ANNUAL REVIEWS

Annual Review of Control, Robotics, and Autonomous Systems

Factor Graphs: Exploiting Structure in Robotics

Frank Dellaert[1,2]

[1]School of Interactive Computing, Georgia Institute of Technology, Atlanta, Georgia 30332, USA; email: frank.dellaert@gatech.edu

[2]Google AI, Mountain View, California 94043, USA

Annu. Rev. Control Robot. Auton. Syst. 2021. 4:141–66

First published as a Review in Advance on January 12, 2021

The *Annual Review of Control, Robotics, and Autonomous Systems* is online at control.annualreviews.org

https://doi.org/10.1146/annurev-control-061520-010504

Keywords

simultaneous localization and mapping, SLAM, estimation, motion planning, graphical models, factor graphs

Abstract

Many estimation, planning, and optimal control problems in robotics have an optimization problem at their core. In most of these optimization problems, the objective to be maximized or minimized is composed of many different factors or terms that are local in nature—that is, they depend only on a small subset of the variables. A particularly insightful way of modeling this locality structure is to use the concept of factor graphs, a bipartite graphical model in which factors represent functions on subsets of variables. Factor graphs can represent a wide variety of problems across robotics, expose opportunities to improve computational performance, and are beneficial in designing and thinking about how to model a problem, even aside from performance considerations. I discuss each of these three aspects in detail and review several state-of-the-art robotics applications in which factor graphs have been used with great success.

1. INTRODUCTION

SLAM: simultaneous localization and mapping

Factor graph: a bipartite graph of variables and factors, representing functions on subsets of those variables

Elimination algorithm: an algorithm that recursively eliminates variables from a problem; it was first developed by Gauss for orbital mechanics but underlies inference in many disparate domains

Many estimation problems in robotics have an optimization problem at their core. For example, **Figure 1** illustrates two common variants of simultaneous localization and mapping (SLAM): one that builds a map of an indoor environment using a sequence of 2D lidar measurements, and one where an autonomous vehicle drives in an urban environment while building a sparse 3D map using both visual and inertial [inertial measurement unit (IMU)] measurements. In these and many other estimation problems, we try to maximize the posterior probability of a set of unknown variables given a set of measurements.

Likewise, in optimal control we try to act optimally by maximizing a performance index or, conversely, minimizing a penalty function. Even in classical planning, we are trying to find an assignment to a set of discrete variables that minimizes the plan length or optimizes for some other desirable property of the plan.

In most of these optimization problems, the objective to be maximized or minimized is composed of many different factors or terms that are local in nature—that is, they depend only on a small subset of the variables. For example, a GPS measurement applies only to the pose of a vehicle at a particular time, and a single IMU measurement relates two vehicle states at adjacent times.

A particularly insightful way of modeling this locality structure is to use the concept of factor graphs (1). Factor graphs are a class of graphical models in which there are variables and factors. The variables represent unknown quantities in the problem, and the factors represent functions on subsets of the variables. Edges in the factor graph between a factor and a set of variables indicate that the factor depends only on those variables.

Using factor graphs when designing algorithms for robotics applications has three main advantages. First, because many optimization problems in robotics have the locality property, factor graphs can model a wide variety of problems across robotics domains, such as tracking, navigation, and mapping; classical planning; reinforcement learning and optimal control; motion planning; and simultaneous trajectory estimation and planning (STEAP).

Second, by clearly exposing the structure of the problem, thinking in factor graphs surfaces many opportunities for improving the performance of key algorithms. Many classical algorithms can be seen as applying the elimination algorithm to a particular type of factor graph, but this algorithm is optimal only for a small class of problems. In many applications, domain-specific knowledge about the structure of the problem can improve the running time of inference by orders

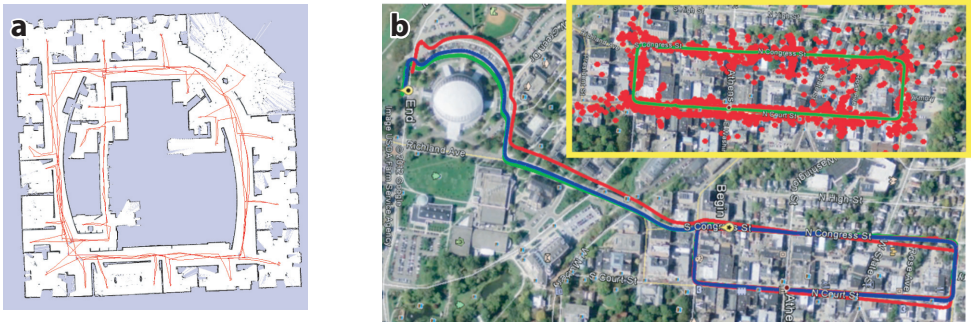

Figure 1

(*a*) An example of mapping with a 2D lidar as a pose graph optimization problem. (*b*) An example of visual–inertial navigation in an urban setting.

of magnitude. Similarly, well-known algorithmic ideas from linear algebra can be generalized to factor graphs—leading, for example, to incremental inference algorithms.

Third, even aside from performance considerations, factor graphs are useful when designing and thinking about how to model a problem, providing a common language in which to express ideas to collaborators and users of a particular algorithm. After working with factor graphs for a while, one starts to identify factor types as a particularly useful unit of design. A factor type specifies how many variables a factor is connected to as well as the semantics associated with the function that is being computed. For example, in tracking, a factor can indicate the likelihood of a particular target pose, given a 2D measurement in an image. Other factors can be associated with different measurements, prior information, probabilistic motion models, and so on. Frequently, in designing new algorithms or algorithm pipelines, thinking about novel factor types is a key element of design.

Maximum a posteriori (MAP) estimate: an estimate that maximizes the posterior probability of unknown variables given measurements and prior knowledge

Below, I discuss each of these three aspects in greater detail, starting with a tour of how factor graphs can encode a variety of different problems within robotics. I then move on to the actual computation to be done with factor graphs and how insight into those factor graphs can lead to algorithmic innovations and dramatic speed improvements. Finally, I review several state-of-the-art applications in which factor graphs have been used with great success.

2. A TOUR OF FACTOR GRAPHS

In this section, I give a formal definition of factor graphs and present several examples from robotics that highlight the different computational structures underlying these applications.

Formally, a factor graph is a bipartite graph $F = (\mathcal{U}, \mathcal{V}, \mathcal{E})$ with two types of nodes: factors $\phi_i \in \mathcal{U}$ and variables $x_j \in \mathcal{V}$, with edges $e_{ij} \in \mathcal{E}$ between them. We denote the set of variables adjacent to a factor ϕ_i as X_i. With these definitions, a factor graph F defines the factorization of a global function $\phi(X)$ as

$$\phi(X) = \prod_i \phi_i(X_i). \qquad 1.$$

In other words, the independence relationships are encoded by the edges e_{ij} of the factor graph, where each factor ϕ_i is a function of only the variables X_i in its adjacency set.

2.1. Tracking

In a tracking application, we are interested in the maximum a posteriori (MAP) estimate of the trajectory X given the measurements Z—that is, we want to maximize

$$p(X|Z) \propto p(x_1)l(x_1; z_1) \prod_{i>1} p(x_i|x_{i-1})l(x_i; z_i), \qquad 2.$$

where the integers i index a set of discrete times t_i at which the measurements were observed. Above, I applied Bayes' law and have factored the posterior density $p(X|Z)$ into a set of unary likelihood factors $\phi(x_i) \propto l(x_i; z_i)$ and binary motion model factors $\phi(x_i, x_{i-1}) \propto p(x_i|x_{i-1})$. The resulting factor graph is shown in **Figure 2*a***. Note that there is an extra unary factor on x_1 corresponding to the prior $p(x_1)$. Technically, maximizing $p(X|Z)$ corresponds to trajectory optimization or smoothing, but incremental inference or tracking is always easy to derive from the smoothing formulation, as discussed below. When the prior, motion model, and measurement are all linear with Gaussian additive noise, doing MAP inference in this factor graph is known as Kalman smoothing, and the tracking version is the Kalman filter.

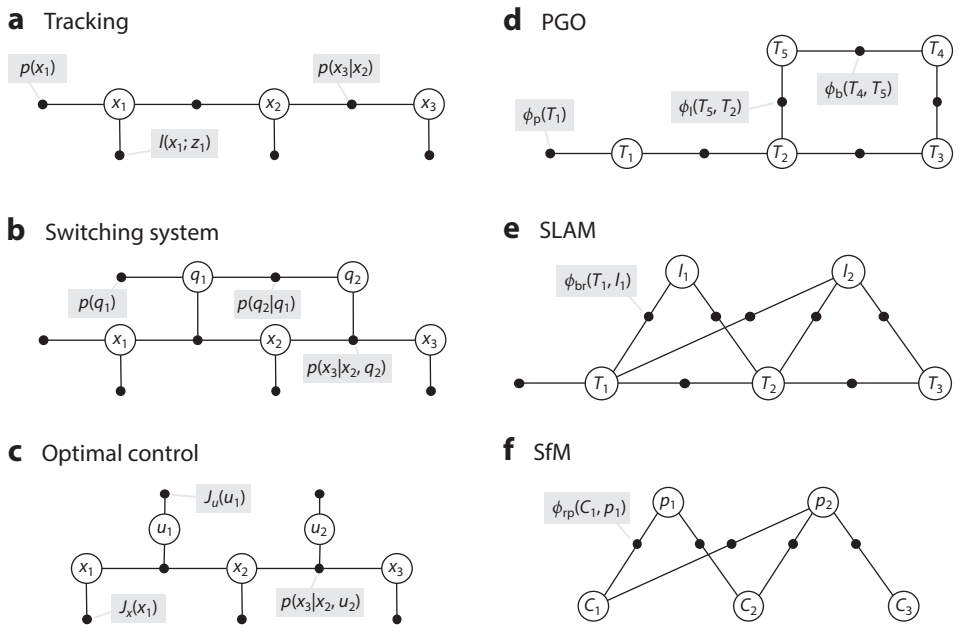

Figure 2

A tour of factor graphs. Abbreviations: PGO, pose graph optimization; SfM, structure from motion; SLAM, simultaneous localization and mapping.

2.2. Switching System

We can add a second Markov chain of discrete variables q_i that switch between different motion models, replacing $p(x_i|x_{i-1})$ with $p(x_i|x_{i-1}, q_{i-1})$. This is shown in **Figure 2*b***, where this new motion model shows up as ternary factors. Note that inference is now more involved, as the set of possible label assignments is exponentially large. Under the same restrictive assumptions underlying the Kalman smoother, the resulting model is known as a switching linear dynamical system.

2.3. Optimal Control

The examples above are estimation problems, but factors can also be used to encode the desirability of a particular variable or state. This is what happens in planning and/or optimal control and is illustrated in **Figure 2*c***. The (stochastic) system dynamics are encoded in the densities $p(x_i|x_{i-1}, u_{i-1})$, now conditioned on continuous control variables u_{i-1}. We now optimize for a future trajectory, maximizing the desirability of a particular outcome as encoded in the unary factors $J_x(x_i)$ and $J_u(u_i)$ on the states X and controls U, respectively. With linear dynamics and quadratic penalties on states and controls, optimizing for the controls yields the finite-horizon linear quadratic regulator, but the same graph structure generalizes to nonlinear dynamics and objectives.

Markov chain: a chain-like graph structure where the state at a given time depends only on the previous state

PGO: pose graph optimization

2.4. Pose Graph Optimization

Pose graph optimization (PGO) is the problem of estimating robot poses T_i over time given relative pose measurements, where both poses and measurements are elements of the special Euclidean manifolds $SE(2)$ or $SE(3)$. An example is an autonomous vehicle equipped with a 3D lidar

scanner. In this case, the relative pose measurements derive from registering lidar scans at successive times. Alternatively, pairwise measurements can also be obtained by tracking features over time in a video stream; in this case, we often use the term visual odometry. When adding inertial sensors (accelerometers and gyroscopes) into the mix, we speak of visual–inertial odometry (VIO). All of these cases can be represented using the factor graph in **Figure 2*d***, where the binary factors $\phi_b(T_i, T_i)$ derive from the relative pose measurements. In addition to these successive measurements, PGO allows for loop closures when we detect that the vehicle has come back to the same place. An example in the figure is the loop closure $\phi_l(T_5, T_2)$.

SfM: structure from motion

PGO is a special case of synchronization problems that include rotation averaging. There, instead of poses T_i, we are looking to find the optimal absolute rotation matrices $R_i \in SO(3)$ given relative orientations. The latter are of special interest because they are frequently used to initialize SLAM or structure from motion (SfM) problems. In SfM, for example, relative orientations can be obtained by estimating the fundamental matrix between pairs of cameras.

2.5. Simultaneous Localization and Mapping

In the estimation problems above, we have one type of unknown variable: the state of the target or the robot. When we also want to estimate knowledge about the environment, we can introduce new variables related to landmarks in the environment, as in the SLAM problem. For example, when we have bearing–range measurements to landmarks, the corresponding SLAM problem in **Figure 2*e*** has new variables l_j and additional binary factors $\phi_{br}(T_i, l_j)$, one for each observation of a specific landmark l_j from a particular pose T_i. The special case in which a camera moves in a continuous trajectory and measurements are 2D projections of 3D landmarks is typically referred to as visual SLAM.

2.6. Structure from Motion

The SfM problem is a related problem, originating in computer vision, where one tries to reconstruct a 3D model of the environment from several photos, typically shot by different people at different times with different cameras. Hence, as shown in **Figure 2*f***, the structure of the problem is very similar to visual SLAM, but without the pairwise relative pose factors. In addition, each camera C_i now has both an unknown pose $T_i \in SE(3)$ and an unknown set of camera calibration parameters. The landmarks are typically 3D points p_j and relate to cameras via binary reprojection factors $\phi_{rp}(C_i, p_j)$.

2.7. A Larger Example

The graph for a larger SLAM problem is shown in **Figure 3**. It was created by simulating a 2D robot moving in the plane for 100 time steps as it observes 22 landmarks. For visualization purposes, each robot pose and landmark is rendered at its ground-truth position in 2D. With this, we see that the odometry factors form a prominent Markov chain backbone, whereas off to the sides binary factors are connected to the landmarks. Factors in these types of problems are typically highly nonlinear.

However, simply examining the factor graph already reveals a great deal of structure with which we can gain insight into a particular instance of the SLAM problem. First, there are landmarks with a large number of measurements, which we expect to be pinned down very well. Others have only a tenuous connection to the graph, and hence we expect them to be less well determined. For example, the lone landmark near the bottom right of **Figure 3** has only a single measurement

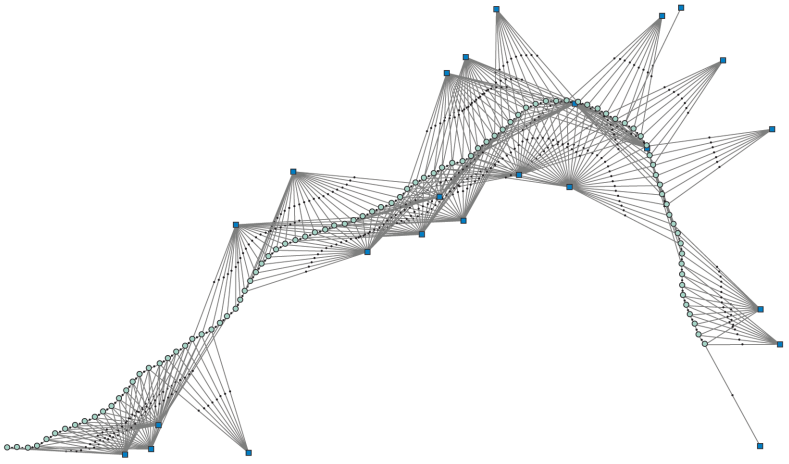

Figure 3

Factor graph for a larger, simulated SLAM example. Abbreviation: SLAM, simultaneous localization and mapping. Figure adapted with permission from Reference 2.

associated with it, and it cannot even be unambiguously reconstructed in the case of bearing-only measurements. In Section 3, I use this example to illustrate the computational aspects of inference in factor graphs.

3. INFERENCE IN FACTOR GRAPHS

Here, I discuss inference in factor graphs, i.e., inferring the most probable value for the unknowns, given the measurements and prior knowledge. A key concept is the elimination algorithm, which is a general algorithm that can be used across many different domains. I then specialize this algorithm more toward robotics applications and show that even in nonlinear optimization problems, it lies at the heart of the computation. Many of these concepts are worked out in more detail in Reference 3.

MAP inference for SLAM problems with Gaussian noise models is equivalent to solving a nonlinear least squares problem (2). Indeed, for an arbitrary factor graph, MAP inference comes down to maximizing the product (3) of all factor graph potentials:

$$X^{\mathrm{MAP}} = \underset{X}{\operatorname{argmax}} \prod_i \phi_i(X_i). \qquad 3.$$

Assume for now that all factors can be modeled by means of measurement functions h_i,

$$\phi_i(X_i) \propto \exp\left\{-\frac{1}{2}\left\|h_i(X_i) - z_i\right\|^2_{\Sigma_i}\right\}, \qquad 4.$$

which includes both Gaussian priors and likelihood factors derived from measurements z_i corrupted by zero-mean Gaussian noise, with covariance Σ_i. Taking the negative log of Equation 3 and dropping the factor 1/2 allows us to instead minimize a sum of nonlinear least squares:

$$X^{\mathrm{MAP}} = \underset{X}{\operatorname{argmin}} \sum_i \left\|h_i(X_i) - z_i\right\|^2_{\Sigma_i}. \qquad 5.$$

Minimizing this objective function performs sensor fusion through the process of combining several measurement-derived likelihood factors, and possibly several priors, to uniquely determine the MAP solution for the unknowns.

Even though the functions h_i are nonlinear, if we have a decent initial guess available, then nonlinear optimization methods such as Gauss–Newton iterations or the Levenberg–Marquardt algorithm will be able to converge to the maximizer of Equation 3. They do so by solving a succession of linear approximations. We can linearize all measurement functions $h_i(\cdot)$ by using a simple Taylor expansion,

$$h_i(X_i) = h_i(X_i^0 + \Delta_i) \approx h_i(X_i^0) + H_i\Delta_i, \quad 6.$$

where the measurement Jacobian H_i is defined as the (multivariate) partial derivative of $h_i(\cdot)$ at a given linearization point X^0.

By collecting the matrices H_i for all individual factors into a single measurement Jacobian A, possibly scaled to account for the measurement covariances, we finally obtain the following standard least squares problem:

$$\Delta^* = \underset{\Delta}{\operatorname{argmin}} \ \|A\Delta - b\|_2^2, \quad 7.$$

where b collects all prediction errors $z_i - h_i(X_i^0)$. The Jacobian A is a large but sparse matrix, with a block structure that mirrors the structure of the underlying factor graph.

The process of solving the sparse linear least squares problem represented by Equation 7 is typically hidden from view in sparse linear algebra libraries, but breaking open this black box turns out to be crucial to understanding the computational ramifications of the problem structure, as codified in its factor graph. In fact, solving sparse linear least squares problems is done using a specialized version of the elimination algorithm, which I discuss next.

Jacobian: a matrix of partial derivatives of errors with respect to variables

Directed acyclic graph (DAG): a directed graph with no directed cycles

3.1. The Elimination Algorithm

The elimination algorithm for inference in general factor graphs is simple to explain. It proceeds in two phases: elimination and back substitution. In the elimination phase, we eliminate the unknown variables one by one to obtain a directed acyclic graph (DAG) in which each variable is expressed as a function of its parents. In the back-substitution phase, we compute the most probable explanation in reverse elimination order.

The elimination algorithm is very general and applies to such diverse problems as Boolean satisfiability, constraint satisfaction and constraint optimization problems (4), inference in Bayes networks (5), and finally linear and nonlinear least squares problems, as shown below.

Most of us are familiar with Gaussian elimination, also known as lower–upper (LU) matrix factorization, which is used to solve linear systems. When we have more equations than unknowns, we can instead strive to find a variable assignment that minimizes the sum of least squares. As an example, consider the rectangular matrix A in **Figure 4*a***, which corresponds to the factor graph from **Figure 3**. When we eliminate a variable, we can ask what the optimal assignment of that variable is given its parents. This computation is known as QR factorization. A faster variant, Cholesky factorization, proceeds instead by first forming A^TA. In both cases, the result is an upper triangular matrix R, for which back substitution can proceed exactly as in Gaussian elimination. **Figure 4** shows two variants of R that differ only with respect to the order in which variables were eliminated.

The elimination algorithm underlies many different classical algorithms that we know under different names in the AI and robotics literature. For example, the discrete equivalent of the tracking problem is known as a hidden Markov model, and the elimination algorithm as applied to hidden Markov models is known as the Viterbi algorithm. Similarly, when measurements in motion models are expressed as least square terms, the special-case elimination algorithm is known as a Kalman smoother. In turn, fixed-lag smoothers and Kalman filters are easily obtained as special

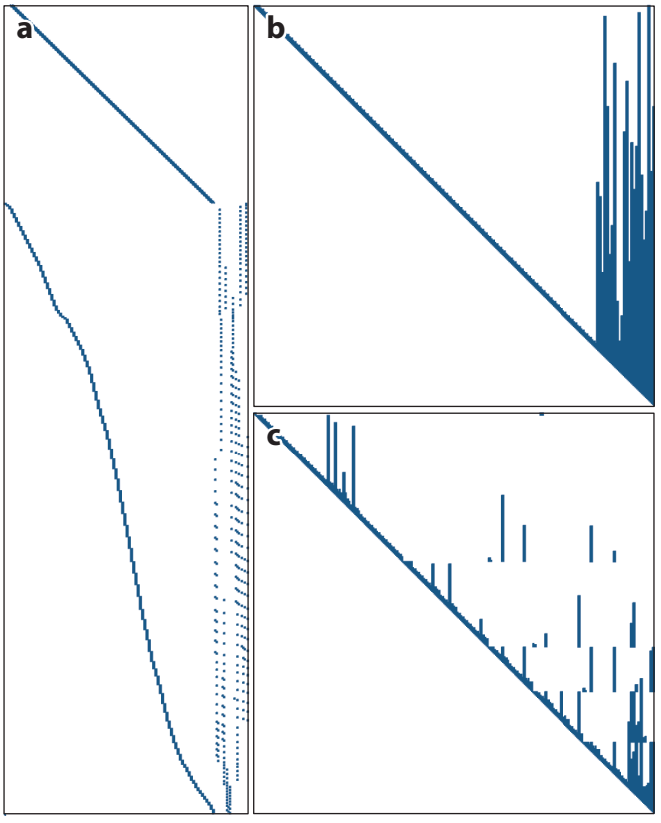

Figure 4

(*a*) The measurement Jacobian A associated with the problem in **Figure 3**, which has $3 \times 95 + 2 \times 24 = 333$ unknowns. The number of rows, 1,126, is equal to the number of (scalar) measurements. (*b*) The upper triangular Cholesky factor R. (*c*) An alternative, sparser factor R obtained with a better variable ordering, COLAMD permutation. Abbreviation: COLAMD, column approximate minimum degree.

cases of a Kalman smoother. Finally, the linear quadratic regulator with finite horizon can be seen as an application of the elimination algorithm, as well.

3.2. Ordering Heuristics

Insight into the graphs underlying robotics inference, and how their sparsity is affected by the implementation choices we make, is crucial for achieving highly performant algorithms. Sometimes it is beneficial to eliminate certain variables in advance, because it can be done efficiently or even in closed form. For example, in the Schur complement trick, which is common in SfM, the 3D landmark variables are eliminated first, and then a reduced problem is solved for the cameras only (6). After that, the back substitution of the 3D landmarks can be done efficiently and in parallel, which is significant if there are many of them. Carlone et al. (7) gave a more general treatment of those so-called conditionally independent sets.

Most importantly, the flop count for sparse QR can vary dramatically for different elimination orderings. While any order will ultimately produce an identical solution, the order in which variables are eliminated matters, as different orderings lead to DAGs with different topologies,

which in turn will affect the computational complexity of the elimination algorithm. Unfortunately, finding an elimination ordering with minimum fill-in that minimizes the cost of the elimination/factorization algorithm is NP hard, and we have to resort to heuristics. Two widely used sparse matrix ordering algorithms for scientific computation are based on the heuristic of first eliminating the least constrained variables of G. This family of algorithms is known as the minimum degree algorithms, and the two popular variants are minimum degree (MMD) and approximate minimum degree (AMD).

As an example, look again at the factor graph shown in **Figure 3** and its Jacobian matrix A in **Figure 4**. **Figure 4*a*** shows the resulting upper triangular Cholesky factor R for two different orderings. These orderings differ in the amount of sparsity they exhibit, and it is exactly this that will determine how expensive it is to factorize A or $\mathcal{I}$. The first version of the ordering comes naturally: It eliminates the poses first and then the landmarks, leading to a sparse R factor with 9,399 nonzeros. By contrast, the sparse factor R in **Figure 4*c*** was obtained by reordering the variables according to the column approximate minimum degree permutation (COLAMD) heuristic (8) and has only 4,168 nonzeros. Yet back substitution gives exactly the same solution for both versions.

In typical robotics applications, even using a domain-independent heuristic such as AMD to eliminate the variables in SLAM yields substantial computational wins. However, because finding an optimal ordering is NP complete, any piece of domain-specific information can help a great deal. A simple domain-specific heuristic is the semantic information from the true factor graph at the level of poses and landmarks, not their scalar components. In the example given above, a further decrease of nonzero elements in R by a factor of two can be achieved by simply applying AMD at the block level instead of the scalar level.

3.3. Nested Dissection

A fundamentally different approach to reordering is to apply a divide-and-conquer paradigm. This is feasible because eliminating a node only induces new constraints for a set of (spatially) direct neighbors. Nested dissection algorithms try to exploit this by recursively partitioning the graph and returning a postfix notation of the partitioning tree as the ordering. In some cases, theoretical guarantees can be given on the resulting elimination complexity (9). The two most important results are as follows: (*a*) For chain-like graphs, we can find constant-size separators, and the resulting factorization will cost $O(n)$ flops (i.e., linear in the number of variables), which explains the fact that inverting or factorizing a band-diagonal matrix is linear, and (*b*) for planar graphs, we can recursively find separators of size less than $2\sqrt{2n}$, and the resulting factorization will cost $O(n^{1.5})$ flops.

The application of nested dissection orderings is especially relevant to SLAM, as environments mapped by mobile robots often contain parts that are spatially separated. A divide-and-conquer scheme is one of the most promising ways to solve challenging SLAM problems, especially in large-scale environments. One advantage of submap-based approaches is that the computation can be done in an out-of-core manner, making it possible to distribute most of the work over multiple computation resources, increasing the scalability in terms of both time and memory.

3.4. Incremental Inference

Many inference problems in robotics are incremental, meaning that measurements arrive as a temporal sequence, and intermediate solutions are needed. It is natural to question whether we can reuse previous computations or must perform a full batch optimization each time. For example,

in SLAM, we do not want to solve a batch problem every time the robot moves and provides a small number of new measurements.

Bayes tree: a directed clique tree that captures the clique structure of a Cholesky factor R for a particular elimination ordering

The desire to generalize incremental inference to nonlinear problems motivates the introduction of the Bayes tree. It is well known that inference in a tree-structured graph is efficient, with linear chain structures as an even more special case. By contrast, the factor graphs associated with typical robotics problems contain many loops. Still, we can construct a tree-structured graphical model in a two-step process: first performing variable elimination on the factor graph to obtain a Bayes net with a special property, and then exploiting that special property to find a tree structure over cliques in this Bayes net.

In particular, the DAG obtained by running the elimination algorithm on a factor graph satisfies a special property: It is chordal, meaning that any undirected cycle of length greater than three has a chord. A chord is an edge connecting two nonconsecutive vertices on the cycle. In AI and machine learning, a chordal graph is more commonly said to be triangulated. By identifying cliques in this chordal graph, the DAG may be rewritten as a Bayes tree. We introduce this new, tree-structured graphical model to capture the clique structure of the DAG, and because it is chordal, the cliques in the Bayes net will form a tree (proof omitted). Listing all these cliques in an undirected tree yields a clique tree, also known as a junction tree in the AI literature (5). The Bayes tree is just a directed version of this tree that preserves information about the elimination order.

Incremental inference corresponds to a simple editing of the Bayes tree. This view provides a better explanation and understanding of the otherwise abstract incremental matrix factorization process. It also allows us to store and compute the square-root information matrix in the form of a Bayes tree, a deeply meaningful sparse storage scheme. Adding a new measurement corresponds to updating a factor (and, hence, to updating the Bayes tree), the affected parts of the tree are converted back into a factor graph, and the new factor associated with the new measurement is added to it. By reeliminating this temporary factor graph, using whatever elimination ordering is convenient, a new Bayes tree is formed, and the unaffected subtrees can be reattached.

Figure 5 shows how the Bayes tree evolves when iSAM (Incremental Smoothing and Mapping) is run on four steps of a Manhattan world-simulated sequence from Reference 11. As the robot explores the environment, new measurements affect only parts of the tree, and only those parts are recalculated.

3.5. Iterative Solvers

I would be remiss in not pointing out that there are alternatives to the elimination algorithm in the inner loop of a nonlinear least squares solver. For densely connected problems, it is often advantageous to switch to a preconditioned conjugate gradient, an iterative approach to solving linear systems (12). However, even in that case, the structural knowledge of the problem given by the factor graph is useful in devising efficient preconditioners. In earlier work, we have developed preconditioning schemes based on subgraphs (13) and generalized subgraphs (14), as have others (15). We can even combine the incremental direct inference methods afforded by the Bayes tree with subgraph-based preconditioning.

4. FACTOR GRAPHS IN NAVIGATION AND MAPPING

Several of the ideas above have now been incorporated into a variety of software packages widely used in the robotics community. GTSAM (Georgia Tech Smoothing and Mapping) (16) is an open source package built entirely around factor graphs and contains many of the algorithms discussed

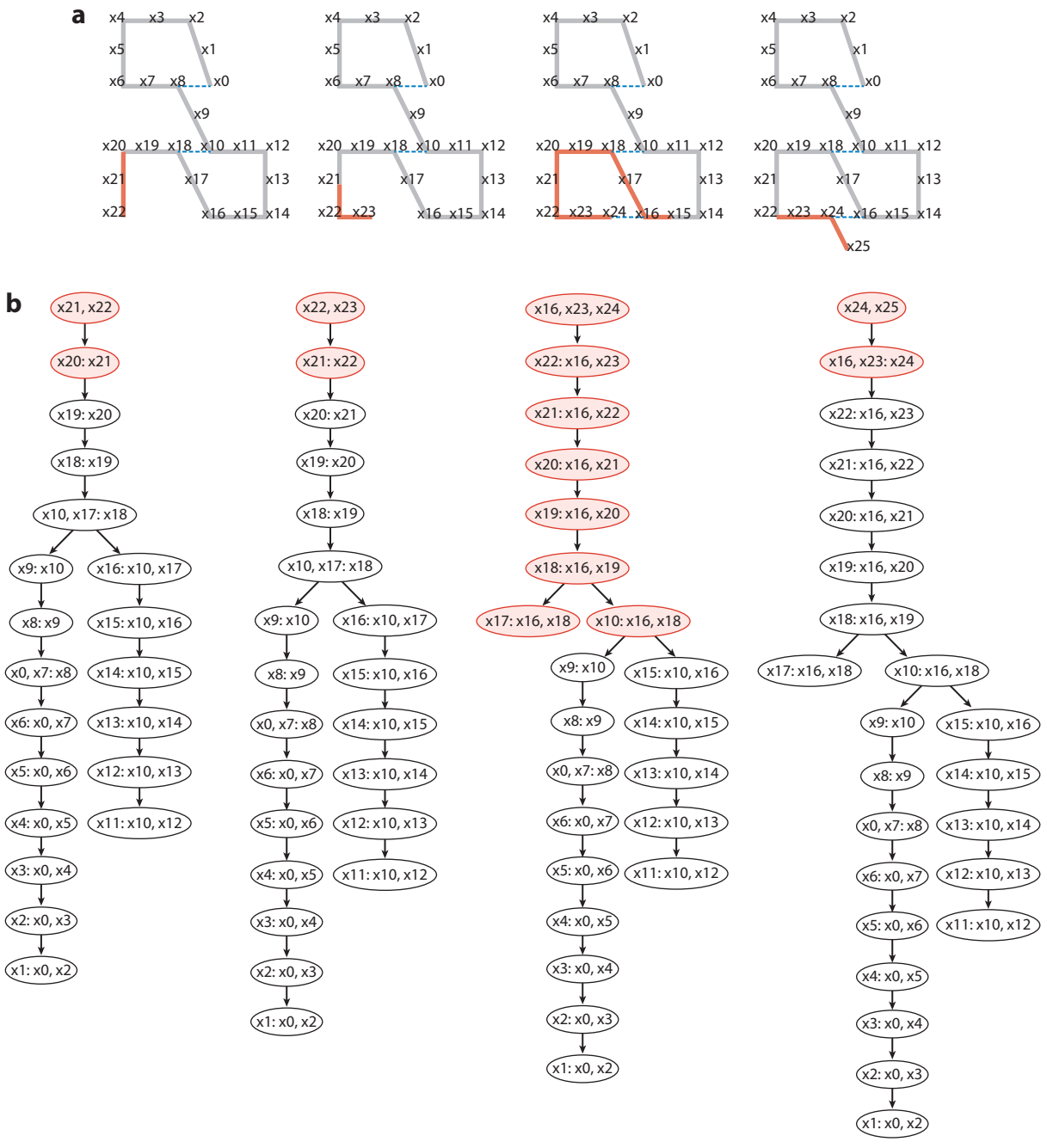

Figure 5

Evolution of the Bayes tree for a simple synthetic SLAM example. (*a*) Four simulation steps from a Manhattan world-simulated sequence. (*b*) The Bayes trees for these four steps, showing how the trees evolve when iSAM is run on each step. Only cliques outlined in red are recalculated. Abbreviations: iSAM, Incremental Smoothing and Mapping; SLAM, simultaneous localization and mapping. Figure adapted with permission from Reference 10.

below. g^{2}o (General Graph Optimization) (17) is a similar package that started out focused on stochastic gradient descent but has since developed many similar capabilities. MOLA (18) builds on GTSAM and provides a modular optimization framework for localization and mapping. And Ceres (19) is a state-of-the-art solver that has been used in many SfM pipelines.

4.1. Nested Dissection in SLAM

The divide-and-conquer nested-dissection approach from Section 3.3 was implemented in the TSAM (Tectonic Smoothing and Mapping) algorithm (20), which combines nested dissection and careful initialization. Some partitioning results from that work are given in **Figure 6**, which shows a recursive partitioning of the well-known Victoria Park data set (21). TSAM2 uses a combination of the METIS graph partitioning package (22) to find a nested dissection ordering at the global level. It then orders the resulting subgraphs locally using AMD. Similar ideas were applied by the same authors in the area of large-scale 3D reconstruction in a method called HyperSfM (23), but there the partitioning was done using hypergraph partitioning, again using METIS.

When multiple robotic platforms need to collaborate on solving a large mapping problem, as illustrated in **Figure 7**, an obvious way to partition the corresponding factor graphs is across multiple vehicles. Factors originate from measurements, and hence it makes sense to keep the factors local to the platform they were taken on. This is the approach adopted by DDF-SAM (Decentralized Data Fusion–Smoothing and Mapping), introduced by Cunningham et al. (24). Each robot optimizes a local factor graph and communicates information about shared variables

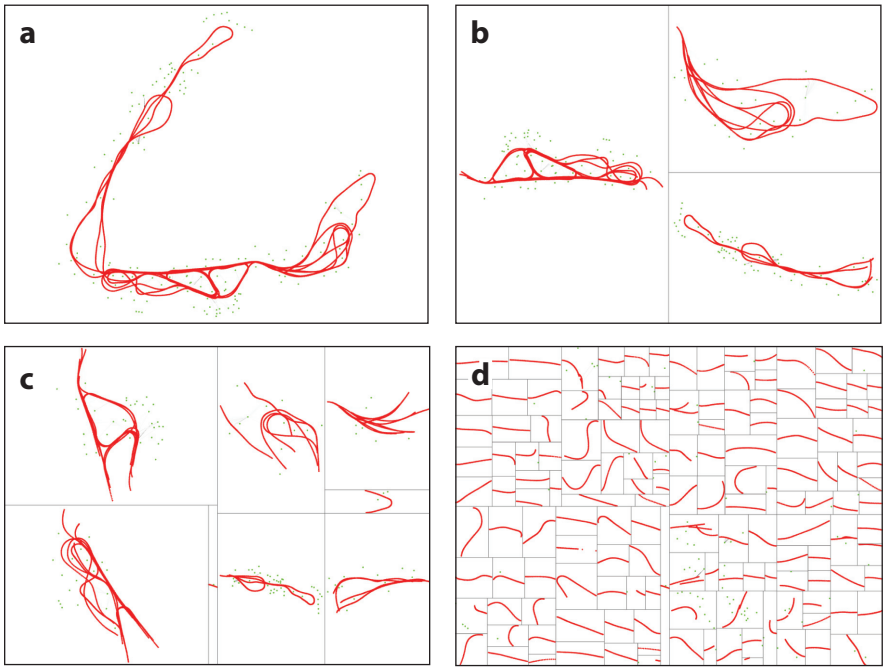

Figure 6

A nested dissection ordering that leads to a hierarchical decomposition of a SLAM problem, here shown for the Victoria Park data set. The four panels show four different levels of partitioning, which results in an efficient elimination ordering. Abbreviation: SLAM, simultaneous localization and mapping. Figure adapted with permission from Reference 20.

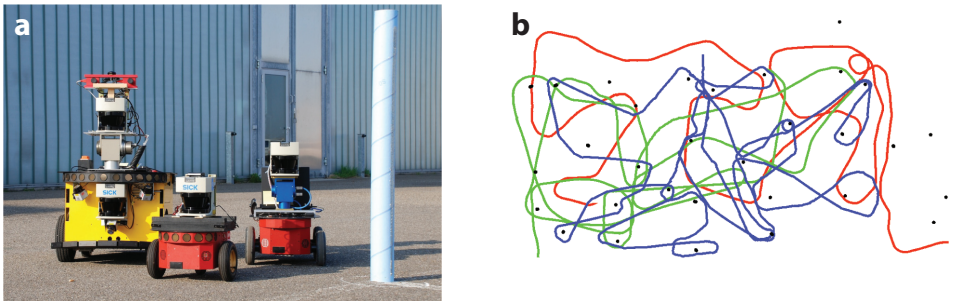

Figure 7

(*a*) Three robots and a synthetic landmark used in a multirobot mapping experiment. (*b*) Final results for all three robots at the end of the experiment. Figure adapted with permission from Reference 24.

of interest to the other robots in the form of a marginal density (which is a factor itself). Coordinate frame transformations are handled using a novel constrained optimization scheme. Results from the experiment are shown in **Figure 7*b***.

4.2. Incremental Smoothing and Mapping

The large-scale mapping problem illustrated in **Figure 8** serves as an example to motivate the need for incremental inference. The figure shows a map generated from multiple sessions of a robot operating in a building, totaling nine hours of operation time. While the problem is initially small and can easily be solved repeatedly, the time to solve grows over time to the point that real-time batch processing is no longer feasible. Even though an approximation is used to significantly reduce the size of the optimization problem (via reduced pose graphs; see Reference 25), repeated batch optimization soon becomes too expensive. Instead, incremental updates that make use of previous calculations are essential.

Putting all of the above together and addressing some practical considerations about relinearization yielded a state-of-the-art incremental, nonlinear approach to MAP estimation in robotics, iSAM2, summarized below but described in full detail by Kaess et al. (10). iSAM uses a Bayes tree representation for the posterior density. It then employs Bayes tree incremental updating as each new measurement comes in, as described above. iSAM2 has been used as the main inference backbone in quite a large number of papers; e.g., Schneider et al. (26) used it for tightly coupled GPS–IMU navigation in unmanned aerial vehicles. The junction tree has also been exploited for distributed inference in SLAM (27–29).

4.3. Inertial Measurement Unit Preintegration

One of the most successful applications of factor graphs is VIO. In a collaboration with the University of Zurich (30), we combined the idea of IMU preintegration (31) with iSAM2, the incremental inference engine that exploits the Bayes tree (10). The idea of preintegration was first proposed by Lupton & Sukkarieh (31) and comes down to fusing several high-frequency IMU measurements (accelerometer and gyroscope) into a single binary factor between $SE(3)$ navigation states that are needed at a slower sampling rate. For example, in VIO, we really want the pose estimates as the time images are acquired and do not want to introduce unknowns at the much higher IMU sampling rate, which can be as high as 1 kHz. The key insight of Lupton & Sukkarieh (31) was how to do so in a way that does not incorrectly integrate gravity. One of the advances introduced by

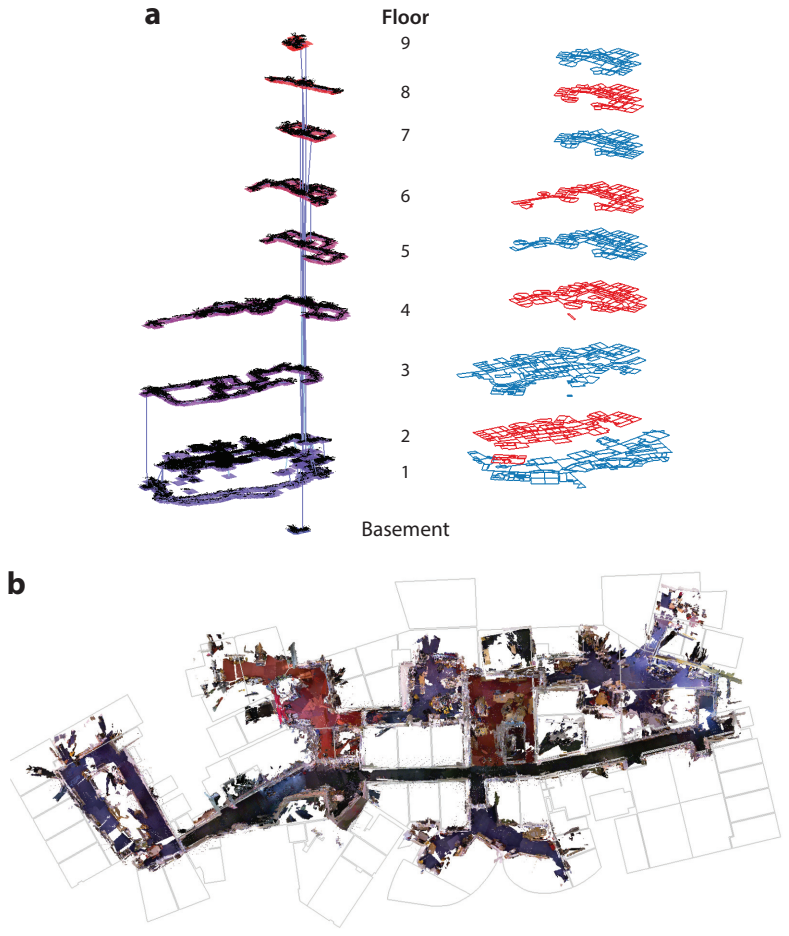

Figure 8

(*a*) A map of 10 floors of the Massachusetts Institute of Technology Stata Center created using a real-time visual SLAM system. The data were collected in 14 sessions over a six-month period. The total distance traveled was 11 km over nine hours of operation time. (*b*) Map for one of the sessions covering the second floor of the Stata Center (approximately 90 m across), with a ground-truth floor plan added for reference. The dense point clouds are RGB-D frames rendered at the solution of the pose graph. Abbreviations: RGB-D, red, green, blue, and depth; SLAM, simultaneous localization and mapping. Figure adapted with permission from Reference 25.

Forster et al. (30) was to take the idea of preintegration but do it on the tangent space of $SO(3)$, as shown in **Figure 9*a***, and then properly update the vehicle's attitude on the $SO(3)$ manifold. Coupled with iSAM2 fast incremental inference, this approach yielded a state-of-the-art VIO system. A sample result from this pipeline is shown in **Figure 9*b***.

5. PUSHING THE BOUNDARIES

In this final section, I discuss several recent efforts in which factor graphs have been used in settings beyond SLAM, pushing the boundaries into new application domains.

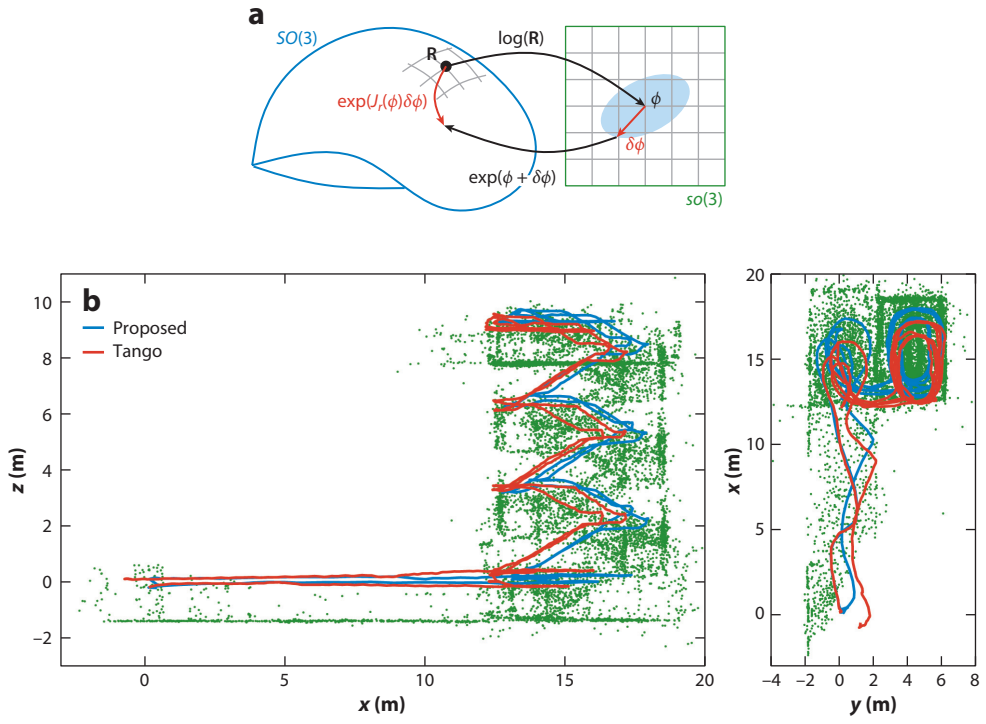

Figure 9

(*a*) Preintegration on the tangent space of $SO(3)$, as done by Forster et al. (30). (*b*) State-of-the-art VIO results with preintegrated IMU measurements and visual odometry fused within iSAM2, as compared with a then state-of-the-art pipeline, Tango. Abbreviations: IMU, inertial measurement unit; iSAM2, Incremental Smoothing and Mapping 2; VIO, visual–inertial odometry. Figure adapted with permission from Reference 30.

5.1. Legged Robots and Dealing with Contact

State estimation of legged robots is particularly challenging because of the switching contact between the robot's feet and the environment. Recent work by Hartley et al. (32, 33) has taken advantage of the factor graph framework to fuse inertial information with visual information and, importantly, domain-specific knowledge about legged robots. To this end, they developed two new factors that they deployed in an incremental iSAM2 estimator. The first factor relates the contact frame of the current stance foot with the base frame of the robot, and is simply a relative pose factor derived from noisy joint encoders. This constrains the range of possible states of the high-degree-of-freedom robot, in their case a Cassie walking robot, as illustrated in **Figure 10**. The second factor is even more interesting: They took the idea of preintegration and applied it to the contact state at the stance feet. This novel factor incorporates knowledge about contacts, slip, and hybrid dynamics and can be integrated through several contact switches, keeping the number of estimated states in the factor graph to a manageable number.

Maurice Fallon's group at Oxford employed a slightly different strategy to fuse visual and kinematic information (34), in this case for the ANYmal quadruped, shown in **Figure 11*a***. The authors implemented a visual–inertial pipeline with the IMU factor from Forster et al. (30) and a leg odometry factor derived from the internal high-frequency state estimator running on the vehicle, provided by ANYbotics (35). In follow-up work (36), this pose constraint was replaced with

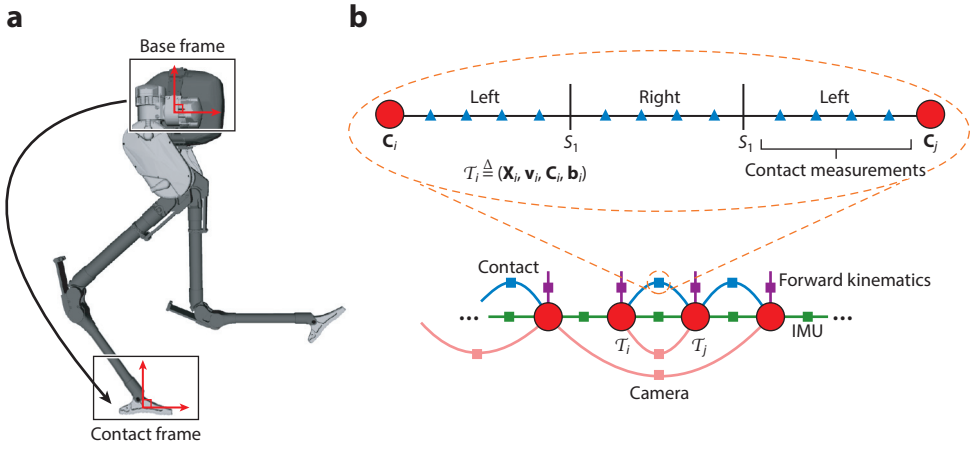

Figure 10

Factor graphs for legged robots. (*a*) The Cassie base frame and contact frame. These components are related by forward kinematics. (*b*) A factor graph fragment showing the forward kinematics factor as well as the preintegrated contact factor, which can integrate foot contacts through multiple phases. Abbreviation: IMU, inertial measurement unit. Figure adapted with permission from Reference 33.

a novel factor, again using the idea of preintegration, but now working with the velocity derived from another extended Kalman filter that fuses velocity estimates from multiple feet contacts; leg odometry drift is now modeled as a bias term in the velocity domain. Convincing experimental results are illustrated in **Figure 12**.

5.2. Motion Planning

Another key area in robotics where factor graphs have made inroads is motion planning. Trajectory optimization approaches to motion planning (37, 38) typically minimize an objective function that encourages trajectories to be both feasible and optimal. In Reference 39, we adopted a continuous-time representation of trajectories by assuming that trajectories are sampled from a Gaussian process and encoding this as a prior in a factor graph framework. Using this

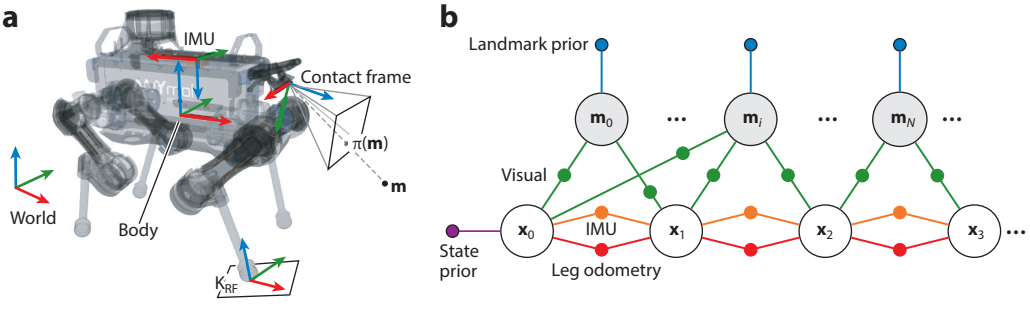

Figure 11

(*a*) An ANYbotics ANYmal quadruped, with relevant coordinate frames. (*b*) Factor graph showing visual landmarks m_i tightly integrated with the IMU factor from Forster et al. (30), but with an additional binary factor derived from the built-in inertial state estimator (running off a different IMU). Abbreviations: IMU, inertial measurement unit, K_{RF}, contact frame for right forward foot. Figure adapted with permission from Reference 34.

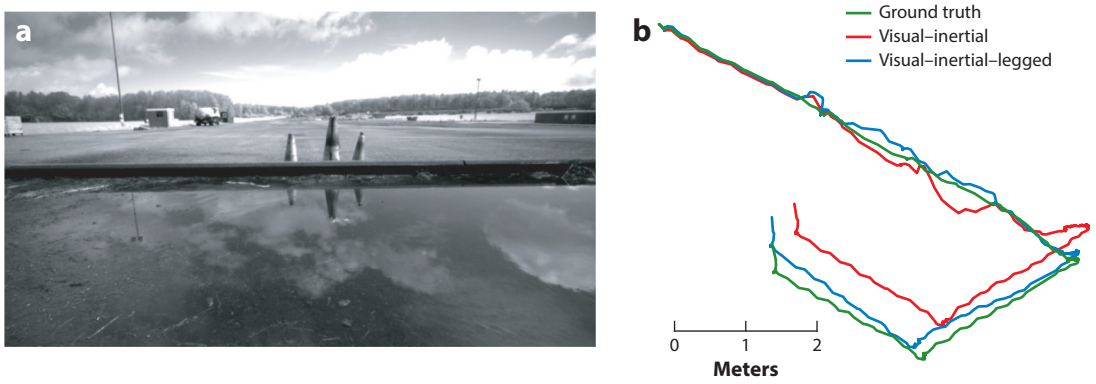

Figure 12

(*a*) Oily puddle environment in which the ANYbotics ANYmal quadruped was tested. (*b*) Performance of a preintegrated velocity bias factor. Figure adapted with permission from Reference 36.

representation, we developed a gradient-based optimization algorithm called GPMP (Gaussian Process Motion Planner) that avoids expensive fine-grained time discretization while still maintaining smoothness.

In contrast to estimation, GPMP uses factors to encourage smoothness and obstacle avoidance, and the trajectory to be optimized represents future actions of the robot. The factor graph approach enables all computational tricks regarding sparsity, elimination ordering, and incremental solvers developed in the SLAM literature (3) to be used in the motion planning context. These versions exploiting sparsity and incremental inference were designated GPMP2 and iGPMP2 (Incremental Gaussian Process Motion Planner 2), respectively, in Reference 39. **Figure 13** illustrates replanning on a WAM (Whole-Arm Manipulator) arm and a PR2 (Personal Robotics 2) robot with iGPMP2, which can be done in a matter of milliseconds, handsomely beating the state of the art in motion planning at that time.

Another important and active research area in robotics is kinodynamic motion planning (40). In recent unpublished work (41), we considered the kinodynamic motion planning as an optimal control problem, using factor graphs as a convenient and intuitive representation. Hence we followed the lead of GPMP2 but now incorporated dynamics as a set of constraint factors, as

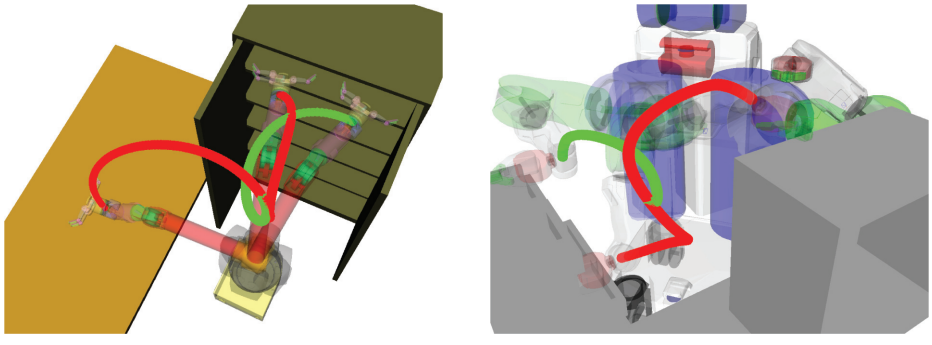

Figure 13

Incremental replanning results in iGPMP2. Abbreviation: iGPMP2, Incremental Gaussian Process Motion Planner 2. Figure adapted with permission from Reference 39.

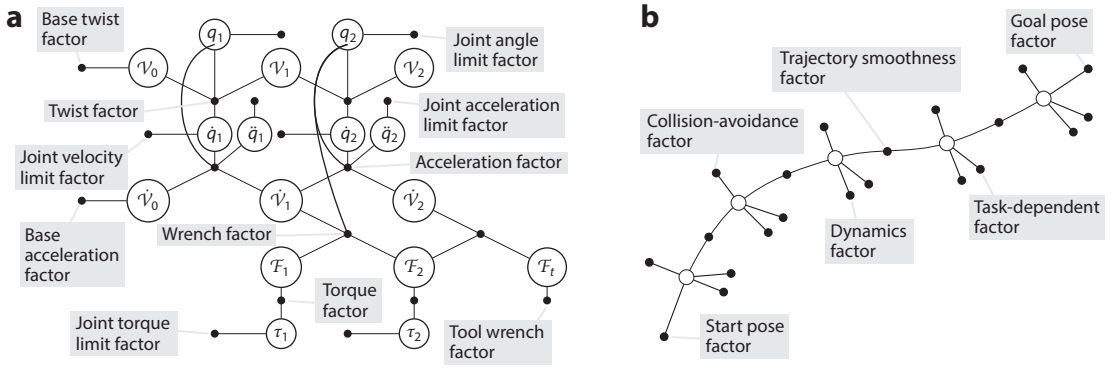

Figure 14

(*a*) A dynamic factor graph for a two-link manipulator. (*b*) A kinodynamic motion planning factor graph. To simplify the notation, we use a dynamics factor to represent all the dynamic constraints at each time slice. Figure adapted from Reference 41.

illustrated in **Figure 14*a***. The figure shows the dynamics factor graph of a two-link manipulator, where the equations of motion are represented as factors and the variables involved in the equations are represented as nodes in the graph. The notation is taken from the recent textbook by Lynch & Park (42). We then constructed an optimal control problem by adding factors representing all the usual trajectory optimization objectives, as well as factors enforcing the dynamics between successive time slices, as shown in **Figure 14*b***.

By combining optimal control and factor graphs, we obtain an intuitive, fast kinodynamic motion planner that is on par with the fastest optimal control methods and adds efficient, incremental kinodynamic replanning. The use of GTSAM (3, 16) as an optimizer ensures that we are exploiting the benefits of both sparsity and automatic differentiation, which was shown to be crucial to the state-of-the-art performance achieved by Zhao et al. (43). Fast replanning is useful in many contexts, such as model predictive control, and as far as we know, no existing kinodynamic motion planning methods can perform fast replanning beyond warm starting, as used in model predictive control.

An important question in all optimal control methods is how to deal with constraints on state and control variables, which the above efforts skirt by using barrier-type constraints. Very promising recent work (44) showed how hard constraints can be integrated by incorporating primal–dual methods into the (incremental) factorization process.

5.3. Simultaneous Trajectory Estimation and Planning

In Reference 45, we combined both estimation and motion planning and presented a unified approach to trajectory estimation and planning. Because both are inherently variants of trajectory optimization, they can be combined to remove the redundancy present in a traditional two-step process. The resulting STEAP method builds on approaches to continuous-time SLAM (46) and continuous-time GPMP2 (39) discussed above. As illustrated in **Figure 15**, STEAP computes the complete continuous-time trajectory from start to goal at each time step such that, given the current time step, the solutions to the estimation problem (history of the trajectory) and the planning problem (future of the trajectory) automatically fall out. Performing this joint optimization allows information to flow between estimation and planning, resulting in mutual benefits. Again we exploit the underlying sparsity of the problem and avoid resolving it from scratch as new information is encountered, using the iSAM2 incremental inference framework (10).

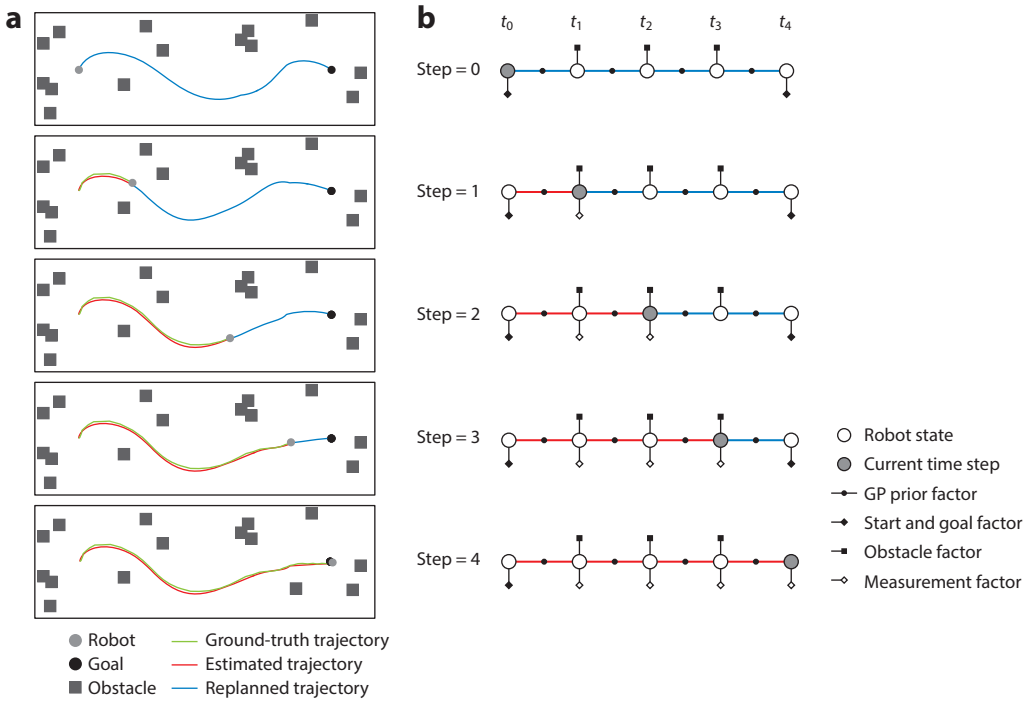

Figure 15

A simple example illustrating STEAP with a robot that navigates to a goal while avoiding obstacles. For each step, panel *a* shows the environment with ground-truth, estimated, and replanned trajectories, and panel *b* shows the corresponding factor graph. Abbreviation: STEAP, simultaneous trajectory estimation and planning. Figure adapted with permission from Reference 45.

5.4. Discrete–Continuous Problems

In the above, aside from the simple elimination examples, we have concentrated mostly on estimation and/or planning with continuous variables. However, a particularly difficult problem in computer vision and robotics is data association: Which measurements are associated with which variables? This is an instance of a problem that is inherently discrete in nature. Contact is another discrete problem that frequently comes up in robotics: For example, in legged locomotion, which feet are in contact with the ground, and which legs are in swing phase? Similar problems arise in the manipulation of objects with dexterous hands. Finally, as a last example, in motion planning discrete choices might have to be made—for example, whether to go to the left or the right of a particular obstacle, or whether to choose this or that grasp. This again introduces discrete choices in what is overwhelmingly a continuous problem, but more choices lead to exponentially more modes to deal with.

Research in this domain is still in its infancy. Early on, the FastSLAM method (47) explicitly modeled data association as discrete variables, and the authors were able to more or less get away with this because of the sampling-based paradigm they used. However, this is not expected to scale, especially because of their use of importance sampling in high-dimensional spaces. One way to get around this problem is the use of Markov chain Monte Carlo sampling, which was used in discrete–continuous inference problems such as bee tracking (48) and even before that in multitarget tracking (49) and SfM (50).

Very interesting recent work by Hsiao & Kaess (51) extended the Bayes tree incremental inference approach from iSAM2 (10) to incorporate discrete modes related to data association

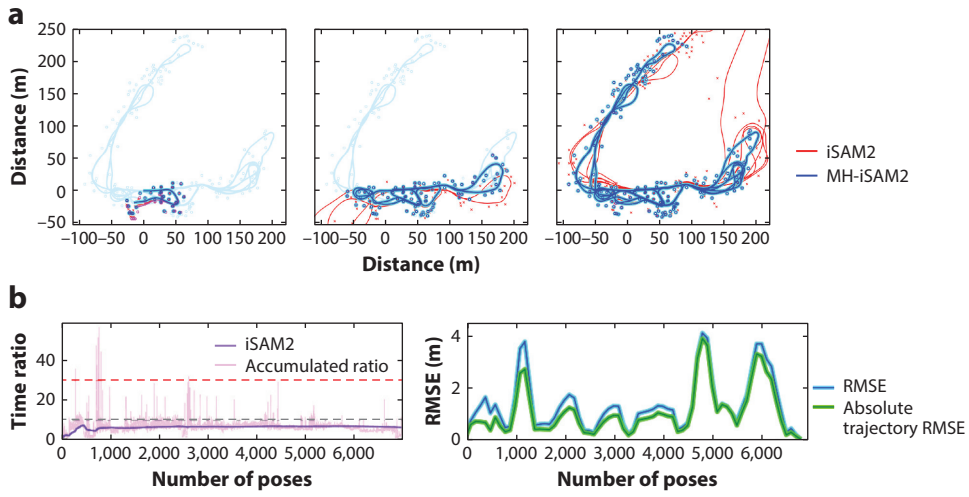

Figure 16

(*a*) An example of the multiple-hypothesis algorithm MH-iSAM2 (51) running on the Victoria Park data set (21) when random outliers are added to the measurements. iSAM2 cannot deal with these outliers gracefully, while MH-iSAM2 can. (*b*) Timing plots that show (*left*) the ratio of the computation against iSAM2 along with (*right*) RMSE and absolute trajectory RMSE. In the left subpanel, the horizontal dashed lines represent bounds that the MH-iSAM2 algorithm tries to stay below. Abbreviations: iSAM2, Incremental Smoothing and Mapping 2; MH-iSAM2, Multiple Hypothesis–Incremental Smoothing and Mapping 2; RMSE, root-mean-square error. Figure adapted with permission from Reference 51.

(**Figure 16**). Despite the nonnegligible overhead related to keeping track of and pruning the multiple hypothesis, the performance is nevertheless impressive considering the exponential complexity of making 412 discrete data association decisions.

5.5. Integration with Deep Learning

Another exciting development is the integration of factor graphs with deep learning. A case in point is the recent work on deep factors (52), which is a real-time SLAM system that is built around iSAM2 (10) but uses the compact code representation from Bloesch et al. (53) to optimize for dense depth at every key frame that is globally consistent. The resulting SfM pipeline is schematically shown in **Figure 17*a***, where in addition to poses p_i there is now a latent code c_i associated with each key frame. **Figure 17*b*** shows a few basis-depth images for a particular image, and it is this mapping from image intensities to basis images that is learned at training time, within a variational autoencoder framework.

An impressive system integrating a GTSAM back end with deep semantic annotations is the Kimera system developed by Rosinol et al. (54), which consists of four models: (*a*) the VIO preintegrated IMU from Forster et al. (30), (*b*) a robust PGO loop closure system also implemented in GTSAM, (*c*) a real-time meshing system, and (*d*) a semantics module that fuses data-driven semantic 2D segmentation in the 3D mesh representation. Some results from Rosinol et al.'s (54) paper are given in **Figure 18**, showing impressive 3D semantic SLAM output.

The factor graph optimization procedure can also be made differentiable. Bhardwaj et al. (55) took this approach to automatically tune the parameters of GPMP2 (39) based on the current state of the motion planner and the distribution and size of the obstacles that are to be avoided. Lv et al. (56) adopted a similar approach earlier in the context of image alignment.

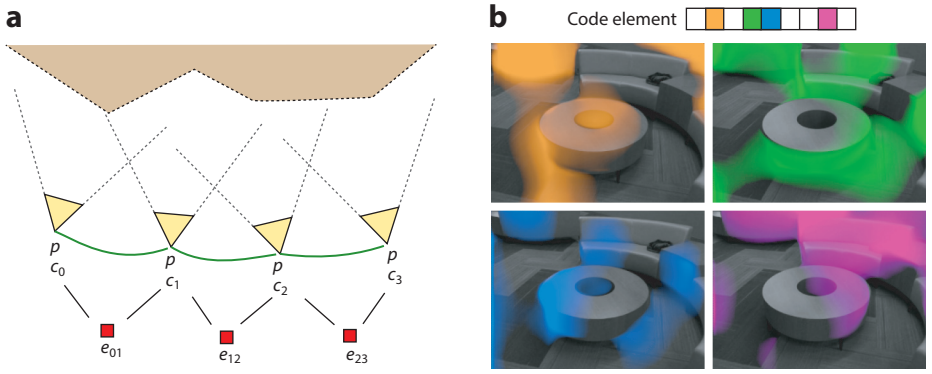

Figure 17

(*a*) SfM pipeline with latent codes c_i as well as pairwise consistency factors e_{ij}. (*b*) Image-intensity-conditioned depth components for different entries of the latent code. Abbreviation: SfM, structure from motion. Figure adapted with permission from Reference 52.

5.6. Toward Spatial AI

I will close this section by mentioning the recent position paper by Davison & Ortiz (57) and the follow-up paper by Ortiz et al. (58) on implementing a factor graph–based SfM computation on specialized hardware by Graphcore. By using loopy belief propagation (59)—something we have also experimented with in the past (60)—they were able to efficiently map the SfM factor graph to the massively parallel hardware, and the computation converges 30 times faster than a state-of-the-art, multithreaded CPU-based version (**Figure 19**).

Also of note is the recent work on 3D dynamic scene graphs (61). This approach leverages factor graphs to obtain a high-level understanding of robots, humans, and objects in a scene.

6. CONCLUSIONS

I hope that it is clear from this review that factor graphs have been used advantageously in many different contexts, ranging from the original SLAM problems, where continuous variables are estimated in a nonlinear least squares framework; to optimal control and motion planning, where factors represent desired states and/or actions; to the integration of data-driven methods such as variational autoencoders and deep semantic segmentation; and even to implementation on specialized hardware that was created for the benefit of those same deep learning methods.

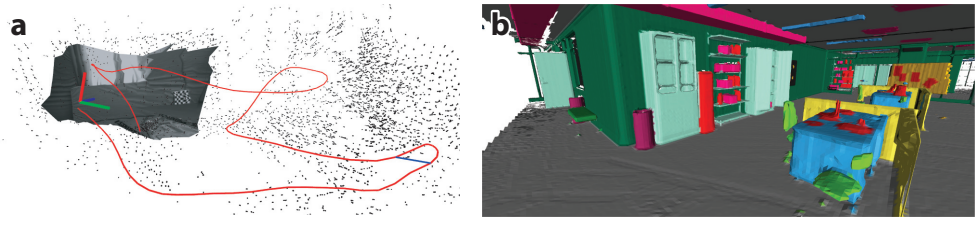

Figure 18

(*a*) A recovered trajectory and (*b*) a semantically annotated mesh from the Kimera system (54). This system builds on the VIO from Section 4.3 and fuses it with a data-driven semantic segmentation pipeline, yielding semantically rich 3D reconstructions that match the ground truth quite closely. Abbreviation: VIO, visual–inertial odometry. Figure adapted with permission from Reference 54.

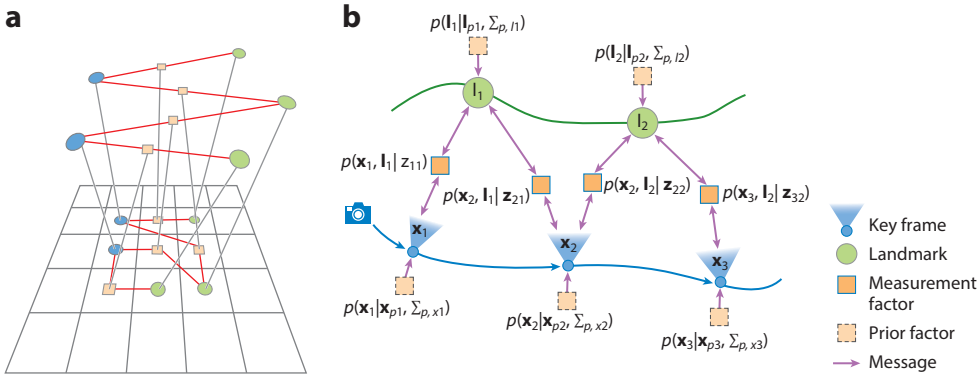

Figure 19

(*a*) A factor graph to compute SfM is mapped to the tiles in a Graphcore IPU. (*b*) Computation proceeds by message passing between factors and variables, as worked out by Yedidia et al. (59). Abbreviations: IPU, image processing unit; SfM, structure from motion. Figure adapted with permission from Reference 58.

Insight into the computational structure of these problems afforded by seeing them in the factor graph framework is beneficial in prompting new ideas and algorithmic advances. The original paper making this connection (2) has stood the test of time and introduced the importance of variable ordering, leading to several lines of work, both at the Georgia Institute of Technology and elsewhere. One of the most fruitful ideas is the notion of the Bayes tree in incremental inference, which is a completely novel graphical interpretation of more specialized algorithms, such as incremental factorization in linear algebra. The resulting iSAM2 algorithm (10) has been used in many different contexts and will no doubt become more important over time, unless loopy belief propagation (57, 60) indeed wins the day.

There are several fruitful areas of research in which I see factor graphs playing a large role over the next few years. The first is how to tackle the computational challenge of discrete–continuous inference, which is useful in many different contexts. The MH-iSAM (Multiple Hypothesis–Incremental Smoothing and Mapping) work by Hsiao & Kaess (51) is very promising in that respect, but it might also be of interest to revisit variational inference, as previously applied to switching linear dynamical systems (48). The second is the application of factor graphs in optimal control, planning, and hybrid dynamics. We have already worked on optimal control in the context of kinematic and kinodynamic motion planning (39, 41, 45), and I believe the structure of these problems is a natural match for factor graphs. Combining advances in discrete–continuous inference with optimal control would open up the road toward task and motion planning and optimal control in hybrid systems, such as legged robots. Finally, blending (differentiable) factor graphs with data-driven deep learning is another recent development where we are already seeing exciting results. This work includes incorporating learned factors (52), integrating deep perception modules (54), and differentiation through factor graph optimization (55, 56).

SUMMARY POINTS

1. Factor graphs can represent a wide variety of problems across robotics.
2. By laying bare the compositional structure of a problem, factor graphs expose opportunities to improve computational performance.

3. Factor graphs are beneficial in designing and thinking about how to model a problem, even aside from performance considerations.

FUTURE ISSUES

1. The computational challenge of discrete–continuous inference needs to be tackled.
2. Factor graphs have potential applications in optimal control, planning, and hybrid dynamics.
3. Blending (differentiable) factor graphs with data-driven deep learning is an area with further potential that is already showing exciting results.

DISCLOSURE STATEMENT

The author is not aware of any affiliations, memberships, funding, or financial holdings that might be perceived as affecting the objectivity of this review.

ACKNOWLEDGMENTS

I would like to sincerely thank my former and present students whose work is represented in these pages: Chris Beall, Alex Cunningham, Jing Dong, Yong Dian Jian, Michael Kaess, Alex Kipp, Peter Krauthausen, Zhaoyang Lv, Kai Ni, Sang Min Oh, Ananth Ranganathan, Richard Roberts, and Mandy Xie. I would also like to thank my postdocs, collaborators, and coadvisors who helped create that work: Doru Balcan, Tucker Balch, Byron Boots, Wolfram Burgard, Luca Carlone, Justin Carlson, Christian Forster, Andreas Geiger, Viorela Ila, Vadim Indelman, Hordur Johannsson, John Leonard, Mustafa Mukadam, Jim Rehg, Davide Scaramuzza, Steve Seitz, Chuck Thorpe, Sebastian Thrun, Kai Wurm, and Xinyan Yan. In addition, many thanks go to the authors who generously provided figures to include in this review: Marco Camurri, Jan Czarnowski, Andy Davison, Ryan Eustice, Maurice Fallon, Jessy Grizzle, Ross Hartley, Maani Ghaffari Jadidi, Michael Kaess, Joseph Ortiz, Antoni Rosinol, Cyrill Stachniss, and David Wisth.

LITERATURE CITED

1. Frey B, Kschischang F, Loeliger HA, Wiberg N. 1997. Factor graphs and algorithms. In *Proceedings of the 35th Allerton Conference on Communications, Control, and Computing*, pp. 666–80. Champaign: Univ. Ill. Press
2. Dellaert F, Kaess M. 2006. Square root SAM: simultaneous localization and mapping via square root information smoothing. *Int. J. Robot. Res.* 25:1181–203
3. Dellaert F, Kaess M. 2017. Factor graphs for robot perception. *Found. Trends Robot.* 6:1–139
4. Dechter R. 2003. *Constraint Processing*. San Francisco: Morgan Kaufmann
5. Koller D, Friedman N. 2009. *Probabilistic Graphical Models: Principles and Techniques*. Cambridge, MA: MIT Press
6. Triggs B, McLauchlan P, Hartley R, Fitzgibbon A. 2000. Bundle adjustment—a modern synthesis. In *Vision Algorithms: Theory and Practice*, ed. W Triggs, A Zisserman, R Szeliski, pp. 298–372. Berlin: Springer
7. Carlone L, Kira Z, Beall C, Indelman V, Dellaert F. 2014. Eliminating conditionally independent sets in factor graphs: a unifying perspective based on smart factors. In *2014 IEEE International Conference on Robotics and Automation*, pp. 4290–97. Piscataway, NJ: IEEE

8. Davis T, Gilbert J, Larimore S, Ng E. 2004. A column approximate minimum degree ordering algorithm. *ACM Trans. Math. Softw.* 30:353–76
9. Lipton R, Tarjan R. 1979. A separator theorem for planar graphs. *SIAM J. Appl. Math.* 36:177–89
10. Kaess M, Johannsson H, Roberts R, Ila V, Leonard J, Dellaert F. 2012. iSAM2: incremental smoothing and mapping using the Bayes tree. *Int. J. Robot. Res.* 31:217–36
11. Olson E, Leonard J, Teller S. 2006. Fast iterative alignment of pose graphs with poor initial estimates. In *Proceedings of the 2006 IEEE International Conference on Robotics and Automation*, pp. 2262–69. Piscataway, NJ: IEEE
12. Björck A. 1996. *Numerical Methods for Least Squares Problems*. Philadelphia: Soc. Ind. Appl. Math.
13. Dellaert F, Carlson J, Ila V, Ni K, Thorpe C. 2010. Subgraph-preconditioned conjugate gradient for large scale SLAM. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2566–571. Piscataway, NJ: IEEE
14. Jian YD, Balcan D, Dellaert F. 2011. Generalized subgraph preconditioners for large-scale bundle adjustment. In *2011 International Conference on Computer Vision*, pp. 295–302. Piscataway, NJ: IEEE
15. Kushal A, Agarwal S. 2012. Visibility based preconditioning for bundle adjustment. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1442–49. Piscataway, NJ: IEEE
16. Dellaert F. 2012. *Factor graphs and GTSAM: a hands-on introduction*. Tech. Rep. GT-RIM-CP&R-2012-002, Ga. Inst. Technol., Atlanta
17. Kuemmerle R, Grisetti G, Strasdat H, Konolige K, Burgard W. 2011. g^2o: a general framework for graph optimization. In *2011 IEEE International Conference on Robotics and Automation*, pp. 3607–13. Piscataway, NJ: IEEE
18. Blanco-Claraco JL. 2019. A modular optimization framework for localization and mapping. In *Robotics: Science and Systems XV*, ed. A Bicchi, H Kress-Gazit, S Hutchinson, pap. 43. N.p.: Robot. Sci. Syst. Found.
19. Agarwal S, Mierle K, et al. 2013. *Ceres Solver*. **http://ceres-solver.org**
20. Ni K, Dellaert F. 2010. Multi-level submap based SLAM using nested dissection. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2558–65. Piscataway, NJ: IEEE
21. Guivant J, Nebot E. 2001. Optimization of the simultaneous localization and map building algorithm for real time implementation. *IEEE Trans. Robot. Automat.* 17:242–57
22. Karypis G, Kumar V. 1998. Multilevel algorithms for multi-constraint graph partitioning. In *SC '98: Proceedings of the 1998 ACM/IEEE Conference on Supercomputing*, p. 28. Piscataway, NJ: IEEE
23. Ni K, Dellaert F. 2012. HyperSfM. In *2012 Second International Conference on 3D Imaging, Modeling, Processing, Visualization and Transmission*, pp. 144–51. Piscataway, NJ: IEEE
24. Cunningham A, Wurm K, Burgard W, Dellaert F. 2012. Fully distributed scalable smoothing and mapping with robust multi-robot data association. In *2012 IEEE International Conference on Robotics and Automation*, pp. 1093–100. Piscataway, NJ: IEEE
25. Johannsson H, Kaess M, Fallon M, Leonard J. 2013. Temporally scalable visual SLAM using a reduced pose graph. In *2013 IEEE International Conference on Robotics and Automation*, pp. 54–61. Piscataway, NJ: IEEE
26. Schneider J, Eling C, Klingbeil L, Kuhlmann H, Förstner W, Stachniss C. 2016. Fast and effective online pose estimation and mapping for UAVs. In *2016 IEEE International Conference on Robotics and Automation*, pp. 4784–91. Piscataway, NJ: IEEE
27. Paskin M. 2003. Thin junction tree filters for simultaneous localization and mapping. In *IJCAI '03: Proceedings of the 18th International Joint Conference on Artificial Intelligence*, pp. 1157–64. San Francisco: Morgan Kaufmann
28. Dellaert F, Kipp A, Krauthausen P. 2005. A multifrontal QR factorization approach to distributed inference applied to multi-robot localization and mapping. In *AAAI '05: Proceedings of the 20th National Conference on Artificial Intelligence*, Vol. 3, pp. 1261–66. Palo Alto, CA: AAAI Press
29. Pinies P, Paz P, Haner S, Heyden A. 2012. Decomposable bundle adjustment using a junction tree. In *2012 IEEE International Conference on Robotics and Automation*, pp. 1246–53. Piscataway, NJ: IEEE
30. Forster C, Carlone L, Dellaert F, Scaramuzza D. 2017. On-manifold preintegration theory for fast and accurate visual-inertial navigation. *IEEE Trans. Robot.* 33:1–21
31. Lupton T, Sukkarieh S. 2012. Visual-inertial-aided navigation for high-dynamic motion in built environments without initial conditions. *IEEE Trans. Robot.* 28:61–76

32. Hartley R, Mangelson J, Gan L, Ghaffari Jadidi M, Walls JM, et al. 2018. Legged robot state-estimation through combined forward kinematic and preintegrated contact factors. In *2018 IEEE International Conference on Robotics and Automation*, pp. 4422–29. Piscataway, NJ: IEEE
33. Hartley R, Ghaffari Jadidi M, Gan L, Huang JK, Grizzle JW, Eustice RM. 2018. Hybrid contact preintegration for visual-inertial-contact state estimation using factor graphs. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3783–90. Piscataway, NJ: IEEE
34. Wisth D, Camurri M, Fallon M. 2019. Robust legged robot state estimation using factor graph optimization. *IEEE Robot. Autom. Lett.* 4:4507–14
35. Bloesch M, Burri M, Sommer H, Siegwart R, Hutter M. 2017. The two-state implicit filter – recursive estimation for mobile robots. *IEEE Robot. Autom. Lett.* 3:573–80
36. Wisth D, Camurri M, Fallon M. 2020. Preintegrated velocity bias estimation to overcome contact nonlinearities in legged robot odometry. In *2020 IEEE International Conference on Robotics and Automation*. Piscataway, NJ: IEEE. Forthcoming
37. Ratliff N, Zucker M, Bagnell J, Srinivasa S. 2009. CHOMP: gradient optimization techniques for efficient motion planning. In *2009 IEEE International Conference on Robotics and Automation*, pp. 489–94. Piscataway, NJ: IEEE
38. Schulman J, Duan Y, Ho J, Lee A, Awwal I, et al. 2014. Motion planning with sequential convex optimization and convex collision checking. *Int. J. Robot. Res.* 33:1251–70
39. Mukadam M, Dong J, Yan X, Dellaert F, Boots B. 2018. Continuous-time Gaussian process motion planning via probabilistic inference. *Int. J. Robot. Res.* 37:1319–40
40. Donald B, Xavier P, Canny J, Reif J. 1993. Kinodynamic motion planning. *J. ACM* 40:1048–66
41. Xie M, Dellaert F. 2020. Batch and incremental kinodynamic motion planning using dynamic factor graphs. arXiv:2005.12514 [cs.RO]
42. Lynch K, Park F. 2017. *Modern Robotics: Mechanics, Planning and Control*. Cambridge, UK: Cambridge Univ. Press
43. Zhao Y, Lin H, Tomizuka M. 2018. Efficient trajectory optimization for robot motion planning. In *2018 15th International Conference on Control, Automation, Robotics and Vision*, pp. 260–65. Piscataway, NJ: IEEE
44. Sodhi P, Choudhury S, Mangelson J, Kaess M. 2020. ICS: incremental constrained smoothing for state estimation. In *2020 IEEE International Conference on Robotics and Automation*, pp. 279–85. Piscataway, NJ: IEEE
45. Mukadam M, Dong J, Dellaert F, Boots B. 2018. STEAP: simultaneous trajectory estimation and planning. *Auton. Robots* 43:415–34
46. Anderson S, Barfoot T, Tong C, Särkkä S. 2015. Batch nonlinear continuous-time trajectory estimation as exactly sparse Gaussian process regression. *Auton. Robots* 39:221–38
47. Montemerlo M, Thrun S. 2003. Simultaneous localization and mapping with unknown data association using FastSLAM. In *2003 IEEE International Conference on Robotics and Automation*, Vol. 2, pp. 1985–91. Piscataway, NJ: IEEE
48. Oh SM, Rehg JM, Balch T, Dellaert F. 2008. Learning and inferring motion patterns using parametric segmental switching linear dynamic systems. *Int. J. Comput. Vis.* 77:103–24
49. Pasula H, Russell S, Ostland M, Ritov Y. 1999. Tracking many objects with many sensors. In *IJCAI '99: Proceedings of the 18th International Joint Conference on Artificial Intelligence*, Vol. 2, pp. 1160–67. San Francisco: Morgan Kaufmann
50. Dellaert F, Seitz S, Thorpe C, Thrun S. 2001. Feature correspondence: a Markov chain Monte Carlo approach. In *Advances in Neural Information Processing Systems 13*, ed. TK Leen, TG Dietterich, V Tresp, pp. 852–58. Cambridge, MA: MIT Press
51. Hsiao M, Kaess M. 2019. MH-iSAM2: multi-hypothesis iSAM using Bayes tree and hypo-tree. In *2019 International Conference on Robotics and Automation*, pp. 1274–80. Piscataway, NJ: IEEE
52. Czarnowski J, Laidlow T, Clark R, Davison A. 2020. DeepFactors: real-time probabilistic dense monocular SLAM. *IEEE Robot. Autom. Lett.* 5:721–28
53. Bloesch M, Czarnowski J, Clark R, Leutenegger S, Davison A. 2018. CodeSLAM—learning a compact, optimisable representation for dense visual SLAM. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2560–68. Piscataway, NJ: IEEE

54. Rosinol A, Abate M, Chang Y, Carlone L. 2020. Kimera: an open-source library for real-time metric-semantic localization and mapping. In *2020 IEEE International Conference on Robotics and Automation*, pp. 1689–96. Piscataway, NJ: IEEE
55. Bhardwaj M, Boots B, Mukadam M. 2020. Differentiable Gaussian process motion planning. In *2020 IEEE International Conference on Robotics and Automation*, pp. 10598–604. Piscataway, NJ: IEEE
56. Lv Z, Dellaert F, Rehg JM, Geiger A. 2019. Taking a deeper look at the inverse compositional algorithm. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4576–85. Piscataway, NJ: IEEE
57. Davison AJ, Ortiz J. 2019. FutureMapping 2: Gaussian belief propagation for spatial AI. arXiv:1910.14139 [cs.AI]
58. Ortiz J, Pupilli M, Leutenegger S, Davison AJ. 2020. Bundle adjustment on a graph processor. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2413–22. Piscataway, NJ: IEEE
59. Yedidia J, Freeman W, Weiss Y. 2002. *Understanding belief propagation and its generalizations*. Tech. Rep. TR-2001-22, Mitsubishi Electr. Res. Lab., Cambridge, MA
60. Ranganathan A, Kaess M, Dellaert F. 2007. Loopy SAM. In *IJCAI '07: Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pp. 2191–96. San Francisco: Morgan Kaufmann
61. Rosinol A, Gupta A, Abate M, Shi J, Carlone L. 2020. 3D dynamic scene graphs: actionable spatial perception with places, objects, and humans. In *Robotics: Science and Systems XVI*, ed. M Toussaint, A Bicchi, T Hermans, pap. 79. N.p.: Robot. Sci. Syst. Found.

Annual Review of Control, Robotics, and Autonomous Systems

Volume 4, 2021

Contents

Errata

An online log of corrections to *Annual Review of Control, Robotics, and Autonomous Systems* articles may be found at http://www.annualreviews.org/errata/control