Experiences in Building a Visual SLAM System from Open Source Components

Christoph Hertzberg†, René Wagner*, Oliver Birbach*, Tobias Hammer†, and Udo Frese*†
†FB3 – Mathematik und Informatik, Universität Bremen, 28359 Bremen, Germany
{chtz,hammer}@informatik.uni-bremen.de
*Deutsches Forschungszentrum für Künstliche Intelligenz GmbH, 28359 Bremen, Germany
{rene.wagner,oliver.birbach,udo.frese}@dfki.de

***Abstract*— This paper shows that the field of visual SLAM has matured enough to build a visual SLAM system from open source components. The system consists of feature detection, data association, and sparse bundle adjustment. For all three modules we evaluate different libraries w.r.t. ground truth.**

We also present an extension of the SLAM system to create dense voxel-maps. It employs dense stereo-matching and volumetric mapping using the poses obtained from bundle adjustment, both implemented with open source libraries.

Apart from quantitative comparison we also report on specific experiences with the various libraries.

I. Introduction

Research into the *simultaneous localization and mapping* (SLAM) problem [1] has seen a certain level of consolidation over the past few years. 2D (indoor-)SLAM based on laser rangefinder data is largely considered a solved problem and the availability of open source packages [2], [3] enables even users unfamiliar with the internals of the underlying algorithms to deploy autonomously navigating mobile robots [4].

Visual SLAM, i.e., SLAM using cameras, is currently approaching a similar consolidation phase – although a complete, accessible solution is not yet available, all required building blocks exist with open source implementations. In this paper, we share our experiences combining these into a fully-functional, deep prototype of a visual SLAM system in C++. The architecture of the system is shown in Fig. 2 and follows the usual pattern. First, keypoints are detected in the images. Then points are matched to establish which observations correspond to the same physical feature (data association). Finally, feature positions and camera poses are estimated, by fitting the measurement equations for the feature observations. The latter has been studied as *bundle adjustment* in photogrammetry [5], as *structure from motion* in computer vision [6], and *visual SLAM* in robotics [7].

For these three modules we contribute a quantitative comparison of the performance of different available libraries w.r.t. ground truth. We extended the system to compute a dense map (Fig. 2, bottom) using stereo-matching and volumetric mapping in open source implementations. Here we present early results, but no extensive evaluation.

Our experiment (Fig. 1) focuses on applications in urban search and rescue scenarios consisting of unstructured, highly cluttered environments as typically found in, e.g., collapsed buildings. The general architecture and the implementations used are also suitable in other application areas.

This work has partly been supported under DFG grant SFB/TR 8 Spatial Cognition and BMBF grant 01IS09044B.

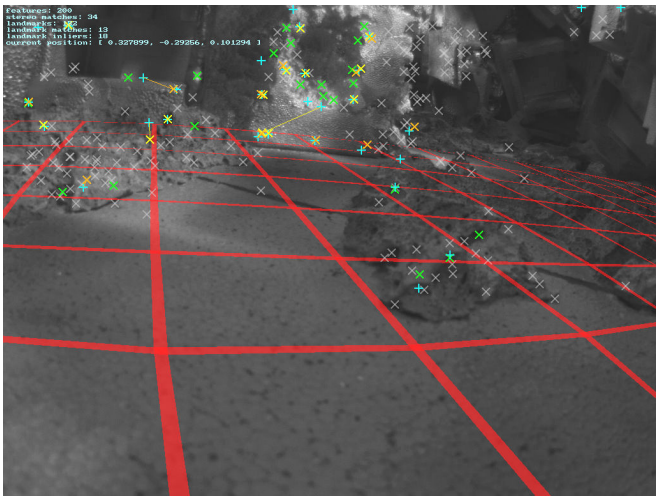

Fig. 1. A screenshot of our visual SLAM system built from open source components. Features with different status are shown as crosses: green: stereo-matched, orange/yellow: fused with the map (monocular/stereo), cyan: reprojected from the map. A red grid is projected onto the ground plane for visually assessing the precision of the computed camera poses. Colors correspond to Fig. 2 and the full experiment is attached as a video.

We use a stereo camera (Fig. 2, left) to avoid scale ambiguity and combine it with an inertial measurement unit (IMU). This has three advantages: Temporary outages of the stereo keypoint input (e.g., due to occlusion) can be compensated for, the required frame rate of the vision component is reduced, and the map has a defined up and down.

This paper is structured as follows. After related work in Section II we discuss the main modules in Sections III to VI, each with module-specific experiments. Section VII presents calibration and Section VIII the overall system along with experiments. Section IX discusses the extension to dense volumetric mapping and we close with lessons learned, conclusions, and an outlook on future work in Section X. All experiments were performed using a dual Intel Xeon E5420@2.5GHz (2×4 cores) running a 32bit Linux 2.6.26.

II. Related Work

There is a vast amount of literature on (visual) SLAM. A good starting point is [1] or the recent special issue [7]. We focus here on a few highlights involving full systems.

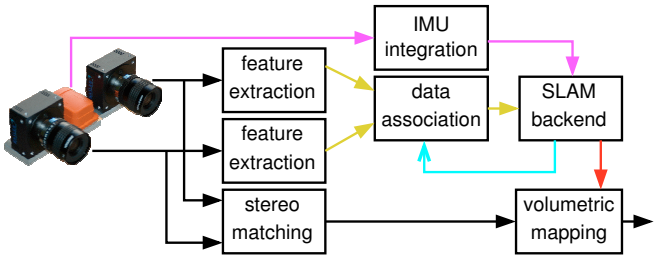

Fig. 2. System architecture: Keypoints (yellow) are extracted from images (black) of a stereo camera, associated left-to-right and to the map and finally fused (cyan) together with integrated IMU information (magenta). Left and right images are also stereo matched and the resulting depth-images are integrated into a voxel map using the poses (red) estimated by SLAM.

Paz et al. [8] realized visual SLAM with a hand-held stereo camera and demonstrate accurate estimation of a 210m indoor and 140m outdoor loop. They find that stereo improves precision in addition to resolving scale ambiguity. However, also including distant bearing-only features improves the precision even more providing orientation information. The system uses image patches around Shi-Tomasi corners and an EKF for building local maps which are later joined.

Milford et al. [9] show an impressive map of a 66km car journey through a suburb with 51 loops of up to 5km. The biologically inspired algorithm "RatSLAM" performs iterative updates thus amortizing loop-closures over time.

Konolige and Agarwal [10] map a 10km outdoor environment. Local bundle-adjustment on center-surround-extrema features provides constraints between regularly spaced poses. This graph of relations is then optimized globally using PCG [10] or sparse Cholesky-decomposition [11].

Kaess and Daellert [12] close a 90m loop through difficult bushy terrain with visual SLAM in a real-time implementation on an autonomous off-road vehicle. They use a point-feature detector without descriptor and sparse bundle adjustment. Their major contribution is reliable data-association and obtaining the covariance information required for that.

Unlike the systems above, PTAM by Klein and Murray [13] is monocular. It operates at video rate but is limited to small environments. Their key idea is to use two concurrent threads: One to localize the camera from FAST feature points w.r.t. the map at frame rate. And another, slower one to extend the map using bundle-adjustment.

In a very impressive demonstration, Weiss et al. [14] use PTAM onboard a quadrocopter for point-to-point navigation. They also showed mapping a 100m path through a small village generating a triangle model from the sparse point features used by PTAM and texturizing that model.

III. Feature Extraction

The feature extraction module needs to find 2D image coordinates of environment features such as object corners or other salient points. To enable data association there must be a way to identify corresponding features both in a stereo frame pair and across frames from different time steps. Recent work in computer vision makes this possible: Feature detectors identify salient points (*keypoints*) in an image, descriptors determine a numerical description of these, usually as a *signature* vector describing the local neighborhood. Both should ideally be invariant to changes in viewing angle, distance (scale), and viewing conditions (illumination, noise). Signatures should be distinctive to allow for reliable matching and be uniformly distributed across the image to yield good geometrical constraints.

The de-facto gold standard in feature detection/description was established by the seminal work of Lowe [15] with the scale invariant feature transform (SIFT). A very accessible introduction is given in [16, §4.1]. For image retrieval, Mikolajczyk and Schmid found SIFT to outperform earlier approaches [17]. For SLAM, it is reportedly superior to other approaches particularly under changes in viewing angle [18]. SIFT feature extraction is not real-time; GPU implementations get close [19] but have their own issues.

SURF [20] picks up the same ideas as SIFT but employs approximations to get much closer to real-time and delivers features not identical yet structurally very similar to SIFT.

More recently, machine learning has been applied to feature extraction. FAST [21] learns a decision tree which is turned into C code to yield an extremely fast corner detector the downside being that features cluster at edges. Calonder et al. [22] reformulate the descriptor task as a classification problem and define the signature as the response of a randomized tree classifier.

A. Towards Constant Computation Times

Depending on the viewing conditions and the detector used, we get 500-10000 features per (monocular) frame, far too many for real-time processing. Thus, we select the $k = 200$ features with the best detector score. This limits computation time per frame and in our application still ensures a reasonable distribution of features.

B. Selecting a Detector/Descriptor Combination

We use Lowe's reference implementation of SIFT for comparison only due to its closed-source nature and real-time requirements (SIFT runs at ∼1Hz on 512x384 images). Actual candidates are a re-implementation of SURF from OpenCV [23] with a minor custom patch to invoke the descriptor only for the k best keypoints, FAST as contributed to OpenCV by Rosten, and the Calonder descriptor from OpenCV. SURF is run with default parameters, FAST operates on four pyramid layers with the parameters also used by PTAM [13], and the Calonder descriptor with default parameters and a classifier tree[1] from its authors.

Concerning the quality of the detectors/descriptors, we are interested in a combination that yields good matches. Since in our case groundtruth is impossible to determine automatically (perspective changes, scene-dependent occlusion), we have manually labeled Euclidean distance nearest neighbor matches (see IV) as close (< 10px w.r.t. 1024×768 resolution), near (< 40px), unrelated (> 40px), and undecided (Fig. 3). Image pairs to be matched consist of 7 stereo frame

[1] http://pr.willowgarage.com/data/calonder_descriptor/current.rtc

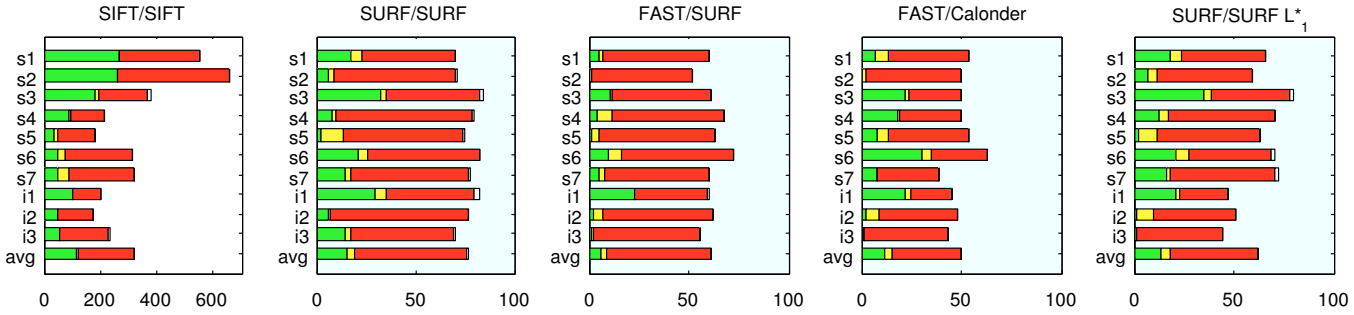

Fig. 3. Stacked number of matches for stereo frame pairs (s1–s7) and inter-frame pairs with zoom in/out situations of increasing difficulty (i1–i3) manually categorized as close ($<$ 10px, green), near ($<$ 40px, yellow), unrelated ($>$ 40px, red) and undecided (white) for different detector/descriptor combinations. See III-B and IV-B for details.

TABLE I

COMPUTATION TIMES OF DIFFERENT FEATURE DETECTOR/DESCRIPTOR COMBINATIONS FOR 2X474 IMAGES DOWNSCALED TO 512X384 FROM THE DATASET ALSO USED IN THE VIDEO. DESCRIPTORS PROCESS THE 200 BEST FEATURES ONLY.

Detector	Descriptor	Time [ms]			
		min	max	μ	σ
SURF	SURF	66.678	93.055	72.158	2.037
FAST	SURF	17.289	32.930	25.208	3.081
FAST	Calonder	17.630	41.643	25.841	3.008

pairs and 3 pairs of left frames from different time steps (inter-frame pairs) with zoom in/out situations of increasing difficulty, all from the log also used for the companion video. Note that the SIFT dataset contains matches of all features, all others those of the 2×200 best features only. Due to the high number of matches the SIFT labeling figures were projected based on a randomly chosen sample of size 25.

Our evaluation criterion is the ratio of correct matches vs. all matches (similar to [17]) as well as the total number of matches per frame pair. Taking the different sizes of the input data into account, SURF/SURF produces a percentage of correct matches that is slightly lower than that of SIFT/SIFT but finds some correct matches on all tested frame pairs. FAST/SURF produces a significantly lower percentage of correct matches, particularly with the more difficult inter-frame pairs which affects overall system performance negatively since constraints across different time steps are weaker. Compared to FAST/SURF, FAST/Calonder improves the percentage of correct matches for stereo frames although the keypoints are the same. The improvement for the inter-frame cases is only marginal. Generally, we conclude that the detector and descriptor must be designed to work well with each other. Thus, although we would like to use the faster alternatives (Table I), we currently prefer SURF as the detector and descriptor.

IV. DATA ASSOCIATION

The data association module forms the bridge between feature extraction and the SLAM back-end by identifying corresponding features in a stereo frame pair (stereo matching) and recognizing previously seen features or adding new features to the map (map management). In both cases, features need to be matched based on their signatures.

A. Feature Matching

Using descriptors, the matching procedure boils down to a nearest neighbors problem. Given two sets of features A and B, $b \in B$ is called the nearest neighbor of $a \in A$ if $b = \operatorname{argmin}_{\bar{b} \in B} \left\| a - \bar{b} \right\|$.

A key problem here lies in determining whether the nearest neighbor match is distinct enough to be reliable. A common strategy [15] is to impose a threshold on the relative distance to the 2nd-nearest neighbor, e.g., 80%. Tuning this threshold to work under different conditions, however, proved difficult. Instead, we require consistency in both directions, i.e., (a, b) match, iff $a = \operatorname{argmin}_{\bar{a}} \left\| \bar{a} - b \right\| \wedge b = \operatorname{argmin}_{\bar{b}} \left\| a - \bar{b} \right\|$. This is checked by keeping track of the current best nearest neighbors in both directions while looping over all $a \in A$ and all $b \in B$ in two nested loops.

Since the matching procedure is time critical we follow the trend towards multi-core CPUs and parallelize the outer loop using the Threading Building Blocks library [24] by dividing the set A into chunks to be processed in parallel. Another optimization is to replace the Euclidean distance ($\mathcal{L}_2$ norm) with the sum of absolute differences (SAD) ($\mathcal{L}_1$ norm). We go one step further and implement an approximation $\mathcal{L}_1^*$ by scaling signatures to sequences of 8bit unsigned integers and computing the SAD with the `PSADBW` SSE2 instruction operating on 16 values in parallel.

B. Stereo Matching

To determine stereo matches we apply the feature matching procedure as above and additionally filter results by an epipolar constraint. Table II lists the computation time required to match the 200 best SURF features from two stereo frames for an $\mathcal{L}_2$/`float` implementation, the $\mathcal{L}_1^*$ implementation and the FLANN library [25] in different modes – brute force search, k-D tree and k-means-based search. Since FLANN determines nearest neighbors in one direction only, we call it twice.

The results show how the parallelized SSE implementation clearly outperforms all others and that in our case the advanced algorithms in FLANN (k-D tree, k-means) do not pay off since a k-D tree does not help with high dimensional

TABLE II

COMPUTATION TIMES OF $\mathcal{L}_2$/FLOAT, $\mathcal{L}_1^*$, AND FLANN (BRUTE FORCE, K-D TREE, K-MEANS) METHODS MATCHING 2X200 FEATURES.

		Time [ms]			
		min	max	μ	σ
$\mathcal{L}_2$/float		7.734	8.193	7.980	0.081
$\mathcal{L}_2$/float	Parallel	1.698	2.171	1.928	0.083
$\mathcal{L}_1^*$		1.157	1.594	1.388	0.076
$\mathcal{L}_1^*$	Parallel	0.805	1.254	1.022	0.073
FLANN	Linear	16.964	17.456	17.183	0.088
	k-D	50.163	60.100	52.061	1.926
	k-means	17.322	30.297	23.180	2.463

spaces [15] and the pre-processing required to build the helper data structures cannot be leveraged as they need to be rebuilt at every time step.

Concerning our optimization strategy, the single largest speed-up is gained by the conversion to SSE instructions. The speed-up due to parallelization is significant but not linear in the number of cores due to overheads induced by the parallelization itself. We have also seen vastly different behavior on different machines with the same number of cores hinting at the exact memory/cache layouts playing an (expectedly) important role.

As for a qualitative comparison, the $\mathcal{L}_1^*$ implementation yields results that are very similar overall but in some cases clearly inferior to the $\mathcal{L}_2$/float matches as highlighted by the inter-frame matches in Fig. 3. Once other parts of the system have been optimized more, we will likely favor accuracy over speed and switch to the parallel $\mathcal{L}_2$/float version especially given its small absolute computation time.

C. Map Management

The map consists of estimated world coordinates of (some) detected features and their signatures. Detected mono and stereo keypoints are searched for in the map using the feature matching algorithm defined above (using an averaged signature in the stereo case). If a match is found and the distance to the reprojected feature is not too large, the match is passed as a feature measurement to the SLAM back-end. Stereo keypoints with no match are added unless they are too close to an existing landmark.

Obviously, this can make loop-closing or reattaching after a blackout period without any features difficult or nearly impossible. In future work, we intend to revisit this decision once the back-end supports longer trajectories (see VIII-B). Re-localization is another topic to look into in this context.

V. SLAM BACK-END

Interpreting SLAM as a (non-linear) least squares (LS) problem is quite straight forward and long known in photogrammetry as bundle adjustment (BA) [5]. It was first applied in robotics by [26] and later formalized in [27, §11] as *GraphSLAM*. The idea is to view odometry, i.e., measurements between consecutive poses, and landmark observations as "constraints" between parts of the state, which consists of all poses and landmark positions (Fig. 4).

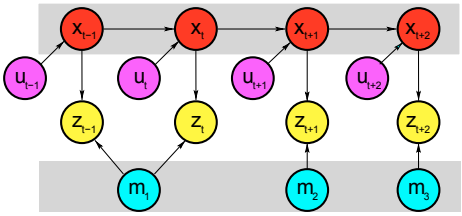

Fig. 4. Bayes net for the SLAM problem solved by the backend (illustration from [27]). Poses X_t (red) are linked by IMU data u_t (magenta). Feature observations z_t (yellow) link poses to feature positions M_i (cyan). The z_t can be either mono (2DOF) or stereo (4DOF) observations. Colors correspond to Fig. 2 and grey indicates unknown random variables.

More formally, the odometry function g, which using odometry u_t maps the previous state X_{t-1} to the current state X_t, can be converted to a set of functions $\tilde{g}_t$, constraining two consecutive poses:

$$X_t = g(X_{t-1}, u_t) \quad \tilde{g}_t(X_{t-1}, X_t) := g(X_{t-1}, u_t) - X_t \quad (1)$$

Analogously, the measurement function h, which maps a landmark M_i at the current pose X_t to its corresponding measurement z_t^i, can be mapped to a constraint $\tilde{h}_t^i$:

$$z_t^i = h(X_t, M_i) \qquad \tilde{h}_t^i(X_t, M_i) := h(X_t, M_i) - z_t^i \quad (2)$$

for all observations $(t, i) \in \mathcal{O}$. All state variables X_t and M_i can be combined to a global state X. The results of $\tilde{g}$ and $\tilde{h}$ are called residues and are accumulated to a cost function

$$f(X) = \sum_t \rho_g(\tilde{g}_t(X_{t-1}, X_t)) + \sum_{(t,i)\in\mathcal{O}} \rho_h(\tilde{h}_t^i(X_t, M_i)), \quad (3)$$

with ρ_* mapping to $\mathbb{R}_{\geq 0}$. For classical LS, we have $\rho(x) = \|x\|_\Sigma^2 = x^\top \Sigma^{-1} x$ with a covariance matrix Σ and (3) can be minimized using textbook math (e.g., Levenberg-Marquardt algorithm). In presence of outliers, robust choices for ρ are better [6, §A6.8]. We use classical LS for odometry links, and a modified Huber cost function $\rho(x) = \begin{cases} \|x\|_\Sigma^2, & \|x\|_\Sigma \leq 1 \\ \|x\|_\Sigma, & \|x\|_\Sigma > 1 \end{cases}$ for camera measurements. The latter behaves like classical LS below the standard deviation and is more robust above.

A. Selecting a Back-End Library

Pure pose adjustment frameworks such as TORO [28] and SPA [11] are unable to solve BA directly and therefore not applicable to our problem. BA is addressed in [6, §18] and [5], and with the *sba* package a well established open source implementation is available [29], implementing monocular BA as an example. Stereo SLAM and usage of robust estimation with *sba* is possible in theory, but technically difficult.

Recently Strasdat et al. [30] published RobotVision, a library focusing on monocular SLAM. It takes advantage of the manifold structure of $SO(3)$ and natively supports robust estimation. Currently it does not support stereo SLAM.

More recently iSAM [31] was made open-source. It can handle arbitrary LS problems, but currently only implements Gauss-Newton optimization, thus it is incapable of optimizing rank-deficient problems such as BA.

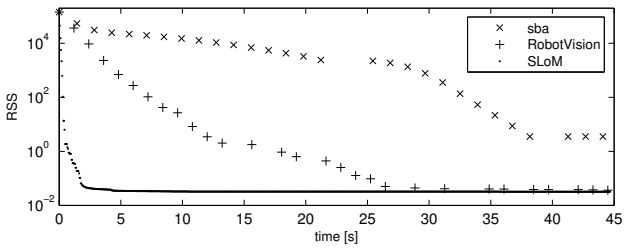

Fig. 5. χ^2-error over computation time for SLoM, sba and RobotVision

B. Sparse Least Squares on Manifolds (SLoM)

The goal of the SLoM framework [32] is to define and optimize general LS problems easily. This is accomplished by providing building blocks in C++ to define state variables (e.g., poses or landmarks) and measurements (or constraints) between states (e.g., odometry). Afterwards instances of states can be added to a problem (in arbitrary order) as well as constraints between those states.

Behind the scenes, SLoM exploits the sparsity of many LS problems, first when calculating Jacobians, and secondly by using the CSparse library [33] when solving the linear system occurring in each Gauss-Newton/Levenberg-Marquardt iteration. States are viewed as so-called manifolds, elegantly avoiding problems with singularities and overparameterization, occurring when working with, e.g., 3D orientations. The manifolds are encapsulated by two operators: $\boxplus$ which adds small perturbations to a state, and its inverse $\boxminus$. This makes it possible to adapt generic sensor fusion algorithms mainly by replacing $+$ and $-$ by $\boxplus$ and $\boxminus$.

In recent work [34], we enhanced the framework by factoring out the handling of manifolds into the separate *Manifold ToolKit* (MTK) so it can be used by other algorithms, too. MTK is capable of stacking manifold primitives such as vectors ($\mathbb{R}^n$) and orientations ($SO(3)$) to compound manifolds such as poses ($SE(n) \simeq SO(n) \times \mathbb{R}^n$), with components being accessible by name instead of index. For matrix/vector operations, MTK uses the open source library Eigen 2 [35] making the implementation of measurement functions straight-forward using a MATLAB-like notation via overloaded operators, and efficient using expression templates. SLoM has previously been released as open source.

C. Comparison with sba and RobotVision

We compared sba, RobotVision, and SLoM on a monocular BA problem extracted from the log also used for the companion video but only using keypoints from the left camera (with the data association obtained from our full system). Measurement outliers were filtered out as well as poses and features with too few measurements. This lead to 349 poses, 188 features and 4152 image projections.

For each algorithm the data was passed through a functionally equivalent BA application. Fig. 5 shows the χ^2-error for each algorithm over time. As the goal was mainly to compare runtime-performance we did no ground truth comparisons.

Surprisingly, SLoM dramatically outperforms both algorithms, mainly because much less time is spent for sparse matrix operations. We assume this is because we do not try to manually exploit any structural sparsity, but let CSparse [33] take advantage of the overall sparsity. The fact that we have more DOFs in poses than in features may contribute to that being untypical for bundle adjustment. Interestingly, RobotVision outperforms SLoM per iteration, which we assume is due to its more sophisticated parameter control in the Levenberg-Marquardt step.

D. Comparison with iSAM

As iSAM cannot handle BA, we compared it to SLoM on problems included with iSAM as sample data sets. On 2D pose adjustment problems, iSAM was about 5% faster with hand-implemented Jacobians, but about three times slower when using numerical differentiation. For 3D problems, iSAM failed to converge in batch mode, probably due to its use of Euler Angles.

VI. IMU Integration

Our odometry is provided by an IMU, i.e., gyroscopes measuring angular velocity ω_t and accelerometers measuring acceleration and gravity a_t. Given a start pose and velocity, IMU measurements can be integrated to an end pose and end velocity using "dead reckoning". Thus, referring to (1), the state elements X_t contain the position ξ_t, orientation $R_t \in SO(3)$ and velocity v_t. The transition function g can then calculate the next pose as follows:

$$R_{t+1} = R_t \cdot \exp(\delta t \cdot \omega_t), \tag{4a}$$

$$v_{t+1} = v_t + \delta t \cdot (R_t \cdot a_t - a_g), \tag{4b}$$

$$\xi_{t+1} = \xi_t + \delta t \cdot v_t, \tag{4c}$$

where a_g is the gravity vector, δt the time between measurements, and exp is the $SO(3)$ exponential map calculating a rotation from a scaled axis (Rodriguez-formula).

IMUs measure much faster than camera frame rates leading to a chain of poses without feature measurements. We remove these to enhance performance. Thus, actually, the state transition function g needs to calculate a pose X_t from a pose X_{t-k} without explicitly calculating $X_{t-k+1}, \ldots, X_{t-1}$. Fortunately, $g(X_t, u_t)$ is linear in X_t, so by induction we get:

$$R_t = R_0 \cdot \overbrace{\textstyle\prod_i \exp(\delta t \cdot \omega_i)}^{\Delta R_t} \tag{5a}$$

$$v_t = v_0 - t\delta t \cdot a_g + R_0 \cdot \overbrace{\delta t \cdot \textstyle\sum_i \Delta R_i \cdot a_i}^{\Delta v_t} \tag{5b}$$

$$\xi_t = \xi_0 + t\delta t \cdot v_0 - \tfrac{t(t+1)}{2}\delta t^2 \cdot a_g + R_0 \cdot \overbrace{\delta t \cdot \textstyle\sum_i \Delta v_i}^{\Delta \xi_t} \tag{5c}$$

The clue is that now it is sufficient to accumulate ΔR_t, Δv_t, $\Delta \xi_t$ and $\Delta t = t\delta t$ only once for each pose relation, but when evaluating g in (1) only (5) needs to be calculated.

VII. Sensor Calibration

Fusing measurements of multiple sensors requires a cross-calibration. For our setup, this is a 6-DOF transformation between both cameras and the IMU.

A. Stereo Camera and Inertial Sensor Calibration

As in [36], we use gravity as a vertical reference. For the IMU frame, we measure gravity using its accelerometers. For the camera, it is measured through the extrinsic parameters while observing a horizontally placed checkerboard pattern. Using these static measurements collected at different poses we can compute the full transformation between both cameras and the rotation between one camera and the IMU. Since our sensors are not rotating fast, a manually determined IMU-to-camera translation is sufficiently precise. In contrast to [36], we formulate this as a combined non-linear LS problem. We use OpenCV's `findChessboardCorners` and `cornerSubPix` to extract the checkerboard features, and `cvFindExtrinsicCameraParams2` provides initial guesses. Optimization is done by SLoM using the camera model provided by `projectPoints`.

B. Tracker-Camera (Hand-Eye) Calibration

For evaluation, we used a Qualisys motion capture system to track position and orientation of a set of tracking markers mounted to the camera. Turning this into the camera ground truth trajectory requires knowledge of the transformation between the marker set and the camera which we determine by adding the tracking system's marker measurements to the LS calibration problem. An initial guess is obtained by the method of [37].

VIII. Overall System Performance

For overall experimental evaluation, we attached a stick to our stereo head, for handling and to rigidly mount tracking markers. We then recorded several trajectories through the mock-up (Fig. 10, top) moving the cameras by hand. We recorded camera data with 1024×768 pixels at 10 Hz, IMU data from an XSens MEMS IMU at 512 Hz and Qualisys motion tracking data (ground truth) at 100 Hz.

Since working at 10 Hz is not realistic using our implementation we evaluated each run at 5 Hz, skipping every other camera frame. We also downscaled camera images to 50% resolution. The complete data sets, source code and tools are available from our website[2].

A. Qualitative Evaluation

We compared each resulting trajectory to ground truth (Fig. 6), obtaining rotational and translational errors over time (Fig. 7). In two runs out of six our algorithm diverged (expectedly) when near the end of the run features could not be matched against the map for an extended period of time – too long to be compensated for by the IMU. Otherwise, we had an average error of about 1.0° and 13 mm, 95% errors below 2.6°/38 mm and outliers up to 10.1°/57 mm. We noted some spurious measurements in the ground truth, which we assume to be responsible for some of the outliers. In retrospect, we doubt that the motion tracking data is sufficiently precise to serve as a source of ground truth.

[2] http://www.informatik.uni-bremen.de/agebv/en/pub/hertzbergicra11/

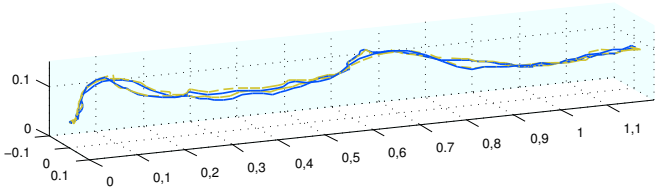

Fig. 6. Estimated trajectory of vSLAM (solid, blue) compared to ground truth (dashed, yellow)

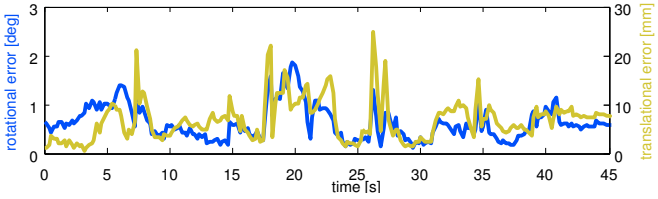

Fig. 7. Error of estimated trajectory compared to ground truth.

B. Real-Time Capability

To check for real-time performance we traced the computation times of each module over time. We found that times for data association and IMU integration are negligible (below 2 ms in total), thus we only further investigated feature extraction and the SLAM back-end. Fig. 8 shows accumulated computational costs for these modules over time when processing data at 5 Hz. At the beginning, feature extraction is clearly dominant, but over time the back-end costs increase. After about 28 seconds the overall cost exceeds 200 ms, thus technically losing real-time capability.

We have experimented with reducing the frame rate further but ultimately SLoM needs to be extended to allow for efficient incremental SLAM, e.g., by a two-level approach such as [10] or [13]. In addition large-scale operation would require global data-association for loop closing [38].

IX. Dense Stereo and Volumetric Mapping

A feature based map is of little use to the human operator, supporting whom was our search and rescue motivation. Thus, we also build a map representing the 3D structure of the environment: We first compute dense range measurements from each stereo image pair, transform these into world coordinates, and register them in a volumetric

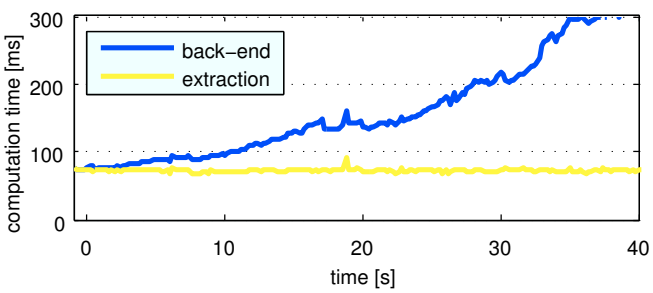

Fig. 8. Stacked computation time of feature extraction and SLAM back-end over time.

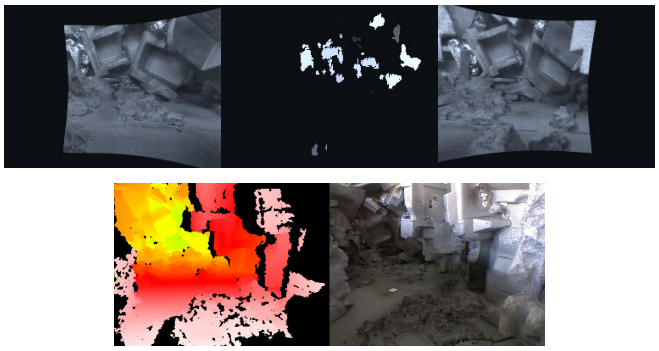

Fig. 9. Top: A typical undistorted and rectified stereo image pair and the resulting disparity map (center). Bottom: Preliminary Kinect experiment.

map (Fig. 10). This is the youngest part of the system but provides a glimpse of what is possible.

A. Determining Dense Stereo Correspondence

For every pixel in one image we seek to find the pixel in the other image of the same stereo pair that corresponds to the same physical location [39]. In a nutshell, the problem is first reduced to line search by undistorting and rectifying the input images such that epipolar curves become parallel lines with identical y-ordinates. For every pixel in the left image the corresponding pixel (if any) in the right can then be found in the same row. A commonly used search method is called *block matching* and looks for the two pixels that minimize a certain distance function (often the SAD) applied to a pixel-neighborhood around both candidates.

We use the OpenCV functions `stereoRectify`, `initUndistortRectify` and `remap` for undistortion and rectification, the `StereoBM` class for block matching, and `reprojectImageTo3D` to transform the resulting disparity map back into metric coordinates which finally is transformed from the rectified camera coordinate system into world coordinates using the poses estimated by SLAM. Block matching requires a pixel-precise calibration. We have found that down-scaling by a factor of two increases robustness against minor, 1-pixel differences and the loss in resolution is negligible.

The algorithm works reasonably well but as illustrated in Fig. 9 (top) the resulting disparity map is not as dense as we had hoped for. It also has the well-known problem of washed out object boundaries and occasional incorrect matches. In future work, we intend to integrate available open source implementations of other, more modern dense stereo algorithms ([40], [41]). An alternative is the recently released Kinect sensor [42]. Preliminary experiments (Fig. 9, bottom) show that it provides nearly dense data except at depth discontinuities and below the minimum range of $\approx 0.6\,\mathrm{m}$.

B. Probabilistic Volumetric Mapping

Our volumetric mapping component is based on OctoMap [43], a probabilistic map data structure that uses an octree to store occupancy probabilites in voxels.

OctoMap implements *mapping with known poses* [27, §9] and was originally designed for use with laser rangefinder

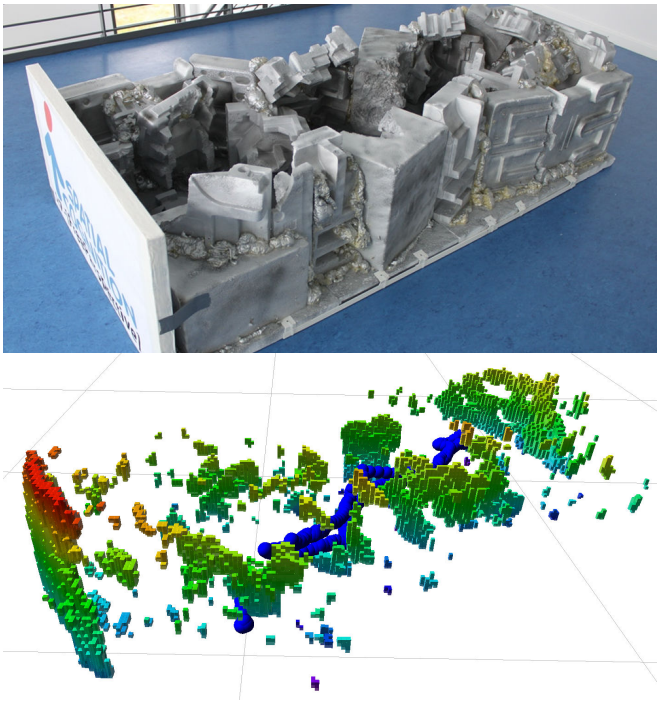

Fig. 10. The test environment and the corresponding volumetric map (1cm voxels, 1m ground grid). Visual inspection reveals that all larger connected blocks and more than half of the small blocks (< 10 voxels) in the map actually exist in reality. Major parts of the environment, including the floor, however, were not stereo-matched and are missing in the dense model. The attached video also shows how the map is built incrementally.

data resulting in an interface that expects range scans. Given the disparity map and the camera pose we can easily convert the dense stereo output into this format – each pixel in the disparity map defines the endpoint of a range ray.

At present, we use OctoMap to store the raw input data required for mapping (camera trajectory, range information) upon a key-press and later generate and visualize the map in the offline viewer `octovis`. In future work, we will investigate real-time operation. Currently, OctoMap spends most time in registering every voxel along a ray as unoccupied.

X. Lessons Learned and Conclusions

Our main conclusion is that it is actually possible to build a dense visual SLAM system from open source components with moderate effort (4 person months in our case). This shows that the field has both a remarkable degree of maturity and a positive attitude towards open source. We follow this line and also release our implementation and dataset.[2]

In detail, we found SURF as detector and descriptor to be the best compromise between speed and reliability. For efficient matching, a brute-force but highly optimized (SSE, multi-core) implementation clearly outperforms more sophisticated algorithms. We believe this is remarkable, because the scientific community prefers complex algorithms over tuned implementations. For the SLAM back-end, SLoM surprisingly outperformed sba and RobotVision. It also was the only one allowing us to integrate IMU data. For dense mapping, the main qualitative bottle-neck is stereo matching,

where the OpenCV block-matcher we used only matched a fraction of the whole environment.

As an outlook, there is still room for improvement. A real-time detector as reliable as SIFT would be very valuable. Our system operates in real-time as long as the SLAM back-end (i.e., batch BA) can handle the map size. There is still no library for *incremental* BA that could overcome this growth in computation time. The related work shows various methods to address this issue, but does not provide a black-box implementation as for batch operation. This should be a challenge for the SLAM community that has a strong tradition in incremental methods.

Acknowledgements

We thank the DFKI Robotics Innovation Center for providing a Qualisys motion capture system for the experiments.

References

[1] S. Thrun and J. J. Leonard, "Simultaneous localization and mapping," in *Springer Handbook of Robotics*, B. Siciliano and O. Khatib, Eds. Springer, New York, 2008, ch. 37.

[2] G. Grisetti, C. Stachniss, and W. Burgard, "Improved techniques for grid mapping with rao-blackwellized particle filters," *IEEE Transactions on Robotics*, vol. 23, no. 1, pp. 34–46, 2007, openslam.org/gmapping.

[3] M. Quigley *et al.*, "ROS: An open-source robot operating system," in *Proc. of the ICRA-Workshop on Open-Source Robotics*, 2009, ros.org.

[4] U. Frese, R. Wagner, and T. Röfer, "A SLAM overview from a user's perspective," *KI-Zeitschrift*, vol. 24, no. 3, pp. 191–198, 2010.

[5] B. Triggs, P. F. McLauchlan, R. I. Hartley, and A. W. Fitzgibbon, "Bundle adjustment – A modern synthesis," in *Vision Algorithms: Theory and Practice*, ser. LNCS, B. Triggs, A. Zisserman, and R. Szeliski, Eds. Springer, 2000, pp. 298–375.

[6] R. I. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. Cambridge University Press, 2004.

[7] J. Neira, A. J. Davison, and J. J. Leonard, Eds., *IEEE Transactions on Robotics Special Issue on Visual SLAM*. IEEE, 2008, vol. 24, no. 5.

[8] L. M. Paz, P. Piniés, J. D. Tardós, and J. Neira, "Large-scale 6-DOF SLAM with stereo-in-hand," *IEEE Transactions on Robotics, Special Issue on Visual SLAM*, vol. 24, no. 5, pp. 946–957, 2008.

[9] M. J. Milford and G. F. Wyeth, "Mapping a suburb with a single camera using a biologically inspired SLAM system," *IEEE Transactions on Robotics, Special Issue on Visual SLAM*, vol. 24, no. 5, pp. 1038–1053, 2008.

[10] K. Konolige and M. Agrawal, "FrameSLAM: from bundle adjustment to realtime visual mappping," *IEEE Transactions on Robotics, Special Issue on Visual SLAM*, vol. 24, no. 5, pp. 1066–1077, 2008.

[11] K. Konolige, G. Grisetti, R. Kümmerle, B. Limketkai, and R. Vincent, "Efficient sparse pose adjustment for 2D mapping," in *Proc. of the IEEE Int. Conf. on Robotics and Automation*, 2010.

[12] M. Kaess and F. Dellaert, "Covariance recovery from a square root information matrix for data association," *Journal of Robotics and Autonomous Systems, RAS*, vol. 57, no. 12, pp. 1198–1210, Dec 2009.

[13] G. Klein and D. Murray, "Parallel tracking and mapping for small AR workspaces," in *Proc. Sixth IEEE and ACM Int. Symposium on Mixed and Augmented Reality (ISMAR'07)*, 2007, pp. 1–10.

[14] S. Weiss, M. Achtelik, L. Kneip, D. Scaramuzza, and R. Siegwart, "Intuitive 3D maps for MAV terrain exploration and obstacle avoidance," in *Int. Conf. on Unmanned Aerial Vehicles, Dubai*, 2010.

[15] D. G. Lowe, "Distincitive image features from scale-invariant keypoints," *Int. Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[16] R. Szeliski, *Computer Vision: Algorithms and Applications*. London: Springer, 2011.

[17] K. Mikolajczyk and C. Schmid, "A performance evaluation of local descriptors," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, vol. 27, no. 10, pp. 1615–1630, 2005.

[18] J. Klippenstein and H. Zhang, "Quantitative evaluation of feature extractors for visual SLAM," in *Fourth Canadian Conference on Computer and Robot Vision, Montreal*, 2007, pp. 157–164.

[19] S. N. Sinha, J.-M. Frahm, M. Pollefeys, and Y. Genc, "GPU-based video feature tracking and matching," in *EDGE 2006, Workshop on Edge Computing Using New Commodity Architectures*, Chapel Hill, May 2006.

[20] H. Bay, T. Tuytelaars, and L. V. Gool, "SURF: Speeded up robust features," *Computer Vision and Image Understanding*, vol. 110, no. 3, pp. 346–359, 2008.

[21] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," in *European Conference on Computer Vision*, vol. 1, May 2006, pp. 430–443.

[22] M. Calonder, V. Lepetit, and P. Fua, "Keypoint signatures for fast learning and recognition," in *10th European Conference on Computer Vision (ECCV)*, ser. LNCS. Springer, October 2008.

[23] "OpenCV," opencv.willowgarage.com, SVN r3545, 2010.

[24] "Intel Threading Building Blocks 2.2 update 3," threadingbuildingblocks.org, 2010.

[25] M. Muja and D. G. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration," in *Int. Conf. on Computer Vision Theory and Application (VISAPP'09)*. INSTICC Press, 2009, pp. 331–340.

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

[27] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. Cambridge, MA: MIT Press, 2005.

[28] G. Grisetti, C. Stachniss, S. Grzonka, and W. Burgard, "A tree parameterization for efficiently computing maximum likelihood maps using gradient descent," in *Robotics Science and Systems*, 2007, openslam.org/toro.

[29] M. I. A. Lourakis and A. A. Argyros, "SBA: A software package for generic sparse bundle adjustment," *Transactions on Mathematical Software*, vol. 36, no. 1, 2009, ics.forth.gr/~lourakis/sba/.

[30] H. Strasdat, J. M. M. Montiel, and A. J. Davison, "Scale drift-aware large scale monocular SLAM," in *Robotics: Science and Systems*, 2010, openslam.org/robotvision.

[31] M. Kaess, A. Ranganathan, and F. Dellaert, "iSAM: Incremental smoothing and mapping," *IEEE Transactions on Robotics*, vol. 24, no. 6, pp. 1365–1378, 2008, openslam.org/iSAM.

[32] C. Hertzberg, "A framework for sparse, non-linear least squares problems on manifolds," Master's thesis, Universität Bremen, 2008, openslam.org/slom.

[33] T. A. Davis, *Direct Methods for Sparse Linear Systems*, ser. Fundamentals of Algorithms. Philadelphia: SIAM, 2006.

[34] C. Hertzberg, R. Wagner, U. Frese, and L. Schröder, "Integrating generic sensor fusion algorithms with sound state representations through encapsulation of manifolds," *Information Fusion*, submitted March 2010, **to appear**.

[35] G. Guennebaud, B. Jacob, *et al.*, "Eigen 2.0.12," eigen.tuxfamily.org, 2010.

[36] J. Lobo and J. Dias, "Relative pose calibration between visual and inertial sensors," *Int. Journal of Robotics Research*, vol. 26, no. 6, pp. 561–575, 2007.

[37] F. C. Park and B. J. Martin, "Robot sensor calibration: Solving AX=XB on the Euclidean group," *IEEE Transactions on Robotics and Automation*, vol. 10, no. 5, pp. 717 –721, 1994.

[38] M. Cummins and P. Newman, "Highly scalable appearance-only SLAM – FAB-MAP 2.0," in *Robotics Science and Systems*, 2009.

[39] M. Z. Brown, D. Burschka, and G. D. Hager, "Advances in computational stereo," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, no. 8, pp. 993–1008, 2003.

[40] D. Scharstein and R. Szeliski, "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms," *Int. Journal of Computer Vision*, vol. 47, no. 1/2/3, pp. 7–42, 2002.

[41] R. Szeliski, R. Zabih, D. Scharstein, O. Veksler, V. Kolmogorov, A. Agarwala, M. Tappen, and C. Rother, "A comparative study of energy minimization methods for Markov random fields with smoothness-based priors," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, no. 6, pp. 1068–1080, 2008.

[42] J. Blake *et al.*, "OpenKinect," openkinect.org, 2010.

[43] K. M. Wurm, A. Hornung, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: A probabilistic, flexible, and compact 3D map representation for robotic systems," in *Proc. of the ICRA 2010 Workshop on Best Practice in 3D Perception and Modeling for Mobile Manipulation*, 2010, octomap.sf.net.