Efficient 6-DOF SLAM with Treemap as a Generic Backend

Udo Frese
SFB/TR 8 Spatial Cognition
Universität Bremen
D-28334 Bremen, Germany
ufrese@informatik.uni-bremen.de

***Abstract*— Treemap is a generic SLAM algorithm that has been successfully used to estimate extremely large 2D maps closing a loop over a million landmarks in 442ms. We are currently working on an open-source implementation that can handle most variants of SLAM. In this paper we discuss the generic part of the algorithm constituting the treemap backend and the variant specific parts acting as a driver. We present their interplay from a software-engineering point of view and show results for the case of 6-DOF feature based SLAM, closing a simulated loop over 106657 3D features in 209ms.**

I. Introduction

Simultaneous Localization and Mapping (SLAM) has been a central topic in mobile robotics research for almost two decades by now [1]. Most of the literature is concerned with generating a 2D map with a sensor moving in the plane (3-DOF). Only in the last few years the problem of generating a 3-D map with a sensor moving in 3D space (6-DOF) has received considerable attention [2]–[5]. Such a system has important applications, for instance rescuing victims from the remains of a collapsed building. So we expect that 6-DOF SLAM will be a growing research area, in particular with the recently emerging 3D cameras.

Many 2D SLAM articles address the problem of efficiency in estimating large maps (see [6] for an overview). It becomes critical when the map is by orders of magnitude larger than the sensor range because then issues such as closing large loops come up. 3D maps always contain a lot of data, so are large from a storage space perspective, but up to now little attention has been paid to large loops and similar issues that make 2D SLAM difficult. Rather, in 6-DOF SLAM the efficiency discussion has mainly focused on the first stages of processing in particular on 3D scan matching [7].

We contributed the treemap algorithm [8] to the efficiency discussion in 2D SLAM. It is designed for computing least square estimates for very large maps efficiently. Using treemap we were able to demonstrate closing a simulated loop with one million landmarks in 442ms [9]. On the one hand, the treemap algorithm is sophisticated but also complicated. On the other hand, it is fairly general mainly estimating random variables of arbitrary meaning. Hence our current project is to develop an open source implementation that – as an *implementation* – can be used to perform most variants of SLAM including 2D, 3D, features and/or poses, and visual SLAM. It consists of the generic treemap backend that is used by all variants plus a variant specific driver that implements the specific observation model.

Part of this article has appeared on the local workshop "Robotic 3D Environment Cognition" in Bremen.

First, the paper contributes this general approach for facilitating software reuse in SLAM research covering both algorithmic and software-engineering questions. Second, it reports first results showing a simulated 6-DOF SLAM experiment (3D features, no odometry) that uses the same implementation as our previous million-landmarks (2D features, odometry, marginalized poses) experiment. By building on the efficiency of treemap as a backend, we where able to close a loop over $n = 106657$ 3D features in 209ms. To our knowledge no comparable result has been reported in 6-DOF SLAM so far. As with the previous 2D experiment we use simulations and concentrate on the least square estimation algorithm because sensor preprocessing and feature extraction depends much more on the specific sensor and setting and will unlikely be reusable. Data association is also not addressed here, i.e. the simulation provides the correct identity of all features. We are confident though to incorporate it in future, because treemap, as a direct equation solver, can provide covariance information.

II. Local and Global Challenges

Many challenges currently addressed in 6-DOF SLAM concern the first stages of sensor processing: matching 3D scans, finding reliable features, matching them, rejecting outliers, filtering range images or handling bearing-only initialization. These problems are local in the sense that they affect only that part of the map that is currently observed by the sensor. In contrast there is also the question how this local information and its uncertainty affects the global map. The most prominent situation is certainly closing a loop. Then the local information that closes the loop leads to back-propagation of the error along the loop. The key point, as we have argued in [6, §12], is that the local uncertainty is small but complex and depends on the actual sensor and the detailed circumstances of observation, whereas the global uncertainty is mostly the composition of local uncertainties. In particular orientation error of the robot leads to correlated position error of the robot and all features later on. This error depends on the magnitude of the orientation error and on the map's geometry. It is mostly independent from local details that lead to that orientation error. Hence the large

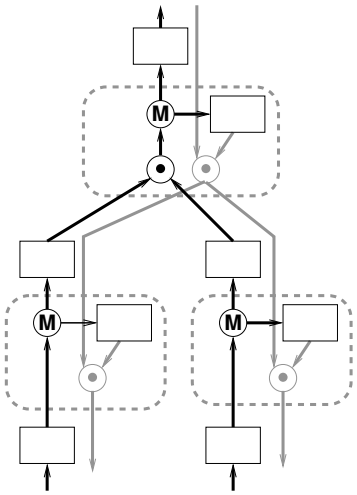

Fig. 1. Data flow of the probabilistic computations performed by treemap.

scale map error is large, rather simple and dominated by the map's geometry.

This insight motivates our treemap approach. In the past it has motivated the design of the treemap algorithm itself that exploits this locality. And now it motivates our idea that many different SLAM variants (2D / 3D, features and/or poses, with/without odometry) can be solved by a specific local preprocessing plus treemap as a global least-square backend. From this motivation our goal is:

> Whenever you can formulate your SLAM approach in a least-square framework such that it works for small maps, you can use treemap as a backend to make it work for large maps.

The paper is organized as follows. Section III briefly sketches the treemap algorithm. Section IV discusses the different SLAM variants. Section V reports on the proposed architecture for achieving re-usability on the software level, section VI discusses the different levels of approximation and finally section VII presents 6-DOF experiments closing a 106657 feature loop in 209ms.

III. The Treemap Algorithm

This section briefly sketches the treemap algorithm [8], [9]. However, for the purpose of this paper treemap is just a black box that incrementally receives a set of local Gaussians and computes the mean of the product of these Gaussians[1]. Treemap is related to TJTF [10] and in some aspects to Dellaert's multifrontal QR-approach [11] and Atlas [12].

Imagine the robot is in a building that is virtually divided into two parts A and B. Consider: *If the robot is in part A, what is the information needed about B?* Only few of B's features are involved in observations with the robot in A. All other features are of no interest at the moment. So intuitively, we need everything observations in B can tell on features also observed from A. Or probabilistically speaking: the marginal distribution of features observed from both A and B conditioned on observations in B. This idea is applied recursively dividing the map into a tree of subregions.

These marginal distributions are computed along the tree (Fig. 1). The computation starts at the leaves which store the local Gaussians input to treemap (black upwards arrows). In each leaf features are marginalized out (Ⓜ) that are not involved outside that leaf any more. Their information is stored as a conditional at the leaf, the marginal is passed to the parent. There it is multiplied (⊙), i.e. integrated, with the corresponding marginal of the other child. Again features only involved below this node are marginalized out (Ⓜ). Their information is stored at the node as a conditional while the marginal is again passed upwards.

To compute an estimate (grey downwards arrows) the mean of the features marginalized out at the root is computed and passed to the root's children. There it is combined with the stored conditional (⊙) to compute the mean of the features marginalized out there and passed downwards again.

The key points for treemap's efficiency are a) many small matrices instead of one large; b) when a new local Gaussian, i.e. a new leaf, is added, only the distributions from this leaf up to the root need to be updated; c) the downward propagation step is essentially a matrix vector product and hence extremely fast. All this depends on the maps topology, requiring that it can be recursively divided into halves with little overlap. This is usually true for buildings, as in the simulation conducted here, but not for a large open plane ("mowing the lawn"). To find such a recursive subdivision treemap executes an optimization algorithm in parallel that tries to improve the tree by moving subtrees around.

IV. Different SLAM Variants

This section discusses different popular variants of SLAM, the random variables involved, their degrees of freedom, whether old robot poses are marginalized out ("forgotten"), and special considerations necessary. An overview is shown in table I. It can be seen that there are many different variants making it very worthwhile to have a single backend implementation with as little variant specific code as possible. The discussion also serves to point out which estimation techniques a generic SLAM algorithm should support.

A. 2D SLAM

In 2D SLAM two variants are most popular [1]. The first is *consistent pose estimation* where 3-DOF poses are estimated from 3-DOF links derived from odometry and scan matching [13]. This is the simplest variant, since no features are needed and nothing is marginalized out. The second is the classical variant with 2D point features and 3-DOF poses where old poses are marginalized out [14]. Marginalization is implicit in the traditional EKF but destroys sparsity in an information matrix based approach. Hence, either poses must be kept at regular intervals or sparsification is required as an additional approximation (cf. section VI).

If line features, e.g. walls, are mapped instead of point features, additionally the problem of how to parameterize lines appears [15]. Some sensors provide no range but

[1]Thanks to the anonymous reviewers for noting this "modular" view.

		features	poses	marginalization	sparsification	bearing only	parametrization
[13]	Consistent pose estimation		3 DOF				
[14]	Point feature based 2D SLAM with odometry	2 DOF	3 DOF	yes	optional		
[15]	Horizontal line feature based 2D SLAM with odometry	2 DOF	3 DOF	yes	optional		yes
[16]	Vertical line feature based 2D visual SLAM	2 DOF	3 DOF	yes	optional	yes	
	3D consistent pose estimation with inclinometer		4 DOF				
	Point feature based 3D SLAM with inclinometer	3 DOF	4 DOF	yes	optional		
[2]	6-DOF consistent pose estimation		6 DOF				yes
	Point feature based 3D SLAM without odometry	3 DOF	6 DOF	yes			yes
	+ inertial rotation	3 DOF	6 DOF	yes	optional		yes
	+ inertial translation	3 DOF	9 DOF	yes	optional		yes
[4]	Point feature Visual SLAM	3 DOF	6 DOF	yes		yes	yes
	+ inertial rotation	3 DOF	6 DOF	yes	optional	yes	yes
[17]	+ inertial translation	3 DOF	9 DOF	yes	optional	yes	yes

TABLE I

OVERVIEW OF DIFFERENT SLAM VARIANTS; NUMBER OF DEGREES OF FREEDOM NEEDED FOR FEATURES AND POSES; WHETHER MARGINALIZATION OR SPARSIFICATION IS USED; WHETHER THE OBSERVATIONS LACK DISTANCE SUCH THAT BEARING ONLY INITIALIZATION IS NEEDED; WHETHER GENERAL MANIFOLDS (ROTATIONS, LINES) NEED TO BE PARAMETERIZED.

bearing only. A notable example in 2D is a camera detecting vertical lines. This leads to the additional problem of how to initialize a feature with unknown distance [16].

B. 3D (4-DOF) SLAM

An intermediate variant between 2D and 3D consists of 3D point features but a robot still moving in 2D. The same happens if the robot pose is 3D but the orientation with respect to gravity is measured by inclinometers. In both cases the robot's orientation can still be represented by a single angle making the pose 4-DOF altogether. This kind of SLAM is basically 2D with an additional Z-coordinate.

C. 3D (6-DOF) SLAM

In full 6-DOF SLAM more variants are possible. Using 3D scan matching [2] one can do 6-DOF consistent pose estimation. 6-DOF feature based SLAM can for instance be conducted using a stereo-camera that measures the position of point features relative to the camera. In contrast to 2D SLAM usually no odometry is available. This gives rise to a simple variant, where poses are marginalized out immediately. Since there is no odometry, sparsity is maintained and no sparsification is needed. Essentially, this means, that each set of observations is converted into relative information on 3D point features (cf. experiments in section VII).

This approach has a major limitation. Without odometry a small sensor blackout or too little overlap between observations will disintegrate the map because no information links the involved two poses anymore. Inertial sensors can help by providing relative orientation (gyros) and absolute inclination (accelerometers) [17]. This is the pendant of classic SLAM with 3D point features and 6-DOF poses marginalized out later. Again, either some poses must be kept or sparsification is needed. Still, with orientation-odometry only, consecutive observations must share one feature. Yet another variant uses the accelerometers as translation-odometry. But when acceleration is integrated the result is relative velocity not relative position, so the poses must be augmented by 3D velocity (9-DOF total) [17].

All 6-DOF SLAM variants share the problem of parameterizing 3D orientation because there is no singularity free parameterization of orientation with 3 parameters.

With a monocular camera [4], no distance can be measured. So, while consistent pose estimation can use the 5-DOF links arising from matching two images [18], additional information is needed for the overall scale. In a feature based approach this leads to the corresponding problem of bearing-only initialization [19].

D. Nonlinearity

Nonlinearity is yet another question. It is rather orthogonal to the different variants because all of them are nonlinear. So it is a matter of sensor noise and map size how large the linearization error is and whether efforts to reduce it are needed. Possible options include repeatedly updating the linearized Gaussians with the current estimate as new linearization point and rotating Gaussians to specifically compensate error in the orientation [20].

V. DRIVER - BACKEND ARCHITECTURE

Figure 2 shows the architecture we propose in this paper consisting of a generic backend and a driver that depends on the specific SLAM variant and eventually even on the application. The driver mainly implements routines for computing linearized Gaussians from the original measurements and an approximation policy that sets appropriate control flags.

When a new observation arrives the driver first checks whether it involves new features or robot poses. If this is the case it allocates the appropriate number of 1-dimensional random variables in the backend. It is important, that the backend only handles plain 1-D Gaussian random variables because in the different variants features and poses have widely varying degrees of freedom. Any hard coded block matrix layout could not handle this. After that, the driver initializes these new random variables with a rough initial estimate in a way that depends on the specific sensor.

The next step is to compute a Gaussian by linearizing the measurement equations and to add it together with the

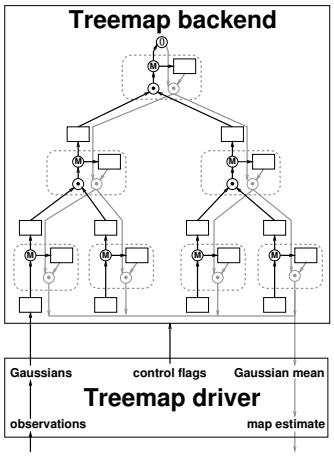

Fig. 2. Data flow between treemap backend and driver. The driver adds new leaves computed from new observations to the tree (here shown for the leftmost leaf). It further sets flags *invalid*, *integrable* for leaves as well as *marginalizable*, *sparsifyable* for random variables to control approximations conducted by the backend. In turn it receives the Gaussians mean and converts it into a map estimate.

original nonlinear measurements to the treemap. As long as the leaf keeps the original measurements the linearized Gaussian can be recomputed whenever desired by the driver's approximation policy to reduce linearization error.

A subtle issue is what linearization point to use when integrating a measurement the first time. Especially when closing a loop the prior estimate can be arbitrarily wrong due to accumulated error and the measurement itself can often provide a much better linearization point. Once the measurement is integrated the estimate incorporates information from the measurement itself and all other measurements so the estimate provides a better linearization point.

After the Gaussian is passed to the treemap backend, it updates its internal data-structures, i.e. the tree and the distributions stored there. Then it computes the mean of the overall Gaussian. The driver then converts the computed mean into a map estimate. Usually this means just to copy numbers. However if the driver uses more sophisticated parameterizations, for instance for lines or rotations, it must convert from these parameters to the representation passed to the application.

The driver further implements a specific approximation policy. It decides when the backend is allowed to integrate two leaves, marginalize out a random variable, or sparsify it out. The strategy involved depends on the variant and eventually on the specific application. It interacts with the treemap backend by setting flags in leaves and random variables whereas the actual operations are performed in the backend. So the driver defines whether an approximation is allowed and the backend both decides when to do it and actually conducts the approximation.

VI. Different Levels of Approximation

This section describes the different levels of approximation the treemap backend offers (Fig. 3). Such a variety of ways to trade computation time vs. accuracy greatly facilitates

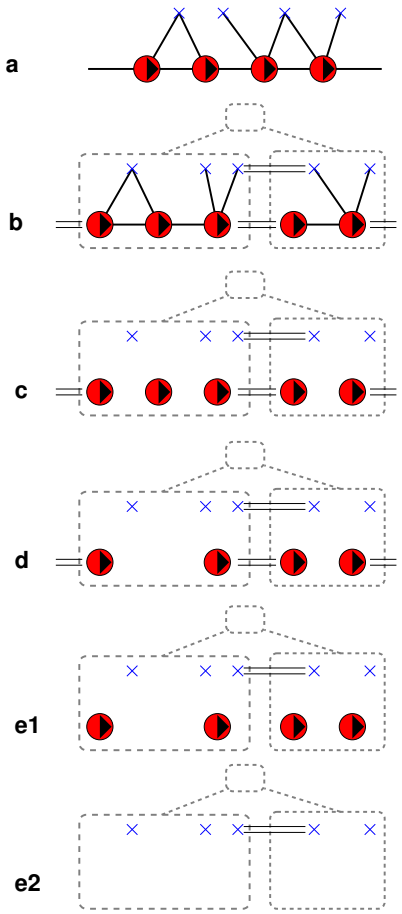

Fig. 3. Different levels of approximation. a) original nonlinear observations b) keep nonlinear observations c) linearize d) marginalize out old poses e) sparsify out old poses by first sacrificing the equality constraint and then marginalizing them out.

generic usage because different application may have different restrictions in both. The original nonlinear observations are illustrated in Fig. 3a. The circles depict robot poses and the crosses depict features. The black lines depict odometry and feature observations respectively.

A. Keep nonlinear observations

Treemap is a linear equation solver, so the original nonlinear observations are assigned to leaves of the tree (dashed oval) and converted into linearized Gaussians for each leaf (Fig. 3b). Some features or robot poses are involved in several leaves. The treemap bookkeeping knows these are the same and takes care of this fact when computing the mean. This is depicted by the '=' connections in the figure. As long as the original nonlinear measurements are kept, the driver can decide to update the linearization. It then recomputes the linearized Gaussian and flags the leaf *invalid* triggering necessary updates in the backend.

B. Linearize

Conversely if the driver decides it will not recompute a leaf any more, it can flag the leaf *integrable*. As a consequence, the treemap backend can integrate this leaf with another

leaf representing both together in a single Gaussian (such as adding information matrices). Any nonlinear information associated with one of the leaves is discarded, finally fixing the linearization point (Fig. 3c). With this strategy observing the same features repeatedly does not lead to a growing representation. When the driver allows to linearize a leaf, the backend checks whether computation time can be reduced by integrating it with another leaf. This is part of the general process that optimizes the tree representation.

C. Marginalize out old poses

Often the application is not interested in old robot poses. The driver can then flag a random variable as *marginalizable* indicating that it is not needed any more. If the random variable is only involved in one leaf it can be marginalized (Fig. 3d) out with the result replacing the original Gaussian. This certainly precludes that the driver can relinerize the Gaussian so the leaf must be flagged *integrable*. If a *marginalizable* random variable is involved in several leaves the treemap backend actively tries to integrate those leaves to be able to marginalize it out.

D. Sparsify out old poses

Not all old robot poses can be marginalized out this way because then all leaves would be integrated into one leaf. Instead, the optimization algorithm implicitly determines a trade-off between getting too large leaves and keeping too many old poses. The result has typically much fewer robot poses than features posing no computational problem. Up to this point the only approximation is linearization and as Eustice et al. noted [21] the result is exactly sparse in information form.

Still the driver can decide to trade in accuracy for efficiency and flag a remaining robot pose *sparsifyable*. Figure 3e discusses the effect. Each occurrence of the robot pose is marginalized out of its leaf independently which is equivalent to treating them as two different poses. So the information that both are the same is sacrificed for the sake of keeping the tree sparse. This sparsification is the same as used by TJTF [10] and related to 'cutting the odometry sequence' [22] and 'relocation' [23]. Remarkably it does not introduce overconfidence.

Since sparsification involves loss of information the optimization algorithm treats it as a last resort used only if there is not other way to improve the tree. Even then, it asks the driver that can implement a policy such as sparsifying out only every n-th robot pose. Note, that sparsification is the reason why we didn't choose a large sparse matrix as the interface to the treemap backend. Because once all local Gaussians are added into one matrix, as with SEIF, it is hard to consistently sparsify without inverting the whole matrix [24].

We used sparsification in our 2D million-landmarks experiment where we kept 48690 poses, sparsified out 285968 and marginalized out 3373643. The experiments reported here do not need sparsification since there is no odometry and hence all robot poses can be marginalized out.

VII. 6-DOF SLAM Experiments

The goal of our current project is to implement all the different SLAM variants. The treemap backend is already finished with the described generic interface so we were able to implement a driver for feature based 2D SLAM [9] in 690 lines of C++ code and a driver for feature based 6-DOF SLAM without odometry in 410 lines of code. Actually the 6-DOF implementation is shorter because no odometry is involved and we needed some special implementation effort to limit the memory consumption in our 2D million-landmarks experiment. This section shows results for the 6-DOF implementation.

A major point for 6-DOF SLAM is how to parameterize rotations and how to compute an initial estimate for linearization. We extend a technique by Castellanos [15] to 3D and use the product

$$\begin{aligned} Q =& Q_0 \begin{pmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{pmatrix} \\ \approx& Q_0 \begin{pmatrix} 1 & -\gamma & \beta \\ \gamma & 1 & -\alpha \\ -\beta & \alpha & 1 \end{pmatrix} \end{aligned} \tag{1}$$

of a fixed orientation Q_0 and three Euler rotations the angles of which are the random variables estimated. Q_0 is initialized with the initial estimate so the Euler angles only parameterize the small *perturbation* of the orientation and are far from singularity. Hence they are always linearized at $\alpha = \beta = \gamma = 0$ and with the linearization shown above.

This technique would in principle allows to reduce the linearization error caused by error in the robot orientation [20]. The distributions passed from a node's children can be rotated according to the current estimate before multiplying them (Fig. 1, $\odot$). We have used this technique for 2D SLAM before [8] closing a loop with $135°$ orientation error, however have not implemented it in the 6-DOF version yet.

Since there is no odometry the initial estimate must be computed from the observed features. We do so by least-square matching all observations of features already in the map using an SVD based closed solution [25]. Then we initialize new features based on the resulting robot pose.

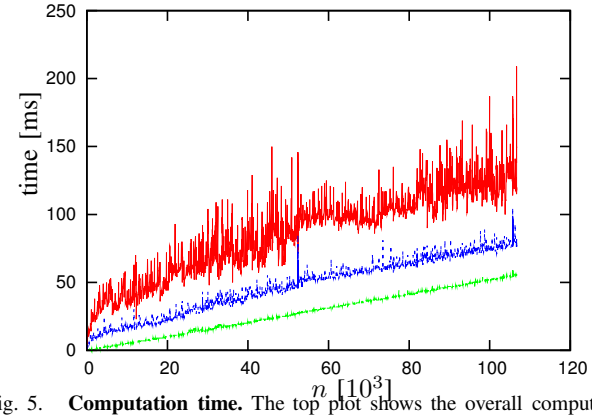

Fig. 5. **Computation time.** The top plot shows the overall computation time per step over the number of features. The two bottom plots divide this time, bottom to top, into downward estimation (mainly backsubstitution), bookkeeping, and upward update (mainly QR-decomposition).

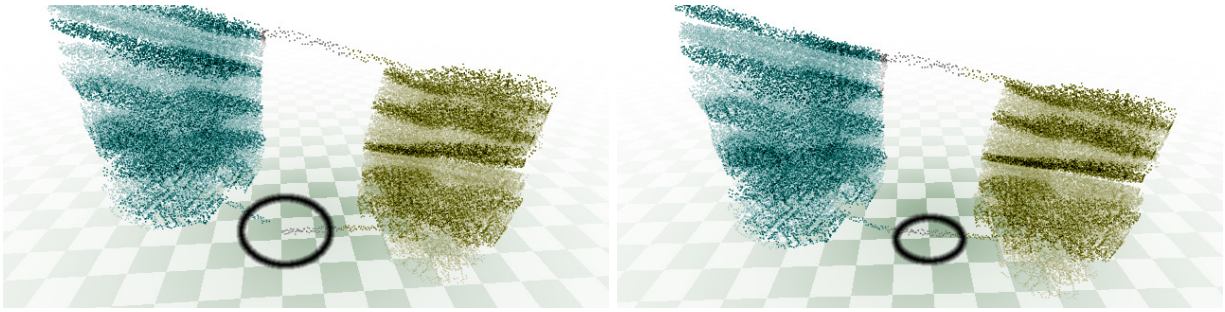

Fig. 4. **6-DOF SLAM map.** before and after closing the large loop (between the two building on the ground level) over all $n = 106657$ features. The black ellipses show the region where the loop is closed. The whole mapping process can be seen as 3D animation accompanying the article.

In our simulated experiment the robot moves through a 20 story building with features on the room's walls (Fig. 4, accompanying video). Then it crosses a bridge on the 19^{th} floor into another 20 story building and maps that building too. Finally it returns to the starting position and closes a loop over all features. The overall map has $n = 106657$ features and $m = 5319956$ observations from $p = 488289$ poses. Poses are not represented in the map. Computation time was extremely fast with at most 209ms (Fig. 5).

The tree has 21743 nodes, i.e. 10871 leaves. Since initially for each pose a new leaf is added, approximately 45 leaves have been integrated into a single leaf by the treemap backend, highlighting the importance of this mechanism.

VIII. Conclusion and Outlook

We have demonstrated that the treemap algorithm in the same generic implementation can be used to solve both 2D and 3D feature based SLAM (without odometry) with high efficiency. We have discussed the general algorithmic approach and software architecture that allows to extend this approach to the remaining SLAM variants. Future work includes implementing those variants, integrating a solution to the bearing-only initialization problem and implementing a 3D variant of the rotation technique used to reduce linearization error. We then plan to publish the implementation as an open source library.

References

[1] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. MIT Press, 2005.

[2] H. Surmann, A. Nüchter, K. Lingemann, and J. Hertzberg, "6D SLAM - preliminary report on closing the loop in six dimensions," in *Proceedings of the 5th Symposium on Intelligent Autonomous Vehicles, Lissabon*, 2004.

[3] J. V. Miro, G. Dissanayake, and W. Zhou, "Vision-based SLAM using natural features in indoor environments," in *Proceedings of the 2005 IEEE International Conference on Intelligent Networks, Sensor Networks and Information Processing*, 2005.

[4] A. Davison, Y. Cid, and N. Kita, "Real time SLAM with wide angle," in *Proc. IFAC Symposium on Intelligent Autonomous Vehicles, Lisbon*, 2004.

[5] K. Ohno and S. Tadokoro, "Dense 3D map building based on LRF data and color image fusion," in *Proceedings of the International Conference on Intelligent Robots and Systems*, 2005, pp. 1774–1779.

[6] U. Frese, "A discussion of simultaneous localization and mapping," *Autonomous Robots*, vol. 20, no. 1, pp. 25–42, 2006.

[7] S. Rusinkiewicz and M. Levoy, "Efficient variants of the ICP algorithm," in *Proceedings of the Third International Conference on 3-D Digital Imaging and Modeling, Quebec City*, 2001, pp. 145 – 152.

[8] U. Frese, "Treemap: An $O(\log n)$ algorithm for indoor simultaneous localization and mapping," *Autonomous Robots*, vol. 21, no. 2, pp. 103–122, 2006.

[9] U. Frese and L. Schröder, "Closing a million-landmarks loop," in *Proceedings of the IEEE/RSJ Intern. Conf. on Intelligent Robots and Systems, Beijing*, 2006, pp. 5032–5039.

[10] M. Paskin, "Thin junction tree filters for simultaneous localization and mapping," in *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, San Francisco, 2003, pp. 1157–1164.

[11] F. Dellaert, A. Kipp, and P. Krauthausen, "A multifrontal QR factorization approach to distributed inference applied to multi-robot localization and mapping," in *Proceedings of the American Association for Artificial Intelligence*, 2005.

[12] M. Bosse, P. Newman, J. Leonard, and S. Teller, "SLAM in large-scale cyclic environments using the Atlas framework," *International Journal on Robotics Research*, vol. 23, no. 12, pp. 1113–1140, 2004.

[13] F. Lu and E. Milios, "Globally consistent range scan alignment for environment mapping," *Autonomous Robots*, vol. 4, no. 4, pp. 333 – 349, 1997.

[14] R. Smith, M. Self, and P. Cheeseman, "Estimating uncertain spatial relationships in robotics," in *Autonomous Robot Vehicles*, I. Cox and G. Wilfong, Eds. Springer Verlag, New York, 1988, pp. 167 – 193.

[15] J. Castellanos, J. Montiel, J. Neira, and J. Tardós, "The SPmap: A probablistic framework for simultaneous localization and map building," *IEEE Transactions on Robotics and Automation*, vol. 15, no. 5, pp. 948 – 952, Oct. 1999.

[16] M. Deans and M. Herbert, "Experimental comparison of techniques for localization and mapping using a bearings only sensor," in *Proc. of the ISER '00 Seventh International Symposium on Experimental Robotics*, 2000.

[17] J. Kim and S. Sukkarieh, "Uav navigation: Airborne inertial SLAM," Tutorial at IROS 2005, 2005.

[18] R. Eustice, H. Singh, J. Leonard, M. Walter, and R. Ballard, "Visually navigating the RMS titanic with SLAM information filters," in *Proceedings of Robotics Science and Systems, Boston*, 2005.

[19] J. Montiel, J. Civera, and A. Davison, "Unified inverse depth parametrization for monocular SLAM," in *Proceedings of Robotics: Science and Systems, Pennsylvania*, 2006.

[20] J. Folkesson, P. Jensfelt, and H. I. Christensen, "Vision SLAM in the measurment subspace," in *Procedings of the IEEE International Conference on Robotics and Automation, Barcelona*, 2005.

[21] R. Eustice, H. Singh, and J. Leonard, "Exactly sparse delayed state filters," in *Proceedings of the Internation Conference on Robotics and Automation, Barcelona*, 2005, pp. 2428–2435.

[22] U. Frese and G. Hirzinger, "Simultaneous localization and mapping - a discussion," in *Proceedings of the IJCAI Workshop on Reasoning with Uncertainty in Robotics, Seattle*, Aug. 2001, pp. 17 – 26.

[23] M. Walter, R. Eustice, and J. Leonard, "A provably consistent method for imposing exact sparsity in feature-based SLAM information filters," in *Proceedings of the 12th International Symposium of Robotics Research*, 2005.

[24] R. Eustice, M. Walter, and J. Leonard, "Sparse extended information filters: Insights into sparsification," in *Proceedings of the International Conference on Intelligent Robots and Systems, Edmonton*, 2005.

[25] D. Eggert, A. Lorusso, and R. Fisher, "Estimating 3-d rigid body transformations: A comparison of four major algorithms," *Machine Vision and Applications*, vol. 9, no. 5/6, pp. 272–290, 1997.