# Chapter 2
# Consumer Robotics: A Platform for Embedding Computer Vision in Everyday Life

**Mario E. Munich, Phil Fong, Jason Meltzer and Ethan Eade**

**Abstract** Consumer robotic devices provide a platform for embedded computer vision algorithms in applications for everyday life. The consumer market is very price-sensitive, so robots must be developed with a single task in mind, aiming to provide the best performance at the lowest cost. Computational resources in consumer robotics are scarce given cost constraints, forcing the design of novel algorithms that elegantly incorporate such constraints. We present a graph-based SLAM approach designed to operate on computationally constrained platforms using monocular vision and odometry. When computation and memory are limited, visual tracking becomes difficult or impossible, and costs for map representation and updating must remain low. Our system constructs a map of structured views using only weak temporal assumptions and performs recognition and relative pose estimation over the set of views. We fuse visual observations and differential measurements in an incrementally optimized graph representation. Using variable elimination and constraint pruning, graph complexity and storage is kept linear in explored space rather than growing over time. We evaluate performance on sequences with ground truth and also compare to a standard graph-SLAM approach.

## 2.1 Introduction

Over the past decade, computer vision algorithms have transitioned from the lab to the marketplace. Improvements in processors, memory density, and image sensor technology enable the deployment of sophisticated algorithms. The introduction

M.E. Munich (✉) · P. Fong · J. Meltzer
iRobot, Pasadena, CA, USA
e-mail: mmunich@irobot.com

P. Fong
e-mail: pfong@irobot.com

J. Meltzer
e-mail: jmeltzer@irobot.com

E. Eade
Microsoft Research, Redmond, WA, USA
e-mail: ethan@ethaneade.com



B. Kisačanin and M. Gelautz (eds.), *Advances in Embedded Computer Vision*,
Advances in Computer Vision and Pattern Recognition,
DOI 10.1007/978-3-319-09387-1_2

of smartphones and tablets has accelerated the pace of this trend. These mobile devices include powerful processors, ample memory, and high-resolution cameras. Coupled with high-level operating systems such as iOS and Android, these enable the quick development of computer-vision-based applications. One can argue that these devices provide a conduit for the deployment of embedded computer vision in the consumer electronics market. However, these devices are fairly expensive, which allow them to provide the resources computation- and memory-hungry computer vision applications require.

Consumer robotic devices, on the other hand, face severe constraints on the cost of computation. The mass consumer market is very price-sensitive, so the retail cost of the robot is key for the success of the product. The consumer electronics industry standard suggests a retail price for the product that is 3–5 times the cost of parts (bill of materials, or BOM). In other words, for a $300 MSRP robot, the BOM should be between $60 and 100, including all mechanical parts, electrical parts, battery, processor, memory, motors, assembly, packaging, user manuals, and miscellaneous items! These strict cost constraints translate into a reduced availability of computational resources, requiring the development of particularly lean algorithms. Consumer robotics thus serves as an important platform for deploying embedded computer vision applications.

Another interesting factor that distinguishes the smartphone from the robotic use case is autonomy. Unlike in a smartphone app, the vision system in a robot must work reliably with no user assistance, continuously and for long periods of time. There is no opportunity for a user to correct an error or ignore a defect; if the vision system fails, the effectiveness of the overall system is reduced.

Visual localization and mapping is attractive for low-cost robotics applications since cameras are data rich, low power, and inexpensive. The challenge lies in designing an algorithm that can efficiently extract relevant information from this high-rate visual data stream. Despite Moore's Law, low-cost embedded platforms are still constrained by limited processing power, memory, and storage. Many state-of-the-art approaches to visual SLAM rely on interframe tracking, which requires high frame rate processing. Additionally, common constraint graph SLAM methods for agglomerating sensor information often incur computation and storage costs that grow with time rather than with space explored. For a robot operating for extended periods within a limited spatial area—typical of practical applications—this is an undesirable trade-off.

This chapter describes the development of a localization system that can enable systematic navigation of domestic robots in a household environment. The target application is a mobile domestic robot with a price lower than $1000, and ideally below $500. We present an approach to visual localization and mapping designed for a low-cost robotic platform equipped with simple odometry and a single camera. Operating primarily as a recognition engine, the visual measurement subsystem requires only occasional, weak assumptions on processing rate, and intrinsically provides robust loop closing when previously-mapped areas are revisited. Visual measurements and odometry are fused in a graph representation and optimized incrementally. Important novel features of this system include techniques for bounding the

SLAM graph complexity during operation, using variable elimination and constraint pruning with heuristic schedules. These methods keep optimization and storage costs commensurate with explored area rather than with time of exploration while causing minimal loss in mapping and localization accuracy.

An instantiation of the approach is demonstrated on real datasets with planar ground-truth reference. The system operates successfully even at frame rates below 2 Hz. Comparing the results with and without complexity reduction demonstrates that the reduced graph yields similar localization accuracy at a small fraction of the computational cost.

## 2.2 Related Work

### *2.2.1 View Recognition for SLAM*

View recognition engines have proven attractive components for SLAM systems because they permit robust and flexible loop closing. Instead of making correspondences between individual features or measurements, visual or otherwise, view recognition engines typically match constellations of features or entire images without requiring feature tracking.

Williams et al. [20] rely on tracking for normal EKF SLAM operation, but use view recognition to recover from failure. Several features are matched to the existing map using appearance and structure constraints in order to reinitialize tracking.

The Parallel Tracking and Mapping(PTAM) [10] system also employs view recognition for recovery from tracking failure. Instead of using feature-based methods for identifying candidate views, the system performs image-to-image correlation using heavily blurred, low-resolution versions of the reference and query images. A crude pose estimate is deduced from the result of the inverse-compositional matching, following which tracking resumes.

Eade and Drummond [4] group subsets of features into local maps during tracking-based SLAM. Correspondences are made between local maps to connect them or to recover from tracking failures. The image-to-map matching first selects a subset of local maps to consider using a bag-of-words ranking, and then performs local matching to determine feature-to-feature correspondences. This two-step process is common to many view recognition systems, often instantiated as a bag-of-words prefilter followed by re-ranking using geometric constraints [17].

The above approaches rely on tracking and use view recognition as an out-of-band method for failure recovery. Our approach instead performs recognition at every time step as the primary source of observations. The system of Karlsson et al. [9] is similar, constructing *landmarks* out of constellations of SIFT [14] features and employing nearest neighbors and a simple Hough transform as the recognition algorithm. The system is further refined by Eade et al. [5] by replacing the particle-filter back end with a graph SLAM back end that is described in further detail in

the following sections. The work of Cummins et al. [2] takes a more sophisticated approach to recognition, building a visual vocabulary offline, and approximating the joint probability distribution of visual words with a Chow-Liu tree. Each view's appearance model is updated upon recognition.

Our view recognition front end bears many similarities to the view-based maps of Konolige et al. [12]. That system constructs views from stereo images and performs two-step recognition using first a vocabulary tree and then a geometric matching stage. Views (called *skeleton frames*) are constructed from the output of visual odometry, which requires a frame rate sufficient for tracking. We require only monocular imagery, constructing structured appearance models from two matched views of the same scene. While Konolige et al. use randomized tree signatures for feature matching, we use a simple variant on SIFT features and local and global feature databases.

### *2.2.2 Graph-Based SLAM*

Storing observations and poses in a constraint graph is now a well-explored technique for localization and mapping. The graph formulation provides a straightforward and flexible representation of the underlying Gaussian Markov random field (GMRF) problem that SLAM attempts to solve. The general framework is described in [18], including a description of a graph relaxation procedure identical to batch bundle adjustment in photogrammetry [19]. Relaxation algorithms for SLAM graphs have received much attention, especially with online operation in mind. Olson et al. [16] suggest a stochastic gradient descent method, and Grisetti et al. [7] review that and related methods for incremental graph optimization.

The system of Eade and Drummond [3] forms a graph where each node is a joint distribution over a local map, and the relative nonlinear constraints between nodes are derived from shared features. The graph is relaxed by imposing cycle constraints using preconditioned gradient descent. The network constructed by PTAM is effectively a graph of relative constraints between keyframes, though the optimization, performed asynchronously to the primary tracking task, acts on individual structure elements.

The view-based mapping of Konolige et al. [12] constructs a reduced graph of poses by consolidating consecutive frames tracked by visual odometry into skeleton frames. Then the constraint graph over skeleton frames is incrementally relaxed using the Toro method [8].

While existing graph-based SLAM methods employ incremental graph optimizers to allow online operation, the number of poses in the graph continues to grow with time. One technique suggested for bounding this growth is that the robot be occasionally virtually "kidnapped," disconnecting its current pose in the graph from previous poses and re-inserting it in using only recent observations [8]. This assumes both that the recent observations are sufficiently accurate to allow relocalization, and that the effective uncertainty of these observations is zero. These assumptions are routinely

violated in practice, especially in visual systems, where the accuracy and uncertainty of relative pose estimates depends heavily on viewpoint and scene structure.

We instead apply probabilistically sound graph reduction methods that limit the complexity of the graph to a linear factor of the complexity of the explored space. Past poses of the robot that are not used for view recognition can be marginalized out of the estimation, and their incident constraints are collapsed back into the graph. The marginalization procedure, equivalent to the update step of the Kalman filter or the variable elimination step of the GraphSLAM algorithm, is described by Konolige in [11].

Marginalization is used to systematically limit graph complexity in Kretzschmar et al. [13], which employs an approximation rather than exact marginalization. The approximate form is used to bound edge connectivity in the graph. Carlevaris–Bianco and Eustice [1] propose an alternative via generic linear constraint node removal. In contrast, we selectively prune edges incident to nodes of high degree, removing their constraints from the GMRF in a conservative manner. The adaptive application of marginalization and edge removal, discussed in Sect. 2.8, is a significant feature of this system.

## 2.3 System Overview

The input to the system is a sequence of images from the camera and a sequence of differential motion estimates, derived from wheel odometry measurements or other differential sensors. We refer to these differential measurements collectively as odometry. The system outputs an incrementally updated estimate of the device's current pose (localization) and estimates of a subset of its previous poses during operation (mapping).

Two high-level components constitute the system: the visual recognition front end and the constraint graph SLAM back end (Fig. 2.1). The front-end processes the video stream, yielding a global appearance database, a set of structured local appearance

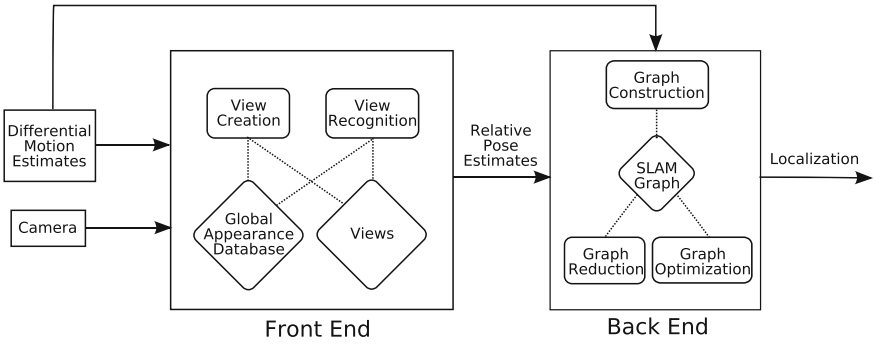


**Fig. 2.1** System structure overview

models called *views*, and a sequence of pose estimates relative to these models. The back end fuses the relative pose and differential motion estimates together in a graph representation, incrementally optimizing and distilling it synchronously with updates. The graph nodes include estimates of current and selected previous poses of the robot.

The front end inherently yields 6DoF relative pose estimates and 3D structured views; the back end can be instantiated in 3DoF for planar robot motion or 6DoF in the general case. This chapter shows results for the 3DoF case (Sect. 2.9).

## 2.4 Viewpoint Invariant Features

The view recognition engine identifies previously constructed appearance models from novel viewpoints based on correspondences between image features. Thus, the image features themselves must have a representation robust to viewpoint and lighting changes. Any efficient feature detector/descriptor combination providing these properties is suitable for this purpose.

We employ Difference-of-Gaussian (DoG) interest points and reduced-dimensionality SIFT [14] descriptors. Our descriptors are computed in a manner similar to 128D SIFT descriptors, but using a $3 \times 3$ spatial grid and four angular histogram bins per cell, instead of the $4 \times 4$ grid and eight angular bins of the standard configuration. We have determined empirically through recognition tasks that these 36D descriptors perform nearly as well as the higher-dimensional variants, but with reduced memory and computational costs.

The detection and description algorithms can be implemented efficiently. Table 2.1 shows the computational viability of our implementation on different platforms.

## 2.5 View Creation

The view creation process extracts from the image sequence a set of structured appearance models. These consist of estimated 3D feature locations and associated appearance descriptors. These sets of features are accessible through a database for use by the view recognition process.

**Table 2.1** Timings for SIFT feature computation

| Task | Core2 (2.4 GHz) | Atom (1.6 GHz) | ARM9 (266 MHz) |
|---|---|---|---|
| Pyramid (ms) | 2.0 | 9.1 | 42 |
| Detector (ms) | 0.6 | 2.0 | 8.9 |
| Descriptors (μs/desc) | 14 | 81 | 219 |

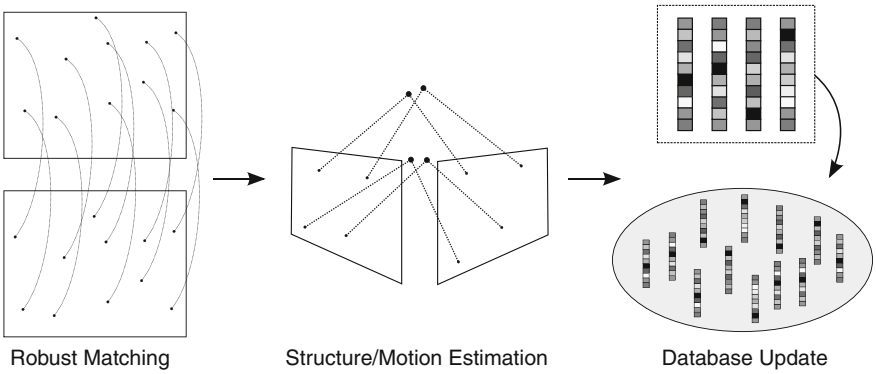


**Fig. 2.2** View creation process

View creation proceeds in three steps (see Fig. 2.2):

1. Robust matching: Correspondences are established between features in two or more temporally local images, while enforcing geometric constraints.
2. Structure estimation: From the feature correspondences, three-dimensional point structure is estimated and stored. The (optional) differential motion estimate is used to determine the common scale of the structure.
3. Database management: The appearance model of the view (comprising a set of feature descriptors) is added to a global database for later recognition.

### *2.5.1 Robust Matching*

The interframe matching procedure for view creation first establishes putative correspondences then partitions these correspondences into inliers (correct matches) and outliers (incorrect matches) using geometric constraints.

Putative correspondences can be generated using only the feature descriptors, or by taking advantage of any differential motion estimates supplied by other sensors, such as wheel odometry. In the first case, each feature in the current image is paired with the feature in a recent older image according to distance in the feature descriptor space using a brute-force or approximate nearest neighbors (ANN) method. In the second case, a motion estimate constrains the search for putative correspondences. The nearest feature in descriptor space that also satisfies the corresponding epipolar constraint is taken as a putative correspondence to the older feature.

Given a set of putative correspondences, geometric constraints are applied iteratively to eliminate outliers. If no prior on camera motion is provided, a starting point for the procedure can be computed using RANSAC [6] and the five-point

algorithm [15]. This yields an estimate of camera motion (up to scale) and a set of inlier correspondences. When a prior differential motion estimate is available, it is a sufficient starting point for iteration.

The iteration proceeds as follows:

1. An error threshold factor $\tilde{r}$ is chosen as a multiple of the desired final acceptance threshold $r$.
2. Inliers are selected by finding all putative correspondences whose matches fall within a threshold distance of the epipolar line, whose descriptor distances are low, and whose depth estimates are positive. The epipolar threshold distance for a feature with scale $s$ is given by $\tilde{r} \cdot s$, modeling larger location uncertainty associated with larger-scale features.
3. The motion estimate is refined by nonlinear maximum-likelihood estimation over the current set of inlier correspondences.
4. The threshold factor $\tilde{r}$ is decreased multiplicatively, and the process is repeated from step 2 until $\tilde{r} \approx r$.

We use this approximation to a standard M-estimator scheme (e.g., iterated reweighted least squares with Tukey weighting) in order to reduce the computational cost on embedded platforms.

### *2.5.2 Structure Estimation*

Given feature correspondences between two views, bundle adjustment [19] is performed over the reprojection objective function to yield joint estimates on structure and camera motion. The scale is left unconstrained by the feature correspondences, so the gauge freedom is eliminated by fixing the camera translation to unit magnitude while performing the optimization. The scale is assigned to the view using the differential odometry between the two views used for estimation. Further views can be added to the optimization either at the point of view creation or upon later observation. In this case of upgrading the structure, the camera translation magnitude is constrained only between the first two views, and all six degrees of freedom vary among the others. The previously computed parameter values are used as a starting point in the new, larger optimization.

### *2.5.3 Database Management*

A global appearance database is maintained to aid view recognition. When a new view is created, its appearance model is added to this database.

The global database could take one of many forms, depending on the desired appearance model representation. We describe a simple but effective approach here.

The database contains descriptors for features in all views in a collection of kd-trees for efficient ANN searches. The time required to add new views to the global database is bounded: upon view creation, all descriptors in the view are added to the current kd-tree, which is then rebalanced. If the number of descriptors in the tree exceeds a predetermined constant bound, a new tree is added to the collection and becomes the current tree. ANN searches of the forest are described below in Sect. 2.6.

In addition to the global database update, a local appearance model is also constructed for each view. The local model supports ANN searches over only the descriptors present in the view, and is queried for the second stage of view recognition.

## 2.6 View Recognition

The view recognition process yields relative pose estimates between single images and existing views. The recognition approach is hierarchical, first performing appearance matching in a global database, and then applying structure constraints at the view-local level.

The input to the view recognition algorithm is a set of viewpoint invariant features extracted from an image, and a global appearance model as described in Sect. 2.5.3. The output is zero, one, or multiple relative pose estimates to existing views.

The recognition method proceeds as follows (see Fig. 2.3).

1. Features in the query image are looked up in the global appearance model database.
2. The results of the database lookup are used to rank potential views by visual similarity, and the $m$ top-ranked views are chosen as candidates (we use $m = 3$ throughout).
3. For each candidate view, correspondences are established between query features and the features in the view.
4. Geometric constraints are applied to these correspondences, using reprojection constraints and the estimated view structure to reject outliers. This yields a rough relative pose estimate.
5. The relative pose and structure estimates are refined using by optimizing over the inlier correspondences and internal correspondences of the view, yielding maximum-likelihood relative pose with covariance.
6. The view's stored structure estimate is optionally updated using the optimization results.

### *2.6.1 Recognition Candidate Selection*

The top $k$ nearest neighbors in the global database for each query feature are determined using ANN search (typically $k = 2$). Then, the putative matches are grouped

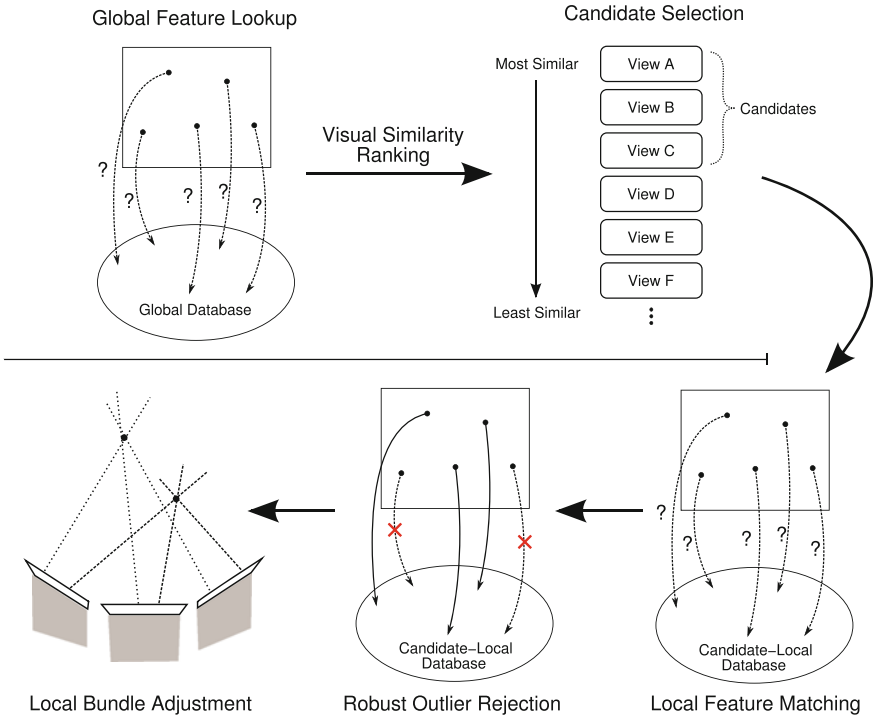


**Fig. 2.3** View recognition process

by view. At this point, crude structure constraints can be applied, using a Hough transform or RANSAC to enforce a loose similarity, affine, homography, or reprojection transformation. The views are then ranked by the number of matches satisfying the constraints. The $m$ highest ranked views are kept as candidates.

### *2.6.2 Robust Matching and Pose Estimation*

For each of the $m$ candidate views chosen by the appearance matching stage, the query features are matched to the view's features using the local view appearance model. Each view feature has an associated three-dimensional structure estimate, allowing the three-point pose algorithm [6] to be applied within a standard RANSAC hypothesize-and-test framework. If enough inliers result from this process, they are passed to the pose estimation stage, along with the relative pose estimate given by the three-point algorithm.

The pose estimation stage takes correspondences between query features and view features, and computes the relative camera pose between the query image and the view's base coordinate frame (the first image of the view pair). The relative pose

estimate is represented by mean and covariance in the Lie group SE(3) of rigid 6DoF transformations. The covariance is represented by a quadratic form in the tangent space se(3).

The maximum-likelihood estimation is performed using Levenberg–Marquardt iteration. The camera motion between the existing view frames is assumed fixed and known, and the feature structure estimates and relative pose to the novel viewpoint are permitted to vary. The data matrix at the point of convergence is taken as the information matrix (inverse covariance) of the optimum, as per the Cramer–Rao lower bound. The structure parameters are marginalized out of this representation, and the resulting $6 \times 6$ matrix is inverted to yield an estimate of the covariance on the relative pose parameters. If the information matrix is singular or poorly conditioned, the pose estimate is under-constrained, and the view recognition is discarded.

## 2.7 Graph Construction and Optimization

The SLAM back end encodes view observations and robot motion in a graph representation of a Gaussian Markov random field (GMRF). The graph is constructed as the robot moves and processes video frames. The graph is continuously and incrementally optimized to improve the state estimate of view and robot poses.

### *2.7.1 Graph Representation*

The SLAM graph [18] consists of nodes and directed edges between pairs of nodes. Each node represents the pose of the robot at a certain time. Edges encode constraints between nodes, arising from differential motion estimates (odometry), view observations, and combinations thereof. All poses and transformations are parametrized in the Lie group SE(2) (for robot pose in 2D space) or SE(3) (for robot pose in 3D space), and any covariances or information matrices are expressed in the respective tangent spaces.

Each node stores the estimated pose of the robot at a certain time. The pose describes the coordinate transformation from the common global frame to the frame of the robot at the specified time. Nodes are created for every timestep when a view is recognized or created. Nodes corresponding to the robot pose at view coordinate frames are called view nodes and nodes corresponding to the robot pose at any other times are called pose nodes.

Each edge stores a rigid transformation estimate, with covariance, describing a constraint between its source and destination endpoint. The constraint means and covariances are represented in the Lie group and algebra, respectively. Edges that encode only differential motion constraints (from odometry) are called motion edges, and connect temporally consecutive nodes. Edges that encode relative pose estimates

from view recognitions are called observation edges. Edges that are formed by combining other edges (described below in Sect. 2.8.2) are called hybrid edges.

### 2.7.2 Graph Construction

After each image is processed by the front end, the graph representation is updated:

- A new pose node is added for the current pose, and the new node is connected to the preceding pose node by a motion edge, encoding the accumulated differential motion estimate between the two poses.
- If an existing view has been observed, an observation edge is created from the observed view node to the pose node, encoding the observation constraint. When the back end is operating in SE(2), the relative pose estimate (in SE(3)) is first projected into SE(2) before creating the observation edge.
- If a new view has been created, one of the recent pose nodes corresponding to the view is promoted to a view node.

### 2.7.3 Incremental Optimization

The graph flexibly represents the GMRF corresponding to the SLAM estimation problem. The negative log-likelihood of the parameter estimates (encoded by the nodes) is the sum residuals of the edges. Denote the edge set by $E = \{e_i\}$. For an edge $e \in E$, the source and destination are given by $s(e)$ and $d(e)$ respectively. The edge's constraint mean is denoted by $\mu(e)$ and the covariance by $\Sigma(e)$. Then the negative log-likelihood $-L$ of the graph (up to a constant offset) is given in terms of the residuals $v_i$ by

$$v_i \equiv \mu(e_i) \cdot s(e_i) \cdot d(e_i)^{-1} \tag{2.1}$$

$$-L = \sum_i v_i^T \left(\Sigma(e_i)^{-1}\right) v_i \tag{2.2}$$

When the node pose estimates better satisfy the constraints encoded in the edges, the negative log-likelihood $-L$ is lower. Graph optimization increases the likelihood of the GMRF parameters by minimizing the negative log-likelihood as function of the node parameters.

Because computation time must be bounded and the graph is continually growing and changing, any feasible graph optimization technique must be incremental. Several methods are described in, e.g., [7]. Any general method for incremental nonlinear optimization can be applied successfully to the graph.

We employ spanning tree and blob-based optimizations, which are run for a fixed number of iterations at each time step following the graph update.

## 2.8 Graph Complexity Reduction

### *2.8.1 Complexity Growth*

The SLAM graph grows every time a view is created or observed. Even when the robot stays within a bounded space, the views there are observed repeatedly, adding pose nodes and edges to the graph and thus increasing the complexity with time. The storage requirements and graph optimization costs grow with the graph complexity, so in order to control these costs, the graph complexity must be bounded.

The view nodes correspond to elements of the front end relative to which pose estimates can be computed. Further, the spatial density of view nodes is bounded by the front end (as existing views will be recognized from nearby viewpoints), so operation within a fixed spatial region implies a bounded number of view nodes. The pose nodes, on the other hand, represent past robot poses that are not directly useful in subsequent operation, except as a data structure for encoding constraints on other nodes. The number of pose nodes grows with the number of observations, instead of with the number of views. The graph complexity can be bounded by removing pose nodes and limiting node connectivity to keep the complexity of the graph linear in the number of views and thus linear in the amount of space explored.

### *2.8.2 Pose Node Marginalization*

The graph represents a GMRF over past poses of the robot, so nodes can be removed in statistically consistent manner by marginalizing out the corresponding pose variables from the GMRF state. The graph directly encodes the Markov property of the system: a node is conditionally independent of all nodes to which it is not directly connected. Thus marginalizing out a node's state involves only the Markov blanket of the node (all of the nodes within one hop in the graph). Further, because the marginal distributions of a Gaussian are also Gaussians, the graph resulting from the removal exactly encodes the appropriate Gaussian distribution over the remaining variables [11].

Removing a node by marginalization induces pairwise constraints between all pairs of nodes connected to the removed node. If a constraint (edge) already exists between such a pair, the new constraint is combined with the existing constraint by multiplication of their Gaussians. A few operations on edges are needed to define the node marginalization procedure:

#### 2.8.2.1 Edge Reversal

An edge $e$ represents an uncertain rigid transformation between its two endpoint nodes, given by a mean and covariance $(\mu, \Sigma)$ in the appropriate Lie group and Lie

algebra respectively. The adjoint operator in a Lie group allows elements of the Lie algebra to be moved from the right tangent space of a transformation to the left. Thus, the reversed edge $e^{-1}$, pointing in the opposite direction in the graph but encoding the same transformation constraint, is given by

$$e^{-1} = \left(\mu^{-1}, \text{Adj}\left[\mu^{-1}\right] \cdot \Sigma \cdot \text{Adj}\left[\mu^{-1}\right]^T\right) \tag{2.3}$$

#### 2.8.2.2 Edge Composition

Given an edge $e_0 = (\mu_0, \Sigma_0)$ from node $a$ to node $b$ and an edge $e_1 = (\mu_1, \Sigma_1)$ from node $b$ to node $c$, the two edges may be composed into one edge from $a$ to $c$ by composing the uncertain transformations, as in a Kalman filter motion update:

$$e_1 \cdot e_0 = \left(\mu_1 \cdot \mu_0, \Sigma_1 + \text{Adj}\left[\mu_1\right] \cdot \Sigma_0 \cdot \text{Adj}\left[\mu_1\right]^T\right) \tag{2.4}$$

#### 2.8.2.3 Edge Combination

Given two edges $e_0 = (\mu_0, \Sigma_0)$ and $e_1 = (\mu_1, \Sigma_1)$ connecting the same two nodes in the same direction, their constraints may be combined by multiplying the associated Gaussian distributions together to yield the resulting Gaussian. Because the exponential map from the tangent space to the transformation manifold is nonlinear, the combination procedure for the mean is iterative. The combined covariance $\Sigma_C$ is computed by summing the information of the two edges:

$$\Sigma_C = \left(\Sigma_0^{-1} + \Sigma_1^{-1}\right)^{-1} \tag{2.5}$$

Let the initial estimate of the combined mean be the first edge's mean:

$$\mu_C^0 = \mu_0 \tag{2.6}$$

Then the combined transformation is updated by taking the information-weighted average between the two transformations and exponentiating the correction into the Lie group:

$$v_j^i = \ln\left(\mu_C^i \cdot \mu_j^{-1}\right), j \in \{0, 1\} \tag{2.7}$$

$$\delta_i = \Sigma_C \cdot \left(\Sigma_0^{-1} \cdot v_0^i + \Sigma_1^{-1} \cdot v_1^i\right) \tag{2.8}$$

$$\mu_C^{i+1} = \exp\left(\delta_i\right) \cdot \mu_C^i \tag{2.9}$$

This update is iterated until convergence (usually three or four iterations), yielding the combined edge:

$$e_C = \left(\mu_C^k, \Sigma_C\right) \tag{2.10}$$

*Node Removal*

Consider a node $n_r$ to be removed by marginalization, with incident edges $E_r = \{e_0, \ldots, e_m\}$. Each pair of such edges $(e_i, e_j)$ is composed into $e_{(i,j)}$ according to the following cases:

$$e_{(i,j)} = \begin{cases} e_i \cdot e_j & s(e_i) = d(e_j) = n_r \\ e_i \cdot e_j^{-1} & s(e_i) = s(e_j) = n_r \\ e_i^{-1} \cdot e_j & d(e_i) = d(e_j) = n_r \\ e_j \cdot e_i & d(e_i) = s(e_j) = n_r \end{cases} \tag{2.11}$$

The resulting composed edge is added to the graph between the two incident nodes that are not $n_r$. If such an edge already exists, the edges are combined, reversing the composed edge if necessary. Finally, all incident edges $E_r$ are deleted from the graph along with the node $n_r$. An example is shown in Fig. 2.4.

### 2.8.3 Edge Pruning

While the node marginalization procedure always decreases the number of graph nodes and attempts to decrease the number of edges, it might fail to bound the degrees of nodes and thus the complexity of the graph. Indeed, marginalizing out all pose nodes results in a completely connected graph over view nodes, with edge cardinality quadratic in the number of views.

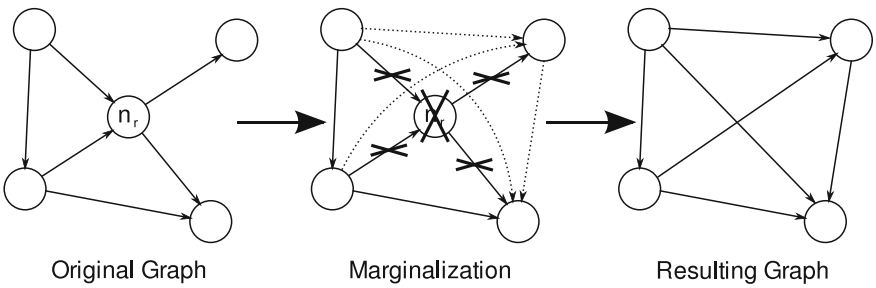


**Fig. 2.4** Graph reduction by marginalizing out a node. In this example, the number of edges in the graph is unchanged

To limit the edge complexity of the graph, edges need to be heuristically pruned during operation. Removing an edge from the graph is equivalent to discarding the information represented by the edge, as though the observation or measurement had never been made.

One simple approach to limiting the number of edges is to maintain a priority queue of nodes with degrees exceeding a fixed, predetermined bound. This queue needs to be updated only when edges are added to the graph (measurements or node removals). Edges are removed from each node in the queue until no node degrees exceed the bound.

The heuristic operates as follows: the edges of a high-degree node $n$ are examined one at a time. If the opposite endpoint through edge $e$ is not connected to $n$ through a path that excludes $e$, with length under a predetermined bound, then $e$ is not eligible for removal, as the graph would be potentially disconnected. The eligible edge with the least residual is deleted. Of the edges incident to $n$, such an edge is in least disagreement with the current state of the graph, and thus its removal should least affect the graph optimum.

This simple, greedy heuristic does not consider the collective effect of removing multiple edges in series. Nonetheless, our evaluation shows that it performs adequately.

## 2.9 Evaluation

We use three indoor sequences (SEQ1, SEQ2, SEQ3) to evaluate the performance of the system and the effects of graph complexity reduction. SEQ1 and SEQ2 were collected using an Evolution Robotics Scorpion, and SEQ3 with an iRobot Roomba. In each instance, the robot was equipped with a web camera and wheel odometry. Fiducials placed in the environment were observed by a SICK NAV200 laser range finder mounted on the robot to provide ground truth.

The sequence parameters are described in Table 2.2, and example images are shown in Fig. 2.5. The ground truth trajectories are shown in Fig. 2.6.

**Table 2.2** Test sequences

| | SEQ1 | SEQ2 | SEQ3 |
|---|---|---|---|
| Environment | Warehouse | Home | Office |
| Frame rate (Hz) | 1.5 | 1.5 | 3.0 |
| Timesteps | 1,035 | 1,822 | 3,896 |
| Extent (m) | 24 × 12 | 20 × 9 | 19 × 10 |
| Views created | 41 | 103 | 140 |

**Fig. 2.5** Example images from the SEQ2 (*left*) and SEQ3 (*right*). The reflector beacons are NAV200 fiducials used for ground truth estimation

### *2.9.1 Metrics*

We measure both the accuracy of the incrementally estimated trajectory and of the final view map. The view map is the set of poses of view nodes in the graph at the end of the run, including incremental optimization but without any post-processing.

Comparing the trajectory to the reference reflects localization accuracy during the run. Comparing the map to the appropriate subset of the reference indicates how well the system can be expected to localize in the same environment given subsequent operation. Though the latter metric is more common, and generally shows smaller errors compared to the reference, it does not necessarily reflect how useful the localization is during online operation.

The estimated and ground truth trajectories are compared by first finding the rigid transformation between them that minimizes sum-squared position error, using RANSAC and least squares. The view map corresponds to a subset of the total robot trajectory, so the same method is used to compute the view map error over that subset.

### *2.9.2 Results*

Figure 2.7 shows a portion of the graph computed for SEQ2 with and without reduction. The node and edge density is significantly lower in the latter.

Table 2.3 shows error metrics and graph complexity for full and reduced graphs. In the "Full" columns, the graph is heavily optimized and no nodes or edges are removed. The "Reduced" columns show the same metrics when the number of pose nodes is bounded by the number of views plus ten, and the maximum permitted node degree is eight. The graph complexity is greatly reduced with little or no loss of localization accuracy. As expected, the error of the view map is smaller than that of the causally estimated trajectory consisting of the best estimate at each timestep, as the map has incorporated all the information up to the end of the run.

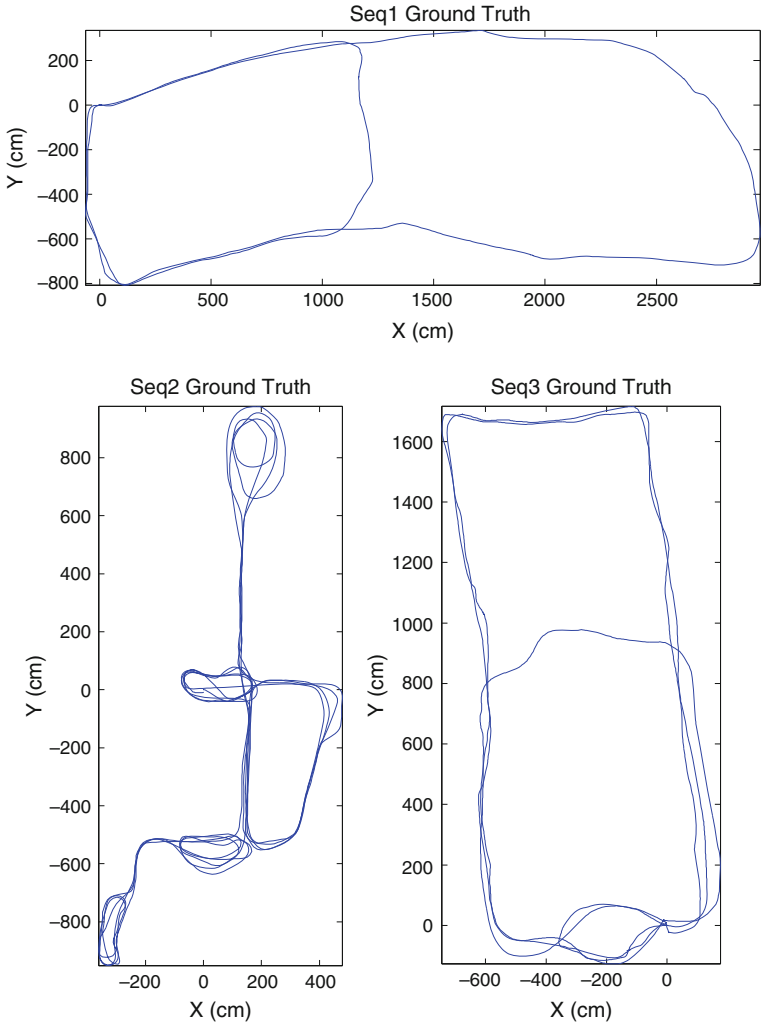


**Fig. 2.6** Ground truth trajectories for the test sequences

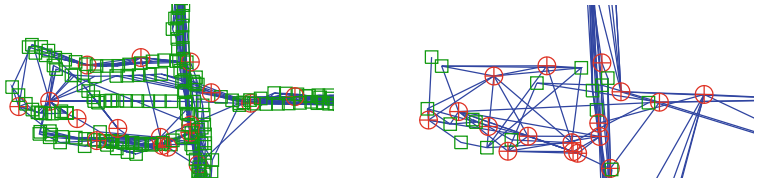

**Fig. 2.7** Detail from middle of full and reduced graphs for SEQ2. View nodes are (*red*) *circles*, pose nodes are (*green*) *squares*, and edges are (*blue*) *lines*. Note the reduced density on the *right*

**Table 2.3** Metrics for full and reduced complexity graphs

| Error (cm) | SEQ1 | | SEQ2 | | SEQ3 | |
|---|---|---|---|---|---|---|
| Odom. RMS | 281 | | 331 | | 469 | |
| Odom. max | 773 | | 667 | | 852 | |
| Error (cm) | Full | Reduced | Full | Reduced | Full | Reduced |
| Traj. RMS | 45 | 44 | 23 | 28 | 59 | 59 |
| Traj. max | 109 | 105 | 81 | 74 | 138 | 149 |
| Map RMS | 24 | 18 | 21 | 20 | 43 | 47 |
| Map max | 41 | 32 | 47 | 46 | 98 | 103 |
| Number of nodes | 709 | 92 | 897 | 216 | 2,491 | 290 |
| Number of edges | 1,471 | 155 | 1,810 | 414 | 5,154 | 501 |

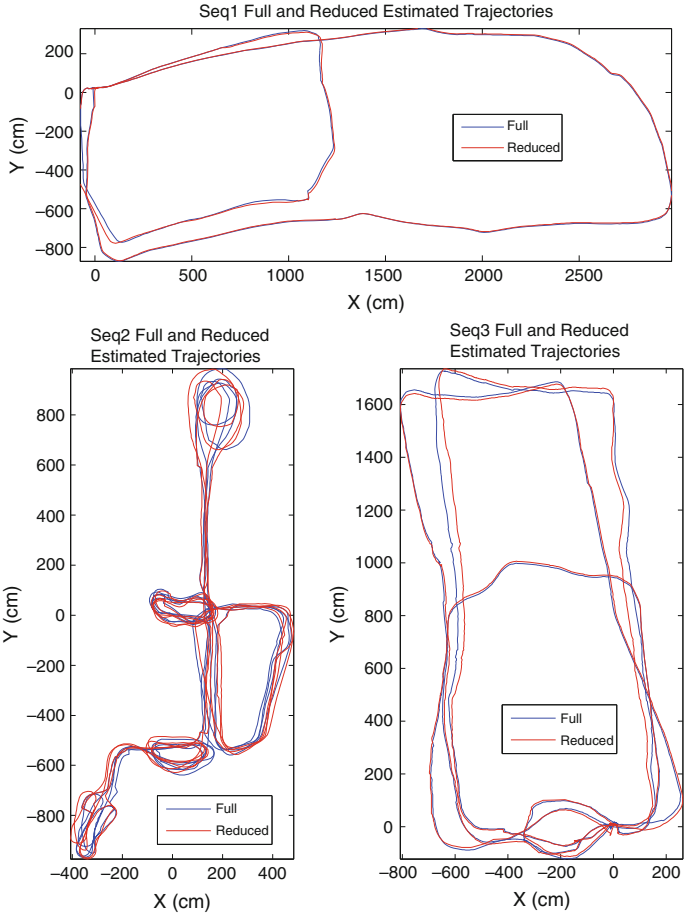


**Fig. 2.8** Causally estimated trajectories: graph reduction yields results similar to those computed with full-complexity graphs

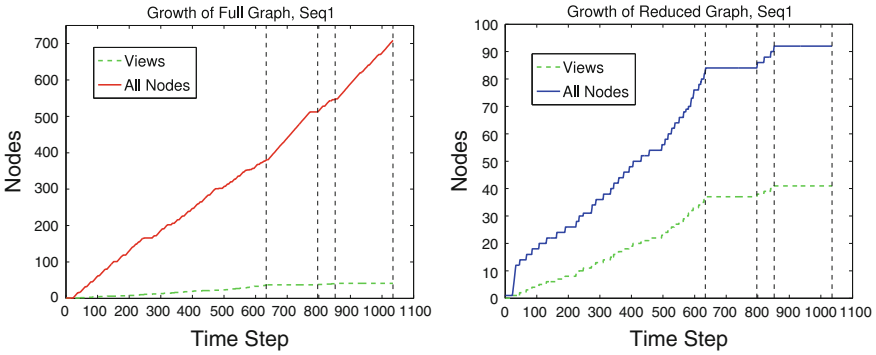


**Fig. 2.9** Graph complexity over time for SEQ1, with and without reduction. The two regions bounded by *vertical dotted lines* are periods of revisitation, during which views are reobserved rather than created. The reduced graph complexity remains constant unless new views are created. Note the difference in vertical scale

Figure 2.8 overlays the trajectories computed using the heavily optimized, fully complex graphs with those computed using reduced graphs. The qualitative similarity of the results reflects the quantitative similarity of the errors to ground truth. The deviation between the two traversals of the large loop in SEQ3 occurs because the robot traverses in opposite directions, so views are not reobserved.

Figure 2.9 shows the growth in number of graph nodes over time for SEQ1, with and without reduction. Reduction keeps the complexity linear with number of views rather than time.

## 2.10 Conclusion

This view-based monocular SLAM system minimizes the computation required for vision-based processing and actively manages the complexity of the SLAM graph to permit operation on constrained computational platforms. Our results show that the complexity reduction methods significantly limit graph node and edge cardinality, while only negligibly affecting localization accuracy. The system uses inexpensive sensors, has low computational requirements, and high reliability, all of which are ideal for low-cost robotic applications.

## References

1. Carlevaris-Bianco N, Eustice RM (2013) Long-term simultaneous localization and mapping with generic linear constraint node removal. In: 2013 IEEE/RSJ international conference on intelligent robots and systems (IROS), IEEE, pp 1034–1041

2. Cummins M, Newman P (2008) Accelerated appearance-only SLAM. In: Proceedings of the IEEE international conference on robotics and automation (ICRA'08), Pasadena
3. Eade E, Drummond T (2007) Monocular slam as a graph of coalesced observations. In: Proceedings of the 11th IEEE international conference on computer vision (ICCV'07), Rio de Janeiro, Brazil
4. Eade E, Drummond T (2008) Unified loop closing and recovery for real time monocular slam. In: Proceedings of the British machine vision conference (BMVC'08), Leeds, BMVA, pp 53–62
5. Eade E, Fong P, Munich ME (2010) Monocular graph slam with complexity reduction. In: IEEE/RSJ international conference on intelligent robots and systems (IROS), pp 3017–3024
6. Fischler MA, Bolles RC (1981) Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. Comm ACM 24(6):381–395
7. Grisetti G, Rizzini DL, Stachniss C, Olson E, Burgard W (2008) Online constraint network optimization for efficient maximum likelihood map learning. In: Proceedings of the 2008 IEEE international conference on robotics and automation (ICRA'08), Pasadena, pp 1880–1885
8. Grisetti G, Stachniss C, Burgard W (2009) Nonlinear constraint network optimization for efficient map learning. Trans Intell Transp Sys 10(3):428–439
9. Karlsson N, di Bernardo E, Ostrowski J, Goncalves L, Pirjanian P, Munich ME (2005) The vslam algorithm for robust localization and mapping. In: Proceedings of the 2005 IEEE international conference on robotics and automation (ICRA'05), Barcelona, Spain, pp 24–29
10. Klein G, Murray D (2008) Improving the agility of keyframe-based SLAM. In: Proceedings of the 10th European conference on computer vision (ECCV'08), Marseille, pp 802–815
11. Konolige K (2005) Slam via variable reduction from constraint maps. In: Proceedings of the 2005 IEEE international conference on robotics and automation (ICRA'05), Barcelona, Spain, pp 667–672
12. Konolige K, Bowman J, Chen JD, Mihelich P, Calonder M, Lepetit V, Fua P (2009) View-based maps. In: Proceedings of robotics science and systems, Seattle
13. Kretzschmar H, Grisetti G, Stachniss C (2010) Lifelong map learning for graph-based slam in static environments. Künstliche Intelligenz
14. Lowe D (2004) Distinctive image features from scale-invariant keypoints. Int J Comput Vis 60(2):91–100
15. Nistér D (2004) An efficient solution to the five-point relative pose problem. IEEE Trans Pattern Anal Mach Intell (PAMI) 26(6):756–777
16. Olson E, Leonard J, Teller S (2007) Spatially-adaptive learning rates for online incremental slam. In: Proceedings of robotics science and systems, Atlanta
17. Philbin J, Chum O, Isard M, Sivic J, Zisserman A (2007) Object retrieval with large vocabularies and fast spatial matching. In: Proceedings of the IEEE international conference on computer vision and pattern recognition (CVPR'07), IEEE Computer Society, Minneapolis, pp 1–8
18. Thrun S, Montemerlo M (2006) The graph slam algorithm with applications to large-scale mapping of urban structures. Int J Robot Res 25(5–6):403–429
19. Triggs B, McLauchlan P, Hartley R, Fitzgibbon A (2000) Bundle adjustment—a modern synthesis. In: Triggs B, Zisserman A, Szeliski R (eds) Vision algorithms: theory and practice, of Lecture Notes in Computer Science, vol 1883. Springer, pp 298–372
20. Williams B, Klein G, Reid I (2007) Real-time SLAM relocalisation. In: Proceedings of the 11th IEEE international conference computer vision

http://www.springer.com/978-3-319-09386-4

Advances in Embedded Computer Vision
Kisačanin, B.; Gelautz, M. (Eds.)
2014, XVI, 287 p. 147 illus., 116 illus. in color.,
Hardcover
ISBN: 978-3-319-09386-4