terrain mapping for a roving planetary explorer. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 997–1002. Institute of Electrical and Electronics Engineers, 1989.

[38] Shoudong Huang and Gamini Dissanayake. Convergence and consistency analysis for extended kalman filter based slam. *IEEE Transactions on robotics*, 2007.

[39] Viorela Ila, Josep M Porta, and Juan Andrade-Cetto. Information-based compact pose slam. *IEEE Transactions on Robotics*, 2010.

[40] Viorela Ila, Lukas Polok, Marek Solony, and Pavel Svoboda. Slam++-a highly efficient and temporally scalable incremental slam framework. *The International Journal of Robotics Research*, 2017.

[41] Hordur Johannsson, Michael Kaess, Maurice Fallon, and John J Leonard. Temporally scalable visual slam using a reduced pose graph. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE, 2013.

[42] Michael Kaess, Ananth Ranganathan, and Frank Dellaert. iSAM : Incremental Smoothing and Mapping. *IEEE Transactions on Robotics*, 2008.

[43] Michael Kaess, Hordur Johannsson, Richard Roberts, Viorela Ila, John J Leonard, and Frank Dellaert. iSAM2 : Incremental Smoothing and Mapping with Fluid Relinearization and Incremental Variable Reordering. *The International Journal of Robotics Research*, 2012.

[44] Alex Kendall, Matthew Grimes, and Roberto Cipolla. Posenet: A convolutional network for real-time 6-dof camera relocalization. In *Proceedings of the IEEE international conference on computer vision*, 2015.

[45] Dong-Ki Kim and Matthew R. Walter. Satellite image-based localization via learned embeddings. In *Proceedings of the IEEE International Conference on Robotics and Automation*, May 2017.

[46] Dong-Ki Kim and Matthew R Walter. Satellite image-based localization via learned embeddings. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017.

[47] Henrik Kretzschmar, Cyrill Stachniss, and Giorgio Grisetti. Efficient information-theoretic graph pruning for graph-based SLAM with laser range finders. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 865–871. Institute of Electrical and Electronics Engineers, 2011.

[48] Rainer Kümmerle, Giorgio Grisetti, Hauke Strasdat, Kurt Konolige, and Wolfram Burgard. g2o: A general framework for graph optimization. In *2011 IEEE International Conference on Robotics and Automation*. IEEE, 2011.

[49] Rainer Kümmerle, Bastian Steder, Christian Dornhege, Alexander Kleiner, Giorgio Grisetti, and Wolfram Burgard. Large scale graph-based slam using aerial images as prior information. *Autonomous Robots*, 2011.

[50] Ngai Ming Kwok and Gamini Dissanayake. An efficient multiple hypothesis filter for bearing-only slam. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2004.

[51] L Lasdon, S Mitter, and A Waren. The conjugate gradient method for optimal control problems. *IEEE Transactions on Automatic Control*, 1967.

[52] Gim Hee Lee, Friedrich Fraundorfer, and Marc Pollefeys. Robust pose-graph loop-closures with expectation-maximization. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 556–563. Institute of Electrical and Electronics Engineers, 2013.

[53] John J Leonard and Hugh F Durrant-Whyte. Simultaneous map building and localization for an autonomous mobile robot. In *Proceedings IROS'91: IEEE/RSJ International Workshop on Intelligent Robots and Systems' 91*. Ieee, 1991.

[54] Jesse Levinson and Sebastian Thrun. Robust vehicle localization in urban environments using probabilistic maps. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. IEEE, 2010.

[55] Jesse Levinson, Michael Montemerlo, and Sebastian Thrun. Map-based precision vehicle localization in urban environments. In *Robotics: Science and Systems (RSS)*, 2007.

[56] Tsung-Yi Lin, Yin Cui, Serge Belongie, and James Hays. Learning deep representations for ground-to-aerial geolocalization. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015.

[57] Kai Lingemann, Andreas Nüchter, Joachim Hertzberg, and Hartmut Surmann. High-speed laser localization for mobile robots. *Robotics and autonomous systems*, 2005.

[58] H-A Loeliger. An introduction to factor graphs. *IEEE Signal Processing Magazine*, 2004.

[59] Will Maddern, Geoffrey Pascoe, and Paul Newman. Leveraging experience for large-scale LIDAR localisation in changing cities. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1684–1691. Institute of Electrical and Electronics Engineers, 2015.

[60] Raj Madhavan and Hugh F Durrant-Whyte. Natural landmark-based autonomous vehicle navigation. *Robotics and Autonomous Systems*, 2004.

[61] Joshua G. Mangelson, Derrick Dominic, Ryan M. Eustice, and Ram Vasudevan. Pairwise consistent measurement set maximization for robust multi-robot map merging. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1–8, Brisbane, Australia, May 2018.

[62] Donald W Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *Journal of the society for Industrial and Applied Mathematics*, 1963.

[63] John McCormac, Ronald Clark, Michael Bloesch, Andrew Davison, and Stefan Leutenegger. Fusion++: Volumetric object-level slam. In *2018 International Conference on 3D Vision (3DV)*, pages 32–41. IEEE, 2018.

[64] J Andvandervorst Meijerink and Henk A Van Der Vorst. An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix. *Mathematics of computation*, 1977.

[65] Michael Montemerlo, Sebastian Thrun, Daphne Koller, Ben Wegbreit, et al. Fastslam: A factored solution to the simultaneous localization and mapping problem. *Aaai/iaai*, 593598, 2002.

[66] Michael Montemerlo, Sebastian Thrun, Daphne Koller, Ben Wegbreit, et al. Fastslam 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges. In *IJCAI*, 2003.

[67] Hans P Moravec and Alberto Elfes. High resolution maps from wide angle sonar. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 19–24, 1985.

[68] José Neira and Juan D Tardós. Data association in stochastic mapping using the joint compatibility test. *IEEE Transactions on robotics and automation*, 2001.

[69] Kai Ni and Frank Dellaert. Multi-level submap based SLAM using nested dissection. 2010.

[70] Kai Ni, Drew Steedly, and Frank Dellaert. Tectonic SAM: Exact, out-of-core, submap-based SLAM. In *Robotics and Automation, 2007 IEEE International Conference on*. IEEE, 2007.

[71] Edwin Olson. Real-time correlative scan matching. In *Robotics and Automation (ICRA), IEEE International Conference on*, 2009.

[72] Edwin Olson. M3RSM: Many-to-Many Multi-Resolution Scan Matching. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, June 2015.

[73] Edwin Olson and Pratik Agarwal. Inference on networks of mixtures for robust robot mapping. *International Journal of Robotics Research*, 32(7):826–840, July 2013.

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

[75] Edwin Olson and Yangming Li. Ipjc: The incremental posterior joint compatibility test for fast feature cloud matching. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, October 2012.

[76] Edwin Olson, Matthew Walter, John Leonard, and Seth Teller. Single cluster graph partitioning for robotics applications. In *Proceedings of the Robotics: Science & Systems Conference*, 2005.

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

[78] Edwin B Olson. Robust and efficient robotic mapping. 2008.

[79] Martin P Parsley and Simon J Julier. Towards the exploitation of prior information in slam. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*. IEEE, 2010.

[80] Martin P Parsley and Simon J Julier. Exploiting prior information in graphslam. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011.

[81] Lukas Polok, Viorela Ila, Marek Solony, Pavel Smrz, and Pavel Zemcik. Incremental Block Cholesky Factorization for Nonlinear Least Squares in Robotics. In *Robotics: Science and Systems (RSS)*, 2013.

[82] Ananth Ranganathan, David Ilstrup, and Tao Wu. Light-weight localization for vehicles using road markings. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2013.

[83] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *Computer Vision (ICCV), IEEE International Conference on*, 2011.

[84] Heinz Rutishauser. The jacobi method for real symmetric matrices. *Numerische Mathematik*, 1966.

[85] Charles Schaff, David Yunis, Ayan Chakrabarti, and Matthew R Walter. Jointly optimizing placement and inference for beacon-based localization. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 6609–6616. Institute of Electrical and Electronics Engineers, 2017.

[86] Alexander Schiotka, Benjamin Suger, and Wolfram Burgard. Robot localization with sparse scan-based maps. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 642–647. Institute of Electrical and Electronics Engineers, 2017.

[87] Robert Schreiber. A new implementation of sparse gaussian elimination. *ACM Transactions on Mathematical Software (TOMS)*, 1982.

[88] Brad Schumitsch, Sebastian Thrun, Gary Bradski, and Kunle Olukotun. The information-form data association filter. In *Advances in Neural Information Processing Systems*, pages 1193–1200, 2006.

[89] Turgay Senlet and Ahmed Elgammal. Satellite image based precise robot localization on sidewalks. In *Proceedings of the IEEE International Conference on Robotics and Automation*. IEEE, 2012.

[90] Heung-Yeung Shum, Qifa Ke, and Zhengyou Zhang. Efficient bundle adjustment with virtual key frames: A hierarchical approach to multi-frame structure from motion. In *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on.*, volume 2, pages 538–543. Institute of Electrical and Electronics Engineers, 1999.

[91] Donald F Specht. A general regression neural network. *IEEE transactions on neural networks*, 1991.

[92] Alexander D Stewart and Paul Newman. Laps-localisation using appearance of prior structure: 6-dof monocular camera localisation using prior pointclouds. In *Proceedings of the IEEE International Conference on Robotics and Automation*. IEEE, 2012.

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

[94] Peter Teunissen. Nonlinear least squares. 1990.

[95] Sebastian Thrun, Yufeng Liu, Daphne Koller, Andrew Y Ng, Zoubin Ghahramani, and Hugh Durrant-Whyte. Simultaneous localization and mapping with sparse extended information filters. *The International Journal of Robotics Research*, 23(7-8): 693–716, 2004.

[96] William F Tinney and John W Walker. Direct solutions of sparse network equations by optimally ordered triangular factorization. *Proceedings of the IEEE*, 1967.

[97] Akihiko Torii, Josef Sivic, and Tomas Pajdla. Visual localization by linear combination of image descriptors. In *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, pages 102–109. IEEE, 2011.

[98] Anirudh Viswanathan, Bernardo R Pires, and Daniel Huber. Vision based robot localization by ground to satellite matching in GPS-denied situations. In *Intelligent Robots and Systems (IROS), IEEE International Conference on*, 2014.

[99] Matthew R Walter, Ryan M Eustice, and John J Leonard. Exactly sparse extended information filters for feature-based slam. *The International Journal of Robotics Research*, 2007.

[100] Sen Wang, Ronald Clark, Hongkai Wen, and Niki Trigoni. Deepvo: Towards end-to-end visual odometry with deep recurrent convolutional neural networks. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017.

[101] Xipeng Wang, Steve Vozar, and Edwin Olson. FLAG: Feature-based localization between air and ground. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, May 2017.

[102] Xipeng Wang, Ryan Marcotte, Gonzalo Ferrer, and Edwin Olson. AprilSAM: Real-time smoothing and mapping. In *Proceedings of the IEEE International Conference on Robotics and Automation*, May 2018.

[103] Jan Weingarten and Roland Siegwart. Ekf-based 3d slam for structured environment reconstruction. In *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2005.

[104] Kai M Wurm, Armin Hornung, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. OctoMap: A probabilistic, flexible, and compact 3D map representation for robotic systems. In *Proc. of the ICRA 2010 workshop on best practice in 3D perception and modeling for mobile manipulation*, volume 2, 2010.

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

[106] Manuel Yguel, Christopher Tay, Meng Keat, Christophe Braillon, Christian Laugier, and Olivier Aycard. Dense mapping for range sensors: Efficient algorithms and sparse representations, 2007.