An Evaluation of the RGB-D SLAM System

Felix Endres[1] Jürgen Hess[1] Nikolas Engelhard[1]
Jürgen Sturm[2] Daniel Cremers[2] Wolfram Burgard[1]

***Abstract*— We present an approach to simultaneous localization and mapping (SLAM) for RGB-D cameras like the Microsoft Kinect. Our system concurrently estimates the trajectory of a hand-held Kinect and generates a dense 3D model of the environment. We present the key features of our approach and evaluate its performance thoroughly on a recently published dataset, including a large set of sequences of different scenes with varying camera speeds and illumination conditions. In particular, we evaluate the accuracy, robustness, and processing time for three different feature descriptors (SIFT, SURF, and ORB). The experiments demonstrate that our system can robustly deal with difficult data in common indoor scenarios while being fast enough for online operation. Our system is fully available as open-source.**

I. INTRODUCTION

Many relevant applications in robotics and computer vision require the ability to quickly acquire 3D models of the environment and to estimate the camera pose with respect to this model. A robot, for example, needs to know its location in the world to navigate between places. This problem is a classical and challenging chicken-and-egg problem because localizing the camera in the world requires the 3D model of the world, and building the 3D model in turn requires the pose of the camera. Therefore, both the camera trajectory and the 3D model need to be estimated at the same time.

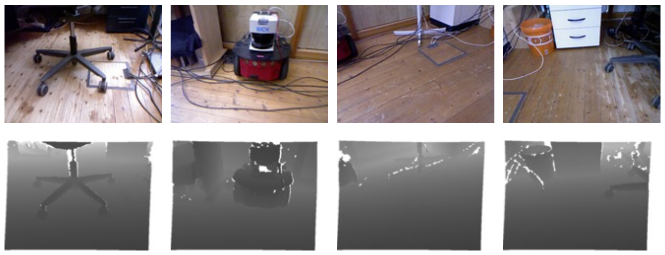
With the introduction of the Microsoft Kinect camera, a new sensor has appeared on the market that provides both color images and dense depth maps at full video frame rate. This allows us to create a novel approach to SLAM that combines the scale information of 3D depth sensing with the strengths of visual features to create dense 3D environment representations.

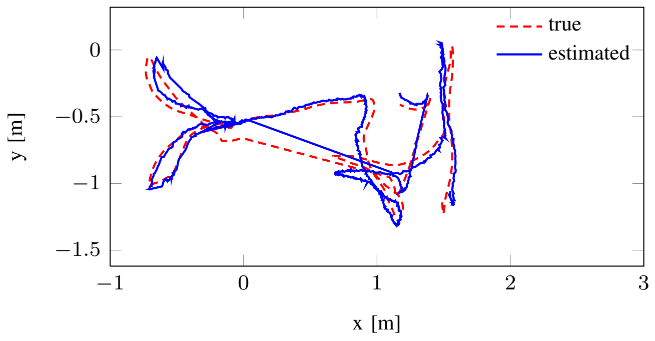
In this paper we present an approach to SLAM based on RGB-D-data that consists of the four processing steps illustrated in Figure 2. First, we extract visual features from the incoming color images. Then we match these features against features from previous images. By evaluating the depth images at the locations of these feature points, we obtain a set of point-wise 3D correspondences between any two frames. Based on these correspondences, we estimate the relative transformation between the frames using RANSAC.

[1] Felix Endres, Jürgen Hess, Nikolas Engelhard and Wolfram Burgard are with the Department of Computer Science, University of Freiburg, Germany. {endres,hess,engelhar,burgard}@informatik.uni-freiburg.de

[2] Jürgen Sturm and Daniel Cremers are with the Computer Science Department, Technical University of Munich, Germany. {sturmju, cremers}@in.tum.de

This work has partly been supported by the European Commission under grant agreement number FP7-248258-First-MM


(a) Input data: Sequence of RGB-D images



(b) Ground truth and estimated camera trajectory projected to 2D

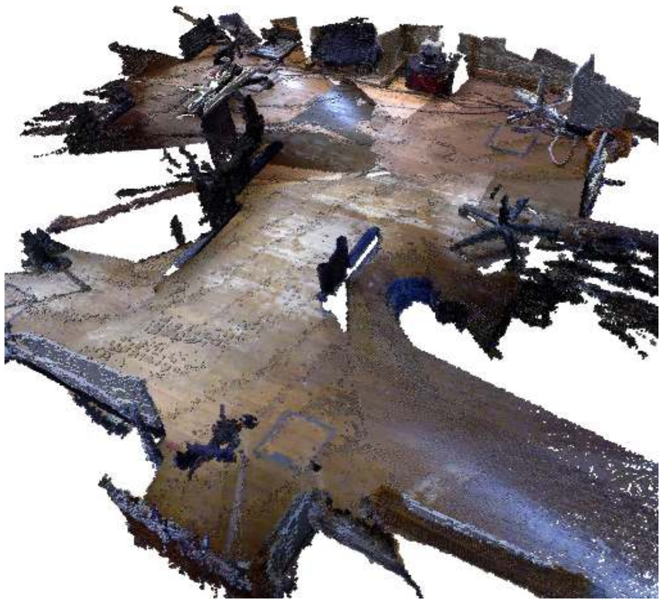

(c) Output: voxel grid (here displayed at 1 cm resolution)

Fig. 1. Our approach registers sequences of RGB-D images (a) to recover the trajectory of the camera (b) and to create globally consistent volumetric 3D models (c).

As the pairwise pose estimates between frames are not necessarily globally consistent, we optimize the resulting pose graph in the fourth step using the $\mathbf{g^2o}$ solver, which is a general open-source framework for optimizing graph-based nonlinear error functions [11]. The output of our algorithm at this stage is a globally consistent 3D model of the perceived environment, represented as a colored point cloud. Finally, we use the Octomap library [33] to generate a volumetric representation of the environment.

The contribution of this paper is twofold. First we describe our RGB-D SLAM system and its components. Our second contribution is a thorough analysis of the performance of our system on a recently published benchmark dataset [29]. The dataset contains both the RGB-D images of the Kinect with time-synchronized ground truth poses obtained from a high-accuracy motion-capture system. As the benchmark comes with a tool that evaluates the accuracy of an estimated trajectory, the measured performance of our algorithm can be objectively compared to those of other systems. Therefore, we hope to establish with our evaluation a baseline for future RGB-D SLAM systems. All code required to reproduce, verify (and improve) on our results is also fully available online. To summarize our results, we found that our system provides the camera pose with an average RMSE of 9.7 cm and 3.95° in a typical office environment, and can very robustly handle even the high speed sequences of the benchmark with average velocities of up to 50 deg/s and 0.43 m/s.

The remainder of this paper is organized as follows. We briefly review related approaches in Sec. II, before we introduce our approach in Sec. III. We evaluate our system in Sec. IV using the RGB-D benchmark sequences.

II. RELATED WORK

The general problem of SLAM has a long history in robotics [30], [22], [10], [6], [15], [9], [23]. Especially methods designed to learn three-dimensional maps of the environment employ laser scanners or Time-of-Flight (ToF) cameras to provide dense point clouds of an environment. Many modern variants apply the iteratively closest point (ICP) algorithm [2], [26], [27] for aligning pairs of local point clouds to establish constraints between observations. These constraints are then used to find maximum likelihood map.

Visual SLAM systems [5], [16], [28] – in the computer vision literature often referred to as *structure and motion estimation* [14], [21] – typically extract sparse keypoints from the camera images. Visual feature points have the advantage of being more informative which simplifies data association. Relevant feature descriptors include SIFT [20], SURF [1], and the recently introduced ORB features [25], as well as parallelized versions thereof like SIFTGPU [32]. In the monocular setting, the absolute scale of the map cannot be determined, so that additional normalization steps are required during optimization. Stereo SLAM systems [17], [24] do not suffer from this limitation as the depth can be calculated from the disparity between the two images. In general, however, the disparity can only be estimated for distinctive points in the image, i.e., surfaces with little or no texture cannot be matched easily.

Novel RGB-D sensors that are based on structured light like the Microsoft Kinect directly provide dense depth maps and color images. Note that in general SLAM approaches that operate on RGB-D images are structurally different from stereo systems as the input is dense RGB-D instead of two color images. Fioraio and Konolige [7] recently presented a system that uses bundle adjustment to align the dense point clouds of the Kinect directly however without further exploiting the RGB images. Most similar to our work is the approach of Henry et al. [12]. Their approach uses sparse keypoint matches between consecutive color images as an initialization to ICP. In their experiments, they found however that often the computationally expensive ICP step was not necessary. Therefore, they improved the algorithm so that ICP was only used if few (or none) keypoint matches could be established. While Henry et al. use sparse bundle adjustment [19] for the optimization of the 2.5D reprojection errors in RGB-D image space, we optimize the 3D pose graph using the $\mathbf{g^2o}$ [18] framework. Finally, the post-processing of the two approaches is different: Henry et al. post-process the resulting point cloud into a surfel representation, while we create a volumetric voxel representation [33] that can directly be used for robot localization, path planning and navigation [13].

Finally, in contrast to all of the above approaches, we evaluate our system on a publicly available benchmark [29]. Therefore, our results can directly be compared to other approaches that are evaluated on the same datasets. We have fully released our code (including the evaluation routines) as open-source[1] to ensure that our results are reproducible and scientifically verifiable.

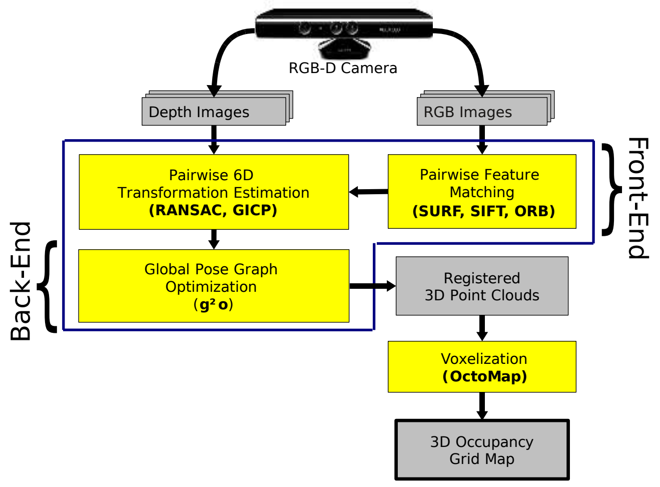
III. APPROACH

This section gives a detailed description of our approach. A schematic overview of our system is given in Figure 2. The trajectory estimation is divided into a front-end and a back-end. Whereas the front-end extracts spatial relations between individual observations, the back-end optimizes the poses of these observations in a so-called pose graph and with respect to a non-linear error function.

In the front-end, we use the visual image of the RGB-D sensor to detect keypoints and extract descriptors. These are matched to previously extracted descriptors and the relative transformation between the sensor poses is computed using RANSAC. Together with the depth information this allows us to register dense point clouds in a common coordinate system.

To compute globally optimal poses for the sensor positions w.r.t. the estimated relative transformations, we use a graph-based optimization routine. After reconstruction of the trajectory, we compute an occupancy voxel grid map.

[1]http://ros.org/wiki/rgbdslam



RGB-D Camera
Depth Images
RGB Images
Pairwise 6D Transformation Estimation (RANSAC, GICP)
Pairwise Feature Matching (SURF, SIFT, ORB)
Front-End
Back-End
Global Pose Graph Optimization (g²o)
Registered 3D Point Clouds
Voxelization (OctoMap)
3D Occupancy Grid Map

Fig. 2. Schematic overview of our approach. We extract visual features that we associate to 3D points. Subsequently, we mutually register pairs of image frames and build a pose graph, that is optimized using $\mathbf{g^2o}$. Finally, we generate a textured voxel occupancy map using the OctoMapping approach.

A. *SLAM Front-End*

The front-end is responsible for establishing spatial relations from the sensor data. Our system computes pairwise relations between camera images by matching of visual features. We rely on OpenCV [3] for detection, description and matching of various feature types, namely SURF, SIFT and ORB. Since SIFT features are computationally much more demanding than SURF and ORB, we also make use of a GPU based implementation of SIFT [32].

ORB is a new keypoint detector and feature descriptor combination recently introduced by Rublee et al. [25]. It is based on the FAST detector [25] and the BRIEF [4] descriptor. ORB computes an unambiguous orientation from the FAST corners and uses it for descriptor extraction, thus making the combination robust to viewpoint changes. Being significantly faster to compute than SIFT and SURF, we added it as an alternative in our system and present an evaluation in Section IV-B.1.

For the SURF keypoint detector, we apply the self-adjusting variant which increases or decreases the threshold on the Hessian to keep the number of keypoints roughly constant. While this slows down keypoint detection in case of fluctuating scene properties, variations in the number of keypoints with a fixed threshold can lead to inaccurate or even failed motion estimations. Too many features slow the system down in the matching step and may lead to many false positives.

After the detection of the keypoints, we project the feature locations from the image to 3D using the depth measurement at the center of the keypoint. The transformation of the camera pose between two frames can then be computed in closed form these correspondences [31].

However, no visual feature provides perfect reliability with respect to repeatability and false positives. Further, the depth data often is inconsistent with the color image, mainly due to a missing synchronization of the shutters of infrared and color camera, but also due to interpolation at depth jumps. Since visually salient points often lie at object borders, the 3D feature positions are prone to be at a wrong depth, making the robust estimation of transformations highly non-trivial.

A well-known approach to cope with noisy data and outliers is the Random Sample Consensus (RANSAC) algorithm [8]: After matching the feature descriptors of two frames, we randomly select three matched feature pairs, which is the minimal number from which a rigid transformation in SE(3) can be computed. Thanks to the full 3D position we can efficiently avoid outliers by refusing sample sets for which the pairwise Euclidean distances do not match. If the samples pass this test, they are used to compute an estimate of the rigid transformation. We apply the transformation to all matched features and count the number of inliers. In our case, consider a feature point an inlier when its mutual distance after the transformation is smaller than 3 cm. Subsequently, we use the inliers to compute a refined transformation. These steps are iterated and the transformation with most inliers is kept.

While the described procedure is very fast (exact timing depends on the number of features and the outlier percentage), after few seconds the number of past frames is too high to compare a new frame against all previous frames. Therefore, we select a subset of twenty frames, consisting of the 3 most recent frames and uniformly sampled earlier frames, and compute the pairwise transformations in parallel using threads.

If a frame could be matched to any predecessor, it is added as a node to the pose graph of the SLAM back-end, the pairwise spatial relations connecting it to the existing pose graph. The process applied when no relation to previous nodes can be found, depends on the application. If it is tolerable that the map is fragmented, a sensible action would be to keep the node even though disconnected, starting a new map fragment possibly to be connected later on through a loop closure. For evaluation purposes, we do not allow fragmentation in our experiments. If a frame cannot be matched, it will be connected to the prior node in the pose graph under the assumption of a constant motion model with high uncertainty. While this usually leads to higher error values than evaluation on the biggest fragment, it facilitates the comparison to other approaches, by reducing the comparison to fully connected trajectories.

B. *SLAM Back-end*

The pairwise transformations between sensor poses, as computed by the front-end, form the edges of a pose graph. Due to estimation errors, the edges form no globally consistent trajectory. To create a globally consistent trajectory we optimize the pose graph using the $\mathbf{g^2o}$ framework [18]. The $\mathbf{g^2o}$ framework is an easily extensible graph optimizer that can be applied to a wide range of problems including several variants of SLAM and bundle adjustment. It performs a minimization of a non-linear error function that can be represented as a graph, as for example the one created by the SLAM front-end described above.

TABLE I

WE EVALUATED OUR SYSTEM ON A LARGE SET OF SEQUENCES FROM THE RGB-D SLAM DATASET [29]. ON AVERAGE, OUR SYSTEM ACHIEVES AN ACCURACY OF 9.7 CM AND 3.95° AND REQUIRES APPROXIMATELY 0.35 S OF PROCESSING TIME PER IMAGE.

Sequence Name	Length	Duration	Avg. Angular Velocity	Avg. Transl. Velocity	Frames	Total Runtime	$\mathbf{g^2o}$ Runtime	Transl. RMSE	Rot. RMSE
FR1 360	5.82 m	28.69 s	41.60 deg/s	0.21 m/s	745	145 s	0.66 s	0.103 m	3.41°
FR1 desk2	10.16 m	24.86 s	29.31 deg/s	0.43 m/s	614	176 s	0.68 s	0.102 m	3.81°
FR1 desk	9.26 m	23.40 s	23.33 deg/s	0.41 m/s	575	199 s	1.31 s	0.049 m	2.43°
FR1 floor	12.57 m	49.87 s	15.07 deg/s	0.26 m/s	1214	488 s	3.93 s	0.055 m	2.35°
FR1 plant	14.80 m	41.53 s	27.89 deg/s	0.37 m/s	1112	424 s	1.28 s	0.142 m	6.34°
FR1 room	15.99 m	48.90 s	29.88 deg/s	0.33 m/s	1332	423 s	1.56 s	0.219 m	9.04°
FR1 rpy	1.66 m	27.67 s	50.15 deg/s	0.06 m/s	687	243 s	10.26 s	0.042 m	2.50°
FR1 teddy	15.71 m	50.82 s	21.32 deg/s	0.32 m/s	1395	556 s	1.72 s	0.138 m	4.75°
FR1 xyz	7.11 m	30.09 s	8.92 deg/s	0.24 m/s	788	365 s	40.09 s	0.021 m	0.90°

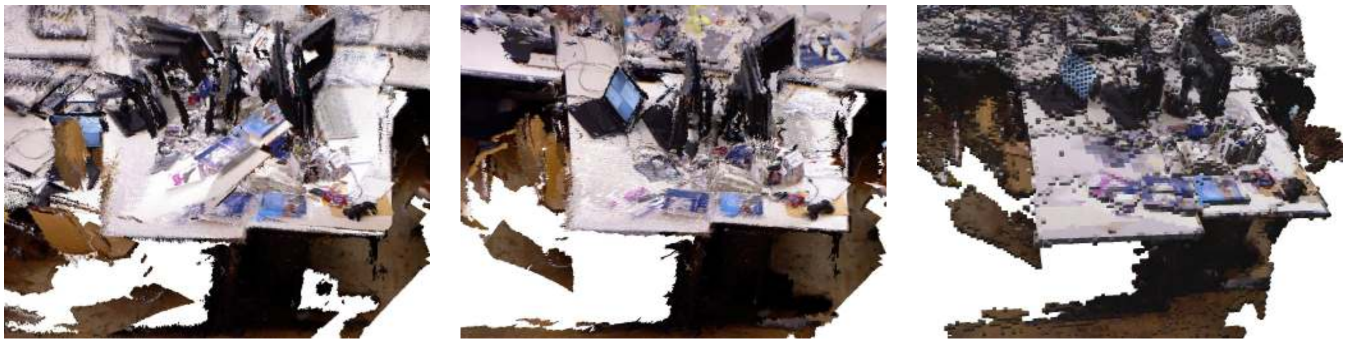

(a) Result of frame-to-frame tracking (no loop closures)
(b) Result after graph optimization (loop closures)
(c) Volumetric 3D map after post-processing

Fig. 3. Using only stepwise tracking leads to undesired results due to the accumulated drift (a). After graph optimization with 5 iterations of $\mathbf{g^2o}$, an excellent alignment of most point clouds is achieved (b). Individual clouds are still slightly misaligned. Through integration of the free-space information these artefact are greatly diminished after computation of occupancy probabilities (c).

Generally, the error function has the form

$$\mathbf{F}(\mathbf{x}) = \sum_{\langle i,j\rangle \in \mathcal{C}} \mathbf{e}(\mathbf{x}_i, \mathbf{x}_j, \mathbf{z}_{ij})^\top \mathbf{\Omega}_{ij} \mathbf{e}(\mathbf{x}_i, \mathbf{x}_j, \mathbf{z}_{ij}) \quad (1)$$

$$\mathbf{x}^* = \underset{\mathbf{x}}{\operatorname{argmin}}\, \mathbf{F}(\mathbf{x}). \quad (2)$$

Here, $\mathbf{x} = (\mathbf{x}_1^\top, \ \ldots \ , \mathbf{x}_n^\top)^\top$ is a vector of pose representations $\mathbf{x}_i$. $\mathbf{z}_{ij}$ and $\mathbf{\Omega}_{ij}$ represent respectively the mean and the information matrix of a constraint relating the poses $\mathbf{x}_j$, i.e., the pairwise transformation computed by the front-end. $\mathbf{e}(\mathbf{x}_i, \mathbf{x}_j, \mathbf{z}_{ij})$ is a vector error function that measures how well the poses $\mathbf{x}_i$ and $\mathbf{x}_j$ satisfy the constraint $\mathbf{z}_{ij}$. It is $\mathbf{0}$ when $\mathbf{x}_i$ and $\mathbf{x}_j$ perfectly match the constraint, i.e., the difference of the poses exactly matches the estimated transformation. For more details on the error function, we refer the interested reader to [18].

Global optimization is especially beneficial in case of a loop closure, i.e., when revisiting known parts of the map, since the loop closing edges in the graph allows to diminish the accumulated error. Unfortunately, large errors in the motion estimation step may impede the accuracy of large parts of the graph. This is primarily a problem in areas of highly ambiguous features. We therefore use a threshold to prune edges with high error values after the initial convergence and continue the optimization.

C. Map Representation

The system described so far computes a globally consistent trajectory. Using this trajectory, the original data can be used to construct a representation of the environment. Projecting all point measurements into a global 3D coordinate system leads to a straightforward point-based representation. Such a model, however, is highly redundant and requires vast computational and memory resources.

To overcome these limitations, we use 3D occupancy grid maps to represent the environment. In our implementation, we use the octree-based mapping framework OctoMap [33]. Voxels are managed in an efficient tree structure that leads to a compact memory representation and inherently allows for map queries at multiple resolutions. The use of probabilistic occupancy estimation furthermore provides a means of coping with noisy measurements and errors in pose estimation. In contrast to a point-based representation, it represents free space and unmapped areas explicitly which is essential for robot navigation and exploration tasks.

IV. EVALUATION

To evaluate our system, we use the RGB-D benchmark [29] which provides a dataset of Kinect sequences with synchronized ground truth. Furthermore, the benchmark provides an evaluation tool that computes the root mean square error (RSME) given an estimated trajectory.

TABLE II

EVALUATION OF THE ACCURACY WITH RESPECT TO THE KEYPOINT DETECTOR AND FEATURE EXTRACTOR

	Success	Transl. RMSE (Avg. ± Std. Dev.)	Rot. RMSE (Avg. ± Std. Dev.)
SIFTGPU	9/9	0.097 m ± 0.063 m	3.95°± 2.47°
SURF	9/9	0.098 m ± 0.078 m	3.39°± 1.55°
ORB	7/9	0.215 m ± 0.189 m	7.75°± 5.55°

For our evaluation we choose the Freiburg1 (FR1) dataset consisting of nine sequences placed in a typical indoor environment. Two of these, the sequences "FR1 xyz/rpy" have very simple motions. The result on these sequences show the capabilities of our approach in the best case. However, results like this can usually be achieved when the sensor can be carefully moved in an indoor environment, e.g., during manual map recording prior to employing a robot. The other datasets are more challenging as they cover larger areas of the office space and unrestricted camera motions.

In this section, we first present our results on the accuracy of our SLAM system and evaluate how the accuracy depends on the chosen feature detector and sensor frame rate. Second, we investigate the influence of various parameters on the runtime of our system.

We evaluated our system on all nine sequences from the FR1 set, see Table I. As can be seen from this table, the average camera velocities range from 9 deg/s to 42 deg/s and from 0.06 m/s to 0.43 m/s.

A. *Accuracy of the Trajectory Estimation*

In our first rounds of experiments, we evaluated the accuracy of our system on all sequences using SIFTGPU feature extraction and approximate matching using FLANN. On the simple "xyz" and "rpy" sequences, we obtain the best values of 2.1 cm and 4.1 cm RMSE error, respectively. We achieved the worst result of 20.1 cm RMSE error on the "room" sequence of 16 m length consisting of a long round through an office space. In general, this evaluation shows that our approach performs well in most of the sequences. High angular and translational velocity pose no obvious difficulty even though frames exhibit less overlap.

Further, we investigated the influence of different choices for the keypoint detectors and feature descriptors. Table II shows the root mean square of the translational and rotational error per sequence. The accuracy achieved is similar for SIFT and SURF. In contrast, ORB features turn out to be less accurate and also less reliable: Using ORB, the trajectory estimation failed for two of the nine sequences. This could not be resolved by adapting the parameters of the feature detector to find more keypoints.

B. *Runtime Evaluation*

In this section we discuss the computational requirements of the presented system. All experiments were carried out on a quad-core CPU with 8 GB of memory. By parallelizing our software we achieved a speed-up by a factor between 2 and 2.5. Graph optimization is started after all frames have been processed and runs on a single core. The runtime results presented in Table I were generated using SIFTGPU and FLANN matching. On average, our system required 0.35 s per frame. The overall runtime performance strongly depends on the configurations presented in the previous sections. Therefore, we evaluated the runtime performance in more detail in the remainder of this section.

TABLE III

RUNTIME ANALYSIS OF FEATURE EXTRACTION WITH RESPECT TO THE CHOSEN INTEREST POINT DETECTOR AND FEATURE DESCRIPTOR. WE FOUND THAT ORB IS FASTER THAN SURF AND SIFT BY ONE ORDER OF MAGNITUDE.

Type	Count Avg. ± Std. Dev.	Runtime Detection + Extraction Avg. ± Std. Dev.
SURF	1733 ± 153	0.34 s + 0.34 s
ORB	1117 ± 558	0.018 s + 0.0086 s
SIFTGPU	1918 ± 599	0.19 s

TABLE IV

RUNTIME ANALYSIS OF PAIRWISE FRAME REGISTRATION. USING AN APPROXIMATELY NEAREST NEIGHBOR METHOD DURING MATCHING REDUCES THE RUNTIME BY A FACTOR OF TWO.

Matcher	Runtime (Avg. ± Std. Dev)
FLANN	0.203 s ± 0.078 s
Brute Force	0.386 s ± 0.120 s

1) Feature Detection and Descriptor Extraction: Our approach has to detect features and extract their descriptors in each incoming image frame. Table III shows a comparison of the computation time required for the different feature types as described in Section III-A. We found that ORB is faster than SURF and SIFTGPU by one order of magnitude. However, the system produced high errors in two of the nine sequences. SIFTGPU is still about 3.5 times as fast as the non-parallelized SURF implementation.

2) Feature Matching and Motion Estimation: Feature matching and motion estimation needs to be computed at least once per frame. If the current frame is only matched against one predecessor, the resulting camera trajectory will quickly accumulate errors over time. Feature matching for many frames is costly to compute, but especially since we do not assume the availability of any odometry information, only with the possibility for loop closures, the system can be accurate over longer trajectories. Further, more information about pairwise relative transformations makes the trajectory estimation more robust to errors in pose estimation. However, a densely connected pose graph also requires more time to optimize. We therefore found matching the current features to those of 20 previous frames a good compromise. Table IV shows the average runtime for matching and motion estimation. We found that FLANN reduces the time required for frame-to-frame registration by a factor of two.

3) Pose Graph Optimization: The optimization of small pose graphs is fast enough to be done in real time, i.e., for every frame. With longer sequences of densely connected poses, the optimization time increases. However, if the motion estimates are reliable, the global optimization is not required in every step. In all sequences, the ratio of graph optimization versus the total runtime was below 6 %.

V. CONCLUSION AND OUTLOOK

We presented a novel approach to visual SLAM from RGB-D sensors. Our approach extracts visual keypoints from the color images and uses the depth images to localize them in 3D. We use RANSAC to robustly estimate the transformations between RGB-D frames and optimize the pose graph using non-linear optimization. Finally, we generate a volumetric 3D map of the environment that can be used for robot localization, navigation and path planning. We evaluated our approach quantitatively on a publicly available RGB-D dataset. On this dataset our system achieves on average an accuracy of 9.7 cm and 3.95°. With an average frame processing time of 0.35 s our approach is suitable for online operation. To allow other researchers to use our software and reproduce the results (and improve on them), we have released all source code required to run and re-evaluate our RGB-D SLAM system as open-source.

During the evaluation of our system, we discovered that sometimes erroneous edges are created that lead to a degradation of the mapping result. It would be interesting to see how such problems can be efficiently detected and possibly repaired autonomously. In the near future, we want to optimize the keypoint matching scheme, for example by adding a feature dictionary, pruning never matched features, and integrating keypoints as landmarks directly during non-linear optimization.

REFERENCES

[1] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool. Speeded-up robust features (SURF). *Comput. Vis. Image Underst.*, 110:346–359, 2008.

[2] P. J. Besl and H. D. McKay. A method for registration of 3-D shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 14(2):239–256, 1992.

[3] G. Bradski and A. Kaehler. *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly Media, 2008.

[4] M. Calonder, V. Lepetit, C. Strecha, and P. Fua. BRIEF: Binary Robust Independent Elementary Features. In *Proc. of the Europ. Conf. on Computer Vision (ECCV)*, 2010.

[5] A.J. Davison. Real-time simultaneous localisation and mapping with a single camera. In *Proc. of the IEEE Intl. Conf. on Computer Vision (ICCV)*, 2003.

[6] F. Dellaert. Square Root SAM. In *Proc. of Robotics: Science and Systems (RSS)*, pages 177–184, 2005.

[7] N. Fioraio and K. Konolige. Realtime visual and point cloud slam. In *Proc. of the RGB-D Workshop on Advanced Reasoning with Depth Cameras at Robotics: Science and Systems Conf. (RSS)*, 2011.

[8] M.A. Fischler and R.C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, 1981.

[9] U. Frese, P. Larsson, and T. Duckett. A multilevel relaxation algorithm for simultaneous localisation and mapping. *IEEE Trans. on Robotics*, 21(2):1–12, 2005.

[10] G. Grisetti, S. Grzonka, C. Stachniss, P. Pfaff, and W. Burgard. Efficient estimation of accurate maximum likelihood maps in 3d. In *Proc. of the Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2007.

[11] G. Grisetti, R. Kümmerle, C. Stachniss, U. Frese, and C. Hertzberg. Hierarchical optimization on manifolds for online 2D and 3D mapping. In *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2010.

[12] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox. RGB-D mapping: Using depth cameras for dense 3D modeling of indoor environments. In *Proc. of the Intl. Symp. on Experimental Robotics (ISER)*, 2010.

[13] A. Hornung, K.M. Wurm, and M. Bennewitz. Humanoid robot localization in complex indoor environments. In *Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2010.

[14] H. Jin, P. Favaro, and S. Soatto. Real-time 3-d motion and structure of point-features: A front-end for vision-based control and interaction. In *Proc. of the IEEE Intl. Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2000.

[15] M. Kaess, A. Ranganathan, and F. Dellaert. iSAM: Incremental smoothing and mapping. *IEEE Trans. on Robotics*, 24(6):1365–1378, 2008.

[16] G. Klein and D. Murray. Parallel tracking and mapping for small AR workspaces. In *Proc. IEEE and ACM Intl. Symp. on Mixed and Augmented Reality (ISMAR)*, 2007.

[17] K. Konolige, M. Agrawal, R. Bolles, C. Cowan, M. Fischler, and B. Gerkey. Outdoor mapping and navigation using stereo vision. In *Experimental Robotics*, volume 39 of *Springer Tracts in Advanced Robotics*, pages 179–190. Springer, 2008.

[18] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. g2o: A general framework for graph optimization. In *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2011.

[19] M.I.A. Lourakis and A.A. Argyros. SBA: a software package for generic sparse bundle adjustment. *ACM Transactions on Mathematical Software*, 2009.

[20] D.G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[21] D. Nister. Preemptive RANSAC for live structure and motion estimation. In *Proc. of the IEEE Intl. Conf. on Computer Vision (ICCV)*, 2003.

[22] A. Nüchter, K. Lingemann, J. Hertzberg, and H. Surmann. 6D SLAM with approximate data association. In *Proc. of the 12th Intl. Conference on Advanced Robotics (ICAR)*, pages 242–249, 2005.

[23] E. Olson, J. Leonard, and S. Teller. Fast iterative optimization of pose graphs with poor initial estimates. In *Proc. of the IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 2262–2269, 2006.

[24] L. M. Paz, P. Pinies, J. D. Tardos, and J. Neira. Large-scale 6-DOF SLAM with stereo-in-hand. *IEEE Transactions on Robotics*, 24(5):946–957, 2008.

[25] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. ORB: an efficient alternative to SIFT or SURF. In *Proc. of the IEEE Intl. Conf. on Computer Vision (ICCV)*, volume 13, 2011.

[26] S. Rusinkiewicz and M. Levoy. Efficient variants of the ICP algorithm. In *Proc. of the Intl. Conf. on 3-D Digital Imaging and Modeling*, 2001.

[27] A. Segal, D. Haehnel, and S. Thrun. Generalized-ICP. In *Proceedings of Robotics: Science and Systems*, 2009.

[28] H. Strasdat, J. M. M. Montiel, and A. Davison. Scale drift-aware large scale monocular slam. In *Proceedings of Robotics: Science and Systems*, 2010.

[29] J. Sturm, S. Magnenat, N. Engelhard, F. Pomerleau, F. Colas, W. Burgard, D. Cremers, and R. Siegwart. Towards a benchmark for RGB-D SLAM evaluation. In *Proc. of the RGB-D Workshop on Advanced Reasoning with Depth Cameras at Robotics: Science and Systems Conf. (RSS)*, 2011.

[30] S. Thrun. Robotic mapping: A survey. In *Exploring Artificial Intelligence in the New Millennium*. Morgan Kaufmann, 2003.

[31] Shinji Umeyama. Least-squares estimation of transformation parameters between two point patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (13), 1991.

[32] Changchang Wu. SiftGPU: A GPU implementation of scale invariant feature transform (SIFT). `http://cs.unc.edu/~ccwu/siftgpu`, 2007.

[33] 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.