Efficient Onboard RGBD-SLAM for Autonomous MAVs

Sebastian A. Scherer[1] and Andreas Zell[1]

***Abstract*— We present a computationally inexpensive RGBD-SLAM solution taylored to the application on autonomous MAVs, which enables our MAV to fly in an unknown environment and create a map of its surroundings completely autonomously, with all computations running on its onboard computer. We achieve this by implementing efficient methods for both tracking its current location with respect to a heavily processed previously seen RGBD image (keyframe) and efficient relative registration of a set of keyframes using bundle adjustment with depth constraints as a front-end for pose graph optimization. We prove the accuracy and efficiency of our system based on a public benchmark dataset and demonstrate that the proposed method enables our quadrotor to fly autonomously.**

I. INTRODUCTION

Autonomous MAVs (micro aerial vehicles) are getting more and more attention within robotics research. For true autonomy, we want a MAV to neither rely on external sensors (i.e. an external tracking system) nor on offloading processing to an external computer for autonomous navigation. RGBD cameras, i.e. cameras that provide registered color and depth images, seem to be the perfect sensor for this task: They are small, light-weight, and provide rich information in hardware already, without requiring additional expensive postprocessing computations by the onboard computer, like stereo cameras do.

One of the most essential problems for autonomous MAVs is localization or pose estimation: Other than ground robots, a MAV typically cannot just stop and wait for pose estimates. In order not to crash, it needs reliable pose estimates in 6D (position and orientation) at a high frequency. If the autonomous MAV should also be able to navigate in previously unkonwn environments, it has to solve the SLAM (simultaneous localization and mapping) problem.

II. RELATED WORK

A. RGBD SLAM

SLAM using RGBD cameras has attracted much attention already. The first RGBD mapping system was presented in the year 2010 in [1], which works by matching 3D points of pairs of RGBD frames using SIFT features for an inital estimate of the relative transformation, followed by refinement using iterative closest point (ICP). The whole map is kept consistent using pose graph optimization (PGO). The open source system RGBD-SLAM[1] described in [2] works in a similar fashion: It registers RGBD frames by matching 3D point pairs using various available image features but without the successive ICP refinement step. There are also some approaches that rely on depth images alone, e.g. KinectFusion [3], which uses coarse-to-fine iterative closest point with projective data association implemented on graphics hardware for registration with the map. The authors themselves in [4] introduced bundle adjustment with depth constraints, which allows us to easily extend monocular visual SLAM systems like the very efficient PTAM system [5] to also utilize depth measurements of RGBD data. This method was shown to enable autonomous flight of a MAV with a stereo camera in [6], but has some systematic limitations: For optimizing the full map, PTAM relies on global bundle adjustment, which quickly becomes computationally infeasible in real time for large numbers of keyframes.

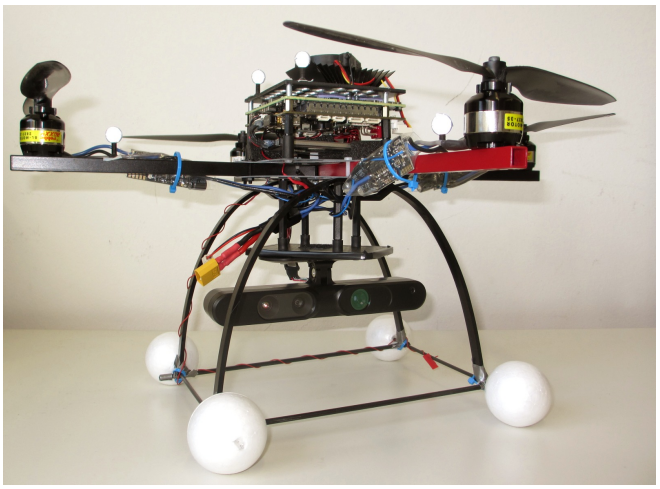


Fig. 1. The RGBD-UAV used for this work.

B. Autonomous MAVs using RGBD Cameras

Notable cases of autonomous MAVs using RGBD cameras include [7], in which the MAV uses the RGBD camera to compute visual odometry onboard and mapping is done on an external computer using the system detailed in [1]. While this is impressive work, it is not a fully autonomous MAV according to our previous definition.

In [8], an RGBD camera mounted on a quadrotor is used for indoor exploration, which is an interesting topic on its own. Pose estimates, however, are provided using a laser range finder also mounted in the MAV.

[1] Sebastian A. Scherer and Andreas Zell are with the Department of Computer Science, Faculty of Science, University of Tuebingen, Tuebingen, Germany. {sebastian.scherer, andreas.zell} at uni-tuebingen.de

[1]http://openslam.org/rgbdslam.html

C. Our Approach

Our goal is an RGBD-SLAM system that should enable a fully autonomous MAV. This means we need it to produce pose estimates that are accurate and fast enough to enable autonomous flight while all processing required for both localization and mapping should run on the onboard computer. This work is customized for the limitations of our RGBD-MAV described in sect. III. The general approach, however, should be almost universally applicable.

III. HARDWARE SETUP

Our RGBD-MAV is a quadrotor helicopter based on the open source Pixhawk platform [9]. It uses a frame, brushless motor controllers and motors from Mikrokopter[2], the pxIMU inertial measurement unit and autopilot and a Kontron microETXexpress-PC single-board computer (SBC) for computer vision tasks hosted on the pxCOMex COM-Express baseboard. Its main sensor is an ASUS Xtion Pro Live RGBD camera, which is similar to a Microsoft Kinect, but smaller, lighter, and provides slightly better synchronized color and depth images.

IV. SOFTWARE ARCHITECTURE

We implement a keyframe-based RGBD-SLAM system focused on being able to use it on our RGBD-MAV described in sect. III. Its main requirements are being able to generate pose estimates that are both accurate and fast enough to allow autonomous navigation on the one hand, and a scalable mapping process that should be able to map environments that are not limited to parts of a room.

We adopt the major idea of PTAM, namely splitting up the SLAM task in one thread for tracking (or localization) running at video rate and a second thread for mapping, i.e. extending and optimizing the map. As opposed to PTAM, however, we employ a relative representation of our map which is inspired by [10]. This means that each map point belongs to one source keyframe, in our case the one in which it was measured for the first time, and has its position stored only relative to its source keyframe.

This relative representation allows the combination of bundle adjustment as a SLAM front-end with pose graph optimization as a back-end and helps keeping the map consistent: If pose graph optimization modifies the pose of one keyframe, the positions of all of its map points will implicitly be modified as well.

A. Map

The map in our case is a set of keyframes, which in turn contain all relevant information: An RGBD image pair, map points, i.e. triangulated interest points of the RGB image, and measurements of map points of other keyframes within its own image. Additionally, each keyframe corresponds to a node in our pose graph. Pose graph edges are added whenever keyframes were registered with respect to each other.

[2]http://www.mikrokopter.de

B. Localization: Tracking Thread

The tracking thread is responsible for processing incoming RGBD pairs and producing pose estimates at video rate. For each new RGBD pair it will predict the current camera pose based on a motion model, choose the best keyframe within the map to track its pose, project all map points of this best keyframe into its current image based on the pose predicted by the motion model, find these map points close to their expected positions within the current image using sparse optical flow, find the camera pose relative to the reference keyframe using robust nonlinear least squares, and finally decide whether it should create a new keyframe from the current RGBD pair.

1) Motion Model: We use a modified version of the decaying-velocity motion model in [5], which assumes the MAV to keep flying at a nearly constant but slowly decaying velocity. Given a previous 6D velocity estimate $v_{t-1} \in se(3)$ and the old relative pose ${}^{C_{t-1}}T_R \in SE(3)$, were C_{t-1} is the camera frame at time $t-1$ and R is the reference frame, it will predict the new pose at time t as:

$$ {}^{C_t}T_R = \exp(v_{t-1}) \cdot {}^{C_{t-1}}T_R \tag{1} $$

Where $\exp : SE(3) \mapsto se(3)$ is the exponential map and $\log : se(3) \mapsto SE(3)$ its inverse. From the equation above it follows that the velocity estimate should be:

$$ v_{t-1} = \log\left({}^{C_t}T_{C_{t-1}}\right) = \log\left({}^{C_t}T_R \cdot {}^{C_{t-1}}T_R^{-1}\right) \tag{2} $$

Since we do not know the true pose ${}^{C_t}T_R$ at the time of prediction, we estimate the velocity for the next prediction after successful localization, assuming it to remain roughly constant:

$$ v_t = \alpha_1\left(\alpha_2 \cdot \log\left({}^{C_t}T_{C_{t-1}}\right) + (1-\alpha_2)v_{t-1}\right) \tag{3} $$

With $\alpha_1, \alpha_1 \in (0,1)$, this is a basic decaying low-pass filter. Special care has to be taken since we track relative poses only: Compared to a previous update step, we could be using a different reference keyframe in the current step, i.e. we have different reference frames R_t and R_{t-1}. The prediction in eq. 1 must then be computed using:

$$ {}^{C_t}T_{R_t} = \exp(v_{t-1}) \cdot {}^{C_{t-1}}T_{R_{t-1}} \cdot {}^{R_{t-1}}T_{R_t} \tag{4} $$

And the transform ${}^{C_t}T_{C_{t-1}}$ required for estimating the relative velocitiy in eq. 2 is in fact:

$$ {}^{C_t}T_{C_{t-1}} = {}^{C_t}T_{R_t} \cdot {}^{R_t}T_{R_{t-1}} \cdot {}^{C_{t-1}}T_{R_{t-1}}^{-1} \tag{5} $$

The relative transform between both references required in both steps has to be recomputed at the moment it is needed based on their current pose estimates relative to a fixed world frame, since it might change due to the mapping thread modifying the map in parallel:

$$ {}^{R_t}T_{R_{t-1}} = {}^{R_t}T_W \cdot \left({}^{R_{t-1}}T_W^{-1}\right) \tag{6} $$

2) Keyframe Selection: From all keyframes within the map, we want to select the one which is best for tracking. In our case, this means we want to find as many map points as possible in the current image, so we select the keyframe with the most map points theoretically visible in the current image. We can compute this by projecting all map points into the current image based on its predicted pose and counting how many map points would be visible, i.e. are in front of the camera and their projection is within the image boundary. Since the number of keyframes might grow too big and this simple heuristic does not consider occlusion or filter out keyframes far away, we restrict our search to keyframes within a certain distance of the predicted pose both in translation and orientation.

3) Finding Map Points using Sparse Optical Flow: After determining the expected positions of all relevant map points within the current image, we need to find their actual locations nearby. PTAM looks for map points at locations of FAST corners by computing the zero-mean sum of squared differences (ZMSSD) between the patches around both. In our experience, this often fails when there is considerable motion blur as the number of FAST corners is limited. We instead decided to use sparse optical flow using the Lucas-Kanade method [11] using the implementation in OpenCV [12] as it requires finding interest points in keyframes only and can cope with at least some motion blur in images in between. Sparse optical flow, however, will always find the local optimum and is therefore usually not able to do wide-baseline matching. When using a proper initialization as provided by applying even a very simple motion model, however, the gaps between expected and actual positions of map points that have to be bridged by are typically very small, which makes finding a wrong local minimum in between rather unlikely. This is illustrated in fig. 2.

Fig. 2. A typical result of tracking: Green and blue lines are distances covered by the prediction step of our motion model, red lines are the gaps that have to be bridged by sparse optical flow. They are difficult to see beacuse most are very short for inliers. Outliers that were removed are marked in blue.

4) Relative Pose Estimation: Determining the pose of the camera relative to the reference keyframes based on the known 3D position of some map points and their 2D image coordinates in the current frame is called the Perspective-n-Point (PnP) Problem.

Since tracking using sparse optical flow always yields a small number of wrong matches, we first determine inliers and outliers using preemptive RANSAC [13] with Gao's solution to the P3P problem [14] to generate hypotheses. After that, we refine the best hypothesis based on all inlier measurements using robust nonlinear least squares. In addition to 2D image coordinates, we also consider depth measurements at the corresponding image locations, if available. The mathematical details of this optimization are described in sect. V-A

5) Deciding about Adding a new Keyframe: The decision whether we should add a new keyframe is made using the following heuristic: If either the distance (both in translation and orientation) relative to the best keyframe or the mean distance of tracked points in 2D image coordinates exceed certain limits, or if the visible ratio of map points of the reference keyframe goes below a certain limit, the current frame should be added as a new keyframe and is passed along to the mapping thread.

C. Mapping

Combining all incoming RGBD frames that are supposed to become keyframes to a consistent and accurate map is the job of the second mapping thread. It will create keyframes from RGBD pairs, refine the pose estimate from tracking using bundle adjustment with depth constraints, try and match more reference keyframes to the latest keyframe, and finally run pose graph optimization to optimize the map.

1) Keyframe Creation: The main task in keyframe creation is selecting map points. Since keyframe creation has to be fast, we apply a FAST corner detector on multiple pyramid levels. As also noticed in [15], FAST corners tend to flock to image regions with high contrast whereas there might be some regions with no FAST corners at all. We tackle this problem by detecting many FAST corners more than needed and assigning each to one of $n \times n$ grid cells within the image. We then sort all corners within a cell by their Harris score and keep only the best m. The result of this process is illustrated in fig. 3.

2) Relative Pose Refinement: From tracking we know a rough pose estimate of the latest keyframe w.r.t. its reference keyframe and a list of measured map points of the reference keyframe in the image of the latest keyframe. This is good enough for tracking, but we want to refine this pose estimate before using it for mapping.

We first find more matches by backward tracking, i.e. trying to find all map points of the new keyframe within the reference keyframe's image using the same method as described in IV-B. Using this higher number of matches, we apply bundle adjustment with depth constraints to find the accurate relative pose between both keyframes. The details of this optimization are described in sec. V-B.

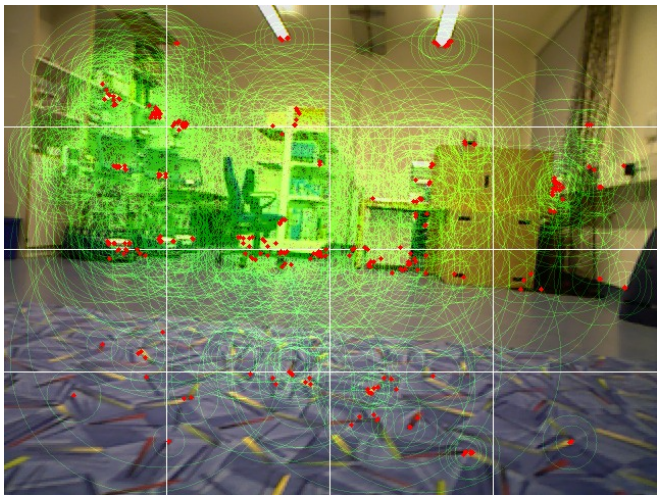


Fig. 3. Keyframe creation: Green circles: FAST corners on various pyramid levels. White lines: Grid layout for map point selection. Red dots: Actually used map points.

Once bundle adjustment is done, we update the refined 3D positions of the relevant map points, update the refined pose of the latest keyframe, and add an edge between the latest keyframe and its reference to the pose graph.

D. Finding and Utilizing More Reference Keyframes

1) Finding Match Candidates: Strictly speaking, the steps described so far compose a visual odometry system, but not a complete SLAM system. For full SLAM, we need to be able to detect and handle loops. We do this by trying to track the latest keyframe in the n_{ref} best keyframes that were again found as described in sect. IV-B.2.

2) Verifying Match Candidates: We have to be extra careful, however, not to add badly registered keyframe pairs to our pose graph as basic pose graph optimization is not robust to outliers.[3] To overcome this problem given a candidate pair of keyframes k_i and k_j, we first try to find the two relative poses ${}^{i}T_j$ and ${}^{j}T_i$ by tracking keyframe i relative to keyframe j and vice versa. Only if both pose estimates roughly agree (i.e. ${}^{i}T_j \approx {}^{j}T_i{}^{-1}$), we continue to refine this pose estimate and add a corresponding edge to the pose graph.

3) Refining Matches: In order to refine such an additional edge, we again utilize bundle adjustment with depth constraints as described in sect. V-B.

E. Global Pose Graph Optimization

We use HOG-MAN [17] to optimize the pose graph consisting of keyframes (nodes) and relative poses (edges) that were added after registering pairs of keyframes.

V. Optimization with Depth Constraints

We use depth constraints as introduced in [4] in two stages of our SLAM system: For tracking by solving a modified perspective-n-point problem, and for registering pairs of keyframes, using a modified variant of relative bundle adjustment.

A. Perspective-n-Point Problem with Depth

For the conventional pespective-n-point problem, we are given n map points ${}^{R}p_i$ of a reference keyframe R w.r.t. the pose of R and their image locations ${}^{C}u_i$ where they were seen in the current camera image C. We then want to find the relative pose ${}^{C}T_R$ that minimizes the following 2D reprojection errors:

$$e_{2D,i} = h({}^{C}T_R\,{}^{R}p_i) - {}^{C}u_i \tag{7}$$

Here, h is the camera projection function that projects 3D points given in the camera coordinate system to pixel coordinates. In our case, we use the typical pinhole-camera model with radial and tangential distortion as also used in OpenCV. The error term $e_{2D,i}$ is thus the difference between the expected and actually measured pixel coordinates.

In addition to 2D image locations, however, we often also measure valid depth values at these pixel positions. We integrate this additional information by adding depth reprojection errors:

$$e_{d,i} = \begin{cases} \left({}^{C}T_R\,{}^{R}p_i\right)_3 - {}^{C}d_i & \text{if } {}^{C}d_i > 0 \\ 0 & \text{else} \end{cases} \tag{8}$$

Both errors are of unrelated dimensions (e_{2D} is in pixels, e_d in meters). They need to be properly scaled according to their expected uncertainties σ_{2D} and σ_d before they can be combined. We use $\sigma_{2D} = 0.5\,\text{px}$ and $\sigma_d(d) = d^2 \cdot 6.331 \times 10^{-3}\text{m}^{-1}$ as determined in [4]. Note that $\sigma_d(d)$ is not constant but a function of depth d.

Since image positions obtained using sparse optical flow always contain a small number of outliers, we need to apply robustified nonlinear least squares optimization. This can be obtained by applying a robustifier ρ which scales all errors. The resulting objective function to be minimized in order to find the optimal ${}^{C}T_R$ is:

$$\sum_i \rho\left(\frac{||e_{2D,i}||^2}{\sigma_{2D}^2}\right) + \rho\left(\frac{||e_{d,i}||^2}{\sigma_d({}^{C}d_i)^2}\right) \tag{9}$$

We implement this optimization using ceres solver[4] [18], an open source c++ library for robust nonlinear least squares problems and choose ρ to be the Huber loss function.

B. Relative Bundle Adjustment with Depth Constraints

Deviating from [4], we need to modify our bundle adjustment formulation to accomodate the relative representation chosen for this work. Given two keyframes, let us call map points that were measured not only in their source keyframe S but also in the other keyframe O *relevant map points*. Each relevant map point was per definition measured at least twice with 2D pixel coordinates and there might be valid depth measurements in none, one, or both depth images.

[3] There is work on robustified variants of pose graph optimization, e.g. [16]. We doubt, however, that it will work well with our small numbers of edges: If there is one inlier and one outlier edge, it might be impossible to decide which one to discard as an outlier.

[4] http://code.google.com/p/ceres-solver/

Each of these up to four measurements leads to a slightly different reprojection error. For each map point at position ${}^S p_i$ which is stored relative to its source keyframe S:

- We always measure it in its own keyframe at a pixel location ${}^S u_i$, which leads to reprojection error $e_{src,2D,i}$ in eq. 10.
- We always measure it in the other keyframe at a pixel location ${}^O u_i$, which leads to $e_{oth,2D,i}$ in eq. 11.
- We might measure valid depth ${}^S d_i$ at its location within its source keyframe, which leads to $e_{src,d,i}$ in eq. 12.
- We might measure valid depth ${}^O d_i$ at its measured location within the other keyframe, which leads to $e_{oth,d,i}$ in eq. 13.

$$e_{src,2D,i} = h({}^S p_i) - {}^S u_i \tag{10}$$

$$e_{oth,2D,i} = h({}^O T_W \, {}^S T_W^{-1} \, {}^S p_i) - {}^O u_i \tag{11}$$

$$e_{src,d,i} = \begin{cases} \left({}^S p_i\right)_3 - {}^S d_i & \text{if } {}^S d_i > 0 \\ 0 & \text{else} \end{cases} \tag{12}$$

$$e_{oth,d,i} = \begin{cases} \left({}^O T_W \, {}^S T_W^{-1} \, {}^S p_i)\right)_3 - {}^O d_i & \text{if } {}^O d_i > 0 \\ 0 & \text{else} \end{cases} \tag{13}$$

Again, 2D and depth reprojection errors need to be properly scaled and the whole system needs to be robust to a small number of outliers so the final optimization problem is:

$$\arg\min \sum_i \rho\left(\frac{||e_{src,2D,i}||^2}{\sigma_{2D}^2}\right) + \rho\left(\frac{||e_{oth,2D,i}||^2}{\sigma_{2D}^2}\right) \tag{14}$$

$$+\rho\left(\frac{||e_{src,d,i}||^2}{\sigma_d({}^C d_i)^2}\right) + \rho\left(\frac{||e_{oth,d,i}||^2}{\sigma_d({}^C d_i)^2}\right) \tag{15}$$

Which we again solve using ceres, choosing ρ as the Huber loss function.

VI. EVALUATION: BENCHMARK DATASET

We first evaluate our system on the *fr3/long_office_household* file from the benchmark dataset described in [19] and available online[5]. Our two main objectives are computational efficiency and accuracy of the pose estimates, so we evaluate both.

A. Computational Cost

We measure and log the time required for each individual step on two different computers: On an average laptop computer with an Intel Core 2 Duo CPU P9400 running at 2.40 GHz, and the onboard single-board computer used on our UAV with an Intel Core 2 Duo SL9400 Low Voltage running at 1.86 GHz. We do this by running our system in a sequential single-threaded mode utilizing only one core. The measured mean times and their standard deviation are shown in table I for tasks of the localization thread and in table II for tasks of the mapping thread.

We can see that tracking is really fast and can be completed in much less than the 33 ms required for video rate (30 Hz) processing, even on the slower onboard single-board computer. The mapping thread has to perform some more complex computations, which does not pose a problem, however, since these only need to be performed when adding a new keyframe. Also, except for keyframe creation, all tasks might be interrupted without breaking the mapping process. Currently, we stop checking for more additional edges as soon as there is a newer keyframe to add, but we could also actively interrupt each task except keyframe creation: Not checking for additional edges a few times is not a problem, and we could even live without reverse tracking or bundle adjustment and just trust the pose estimate from tracking.

task	laptop [ms]	SBC [ms]
sparse optical flow	6.38 ± 1.06	8.38 ± 1.23
preemptive RANSAC	3.80 ± 0.46	5.23 ± 0.28
robust optimization	1.90 ± 0.93	2.44 ± 1.18
total	12.08 ± 1.46	16.05 ± 1.62

TABLE I
COMPUTATION TIMES REQUIRED FOR LOCALIZATION (TRACKING THREAD).

task	laptop [ms]	SBC [ms]
keyframe creation	12.54 ± 2.02	15.60 ± 2.20
reverse tracking	13.05 ± 4.16	16.41 ± 1.32
refinement (BA)	32.96 ± 15.20	43.62 ± 19.27
additional edge accepted	51.95 ± 12.50	67.46 ± 15.74
additional edge rejected	22.11 ± 2.08	30.03 ± 2.63
pose graph optimization	16.20 ± 9.61	21.70 ± 12.66

TABLE II
COMPUTATION TIMES REQUIRED FOR PROCESSING A NEW KEYFRAME (MAPPING THREAD).

B. Accuracy

We evaluate the accuracy of our pose estimates using the tool included with the benchmark dataset. It reports an absolute position root-mean-square error (RMSE) of 0.136 m comparing the full estimated trajectory to ground truth. As we do both tracking and refinement relative to reference keyframes, we are mainly interested in relative position and orientation errors relative to the corresponding reference keyframe. This is shown in table III. Note that the errors for tracking and refinement are not directly comparable as the transforms estimated in refinement can typically represent considerable steps in translation and/or rotation, whereas the

method	tracking	refinement (BA)
position RMSE	1.8 cm	1.4 cm
position MAE	1.2 cm	1.1 cm
orientation RMSE	0.95°	0.85°
orientation MAE	0.67°	0.60°

TABLE III
ERRORS OF POSE ESTIMATES RELATIVE TO REFERENCE KEYFRAME ON THE FREIBURG3 DATASET.

[5] http://vision.in.tum.de/data/datasets/rgbd-dataset

transforms estimated in tracking usually consist of very small motions.

Fig. 4. Visualization of the map obtained by applying our proposed system on the freiburg3 dataset: Keyframe poses (red), edges between keyframes (golden), and map points (in their original color).

A visualization of the resulting map obtained by running our SLAM system on the freiburg3 dataset is shown in fig. VI-B. The trajectory starts at the keyframe all the way at the bottom and ends where the colored pose marker is. It is interesting to see that our system is accurate enough that it implicitly closed several loops without active loop closure detection, simply because rather old keyframes were among the closest.

VII. EVALUATION: AUTONOMOUS FLIGHT

We use the same system running in real-time on the onboard computer of our MAV while it is flying autonomously in our laboratory. We let it follow a predefined list of waypoints arranged on a rectangular pattern. Position control is achieved by the nested PID controller implemented on the pxIMU autopilot. Whenever the MAV is close enough to its current waypoint, it uses the successive waypoint as the set point of its position controller.

We let it fly autonomously based on pose estimates provided by an external tracking system, run our SLAM system onboard, and record all data for offline evaluation and inspection. We captured this flight on camera and show the logged data (visualizations of the tracking and mapping processes) in the video attached to this paper. A higher-quality version of this video will also be available on our youtube channel.[6]

A. Accuracy

Since the computational cost does not change with different data, we only evaluate the accuracy based on ground truth poses provided by an external tracking system. We use an Optitrack tracking system by Naturalpoint, currently using a total of 7 V100:R2 cameras to provide pose estimates at 100Hz. We again compute relative errors in position and orientation for both tracking and refinement. The resulting errors are shown in table IV. They are slightly higher in this case, which is mainly due to the starting and landing phases, during which the MAV moves very quickly, which introduces considerable motion blur.

[6]http://www.youtube.com/user/ZellTuebingen

method	tracking	refinement (BA)
position RMSE	1.7 cm	2.5 cm
position MAE	1.5 cm	2.3 cm
orientation RMSE	1.71°	1.00°
orientation MAE	1.30°	0.79°

TABLE IV
ERRORS ON THE AUTONOMOUS MAV DATASET.

VIII. CONCLUSIONS

We present a very efficient RGBD-SLAM system which is able to run in real-time on the onboard single-board computer of our autonomous MAV. This is achieved by a combination of fast tracking and localization relative to a single keyframe and bundle adjustment with depth constraints as a SLAM front-end for pose graph optimization.

Future work will focus on making this system more robust, especially during the critical phases of takeoff and landing. We also want to test this system for larger-scale mapping and take advantage of the resulting map for autonomous navigation, i.e. path planning.

ACKNOWLEDGMENT

The authors would like to thank Konstantin Schauwecker for providing his code for waypoint following, Shaowu Yang for helping out when we had hardware problems and Prof. Dr. Andreas Schilling for letting us use his tracking system.

REFERENCES

[1] 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 *the 12th International Symposium on Experimental Robotics (ISER)*, vol. 20, 2010, pp. 22–25.

[2] F. Endres, J. Hess, N. Engelhard, J. Sturm, D. Cremers, and W. Burgard, "An evaluation of the RGB-D SLAM system," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 2012, pp. 1691–1696.

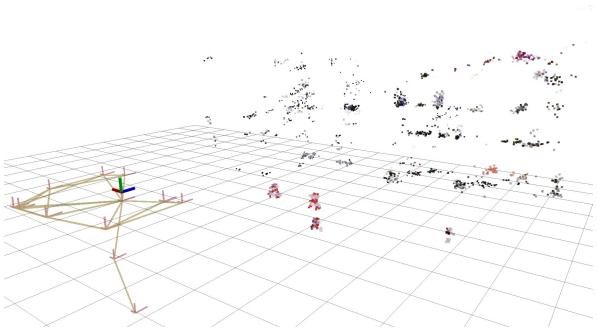
[3] R. A. Newcombe, A. J. Davison, S. Izadi, P. Kohli, O. Hilliges, J. Shotton, D. Molyneaux, S. Hodges, D. Kim, and A. Fitzgibbon, "KinectFusion: Real-time dense surface mapping and tracking," in *Mixed and Augmented Reality (ISMAR), 2011 10th IEEE International Symposium on*. IEEE, 2011, pp. 127–136.

[4] S. A. Scherer, D. Dube, and A. Zell, "Using depth in visual simultaneous localisation and mapping," in *Proceedings of the IEEE International Conference on Robotics and Automation*, St. Paul, Minnesota, USA, May 2012.

[5] G. Klein and D. Murray, "Parallel tracking and mapping for small AR workspaces," in *Proc. Sixth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'07)*, Nara, Japan, November 2007.

[6] K. Schauwecker, N. R. Ke, S. A. Scherer, and A. Zell, "Markerless Visual Control of a Quad-Rotor Micro Aerial Vehicle by Means of On-Board Stereo Processing," in *22nd Conference on Autonomous Mobile Systems (AMS)*. Stuttgart, Germany: Springer, September 2012, pp. 11–20.

[7] A. S. Huang, A. Bachrach, P. Henry, M. Krainin, D. Maturana, D. Fox, and N. Roy, "Visual odometry and mapping for autonomous flight using an RGB-D camera," in *International Symposium on Robotics Research (ISRR)*, 2011.



(a) Visualization of the actual map: Keyframe poses, edges between keyframes, and map points.

(b) Reconstruction based on the full point clouds of all keyframes transformed according to their keyframe poses. The estimated trajectory is shown as red dots.

Fig. 5. Mapping result obtained while the MAV flies autonomously within our laboratory.

[8] S. Shen, N. Michael, and V. Kumar, "Autonomous Multi-Floor Indoor Navigation with a Computationally Constrained MAV," in *Proceedings of the IEEE International Conference on Robotics and Automation*. IEEE, 2011, pp. 20–25.

[9] L. Meier, P. Tanskanen, F. Fraundorfer, and M. Pollefeys, "PIXHAWK: A system for autonomous flight using onboard computer vision," in *Proceedings of the IEEE International Conference on Robotics and Automation*, May 2011, pp. 2992–2997.

[10] G. Sibley, C. Mei, I. Reid, and P. Newman, "Adaptive relative bundle adjustment," in *Robotics Science and Systems Conference*, 2009, pp. 1–8.

[11] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in *Proceedings of the 7th international joint conference on Artificial intelligence*, 1981, pp. 674–679.

[12] J.-Y. Bouguet, "Pyramidal implementation of the lucas kanade feature tracker: Description of the algorithm," Tech. Rep., 2001.

[13] D. Nistér, "Preemptive RANSAC for live structure and motion estimation," *Machine Vision and Applications*, vol. 16, no. 5, pp. 321–329, 2005.

[14] X.-S. Gao, X.-R. Hou, J. Tang, and H.-F. Cheng, "Complete solution classification for the perspective-three-point problem," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, no. 8, pp. 930–943, 2003.

[15] K. Schauwecker, R. Klette, and A. Zell, "A new feature detector and stereo matching method for accurate high-performance sparse stereo matching," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Vilamoura, Algarve, Portugal: IEEE, October 2012, pp. 5171–5176.

[16] N. Sünderhauf and P. Protzel, "Towards a robust back-end for pose graph SLAM," in *Proceedings of the IEEE International Conference on Robotics and Automation*. IEEE, 2012, pp. 1254–1261.

[17] G. Grisetti, R. Kümmerle, C. Stachniss, U. Frese, and C. Hertzberg, "Hierarchical optimization on manifolds for online 2d and 3d mapping," in *Proceedings of the IEEE International Conference on Robotics and Automation*, Anchorage, AK, USA, May 2010.

[18] S. Agarwal and K. Mierle, *Ceres Solver: Tutorial & Reference*, Google Inc.

[19] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of RGB-D SLAM systems," in *Proceedings of the 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Vilamoura, Algarve, Portugal, October 2012.