2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
Daejeon Convention Center
October 9-14, 2016, Daejeon, Korea

Pose Fusion with Chain Pose Graphs for Automated Driving

Christian Merfels Cyrill Stachniss

***Abstract*— Automated driving relies on fast, recent, accurate, and highly available pose estimates. A single localization system, however, can commonly ensure this only to some extent. In this paper, we propose a multi-sensor fusion approach that resolves this by combining multiple localization systems in a plug and play manner. We formulate our approach as a sliding window pose graph and enforce a particular graph structure which enables efficient optimization and a novel form of marginalization. Our pose fusion approach scales from a filtering-based to a batch solution by increasing the size of the sliding window. We evaluate our approach on simulated data as well as on real data gathered with a prototype vehicle and demonstrate that our solution runs comfortably at 20 Hz, provides timely estimates, is accurate, and yields a high availability.**

I. Introduction

Navigation for automated vehicles requires a precise knowledge of the car's pose to make informed driving decisions. A large variety of systems and algorithms has been proposed in the past to solve the localization task, including systems based on Global Positioning System (GPS), vision, and lidar. It is important to realize that the overwhelming part of localization systems operates within limited system boundaries and can not guarantee 100% availability under real-world conditions. A GPS-based localization system for example will likely fail in satellite-denied regions such as in tunnels or parking garages, whereas a vision-based localization system is likely to fail in darkness or other extreme lighting conditions. There is the potential that the availability, robustness, and accuracy of the localization increase if multiple pose estimation procedures with orthogonal sensors are combined, adding also to the versatility and fail-safe behavior of the system.

In this paper, we present an approach to multi-sensor data fusion that decouples the localization from the fusion task and can be executed online. Merfels *et al.* [1] propose a concept for sliding window pose graph fusion that we develop to the more sophisticated chain pose graph approach in this paper. The proposed system, called the *PoseGraphFusion*, enables the combination of multiple global pose sources (e.g., GPS) with multiple odometry sources (e.g., wheel odometry) to estimate the current pose, as illustrated in Fig. 1 and Fig. 2. A key advantage of decoupling the fusion from the localization is the ability to incorporate third-party localization modules for which source code is unavailable. The contribution of this paper is an online pose estimation algorithm based on fusing multiple pose sources.

Christian Merfels is with Volkswagen Group Research, Wolfsburg, and Institute of Geodesy and Geoinformation, University of Bonn, Germany. Cyrill Stachniss is with Institute of Geodesy and Geoinformation, University of Bonn, Germany.

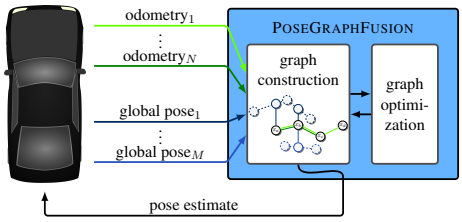


Fig. 1. Overview of the proposed multi-sensor data fusion: multiple odometry and global pose sources are fused in a graph-based optimization to provide a single pose estimate.

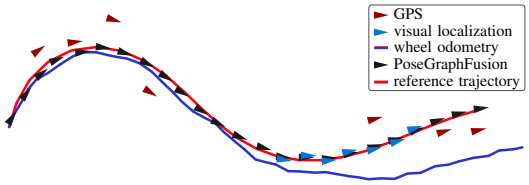


Fig. 2. A coarse localization (red triangles), a precise but only temporary available localization (blue triangles), and odometry as dead reckoning trajectory (blue) are used to estimate the true trajectory (red) of a vehicle. The estimated poses are shown as black triangles: the goal is to approximate the unknown red line as closely as possible with the black triangles.

The algorithm avoids overconfidence by performing delayed marginalization. The pose estimation is formulated as a sliding window graph-based optimization, which leads to the maximum likelihood (ML) estimate over the joint probability of vehicle poses in the current window. It converges to the online ML estimate for increasing sizes of the sliding window. Our pose fusion combines exchangeable input sources in a generic way. It deals with multi-rate sources, nonconstant input frequencies, out-of-sequence estimates, and time-varying latencies in a straightforward manner. Efficiency is a major design criterion as the proposed system runs online in an automated vehicle. Different parametrizations make it possible to scale from the (iterated) Extended Kalman Filter (EKF) to the online batch solution and thus to balance runtime versus accuracy.

In brief, the key contributions of this paper are:

- the presentation of an efficient sensor fusion algorithm with generic odometry and global pose inputs offering an intuitive architecture for pose estimation, which effortlessly resolves typical timing issues;
- the description of a graph construction algorithm designed to produce a sparse block-tridiagonal structure of the system matrix, which therefore offers a fast solution;
- the insight that marginalization on such a matrix structure can exactly and efficiently be carried out without an additional fill-in, so that marginalization can be interpreted as adding a prior node to the graph.

978-1-5090-3761-2/16/$31.00 ©2016 IEEE

II. Related Work

Multi-sensor data fusion for navigation systems enables the integration of information from multiple sources to estimate the state of the system. This is conventionally achieved by using filtering-based approaches such as the Kalman filter and its variants or, alternatively, sliding window smoothing algorithms. For example, Kubelka *et al.* [2] use an error state EKF to fuse information from four different odometry sources. Weiss *et al.* [3] propose an EKF to fuse inertial measurement unit (IMU) with GPS data and a camera-based pose estimate. Their work is generalized by Lynen *et al.* [4] to a Multi-Sensor-Fusion EKF. The filtering-based approaches have in common that they rely at a very early stage on the Markov assumption and marginalize all older information, thus prematurely incorporating the linearization error. Strasdat *et al.* [5] show that mainly because of that reason filtering performs suboptimal even for short time frames when compared to smoothing.

In contrast to filtering techniques, smoothing approaches compute the ML estimate by nonlinear least squares optimization to a Bayesian network, Markov random field (MRF), or factor graph. Offline batch optimization of this form assumes additive, white Gaussian noise with zero mean. It considers past and future measurements. Online batch optimization only takes into account all past states up to the current one. Although these are both computationally expensive operations, online batch optimization becomes feasible through the usage of incremental smoothing techniques, such as iSAM2 [6], that recalculate only the part of the graph that is affected by new measurements.

In this context, Chiu *et al.* [7] combine a long-term smoother using iSAM2 and a short-term smoother using so-called Sliding-Window Factor Graphs to fuse pose sources. Indelman *et al.* [8] use the incremental smoothing technique [6] to fuse multiple odometry and pose sources. Cucci and Matteucci propose the graph-based ROAMFREE framework [9] for multi-sensor pose tracking and sensor calibration. They keep the size of the graph bounded by simply discarding older nodes and edges, thus potentially obtaining overconfident estimates.

The approach that we present in this paper is from a methodical point of view closest to the approach of Sibley *et al.* [10], who are the first to introduce the concept of a sliding window filter in the context of robotics. They apply it to planetary entry, descent, and landing scenarios, in which they estimate surface structure with a stereo camera setup. Our contribution is based on a similar methodology but applies it to our use case of pose fusion. Furthermore, the structure of our problem is explicitly kept less complex by design and thus sparser due to the way our graph is constructed. This leads to a faster way of solving the nonlinear least squares equations, performing marginalization, and estimating the uncertainty of the output. Furthermore, we provide a way of semantically reasoning about the prior information arising from marginalization by deriving a prior node.

III. PoseGraphFusion

The input data to the sensor fusion are pose measurements, which are subject to noise. We assume the noise to be additive, white, and normally distributed with zero mean. Many estimation and data fitting problems can be formulated as nonlinear least squares problems. Our approach exploits the state-of-the-art graph optimization framework g2o [11] and for the most part, we adopt the notation of Kümmerle *et al.*

The key idea is that given the state vector $\boldsymbol{x} = (\boldsymbol{x}_1^\top, \dots, \boldsymbol{x}_m^\top)^\top$ and a set of measurements, where $\boldsymbol{z}_{ij}$ is the mean and $\boldsymbol{\Omega}_{ij}$ the information matrix of a single measurement relating $\boldsymbol{x}_i$ to $\boldsymbol{x}_j$ (with $\mathcal{C}$ being all pairs of indices for which a measurement is available), least squares estimation seeks the state

$$\boldsymbol{x}^* = \operatorname*{argmin}_{\boldsymbol{x}} \sum_{\langle ij \rangle \in \mathcal{C}} \boldsymbol{e}_{ij}^\top \boldsymbol{\Omega}_{ij} \boldsymbol{e}_{ij} \tag{1}$$

that best explains all measurements given the ℓ_2 norm. The vector error function $\boldsymbol{e}(\boldsymbol{x}_i, \boldsymbol{x}_j, \boldsymbol{z}_{ij})$ measures how well the constraint from the measurement $\boldsymbol{z}_{ij}$ is satisfied and we abbreviate it as $\boldsymbol{e}_{ij} = \boldsymbol{e}(\boldsymbol{x}_i, \boldsymbol{x}_j, \boldsymbol{z}_{ij})$. Solving (1) requires iteratively solving a linear system with the system matrix $\boldsymbol{H}$ and the right-hand side vector $\boldsymbol{b}$ such that

$$\boldsymbol{H} = \sum_{\langle ij \rangle \in \mathcal{C}} \boldsymbol{J}_{ij}(\boldsymbol{x})^\top \boldsymbol{\Omega}_{ij} \boldsymbol{J}_{ij}(\boldsymbol{x}), \tag{2}$$

$$\boldsymbol{b}^\top = \sum_{\langle ij \rangle \in \mathcal{C}} \boldsymbol{e}_{ij}^\top \boldsymbol{\Omega}_{ij} \boldsymbol{J}_{ij}(\boldsymbol{x}), \tag{3}$$

where $\boldsymbol{J}_{ij}(\boldsymbol{x})$ refers to the Jacobian of the error function computed in state $\boldsymbol{x}$. For more details, we refer the reader to [11].

A. Sliding Window Chain Pose Graph Fusion

In contrast to general nonlinear least squares estimation, which is commonly taking into account all available information within the full pose graph, it is necessary for an online state estimation system to limit the considered information to keep the problem computationally tractable. The proposed approach achieves this by marginalizing out prior state variables and thus only considering a fixed amount of state variables. More formally, the state vector $\boldsymbol{x}$ in a sliding window pose graph is reduced to the M most recent states $\boldsymbol{x} = (\boldsymbol{x}_{t-M+1}^\top, \dots, \boldsymbol{x}_t^\top)^\top$. The size of the system matrix $\boldsymbol{H}$ is therefore bounded by $\mathbb{R}^{3M \times 3M}$. The optimization result from the last time step provides the initial guess for the optimization in the current time step. This leads to an effective and efficient solution in practice as a single optimization is usually sufficient to integrate the additional information of the current time step.

The state variables consist of a position plus a heading. They are defined in a two-dimensional Cartesian coordinate system for which we choose the Universal Transverse Mercator coordinate system. We refer to pose sources, which measure poses within this coordinate system, as *global pose sources* (e.g., GPS), and poses in this system as *global poses*. Pose sources, which measure spatial transformations relative

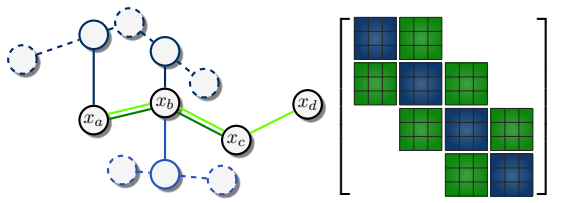
to the previous pose, are dubbed *local pose sources* or simply odometry.

In the spirit of MRFs, we refer to nodes representing state variables as *hidden nodes*. In contrast, *global* pose constraints are encoded in so-called *observed nodes* and denoted with $\bar{x}$. They are connected to hidden nodes to constrain them in the global coordinate frame. The observed nodes therefore "pull" the hidden nodes towards them. Their constraints equal zero if and only if the hidden nodes are identical to the observed nodes. Similarly, we map odometry measurements to edges between hidden nodes. They "push" the hidden nodes to relative poses which are equal to the relative transformation encoded in the edge.

In contrast to related graph-based approaches [9], [10], we neither generate a hidden node every time a measurement arrives nor tie their generation to a specific pose source. Instead, we construct a hidden node every time step, i.e., Δt seconds (the *temporal resolution*). For each hidden node, we query all global pose sources for measurements and interpolate one observed node per source at the timestamp of the hidden node if measurements are available. Additionally, we query each odometry source to interpolate the edges between all two successive hidden nodes. The motivation behind this is to enforce a certain matrix structure for $\boldsymbol{H}$, to include all measurement sources in a generic way independently of their specific output frequencies, and to a priori relate the number of state variables to the length of the interval of the sliding window. We refer to the resulting form of the graph as *chain pose graph*.

The block structure of $\boldsymbol{H}$ reflects the connections of poses and edges in the graph as it is its adjacency matrix [12]. Its structure changes slightly with the availability of measurements. In general, the block structure of a chain pose graph is a block-tridiagonal matrix. An example graph in Fig. 3 illustrates how to integrate multiple hidden and observed nodes as well as odometry constraints. It additionally shows the resulting block-tridiagonal matrix structure of the corresponding system matrix. The diagonal entries in the system matrix are influenced by odometry and global pose constraints, while the off-diagonal entries are only affected by odometry constraints. Loop closures are not considered as they arise rarely when driving straight from destination to target and break with the treatment of input sources as black boxes.

The block-tridiagonal structure is a consequence of the linear temporal ordering of the state variables combined with the fact that edges are at most constructed between successive nodes. We specifically design our solution to produce a block-tridiagonal matrix because this structure does not produce fill-in in $\boldsymbol{H}$ after marginalization of the oldest state variables. Beyond that, even the Cholesky factorization $\boldsymbol{H} = \boldsymbol{R}^\top \boldsymbol{R}$, which we perform to solve the linear system, does not suffer from fill-in in its triangular matrix $\boldsymbol{R}$. In fact, $\boldsymbol{R}$ becomes a band matrix. As a consequence, costly variable reordering techniques are unnecessary as $\boldsymbol{R}$ already contains the minimum number of nonzero elements necessary to reconstruct $\boldsymbol{H}$.



(a) A chain pose graph. (b) Corresponding block-tridiagonal structure of the system matrix.

Fig. 3. A chain pose graph and the corresponding structure of the system matrix. The black circles are hidden nodes, the dashed blue circles are global pose measurements from two different sources, the non-dashed blue circles are observed nodes, and the green edges are odometry constraints. Note how the raw global pose measurements are interpolated (dashed blue lines) at the same timestamps as the hidden nodes to obtain the observed nodes.

Furthermore, the computational complexity of the Cholesky factorization of a block-tridiagonal matrix is $\mathcal{O}(n)$ (with n being the number of nonzero entries in $\boldsymbol{H}$). This is a substantial improvement as for arbitrary (dense) matrix structures their decomposition or inversion becomes as costly as approximately $\mathcal{O}(n^{2.4})$. For experiments concerning the runtime for general graph-based optimization with g2o, we refer the reader to the runtime evaluations in [11, Fig. 8], where the time complexity for optimizing the Manhattan3500 dataset is clearly higher than linear in the number of nodes although the same sparse matrix techniques have been used that are being used in our implementation. For our approach, we demonstrate the linear time complexity for a practical experiment in the evaluations section.

In summary, our chain pose graph approach prevents fill-in after marginalization in $\boldsymbol{H}$, during the factorization in $\boldsymbol{R}$, makes common variable reodering strategies unnecessary, and is efficiently solvable in $\mathcal{O}(n)$.

B. Time behavior

After detailing how and for which timestamps we construct hidden and observed nodes, we turn our attention to the questions how our system handles the time behavior of input sources and how we design the time behavior of the pose fusion. In this paper, we define time behavior as the latency, frequency, and availability of estimates.

Integrating input sources with unknown time behavior is difficult as we deal with multi-rate sources, nonconstant input frequencies, out-of-sequence estimates, and time-varying latencies. Our approach consists in buffering all incoming data and preprocessing it. This does not introduce any delay as we do not need the measurements before the next graph construction phase. The preprocessing includes detecting missing pose estimates by estimating the recent input frequency of each source and comparing the number of estimates we should have received to the number of estimates we actually received. Sorting the data by time enables the integration of out-of-sequence data.

Instead of being data-triggered, the output of the PoseGraphFusion is time-triggered. Its output frequency f is decoupled from the temporal resolution Δt. Every $1/f$ the cycle of graph construction and optimization is triggered.

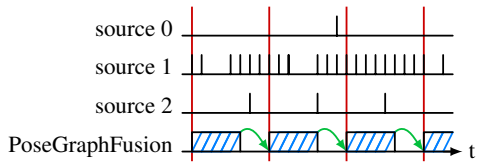


Fig. 4. Time behavior of the input data for three examples sources 0 to 2 and the corresponding time behavior of the output of the PoseGraphFusion. The input sources show occasional activity (source 0), data dropouts (source 1), and different data rates. The PoseGraphFusion is able to handle these characteristics and incorporates all input data by first buffering it. At the start of each cycle (directly after the red vertical line), the graph is constructed and optimized. The time necessary for that is the *computation time* (teal hatched). After the optimization, the estimated pose is propagated to the beginning of the next cycle (green arrows) so that it can be transmitted immediately.

The time spent for constructing and optimizing the graph is summed up as the *computation time*. The most recent state is estimated with a sliding window pose graph over the current set of measurements. It is subsequently propagated into the future to the start of the next cycle as depicted in Fig. 4 with a constant turn rate and velocity model. The propagation ensures that at that point, the now already computed pose estimate can directly be sent out. This in turn guarantees a low latency of the pose estimate, where we define the latency to be the difference of the time when the pose has been computed and the time for which it is valid. Depending on the application, a slightly higher latency might be tolerable in exchange for a propagation-free pose estimate, in which case the pose propagation is turned off.

A conventional Kalman filtering approach has difficulty to generically incorporate multiple input sources with unknown data rates. To integrate a new measurement, it has to propagate its state back in time to the time of the measurement, apply the measurement, and re-apply all other stored measurements. A pose graph approach with the described characteristics does not suffer from repeated backward-forward computations and is able to elegantly resolve time behavior issues and treat all sources in a homogeneous manner.

C. Marginalization in the form of a prior node

As stated in Section III-A, it is mandatory to limit the amount of hidden nodes to maintain constant runtime complexity. Simply removing edges and nodes leads to information loss and is equivalent to conditioning, which potentially leads to overconfidence. Therefore, we marginalize the oldest nodes. This truncates the graph but retains the same information (given the linearization point). The common approach for that is computing the Schur complement on the system matrix $\boldsymbol{H}$. In general, the disadvantage of this operation is the introduction of conditional dependencies between state variables that are connected. As we design our problem structure to be a chain pose graph, we are able to retain the same sparsity pattern and do not suffer from a denser system matrix after marginalization.

We examine the effect of the Schur complement and show that we can exploit the knowledge of the particular block-tridiagonal matrix structure to derive the concept of a *prior node*, which carries the same information as introduced by the Schur complement. In general, using a representation in the form of a graph is beneficial compared to directly solving the least squares equations because of the possibility to visually understand the problem, more possibilities for data inspection, and a more intuitive way to manipulate the problem structure. These are the same reasons why it is advantageous to construct a prior node for marginalization instead of performing the Schur complement. The user has the possibility to understand how the prior information affects the rest of the graph, thus allowing him to manipulate this information if desired. If one was to repeatedly perform the Schur complement, it would become untraceable to understand the optimization result of the graph as not all necessary information is conceptually represented herein. The concept of a prior node is also supported by a more pragmatic reason: it allows us to store and load the optimization problem with solely the help of its graph representation. Furthermore, it opens up the possibility to explicitly apply a robust kernel on the cost function of the prior node and to adjust the uncertainty of the prior information based on context. In total, marginalization by using a prior node is making the graph construction logic aware of the marginalization process, allows the understanding and manipulation of the prior information, and is thus preferable over marginalization by performing the Schur complement on the system matrix.

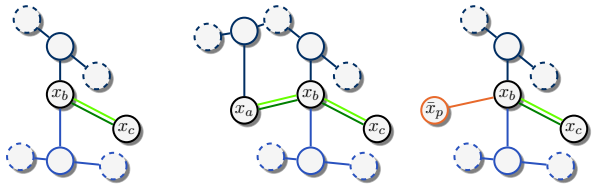
In the following, we will first analyze the impact of the Schur complement on our chain pose graphs and subsequently show, how to compute the uncertainty and mean estimate of the prior node to obtain the same result. The derivation is detailed for the marginalization of a single hidden node but can easily be applied iteratively if multiple nodes shall be marginalized.

1) Effect of the Schur complement on chain pose graphs: It is useful for the derivation to start with a graph with only two hidden nodes (see Fig. 5a) and ignore the node x_a for now to see how the different terms are affected by the marginalization. Consider a graph $\mathcal{G}^{\text{small}}$ where x_b is linked to x_c and additionally to one or more observed nodes (Fig. 5a shows an example for two connected observed nodes, see the blue circles). The corresponding system matrix $\boldsymbol{H}^{\text{small}}$ is given as a block matrix and the optimization solves the equation

$$\begin{bmatrix} \boldsymbol{H}_{bb}^{\text{small}} & \boldsymbol{H}_{bc}^{\text{small}} \\ \boldsymbol{H}_{cb}^{\text{small}} & \boldsymbol{H}_{cc}^{\text{small}} \end{bmatrix} \Delta\boldsymbol{x}^{\text{small}} = - \begin{bmatrix} \boldsymbol{b}_{b}^{\text{small}} \\ \boldsymbol{b}_{c}^{\text{small}} \end{bmatrix}. \tag{4}$$

If we additionally include the hidden node x_a, the graph structure and the system matrix change. Let x_a also be connected to one or more observed nodes and consider the one or more odometry constraints between x_a and x_b. The resulting graph $\mathcal{G}^{\text{full}}$ (see Fig. 5b) is defined by the system matrix

$$\boldsymbol{H}^{\text{full}} = \begin{bmatrix} \bar{\boldsymbol{H}}_{aa} + \hat{\boldsymbol{H}}_{aa} & \hat{\boldsymbol{H}}_{ab} & \\ \hat{\boldsymbol{H}}_{ba} & \boldsymbol{H}_{bb}^{\text{small}} + \hat{\boldsymbol{H}}_{bb} & \boldsymbol{H}_{bc}^{\text{small}} \\ & \boldsymbol{H}_{cb}^{\text{small}} & \boldsymbol{H}_{cc}^{\text{small}} \end{bmatrix} \tag{5}$$



(a) Graph $\mathcal{G}^{\text{small}}$. (b) Graph $\mathcal{G}^{\text{full}}$. (c) Graph $\mathcal{G}^{\text{prior}}$.

Fig. 5. Graph marginalization can be understood as prepending a *prior node* $\bar{x}_p$ to the graph. Blue non-dashed circles represent observed nodes. For understanding how the hidden node x_a influences the optimization, it is useful to start with a graph in (a) without x_a. Considering x_a in (b) leads to additional terms in the system matrix. Marginalization in (c) with a prior node $\bar{x}_p$ leads to the same system matrix as performing the conventional Schur complement on $\mathcal{G}^{\text{full}}$.

and the coefficient vector

$$\boldsymbol{b}^{\text{full}} = \begin{bmatrix} \boldsymbol{b}_a^{\text{full}} \\ \boldsymbol{b}_b^{\text{full}} \\ \boldsymbol{b}_c^{\text{full}} \end{bmatrix} = \begin{bmatrix} \bar{\boldsymbol{b}}_a + \hat{\boldsymbol{b}}_a \\ \boldsymbol{b}_b^{\text{small}} + \hat{\boldsymbol{b}}_b \\ \boldsymbol{b}_c^{\text{small}} \end{bmatrix}, \tag{6}$$

where $\bar{\boldsymbol{H}}_{ij}$ and $\hat{\boldsymbol{H}}_{ij}$ capture the sum of all entries of $\boldsymbol{H}_{ij}$ stemming from observed nodes and odometry constraints, respectively. The terms $\bar{\boldsymbol{b}}_i$ and $\hat{\boldsymbol{b}}_i$ have the same function for the vector entry $\boldsymbol{b}_i$. All of these terms are readily available as they have been computed in the last iteration, including any robust cost function applied on them.

The common way to marginalize x_a consists in computing the Schur complement of $\boldsymbol{H}^{\text{full}}$ and $\boldsymbol{b}^{\text{full}}$. This leads to the marginalized system matrix $\boldsymbol{H}^{\text{marg}}$, which is identical to $\boldsymbol{H}^{\text{small}}$ except for the upper left block which changes to

$$\boldsymbol{H}_{bb}^{\text{marg}} = \boldsymbol{H}_{bb}^{\text{small}} + \boldsymbol{H}_{\text{schur}}, \tag{7}$$

$$\boldsymbol{H}_{\text{schur}} = \hat{\boldsymbol{H}}_{bb} - \hat{\boldsymbol{H}}_{ba}(\bar{\boldsymbol{H}}_{aa} + \hat{\boldsymbol{H}}_{aa})^{-1}\hat{\boldsymbol{H}}_{ab}. \tag{8}$$

The corresponding marginalized coefficient vector is

$$\boldsymbol{b}^{\text{marg}} = \begin{bmatrix} \boldsymbol{b}_b^{\text{small}} + \boldsymbol{b}_{\text{schur}} \\ \boldsymbol{b}_c^{\text{small}} \end{bmatrix}, \tag{9}$$

$$\boldsymbol{b}_{\text{schur}} = \hat{\boldsymbol{b}}_b - \hat{\boldsymbol{H}}_{ba}(\bar{\boldsymbol{H}}_{aa} + \hat{\boldsymbol{H}}_{aa})^{-1}(\bar{\boldsymbol{b}}_a + \hat{\boldsymbol{b}}_a). \tag{10}$$

We remark that after the marginalization, the structure of the system matrix is still block-tridiagonal due to the particular design of our chain pose graph, meaning that the sparsity pattern of $\boldsymbol{H}$ is retained after marginalization without fill-in. Moreover, the information is conserved in a consistent way. Any other marginalization method that claims to be exact, needs to produce the same result. In the remainder of this section, we show that an alternative marginalization technique consists in replacing x_a and its connected observed nodes and edges with a prior node $\bar{x}_p$, which behaves like an observed node. To this end, two questions have to be answered: how to compute its correct mean estimate and how to compute the uncertainty of this prior node?

2) Derivation of the prior node: Our goal is now to derive the equations for computing the prior node $\bar{x}_p$ as depicted in Fig. 5c. First of all, we observe that in the special cases that x_a is not directly connected to any observed node or that there exist no edges between x_a and x_b, x_a does not influence the estimate of x_b and we can omit constructing a prior node. The derivations in the following treat the general case in which x_a is connected to at least one observed node (which can for example be the prior node from the last cycle) and is additionally linked via at least one edge to x_b.

The first step is to remove x_a as well as all observed nodes and edges connected to it. Connecting the prior node $\bar{x}_p$ with the information matrix $\bar{\boldsymbol{\Omega}}_p$ to x_b leads to the graph $\mathcal{G}^{\text{prior}}$ (see Fig. 5c). The prepended node yields an additional addend $\bar{\boldsymbol{H}}_p$ in the matrix $\boldsymbol{H}^{\text{prior}}$ of $\mathcal{G}^{\text{prior}}$ such that

$$\boldsymbol{H}^{\text{prior}} = \begin{bmatrix} \boldsymbol{H}_{bb}^{\text{small}} + \bar{\boldsymbol{H}}_p & \boldsymbol{H}_{bc}^{\text{small}} \\ \boldsymbol{H}_{cb}^{\text{small}} & \boldsymbol{H}_{cc}^{\text{small}} \end{bmatrix}, \tag{11}$$

and an additional addend $\bar{\boldsymbol{b}}_p$ in $\boldsymbol{b}^{\text{prior}}$ such that

$$\boldsymbol{b}^{\text{prior}} = \begin{bmatrix} \boldsymbol{b}_b^{\text{small}} + \bar{\boldsymbol{b}}_p \\ \boldsymbol{b}_c^{\text{small}} \end{bmatrix}. \tag{12}$$

We want to create the prior node so that it behaves like any other observed node in the graph. This already determines that the mean estimate of the edge, which relates $\bar{x}_p$ to x_b, is equal to the mean estimate of any observed node, i.e., $(0, 0, 0)$. This trick simplifies the calculation of the the error function $\boldsymbol{e}_{pb}$ and consequently its Jacobian $\boldsymbol{J}_{pb}(\boldsymbol{x})$ while not limiting the generality of the solution. We obtain

$$\boldsymbol{e}_{pb} = \begin{bmatrix} \boldsymbol{R}_{\theta_p}^\top & \boldsymbol{0} \\ \boldsymbol{0}^\top & 1 \end{bmatrix} (\boldsymbol{x}_b - \bar{\boldsymbol{x}}_p), \tag{13}$$

$$\boldsymbol{J}_{pb}(\boldsymbol{x}) = \begin{bmatrix} \boldsymbol{R}_{\theta_p}^\top & \boldsymbol{0} \\ \boldsymbol{0}^\top & 1 \end{bmatrix}, \tag{14}$$

where $\boldsymbol{R}_{\theta_p}$ is the standard two-dimensional rotation matrix. We need these two terms to compute $\bar{\boldsymbol{H}}_p$ and $\bar{\boldsymbol{b}}_p$:

$$\bar{\boldsymbol{H}}_p = \boldsymbol{J}_{pb}(\boldsymbol{x})^\top \bar{\boldsymbol{\Omega}}_p \boldsymbol{J}_{pb}(\boldsymbol{x}), \tag{15}$$

$$\bar{\boldsymbol{b}}_p = \boldsymbol{J}_{pb}(\boldsymbol{x})^\top \bar{\boldsymbol{\Omega}}_p \boldsymbol{J}_{pb}(\boldsymbol{x})(\boldsymbol{x}_b - \bar{\boldsymbol{x}}_p). \tag{16}$$

In total, we derived how $\bar{x}_p$ influences $\boldsymbol{H}^{\text{prior}}$, $\boldsymbol{b}^{\text{prior}}$ through $\bar{\boldsymbol{H}}_p$, $\bar{\boldsymbol{b}}_p$. We have also shown that the Schur complement influences $\boldsymbol{H}^{\text{marg}}$, $\boldsymbol{b}^{\text{marg}}$ through $\boldsymbol{H}_{\text{schur}}$, $\boldsymbol{b}_{\text{schur}}$. The last step consists in postulating $\bar{\boldsymbol{H}}_p = \boldsymbol{H}_{\text{schur}}$ and $\bar{\boldsymbol{b}}_p = \boldsymbol{b}_{\text{schur}}$ to guarantee that the effect of the prior node is equivalent to the exact marginalization. We solve the resulting system of equations by inserting $\bar{\boldsymbol{H}}_p$ into $\bar{\boldsymbol{b}}_p$, which leads to

$$\bar{\boldsymbol{b}}_p = \bar{\boldsymbol{H}}_p(\boldsymbol{x}_b - \bar{\boldsymbol{x}}_p) \tag{17}$$

$$\Leftrightarrow \boldsymbol{b}_{\text{schur}} = \boldsymbol{H}_{\text{schur}}(\boldsymbol{x}_b - \bar{\boldsymbol{x}}_p) \tag{18}$$

$$\Leftrightarrow \bar{\boldsymbol{x}}_p = \boldsymbol{x}_b - \boldsymbol{H}_{\text{schur}}^{-1}\boldsymbol{b}_{\text{schur}}. \tag{19}$$

As this allows us to calculate $\bar{\boldsymbol{x}}_p$ by using (8) and (10), we can now compute $\bar{\boldsymbol{\Omega}}_p$:

$$\bar{\boldsymbol{\Omega}}_p = (\boldsymbol{J}_{pb}(\boldsymbol{x})^\top)^{-1}\boldsymbol{H}_{\text{schur}}\boldsymbol{J}_{pb}(\boldsymbol{x})^{-1}. \tag{20}$$

These analytic closed-form expressions for $\bar{\boldsymbol{x}}_p$ and $\bar{\boldsymbol{\Omega}}_p$ allow us to position the prior node $\bar{x}_p$ in such a way that the resulting pose estimates for the rest of the graph are identical to the Schur complement marginalization. The mean in combination with the uncertainty estimate provide us with the insight how the marginalization affects the graph. As motivated above, this knowledge can (amongst others) be used to visualize the prior information, adapt its uncertainty, or apply a robust cost function on it.

D. Assessing the uncertainty of the fused estimate

Modeling uncertainty is a key element of probabilistic robotics and obtaining an estimate of the uncertainty of the fused pose estimate is vital for most sophisticated motion planners. We described in the previous sections the process of preprocessing the input data (Section III-B) and constructing a sliding window chain pose graph (Section III-A). The optimization assigns to the hidden nodes the global poses which best satisfy all constraints. In this section we show that we can efficiently recover the uncertainty of the optimized hidden nodes.

Solving the nonlinear least squares problem of the pose fusion with a sparse solver typically involves computing the Cholesky factorization $\boldsymbol{H} = \boldsymbol{R}^\top \boldsymbol{R}$, where $\boldsymbol{R}$ is an upper triangular matrix with entries r_{ij}. Following [13], the uncertainty matrix of the hidden nodes $\boldsymbol{H}^{-1}$ with entries $\boldsymbol{H}_{ij}^{-1}$ is obtained by the recursive formula

$$\boldsymbol{H}_{ii}^{-1} = \frac{1}{r_{ii}} \left[\frac{1}{r_{ii}} - \sum_{\substack{k=i+1 \\ r_{ik} \neq 0}}^{n} r_{ik} \boldsymbol{H}_{ki}^{-1} \right], \tag{21}$$

$$\boldsymbol{H}_{ij}^{-1} = \frac{1}{r_{ii}} \left[- \sum_{\substack{k=i+1 \\ r_{ik} \neq 0}}^{j} r_{ik} \boldsymbol{H}_{kj}^{-1} - \sum_{\substack{k=j+1 \\ r_{ik} \neq 0}}^{n} r_{ik} \boldsymbol{H}_{jk}^{-1} \right]. \tag{22}$$

The formula yields an $\mathcal{O}(n)$ time complexity (with n being the number of state variables) because it operates on the nonzero entries of the sparse band matrix $\boldsymbol{R}$. It becomes constant time for sliding window pose graphs as the number of state variables is upper bounded by a constant value. As we are only interested in the estimated uncertainty of the last hidden node, only a single and comparably trivial calculation has to be performed. We can therefore compute the uncertainty of our pose estimates, and even more, do so efficiently.

E. Noise correlations between input sources

We need to apply appropriate preprocessing if noise is correlated between input sources. This occurs when different input sources build up on the same sensor data or when the same algorithm runs on two physically different sensors. We explicitly account for correlated noise between sources to avoid overconfident estimates. As our approach is agnostic to the specific type of input source, we perform fusion under unknown correlation and employ covariance intersection [14] in a preprocessing step (see [1] for further details). To this end, we identify groups of sources with correlated noise, buffer their measurements, and combine them into a single consistent estimate per group using covariance intersection. These estimates are subsequently entered into the pose fusion process. This results in a reduced number of input sources, in conservative covariance estimates, and avoids the potential threat of divergent or overconfident fusion estimates.

IV. Evaluation

The experimental section is designed to support our claims that the presented sensor fusion approach is an efficient pose estimation algorithm, converges accurately towards the online ML estimate, scales from a filtering to a batch least squares solution, and handles different time behaviors without difficulty. We provide experiments on data gathered on a real prototype vehicle and on simulated data.

A. Experiment on a real prototype vehicle

The following experiment is designed to show that the pose fusion is able to run online on a car and effectively generate pose estimates given a set of real-world input sources. The prototype vehicle is an Audi A6 Avant equipped with a front-facing monocular camera, a front-facing automotive-grade lidar scanner, and four fisheye top view cameras. Three localization systems are incorporated as global pose sources and one system is considered for odometry. The first global pose source (referenced as *pose source 0*) is computed by matching coarse lidar scans to a globally referenced point cloud. The second global pose source is a GPS (*pose source 1*), whereas the third source (*pose source 2*) is a visual localization system, which computes a pose based on comparisons of visual features with a globally referenced feature map. The odometry source is provided by wheel odometry.

The PoseGraphFusion takes these four sources as input and computes pose estimates for the planner. As the noises of the odometry source and the global pose sources are not correlated, we directly fuse them and do not need to apply covariance intersection in this setup. We use $M = 1000$ hidden nodes and set the temporal resolution to $\Delta t = 25\,\mathrm{ms}$, which results in a sliding window over the last $25\,\mathrm{s}$. The test drive was carried out on a route of about $16\,\mathrm{km}$ in rural and urban areas in Germany.

In practice, these sources do not exhibit white Gaussian noises with zero mean. To show this, we evaluate the auto-correlations of their noises and find significant components other than the zero component, meaning that the noises are not independently distributed. Furthermore, the means are unequal to zero. Moreover, pose sources 0 and 1 are third-party black box modules without proper uncertainty models. We therefore assume constant uncertainties for their estimates. Despite this input data not being ideal for the presented fusion approach, we demonstrate the applicability of the fusion and show a decent performance. A detailed analysis with the help of simulated data in Section IV-B evaluates the performance for controlled noise.

1) Pose estimation quality: We evaluate the estimation quality of the PoseGraphFusion by comparing its output to the batch solution. Fig. 6 shows the development of the position error over time. Pose sources 0 and 1 provide only coarse localization estimates with RMS errors of $1.06\,\mathrm{m}$ and $1.23\,\mathrm{m}$ respectively, whereas pose source 2 performs better with an RMS error of $0.28\,\mathrm{m}$. The PoseGraphFusion stays close to the performance of the best pose source with an RMS position error of $0.38\,\mathrm{m}$ and heading error of $1.16°$. Theoretically, the output of the fusion is supposed to improve compared to the input sources. However, the noises are not independently distributed with zero mean and the uncertainty

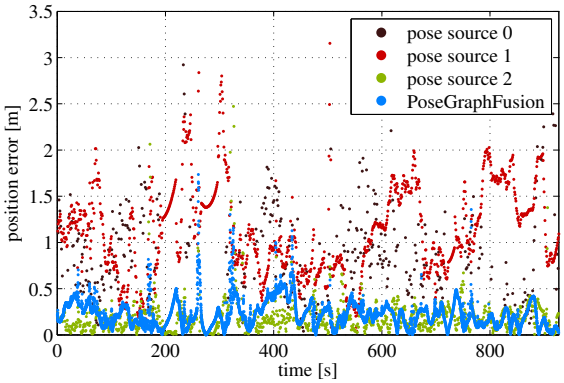


Fig. 6. Position error over time for the three input sources and the fused estimate. The performance of the PoseGraphFusion is linked to the performance of the input sources. Its RMS error is 0.38 m.

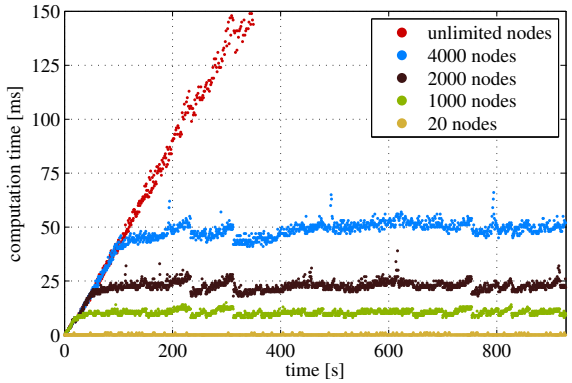


Fig. 7. Computation time for four input sources. The red curve corresponds to a setup in which the number of hidden nodes is unlimited such that nodes never get marginalized out. The resulting unbounded demand for computation time disqualifies it for online usage. The computation time is roughly constant in all other configurations.

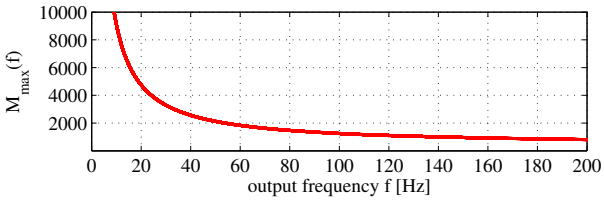


Fig. 8. Parameter space (assuming four input sources). The red curve marks the maximum number of hidden nodes $M_{\max}$ that can still be processed fast enough to attain the corresponding output frequency f.

models are rough as mentioned above. Additionally, the input pose sources are providing estimates in the past and are not always available as we will see in the next evaluation.

2) Latency and availability: As for the current real-world experiment, we demonstrate the capability of our approach to combine the different time behaviors of the pose sources and generate a consistent behavior as output. For this purpose, we examine the latencies of their estimates: the pose sources 0, 1, and 2 all exhibit a latency of up to 0.3 s. This implies that their pose estimates are too old to directly feed them into our planner. In 95% of the time the PoseGraphFusion exhibits a pose latency of 10 ms or less. Moreover, the pose sources are not always able to calculate and provide a pose estimate due to for example sensor or model failures. Consequently, the pose sources have a limited availability, ranging from 66.98% (pose source 0) over 97.76% (pose source 2) to 100% (pose source 1 and odometry) of the time of this experiment. The PoseGraphFusion achieves an availability of 100% of the time. Note that this influences the position accuracy as the pose sources can simply refuse to send a pose estimate in difficult situations whereas the PoseGraphFusion is bound to deliver.

Having shown that our system is capable of integrating different time behaviors into a single consistent output behavior, we study the question whether the proposed algorithm runs fast enough for online usage on a car.

3) Runtime performance: Next we concern ourselves with the runtime performance of the PoseGraphFusion. The software was repeatedly run on a single core of a laptop with an Intel i7-4800QM processor. Fig. 7 shows the computation time at each time step for different numbers of hidden nodes in the sliding window pose graph. The red curve illustrates the need for limiting the size of the graph as otherwise the computation time grows unboundedly and gets quickly too demanding for a decent output frequency. A choice of $M = 4000$ hidden nodes allows us to set the output frequency to $f \approx \frac{1}{50\,\mathrm{ms}} = 20\,\mathrm{Hz}$.

The near-constant computation time once the graph attains its full size is expected as the optimization of a chain pose graph of fixed size is constant as detailed in previous sections. Also, the linear increase in computation time before the number of nodes equals M is in line with our theoretical expectations as the solution of a problem in the form of a chain pose graph has a runtime complexity of $\mathcal{O}(n)$. An empirical analysis of the data depicted in Fig. 7 reveals the reciprocal relationship between the maximum number of hidden nodes $M_{\max}$ and the attainable output frequency f, as shown in Fig. 8. Different parametrizations allow us to balance the need for a high output frequency versus the desire for more hidden nodes. For our requirements we usually choose an output frequency first and subsequently set the number of hidden nodes.

In total, the PoseGraphFusion generates a smooth trajectory and continuously provides accurate pose estimates with a low latency at a configurable output frequency under real-world conditions. Two last questions remain unanswered so far: how many hidden nodes are needed at least to fulfill a given accuracy requirement, and do more hidden nodes actually reduce the estimation error? To answer these questions, we perform two experiments with simulated input data.

B. Simulated input data

The next experiment is conducted on simulated input data and serves to investigate the estimation quality as a function of the number of hidden nodes M. This engages the question of the required number of hidden nodes, illustrates the need of delayed marginalization, and most importantly demonstrates that the sliding window estimate converges to

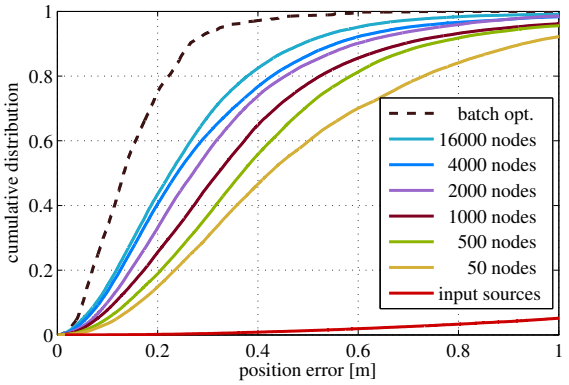


Fig. 9. The cumulative distribution of the position error for different numbers of hidden nodes M. The batch optimization (dashed black line) depicts the theoretically attainable upper bound of accuracy if future information were available. The other configurations demonstrate that the accuracy increases for higher values of M, thus showing the need for delayed marginalization.

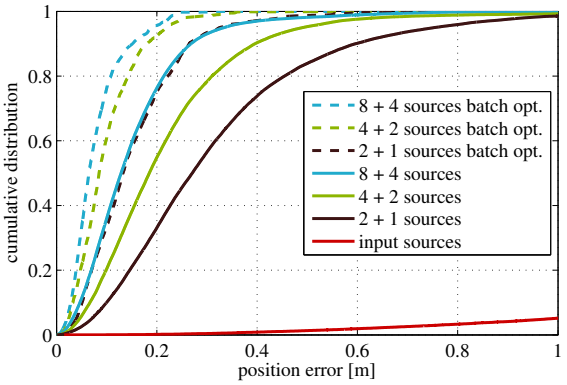


Fig. 10. The cumulative distribution of the position error for different numbers of input sources. The legend entries *"a + b sources"* denote a global pose sources (e.g., GPS) and b odometry sources. The dashed curves are the offline batch optimization results. The red curve shows an example for the quality of a noisy input source. Two of these sources paired with an odometry source suffice as input to the PoseGraphFusion to obtain the accuracy presented as black curve.

the online batch solution for an increasing number of hidden nodes. All inputs are sampled around the true values from a Gaussian distribution with a standard deviation of $3.0\,\mathrm{m}$, $3.0\,\mathrm{m}$ and $4°$ respectively in lateral and longitudinal direction and heading orientation of the vehicle. Fig. 9 shows that choices for M with a decent resulting accuracy lie well within the space of possible parametrizations. Furthermore, the cumulative distribution of the position error improves for more hidden nodes and approaches the batch optimization.

The second experiment with simulated data is designed to show that the position error decreases for an increasing number of sources. The noise terms are identical for all sources and equal to the values mentioned above. Fig. 10 shows how the fusion of inaccurate pose estimates (red curve) leads to a much more accurate estimate and improves further for increasing number of input sources. The online combination of eight global pose and four odometry sources is even roughly as accurate as the offline batch optimization of two global pose sources and one odometry source. This result reinforces the motivation for a pose fusion algorithm.

In summary, a higher number of hidden nodes and more input sources are both advantageous with respect to the position accuracy given certain noise assumptions.

V. Conclusion

The pose fusion concept presented in this paper is motivated by the need for fast, recent, accurate, and highly available pose estimates. We approached these requirements by proposing a sliding window graph-based optimization scheme, which scales from a fast incremental solution to an online batch solution by increasing the sliding window size. Furthermore, we described the design of chain pose graphs for efficient optimization. These graphs do not suffer from additional fill-in after marginalization and allow us to collapse all marginalization information in a prior node. We have shown with experiments on a real vehicle that the PoseGraphFusion is fast, provides timely estimates, is accurate, and yields a high availability. The proposed system works with generic input sources and is general enough to be applied to other autonomous systems.

References

[1] C. Merfels, T. Riemenschneider, and C. Stachniss, "Pose Fusion with Biased and Dependent Data for Automated Driving," in *Proc. Conf. Positioning and Navigation for Intelligent Transportation Syst. (POSNAV)*, 2016, ISSN: 2191-8287.

[2] V. Kubelka, L. Oswald, F. Pomerleau, F. Colas, T. Svoboda, and M. Reinstein, "Robust Data Fusion of Multimodal Sensory Information for Mobile Robots," *Journal of Field Robotics*, pp. 447–473, 2014.

[3] S. Weiss, M. W. Achtelik, M. Chli, and R. Siegwart, "Versatile Distributed Pose Estimation and Sensor Self-calibration for an Autonomous MAV," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, 2012, pp. 31–38.

[4] S. Lynen, M. W. Achtelik, S. Weiss, M. Chli, and R. Siegwart, "A Robust and Modular Multi-Sensor Fusion Approach Applied to MAV Navigation," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*, 2013, pp. 3923–3929.

[5] H. Strasdat, J. Montiel, and A. J. Davison, "Visual SLAM: Why filter?" *Image and Vision Computing*, vol. 30, no. 2, pp. 65–77, 2012.

[6] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. Leonard, and F. Dellaert, "iSAM2: Incremental Smoothing and Mapping using the Bayes tree," *Int. Journal of Robotics Research*, pp. 216–235, 2012.

[7] H. P. Chiu, S. Williams, F. Dellaert, S. Samarasekera, and R. Kumar, "Robust vision-aided Navigation using Sliding-Window Factor Graphs," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, 2013, pp. 46–53.

[8] V. Indelman, S. Williams, M. Kaess, and F. Dellaert, "Factor Graph Based Incremental Smoothing in Inertial Navigation Systems," in *Proc. Int. Conf. Inform. Fusion (FUSION)*, 2012, pp. 2154–2161.

[9] D. Cucci and M. Matteucci, "A Flexible Framework for Mobile Robot Pose Estimation and Multi-Sensor Self-Calibration," in *Proc. Int. Conf. Inform. in Control, Automation and Robotics (ICINCO)*, 2013, pp. 361–368.

[10] G. Sibley, L. Matthies, and G. Sukhatme, "Sliding Window Filter with Application to Planetary Landing," *Journal of Field Robotics*, vol. 27, no. 5, pp. 587–608, 2010.

[11] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, "g2o: A General Framework for Graph Optimization," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, 2011, pp. 3607–3613.

[12] R. Kümmerle, "State Estimation and Optimization for Mobile Robot Navigation," Ph.D. dissertation, University of Freiburg, 2013.

[13] M. Kaess and F. Dellaert, "Covariance Recovery from a Square Root Information Matrix for Data Association," *Robotics and Autonomous Systems*, pp. 1198–1210, 2009.

[14] S. J. Julier and J. K. Uhlmann, "A non-divergent estimation algorithm in the presence of unknown correlations," in *Proc. Amer. Control Conf. (ACC)*, 1997, pp. 2369–2373.