Robust Onboard Visual SLAM for Autonomous MAVs

Shaowu Yang, Sebastian A. Scherer and Andreas Zell

Department of Computer Science, University of Tuebingen, Tuebingen, Germany
{shaowu.yang,sebastian.scherer,andreas.zell}@uni-tuebingen.de

Abstract. This paper presents a visual simultaneous localization and mapping (SLAM) system consisting of a robust visual odometry and an efficient back-end with loop closure detection and pose-graph optimization. Robustness of the visual odometry is achieved by utilizing dual cameras pointing different directions with no overlap in their respective fields of view mounted on an micro aerial vehicle (MAV). The theory behind this dual-camera visual odometry can be easily extended to applications with multiple cameras. The back-end of the SLAM system maintains a keyframe-based global map, which is used for loop closure detection. An adaptive-window pose-graph optimization method is proposed to refine keyframe poses of the global map and thus correct pose drift that is inherent in the visual odometry. The position of each map point is then refined implicitly due to its relative representation to its source keyframe. We demonstrate the efficiency of the proposed visual SLAM algorithm for applications onboard MAVs in experiments with both autonomous and manual flights. The pose tracking results are compared with the ground truth data provided by an external tracking system.

1 INTRODUCTION

Micro aerial vehicles (MAVs) are potentially able to efficiently navigate in complex 3D environments with different types of terrains, which might be inaccessible to ground vehicles. A basic requirement for MAVs to autonomously operate in such environments is robust pose tracking, which is still a challenging task when the environment is unknown. Meanwhile, mapping the environment can provide support to path planning for safe autonomous navigation. Recently, more focus has been on using onboard visual solutions to address these issues, especially using visual simultaneous localization and mapping (SLAM) systems, duo to the advantages offered by relying on cameras: their superior potential for environment perception, being lightweight and energy efficient which is especially important for MAVs.

In [12], we implemented a visual SLAM system, which can utilize measurements from multiple cameras. Those cameras looking in different directions can provide more reliable image features for pose tracking, compared to a monocular camera. The expanded field of view (FOV) of the vision system facilitates more robust pose tracking when an MAV flies in complex environments. However, the proposed SLAM system did not scale well for large-scale environment operations, and in such cases, will hardly be able to build a consistent map when flying around with loops, i.e. re-visiting some places during an exploration. In this paper, we first modify this visual SLAM system to operate as a robust visual odometry with constant time cost in large-scale operations.

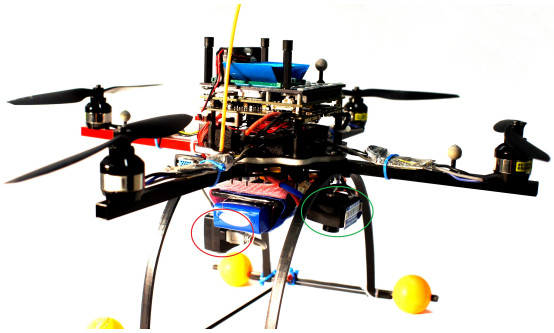


Fig. 1: Our MAV platform, with two cameras mounted looking in two different directions: downward (green ellipse) and forward (red ellipse).

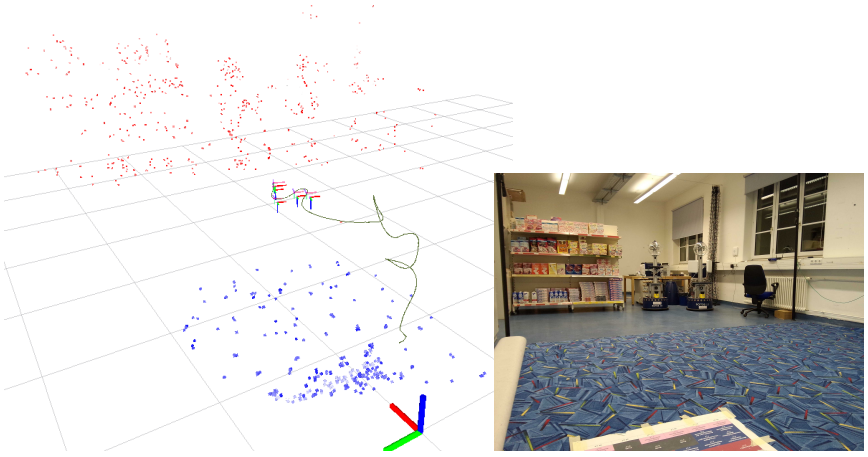
Our final implementation uses two cameras pointing forward and downward, respectively, for the visual odometry, as shown in Fig. 1. The choice of the number of cameras is resulted from a compromise between tracking robustness and onboard computation capability. Furthermore, we implement an efficient back-end for loop closure detection and correcting pose drift inherent in the visual odometry by using pose-graph optimization (PGO). It maintains a consistent global map organized as keyframes, each of which is associated with some map points represented using positions relative to it.

The work in [9] proposed a double-window graph structure for optimizing its global map. Bundle adjustment within a small inner window and pose-graph optimization within a large window are integrated within one optimization problem. In our work, we decouple bundle adjustment and PGO into a visual odometry front-end and a separate back-end, so that accurate pose tracking and local map can be achieved in constant time, without losing the benefit of PGO in building a consistent global map. The robot position and map in [5] are represented in a continuous relative representation (CRR) framework. It allows relative bundle adjustment for map refinement and real-time loop closure. In the global map of our SLAM system, we just keep map points in a relative representation. Thus, they can be implicitly updated by PGO operations.

2 Robust Multi-Camera Visual Odometry

The implementation of the visual SLAM system we proposed in [12] is based on the open source system Parallel Tracking and Mapping (PTAM) [3]. In order to achieve real-time operation, the main idea proposed in PTAM is to split tracking and mapping into two separate threads, which can be processed in parallel on a dual-core computer. The first *tracking* thread is responsible for real-time tracking of the camera motion relative to the current map. The second *mapping* thread extends the map which consists of 3D point features organized in keyframes, and refines the map using bundle adjustment.

Since global bundle adjustment in PTAM is computationally intensive, we retain only the local bundle adjustment in [12]. In this paper, we further reduce the complexity



(a) (b)

Fig. 2: (a) A local map built by our dual-camera visual odometry during an exploration, with $n_L = 4$. Map points from the forward-looking camera are marked in red color, and those from the downward-looking camera in blue. The trajectory of the forward-looking camera is plotted in green. (b) A scene of the actual lab environment where this experiment was performed, in a similar perspective.

of the SLAM system proposed in [12] by fixing the size of keyframes from each camera to be a constant number n_L, by removing the oldest keyframe when a new keyframe is added to the map. Bundle adjustment is performed within all $m \cdot n_L$ keyframes in a m-camera case. This changes the dual-camera SLAM system proposed in [12] to be an efficient constant-time visual odometry. An example map built by the visual odometry in our dual-camera setting is shown in Fig. 2a.

3 Back-End of the SLAM System

The visual odometry only maintains a local map for pose tracking. We further implement a back-end for the SLAM system to manage and refine a global map. The back-end mainly performs loop closure detection and pose graph optimization.

3.1 The global map representation

We use a keyframe-based global map representation proposed in [8]. It is very similar to the one used in the original PTAM. However, we store positions of map points relative to their associated keyframes within the global map, so that their global pose will be implicitly updated after the keyframe poses are updated in PGO described in Sec. 3.3. Feature descriptors, i.e. BRIEF descriptors [1], of map points and other feature corners of a keyframe are also computed and stored in the global map to facilitate loop closure detection.

3.2 Loop-closure detection

Loop closures provide additional pose constraints (edges) to the pose graph. It is of vital importance for correcting pose drift and building a consistent map during explorations in large-scale environments. We detect the following two types of loop closures.

Appearance-based large loop closure A hierarchical Bag-of-Word method developed in [2] is used for large loop closure detection. The word/feature vocabulary used in our system is trained off-line using a large number of images taken from different environments. After a large loop is detected between the current keyframe $\mathcal{K}_A$ and a previous keyframe $\mathcal{K}_B$, we match feature corners in $\mathcal{K}_A$ to map points measured in $\mathcal{K}_B$ to retrieve 2D-3D correspondences. If there are enough 2D-3D correspondences, we estimate the relative pose E_{AB} between these two keyframes. This can be done efficiently using a P3P plus RANSAC method and further refining the result by robust optimization which minimizes 2D reprojection errors of all inlier correspondences. Then the edge $\mathcal{E}_{AB}$ is added to the pose graph. We do not compute the otherwise relative pose E_{BA}, since we expect that the keyframe $\mathcal{K}_A$ would have significant pose drift relative to $\mathcal{K}_B$, and thus positions of map points measured in $\mathcal{K}_B$ should be trusted.

Local loop closure A potential local loop closure can be detected by trying to register the current keyframe $\mathcal{K}_A$ to its best neighbor $\mathcal{K}_B$ (in the sense of co-visibility of common features) within a certain geometric distance range, and estimating their relative pose in a way similar to the method we used after an appearance-based loop closure is detected. In this case, we compute both E_{AB} (from 2D-3D correspondences matching feature corners in $\mathcal{K}_A$ to map points measured in $\mathcal{K}_B$) and E_{BA} (from 2D-3D correspondences matching feature corners in $\mathcal{K}_B$ to map points measured in $\mathcal{K}_A$). We expect E_{AB} and E_{BA} to agree with each other if a true local loop closure has been detected, as proposed in [8].

3.3 Adaptive-window pose-graph optimization

We apply pose-graph optimization at each time when a new keyframe is added to the global map, and we adaptively define a portion of the whole global graph to be optimized depending on whether new edges are added from loop closures.

The graph structure For the purpose of PGO, the graph structure definition of our SLAM system is straightforward: It consists of a set of keyframe-pose vertices ($\mathcal{V}_i$) and relative edges ($\mathcal{E}_{ij}$) describing the relative pose-pose constraints (E_{ij}) among those vertices. Each vertex $\mathcal{V}_i$ stores an absolute pose E_i of its corresponding keyframe $\mathcal{K}_i$ in the world frame of the SLAM system. The edges consist of constraints obtained from the bundle adjustment in the visual odometry and the two types of loop closures.

We notice that each bundle adjustment operation in the visual odometry produces pose constraints among all n_L keyframes in the local map. When a new keyframe $\mathcal{K}_n$ is added to the local map of the visual odometry, the oldest keyframe $\mathcal{K}_r$ will be removed before the next bundle adjustment operation. As a result, poses of the remaining $n_L - 1$

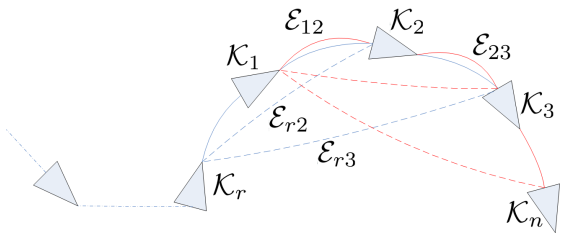


Fig. 3: Edges added to the pose graph after a new keyframe $\mathcal{K}_n$ is added, in the case of four keyframes involved in bundle adjustment. Edge $\mathcal{E}_{r2}$ and $\mathcal{E}_{r3}$ will also be added to the graph before $\mathcal{K}_r$ is removed from the local map. Edge $\mathcal{E}_{12}$ and $\mathcal{E}_{23}$ will be replaced by new constraints (in red) after a new bundle adjustment operation.

keyframes will be re-adjusted during the next bundle adjustment step without considering the pose constraints to $\mathcal{K}_r$. This means that pose constraints between $\mathcal{K}_r$ and the other $n_L - 1$ nodes may actually contain useful information and should be considered in PGO. Thus, we not only add pose constraints among consecutive keyframes to the pose graph, but also add those between $\mathcal{K}_r$ and all other keyframes within the current local map, as depicted in Fig. 3.

Defining the sub-graph for optimization In PGO, we always consider a window of the whole pose graph $\mathcal{G}$ to be adjusted. We perform uniform-cost search for choosing sub-graph vertices, beginning with the latest added vertex. The cost is measured by geometric distances among vertices. The sub-graph to be adjusted (denoted as $\mathcal{G}_s$) consists of all those sub-graph vertices and constrains among them. We define the size of the sub-graph vertices in the following adaptive way depending on whether there is a loop closure been detected: During regular exploration of the SLAM system without loop closure been detected, only a relatively small window (with no more than n_s vertices) of the whole pose graph is to be adjusted. When a loop closure between two keyframes is detected, an edge between their corresponding vertices will be added to the pose graph. Then we expand the graph window $\mathcal{G}_s$ to contain a large number of vertices until it has included all vertices in the detected loop or a maximal number (n_m) of vertices.

Pose-graph optimization Given a n-vertices sub-graph $\mathcal{G}_s$ we previously defined, the pose-graph optimization is to minimize the following cost function:

$$F(\mathbf{E}) = \sum_{\mathcal{E}_{\mathcal{G}_s}} \Delta E_{ij}^T \Omega_{ij} \Delta E_{ij}, \tag{1}$$

with respect to all vertex poses $\mathbf{E} = (E_1, E_2, ..., E_n)$, where $\mathcal{E}_{\mathcal{G}_s}$ contains all edges in $\mathcal{G}_s$, $\Delta E_{ij} := log(E_{ij} \cdot E_j^{-1} \cdot E_i)$ is the relative pose error in the tangent space of SE(3) and Ω_{ij} is the information matrix of the pose constraint E_{ij}. We estimate Ω_{ij} coarsely as a diagonal matrix in a way proposed in [9], with the rotation component to be a constant and the translation component to be proportional to the translation $\mathbf{t}_{ij}$ normalized by the average scene depth. Here, $\mathbf{t}_{ij}$ is the translation elements of E_{ij}.

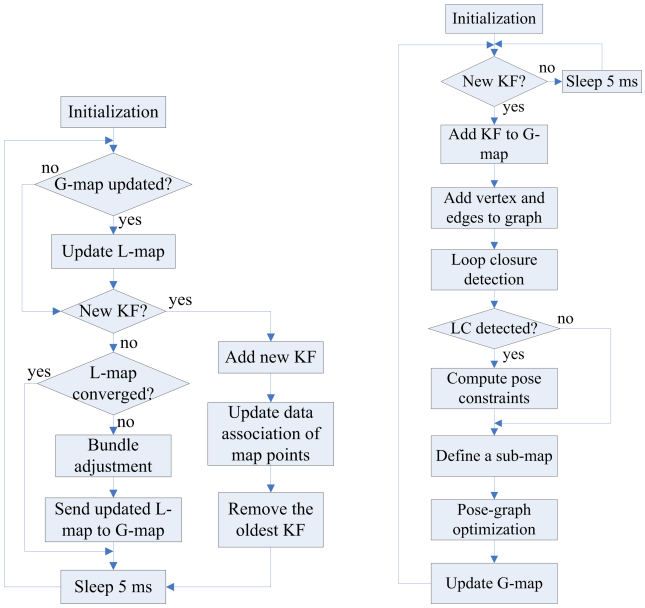


Fig. 4: (a) the mapping thread of the visual odometry, and (b) the back-end thread of the SLAM system. Abbreviations: KF (keyframes), L-map (local map), G-map (global map), and LC (loop closure).

4 Implementation

Our visual SLAM system mainly consists of three threads: two threads for the visual odometry, and a third thread working for the back-end. The mapping thread of the visual odometry manages a local map and handles most of the interactions with the back-end. The global map and the pose graph is managed by the back-end thread. A general view of operations of the mapping thread and the back-end is illustrated in Fig. 4. In this section, we mainly present more details about the modifications made to the visual odometry for interactions with the back-end.

4.1 The visual odometry

Automatic initialization The visual odometry is initialized in the same way as we did in [12], with a circular pattern with known size. The downward camera C_1 is responsible for the initialization, which is done during the takeoff phase of the MAV.

Motion-model update of the tracker In the tracking thread, we assume a slowly descending velocity model of the cameras. In each tracking process, we estimate a

prior pose E_p of the camera C_1 as the initial guess in the pose optimization, based on the camera velocity v_c and the camera pose E_l in the last tracking process. Once the local map is updated by the mapping thread according to the updated global map, we also need to update the prior estimation E_p to avoid tracking failure. We simply retain the velocity v_c unchanged. However, we update E_p by assuming a constant relative pose E_{ip} to its neighboring keyframe $\mathcal{K}_i$. Thus, we have the new prior pose $E'_p = E_p \cdot E_i^{-1} \cdot E'_i$, where E_i and E'_i are respectively the poses of K_i before and after the local map update.

Adding local map to the back-end We add the local sub-map (in the form of keyframes) built from images of one specific camera to the back end to form the global map. In this paper, we set this camera to be the forward-looking camera C_2. There are three reasons for this decision: First, since the dual cameras are rigidly connected, pose constraints of one camera are sufficient for pose-graph optimization. Second, we mainly use our MAV in low-altitude applications. Thus, we expect more interesting views taken from the forward-looking camera, which will be used for loop closure detection. Third, this is again a compromise for real-time performance: Keyframes from the downward-looking camera might later be included in the global map if the onboard computation capability improves.

Removing old keyframes of the local map If the number of keyframes from camera C_i exceeds n_L after a new keyframe is added to the local map, we remove the oldest keyframe $\mathcal{K}_0$ from C_i. Before that, we first update the data association of the map points measured by $\mathcal{K}_0$. Each keyframe $\mathcal{K}_i$ is associated with some map points p_j. We call $\mathcal{K}_i$ the source keyframe of p_j since it was used to triangulate p_j together with an earlier keyframe. When $\mathcal{K}_i$ should be removed, we only remove those map points which are not measured by any other keyframe in the local map. Other map points could be important for later pose tracking. If p_j is measured by another keyframe $\mathcal{K}_m$, we transfer its source keyframe identity to $\mathcal{K}_m$, and update its related data with the measurement in $\mathcal{K}_m$, which will be used in pose tracking. Meanwhile, we mark p_j indicating that it has been sent to the global map already, in order to avoid re-sending it with $\mathcal{K}_m$ in the future.

Updating the local map After each pose-graph optimization process, we update the local map according to the global map, including updating keyframe poses and their measured map point poses. Here, we distinguish keyframe $\mathcal{K}_{1j}, j \in \{1, 2, ..., n_L\}$ from the downward-looking camera C_1 and keyframe $\mathcal{K}_{2j}, j \in \{1, 2, ..., n_L\}$ from the forward-looking camera C_2. We assume that the pose of $\mathcal{K}_{2j}$ is identical to the pose of its corresponding keyframe in the global map, while the $\mathcal{K}_{1j}$ pose is updated by assuming a calibrated rigid transform to $\mathcal{K}_{2j}$. Since the map points in the local map are stored with absolute coordinates, their positions need to be updated individually. We do this by assuming unchanged relative translations to their current source keyframe.

4.2 The back-end

The back-end thread runs in an endless loop as illustrated in Fig. 4b. It is activated whenever a new keyframe is added to the waiting queue of the global map by the map-

Table 1: MAV pose RMSEs in the autonomous flight (Auto) and the manual flight (Manual) experiments using the proposed visual SLAM system, and using only the visual odometry (VO), with position errors in *millimeters* and attitude errors in *degrees*.

RMSEs	x	y	z	3D	roll	pitch	yaw
Auto	103.0	117.0	82.7	176.5	1.70	1.45	0.80
Manual	50.6	89.9	114.9	150.4	2.06	1.34	1.92
VO	69.2	168.2	106.7	206.9	2.37	1.35	2.28

ping thread. After the PGO is done, the global map is updated according to the pose graph: The keyframe poses are updated by directly copying the corresponding vertex poses of the graph, leaving the associated map points only implicitly updated. The implementation of our PGO is based on the open source library g^2o described in [4].

5 Experiments

In this section, we evaluate our visual SLAM system in an indoor environment (our robotics laboratory), as shown in Fig. 2b. An external tracking system available here provides accurate measures of the pose tracking errors of our onboard SLAM system.

5.1 Experimental setup

Quadrotor MAV platform Our MAV is based on the open source and open hardware quadrotor platform described in [6], as shown in Fig. 1. The onboard computer features an Intel Core 2 Duo 1.86GHz CPU, 2 GB DDR3 RAM and a 32GB SSD. The pxIMU inertial measurement unit and autopilot board mainly consists of a microcontroller unit (MCU) for position and attitude control, and sensors including a tri-axis accelerometer and a tri-axis gyroscope. The two synchronized cameras utilized on our MAV are two PointGrey Firefly MV monochrome cameras, each of which weighs only 37 *grams*. Each camera has an image resolution of 640×480 pixels, a maximum frame rate of 60 *fps*, and both lenses we use have viewing angles of approximately 90 *degrees*. The extrinsic parameters of the two cameras are calibrated off-line as we described in [12].

Quadrotor controller We use a nested PID pose controller and PD trajectory controller described in previous work [11] for autonomous navigation of our quadrotor. Those controllers are implemented based on the work in [7]. The 3D position estimates and the yaw angle estimates of the visual SLAM system are fed to the position controller at frame rate. The attitude controller runs at a frequency of 200 Hz, using the roll and pitch estimates provided by the IMU.

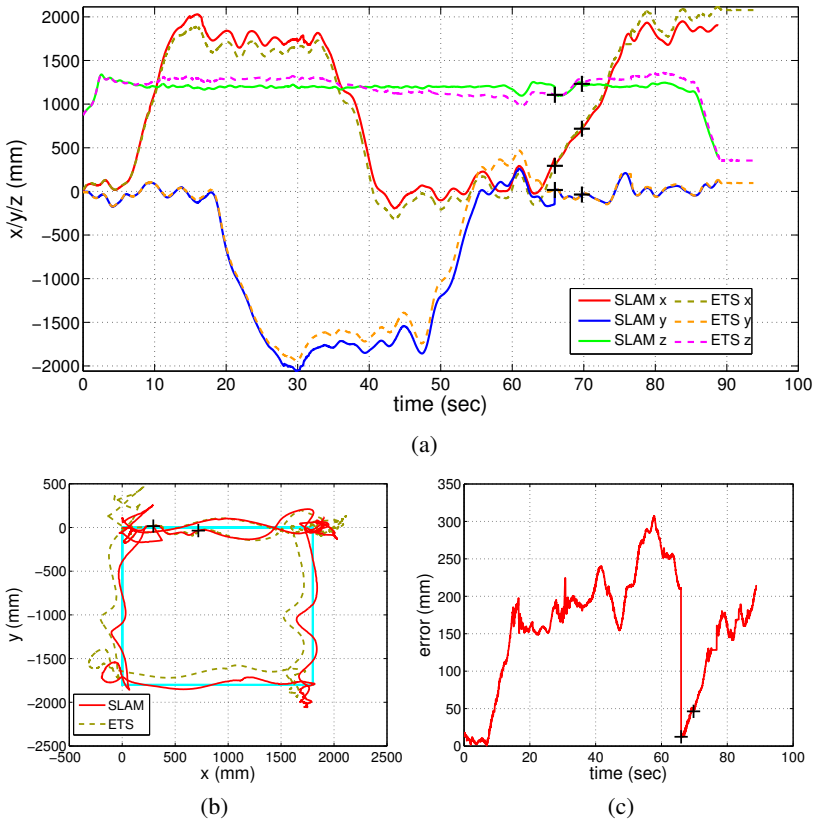
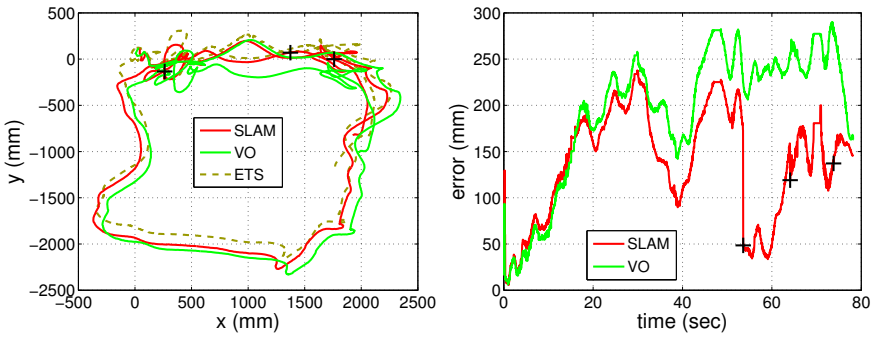


Fig. 5: The pose tracking results during the autonomous flight, compared with ground truth data. Pose corrections after loop closures been detected and PGO been processed, are marked with black crosses.

5.2 Enabling autonomous navigation

In this experiment, we demonstrate the efficiency of our SLAM system to enable autonomous navigation of our MAV. The MAV autonomously navigates along a predefined rectangular path with a height of 1.2 *m* (plotted in cyan in Fig. 5b) in a clockwise direction, taking off above the origin of the world frame and landing on the top-right corner. It turns 90 *degrees* at each corner in order to head to the forward direction.

Fig. 5 shows the pose tracking results of the SLAM system (SLAM) compared with the ground truth data provided by the external tracking system (ETS). The visual odometry slowly drifts during explorations. However, the local-map update process will correct the drift after a loop closure is detected and the pose-graph optimization is performed. We can clearly recognize the effect of such corrections in Fig. 5: The black crosses are marked to the pose tracking results after local-map updates. The first loop-



(a) (b)

Fig. 6: MAV pose-estimation results using the proposed SLAM system and visual odometry during the manual flight compared with ground truth data: (a) MAV position on $x_W - y_W$ plane, (b) the translation errors. Pose corrections after loop closures been detected and PGO been processed, are marked with black crosses.

closure provides an obvious correction to the pose tracking of the visual odometry. Fig. 5c shows the 3D translation errors of the pose tracking results compared with the ground truth data during the flight. Those very short horizontal lines in Fig. 5c are resulted from missing ground truth data during those periods of time, when the MAV is not flying in the effective field of view of the external tracking system. The RMSEs of the 6DOF pose estimates of the SLAM system to the ground truth data during the whole flight are listed in Tab. 1 (in the row of Auto).

Pose estimates of a visual odometry are subjected to drift during explorations. In our case, two metric scale related issues which contribute to the pose drift should be noted: First, although dual cameras are used in our visual odometry, they have no overlap in their respective fields of view. Thus, the visual odometry cannot make stereo triangulation to track the metric scale of the environment, which results in scale drift in the pose estimation. Second, the accuracy of our automatic initialization module mentioned in Sec. 4.1 could be affected by the vibration of the MAV. The initialization errors in the metric scale and the attitude will be accumulated in the whole flight trajectory of the MAV. Another factor which could affect the pose tracking accuracy of the visual odometry is the errors in extrinsic calibration of the dual cameras.

5.3 Further evaluations with manual flight data

We manually control the quadrotor to fly in a similar way as we have done in Sec. 5.2, and record all necessary onboard data in a ROS-bag file. We process this logfile in post-processing on the onboard computer, to gain more insights into the performances of both the visual odometry and the back-end of the SLAM system. The results of this experiment are shown in Fig. 6 and Fig. 7.

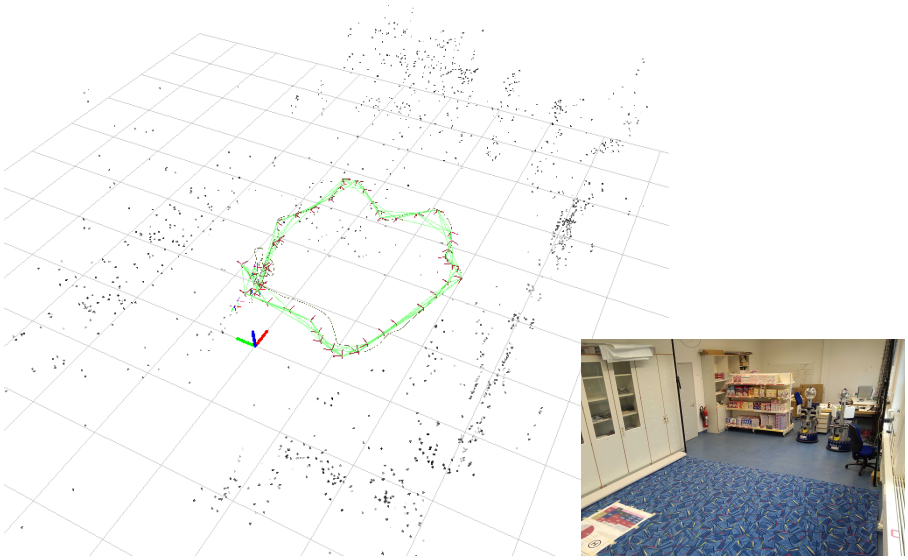


Fig. 7: A view of the built map of the visual SLAM system during a manual flight in the lab. The grid cell size is $1m \times 1m$. The global map points are shown in grey-scale. The vertices of the pose graph are illustrated with red tri-axes, and the edges are plotted in green.

Fig. 6a provides a top view of the MAV trajectory on $x_W - y_W$ plane. Here, we have processed the ROS-bag file twice using our SLAM system: firstly, using the full SLAM system (SLAM), and secondly, using only the visual odometry (VO) without the back-end. The results of the two processes are compared with pose estimates from the external tracking system (ETS). Fig. 6b shows the position estimation errors of these two processes. Without the back-end, the visual odometry would result in larger pose drift. Loop-closure detection of the back-end can obviously benefit the pose tracking with drift corrections. RMSEs of the pose tracking results of the two processes are listed in Tab. 1 (in the row of Manual and the row of VO, respectively).

Fig. 7 illustrates a view of the resulting global map of the SLAM system, with references of the real-world pictures. In this figure, we can find the pose graph with nodes (keyframe poses) and edges (in green), and map points at their absolute positions which are only computed for visualization purpose.

6 Conclusions and Discussions

Our proposed visual SLAM system utilizes dual cameras to achieve a constant-time visual odometry, which can provide robust pose tracking for autonomous navigation of our MAV. The back-end of the SLAM system performs loop-closure detection and adaptive-window pose-graph optimization to correct pose drift of the visual odometry

and to maintain a consistent global map. Autonomous navigation of an MAV following a predefined path has been achieved using the proposed visual SLAM system.

In general, doing PGO in the Similarity space Sim3, instead of SE3, can provide better scale corrections to the SLAM system [10], which could be a near future work. A map merging strategy would also be considered to achieve a more consistent map after loop closures. Then the performance of the resulting visual SLAM system in large-scale outdoor environments would be evaluated in the future. Currently, the keyframes and map points in the global map are not used in the visual odometry. It would be worthwhile to investigate how to more efficiently merge the back-end with the visual odometry, so that the updated global map can be directly used by the visual odometry. Additional sensor which can provide metric scale measurements could be added to the system to tackle the scale drift issue before a loop is detected, and to initialize the metric scale of the SLAM system.

References

1. Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. Brief: Binary robust independent elementary features. *Computer Vision–ECCV 2010*, pages 778–792, 2010.
2. D. Gálvez-López and J.D. Tardós. Bags of binary words for fast place recognition in image sequences. *Robotics, IEEE Transactions on*, 28(5):1188–1197, 2012.
3. Georg Klein and David 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.
4. R. Kummerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. G2o: A general framework for graph optimization. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 3607–3613, 2011.
5. Christopher Mei, Gabe Sibley, Mark Cummins, Paul Newman, and Ian Reid. Rslam: A system for large-scale mapping in constant-time using stereo. *International journal of computer vision*, 94(2):198–214, 2011.
6. L. Meier, P. Tanskanen, L. Heng, G. Lee, F. Fraundorfer, and M. Pollefeys. PIXHAWK: A Micro Aerial Vehicle Design for Autonomous Flight Using Onboard Computer Vision. *Autonomous Robots*, pages 1–19, 2012.
7. Daniel Mellinger, Nathan Michael, and Vijay Kumar. Trajectory generation and control for precise aggressive maneuvers with quadrotors. *The International Journal of Robotics Research*, 31(5):664–674, 2012.
8. Sebastian A. Scherer, Shaowu Yang, and Andreas Zell. DCTAM: Drift-corrected tracking and mapping for autonomous micro aerial vehicles. In *Intelligent Robots and Systems (IROS), 2014 IEEE/RSJ International Conference on*, 2014. Submitted.
9. H. Strasdat, A.J. Davison, J. M M Montiel, and K. Konolige. Double window optimisation for constant time visual slam. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2352–2359, 2011.
10. Hauke Strasdat, JMM Montiel, and Andrew J Davison. Scale drift-aware large scale monocular slam. In *Robotics: Science and Systems*, volume 1, 2010.
11. Shaowu Yang, Sebastian A. Scherer, Konstantin Schauwecker, and Andreas Zell. Autonomous landing of mavs on an arbitrarily textured landing site using onboard monocular vision. *Journal of Intelligent & Robotic Systems*, 74(1-2):27–43, 2014.
12. Shaowu Yang, Sebastian A. Scherer, and Andreas Zell. Visual SLAM for Autonomous MAVs with Dual Cameras. In *2014 International Conference on Robotics and Automation (ICRA'14)*, Hongkong, China, May 2014. Accepted.