DCTAM: Drift-Corrected Tracking and Mapping for Autonomous Micro Aerial Vehicles

Sebastian A. Scherer[1], Shaowu Yang[2] and Andreas Zell[1]

***Abstract*— Visual odometry, especially using a forward-looking camera only, can be challenging: It is doomed to fail from time to time and will inevitably drift in the long run. We accept this fact and present methods to cope with and correct the effects for an autonomous MAV using an RGBD camera as its main sensor. We propose correcting drift and failure in visual odometry by combining its pose estimates with information about efficiently detected ground planes in the short term and running a full SLAM back-end incorporating loop closures and ground plane measurements in pose graph optimization. We show that the system presented here achieves accurate results on several instances of the TUM RGB-D benchmark dataset while being computationally efficient enough to enable autonomous flight of an MAV.**

I. INTRODUCTION

A. Problem Definition

Vision-based aerial robots are an increasingly popular research topic. Its focus is on the goal of enabling autonomous flight of robots using cameras as their main sensors, preferably in previously unknown environments. A very crucial problem is pose estimation, since accurate pose estimates are required for even most basic tasks, e.g. controlling a MAV to hold its position or to follow a desired trajectory. This is typically addressed by employing visual odometry on vision-based aerial robots. Robots flying in unknown environments are also required to map their surroundings. In order to create consistent maps, it is thus also required to solve the SLAM (simultaneous localization and mapping) problem.

B. Related Work

There is a plethora of work aimed at enabling MAVs with RGBD or stereo cameras to fly autonomously already, but we need to limit ourselves to the work most relevant to this research. Bachrach et al. in [1] describe an autonomous quadrotor using an RGBD camera as its main sensor, which computes visual odometry on its on-board computer and relies on a separate external computer for SLAM and planning paths, which are then sent back to the MAV.

The MAV described by Schmid et al. in [2] relies on stereo vision, with dense stereo matching performed in hardware on an FPGA, and computes visual odometry and even 3D reconstructions for semi-autonomous navigation with obstacle avoidance on-board. It does not perform full SLAM, which is not a problem as long as it does not revisit places after long flights.

We previously implemented and demonstrated an efficient RGBD-SLAM system running on our autonomous MAV in [3], which to our knowledge was the first time for an MAV performing full RGBD-SLAM, which includes closing detected loops and running pose graph optimization on-board while flying.

C. General Approach

Our MAV is a quadrotor helicopter equipped with a forward-looking Asus Xtion Pro Live RGBD camera as its main sensor and a single-board computer with an Intel Core 2 Duo CPU hosted by a pxCOMex base board [4], which performs all processing on-board. The main sensor is used for visual odometry and SLAM.

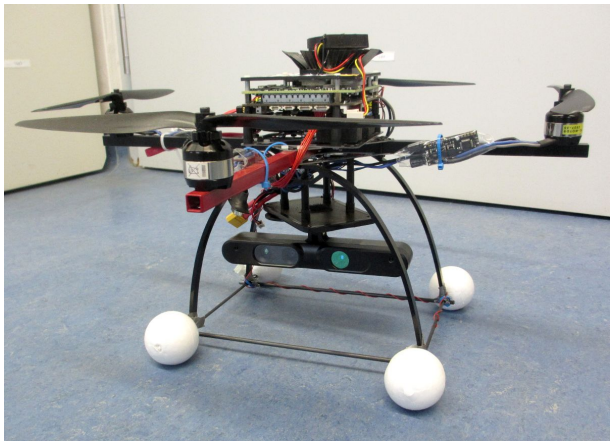


Fig. 1. The quadrotor helicopter used in this work with its RGBD camera and onboard computer.

In this work, we combine a modified version of the widely-used visual odometry system PTAM[1] (parallel tracking and mapping, [5]) with a SLAM back-end to handle tracking failures and closed loops to produce consistent maps in real time on the on-board computer. We also embrace the fact that visual odometry, especially using a forward-looking camera only, will sometimes fail in the short term and drift in the long run. We employ ground plane detection to make the MAV robust to both tracking failure and long-term drift.

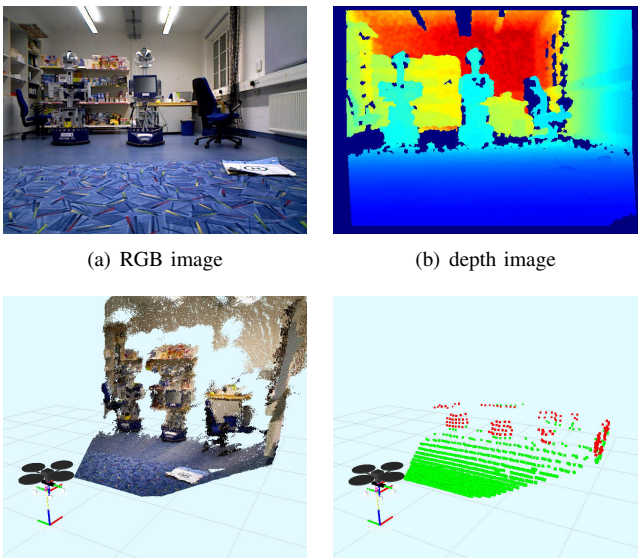
[1]Sebastian A. Scherer and Andreas Zell are with the Department of Computer Science, Faculty of Science, University of Tuebingen, Tuebingen, Germany. {sebastian.scherer, andreas.zell}@uni-tuebingen.de

[2]Shaowu Yang is with the State Key Laboratory of High Performance Computing (HPCL), College of Computer, National University of Defense Technology, Changsha, China. shaowu.yang@nudt.edu.cn

[1]PTAM can be considered both a visual odometry and visual SLAM system. We do not consider it a *full* visual SLAM system here, since it cannot explicitly find and close loops. This is not a serious problem in some applications, especially when used in conjunction with a downward-looking camera.

D. Contribution of this Paper

The contributions of this paper are as follows: We present an efficient, accurate, and robust ground plane detection algorithm based on RANSAC with a novel inlier/outlier/worse outlier model described in Sect. III. We propose a fusion method combining information about detected ground planes with visual odometry, which allows to correct its drift in attitude and altitude in Sect. IV. We use a heavily modified version of PTAM as a visual odometry system, which is described in Sect. II. We then go on to extend PTAM with a SLAM back-end to turn it into a full SLAM system with loop closing, pose graph optimization and incorporating ground plane measurements, which is described in Sect. V. We prove the accuracy of the proposed system by evaluating it on a public RGBD benchmark dataset in Sect. VII and finally demonstrate that our system enables a MAV to fly completely autonomously using its on-board RGBD camera while performing the full SLAM task on its on-board computer in Sect. VIII. We publish our full software stack under an open-source license, since we believe this stack can be useful not only to other MAV groups: The SLAM back-end to PTAM can also be used without ground-plane detection for bulding consistent maps which is not possible with PTAM alone.



(a) RGB image (b) depth image

(c) full point cloud (d) ground plane visualization

Fig. 2. A typical RGBD scene encountered while flying indoors

II. PTAM AS VISUAL ODOMETRY

We use our heavily modified fork of PTAM (Parallel Tracking and Mapping) [5] as a visual odometry system. Our previous modifications include porting PTAM to ROS and using depth measurements (if available) in addition to 2D measurements for map points initialization, tracking, and bundle adjustment as described in [6]. We remove old keyframes and corresponding map points and only keep the most recent keyframes for tracking and local bundle adjustment [7]. When tracking fails for a few frames, we blindly predict the pose based on the last known velocity before giving up on the current map and reinitializing it.

For this work, we could further improve tracking speed by making sure that images are never copied on their way from the camera driver to PTAM and limiting the number of map points per keyframe by an upper boundary.

Limiting the number of map points per keyframe is necessary since their number and thus the time required for tracking may vary uncontrollably for different environments. We fix this problem by disregarding all but the n best map points per image pyramid level, ordered according to their Shi-Tomasi score [8].

As mentioned above, we remove keyframes from PTAM in order to limit its computational complexity. Just deleting these keyframes forever, however, wastes useful information. Instead, we now publish all keyframes with all their information before they are deleted so they can be picked up by a SLAM back-end (see. Sect. V).

III. GROUND PLANE DETECTION

During a typical indoor flight, the ground plane can be clearly identified in most RGBD point clouds and thus also depth images seen by the on-board camera, which is demonstrated in Fig. 2. Our ground plane detection method consists of the following steps: Sampling a number of sparse 3D points, detecting inliers and outliers using a custom RANSAC scheme, robust refinement of the most promising hypothesis based on inliers.

A. Sampling 3D Points

We sample the depth image at few sparse locations to significantly reduce the number of depth measurements that need to be considered. We obtain good results with considering one depth value out of a 10×10 grid. For a typical flight with a forward-looking camera, we can usually disregard the upper half of the image, since the ground plane can be expected in the lower half of the image. Only if there are very few valid depth measurements in the lower half, which might be the case because the MAV is flying very close to the ground, we fall back to considering the full depth image.

Using the intrinsic camera calibration, depth pixels are then reprojected to 3D points.

B. Inlier/Outlier Detection

Out of all 3D points determined by sampling, only a minor fraction might actually lie on the ground plane. We employ a preemptive RANSAC scheme [9] with a custom inlier/outlier/worse outlier model to quickly arrive at a well-supported hypothesis.

We also consider the attitude estimate provided by the on-board inertial measurement unit (IMU) to immediately disregard hypotheses that are far off from its not always accurate but rarely completely-off attitude estimate.

Inlier/Outlier/Worse Outlier Model: RANSAC (RANdom SAmple Consensus) as originally described in [10] tries to find a hypothesis which maximizes the size of its consensus set (i.e. the number of inliers). This is also called using an inlier/outlier model in [9], since it only considers the fact whether a measurement is to be considered an inlier or outlier.

Early on in our experiments, however, we found that RANSAC with using the inlier/outlier model often fails in ground plane detection when facing a vertical wall or in front of a wide obstacle. A simplified 2D sketch of such a failure case is shown in Fig. 3. Both hypotheses lead to exactly the same numbers of inliers (7) and outliers (2), so it is impossible to distinguish the better solution.

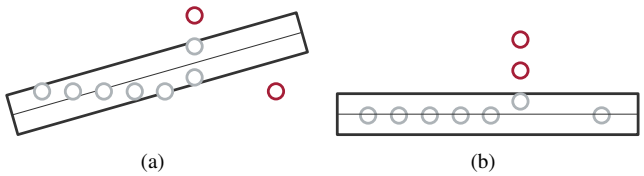


Fig. 3. (a) A typical failure case of the basic inlier/outlier model for RANSAC in ground plane detection and (b) the desired estimate obtained using our proposed inlier/outlier/worse outlier model.

When trying to find ground planes, however, there are two very different kinds of outliers which are very easy to distinguish: Points above the ground plane are ubiquitous, since we see walls and obstacles all the time. Points below the ground plane, however, should be very rare: During indoor flight on a single floor of a building, we expect to see none except for some completely random sensor failures.

We thus introduce an inlier/outlier/worse outlier model, which adds the following values to the score of a hypothesis: $+1$ if the observation is an inlier, $+0$ if the observation is a "good" outlier above the ground plane, and $-c_{bad}$ with $c_{bad} > 1$ if it is a worse outlier below the ground plane. We choose $c_{bad} = 10$, which worked well in all of our experiments. In the toy example above, hypothesis (a) would score $s_a = -3$ and (b) $s_b = 7$. This means our proposed model correctly solves this problem.

C. Robust Refinement

RANSAC will return a hypothesis which is best with regards to its scoring function, which in turn only depends on the number of inliers and (different kinds of) outliers, but not necessarily the most accurate one. For accuracy, it is required to further refine this hypothesis based on its inliers. Since there will always be some points classified as inliers by RANSAC that do not belong to the ground plane but to the lower parts of obstacles, it is advisable to use a robust or robustified method for this refinement instead of basic least squares.

We robustify [11] the PCA method for plane fitting [12] to refine the rough ground plane estimate obtained using RANSAC based on its inliers.

Given N points $p_i = (x_i, y_i, z_i)^T$, the PCA method assumes the plane to pass through their arithmetic mean:

$$\mu = \frac{\sum_{i=1}^{N} w_i \cdot p_i}{\sum_{i=1}^{N} w_i} \tag{1}$$

And the normal of the plane should correspond to the smallest Eigenvalue of the covariance matrix:

$$A = \frac{\sum_{i=1}^{N} w_i^2 \cdot (p_i - \mu) \cdot (p_i - \mu)^T}{\left(\sum_{i=1}^{N} w_i^2\right) - 1} \tag{2}$$

Where w_i are weights for each point. The basic PCA method for plane fitting treats all points equally, i.e.

$$w_i = 1 \quad \forall i \tag{3}$$

In this case, equations 1 and 2 actually compute the basic sample mean and sample covariance matrix.

A robust estimator of mean and covariance as described in [11], however, computes the weights w_i based on the Mahalanobis distance of point p_i from the current hypothesis:

$$w_i = \frac{\omega(d_i)}{d_i} \tag{4}$$

Where d_i is the Mahalanobis distance between p_i and the current estimate of the mean and ω is an influence function:

$$\omega(d) = \begin{cases} d & \text{if } d < d_0 \\ d_0 \exp\left(-\frac{(d-d_0)^2}{b_2^2}\right) & \text{else} \end{cases} \tag{5}$$

Once the ground plane is found and refined, we can represent it using its normal vector pointing up in the current camera frame ${}^{C}\boldsymbol{up}$ and the camera height h with respect to the plane.

IV. DRIFT CORRECTION

Given visual odometry pose estimates that drift over time and occasional ground plane detections, we want to combine both of them to a drift-corrected pose estimate such that

- corrected pose estimates are consistent with the latest ground plane estimate, and
- correcting visual odometry pose estimates introduces as few disturbances as possible, i.e. the correction transformation applied to to visual odometry pose estimates should be minimal in translation and rotation.

Using visual odometry alone, we can only estimate the camera pose with respect to a fixed reference frame by incrementally combining all sequential relative pose estimates in a chain:

$${}^{W}\boldsymbol{T}_{C_i} = {}^{W}\boldsymbol{T}_{C_0} \cdot {}^{C_0}\boldsymbol{T}_{C_1} \cdots {}^{C_{i-1}}\boldsymbol{T}_{i} \tag{6}$$

Ground plane measurements, on the other hand, put constraints on parts of the absolute pose ${}^{W}\boldsymbol{T}_{C_i}$ and are shortcuts in the chain above.

Our proposed method of combining both for drift-corrected pose estimates ${}^{W}\hat{\boldsymbol{T}}_{C_i}$ consists of two steps:

- Prediction using the relative pose estimate inferred from visual odometry:

$$ {}^{W}\bar{\boldsymbol{T}}_{C_i} = {}^{W}\hat{\boldsymbol{T}}_{C_{i-1}} \cdot {}^{C_{i-1}}\boldsymbol{T}_{C_i} \tag{7} $$

- Correcting the predicted pose estimate by applying a minimal correction transform such that the resulting pose estimate is more consistent with the measured ground plane:

$$ {}^{W}\hat{\boldsymbol{T}}_{C_i} = {}^{W}\bar{\boldsymbol{T}}_{C_i} \cdot \boldsymbol{T}_{corr} \tag{8} $$

Correction Transform Computation: The drift-correction transform $\boldsymbol{T}_{corr}$ consists of rotation component $\boldsymbol{R}_{corr}$ and translation $\boldsymbol{t}_{corr}$. The rotation part should bring the measured up vector in camera coordinates ${}^{C_i}\boldsymbol{up}$ parallel to the up direction expected based on the current camera pose:

$$ {}^{W}\bar{\boldsymbol{R}}_{C_i} \cdot \boldsymbol{R}_{corr} \, {}^{C_i}\boldsymbol{up} \stackrel{!}{=} {}^{W}\boldsymbol{up} \tag{9} $$

$$ \boldsymbol{R}_{corr} \cdot {}^{C_i}\boldsymbol{up} \stackrel{!}{=} {}^{W}\bar{\boldsymbol{R}}_{C_i}^{-1} \cdot {}^{W}\boldsymbol{up} \tag{10} $$

There is a closed-form solution to compute the shortest rotation $\boldsymbol{R}_{a,b}$ given two unit vectors $\boldsymbol{a}, \boldsymbol{b}$ such that $\boldsymbol{R}_{a,b} \cdot \boldsymbol{b} = \boldsymbol{a}$, which can be derived from the fact that $\boldsymbol{R}_{a,b}$ should rotate around an axis orthogonal to $\boldsymbol{a}$ and $\boldsymbol{b}$ by the angle between both vectors.

$$ \boldsymbol{R}_{a,b} = \boldsymbol{I}_3 + \boldsymbol{S} + \frac{1}{1+d} \cdot \boldsymbol{S} \cdot \boldsymbol{S} \tag{11} $$

Where $\boldsymbol{S} = [\boldsymbol{a} \times \boldsymbol{b}]_\times$ is the cross-product matrix of the cross product and $d = \langle \boldsymbol{a}, \boldsymbol{b} \rangle$ is the dot product of $\boldsymbol{a}$ and $\boldsymbol{b}$. To compute $\boldsymbol{R}_{corr}$ above, we choose $\boldsymbol{a} = {}^{C_i}\boldsymbol{up}$ and $\boldsymbol{b} = {}^{W}\bar{\boldsymbol{R}}_{C_i}^{-1} \cdot {}^{W}\boldsymbol{up}$

The translation component on the other hand, should be a small translation along the plane normal $\boldsymbol{t}_{corr} = s \cdot {}^{C_i}\boldsymbol{up}$ that brings the corrected camera pose to the measured height h above the ground plane:

$$ h \stackrel{!}{=} \left({}^{W}\hat{\boldsymbol{T}}_{C_i} \right)_{3,4} = \left({}^{W}\bar{\boldsymbol{T}}_{C_i} \cdot \boldsymbol{T}_{corr} \right)_{3,4} \tag{12} $$

$$ \stackrel{!}{=} \left({}^{W}\bar{\boldsymbol{R}}_{C_i} \cdot \boldsymbol{t}_{corr} + {}^{W}\boldsymbol{t}_C \right)_z \tag{13} $$

$$ \stackrel{!}{=} s \left({}^{W}\bar{\boldsymbol{R}}_{C_i} \cdot {}^{C_i}\boldsymbol{up} + {}^{W}\boldsymbol{t}_C \right)_z \tag{14} $$

Which can easily be solved for s.

Correction Filtering: Even though ground plane estimates are immune to long term drift, they still suffer from high-frequency measurement errors. Applying the full correction would introduce these errors into the corrected pose estimates. We instead limit the influence of a single attitude measurement by scaling it with a gain factor $\alpha < 1$:

$$ \boldsymbol{T}_{corr} = \exp\left(\alpha \cdot \log\left[\begin{pmatrix} \boldsymbol{R}_{corr} & \boldsymbol{t}_{corr} \\ 0 & 1 \end{pmatrix} \right] \right) \tag{15} $$

V. SLAM BACKEND

The SLAM back-end is in charge of building maps given keyframes that were deleted by the odometry front-end (i.e. PTAM), including detecting and closing loops to counter long-term drift and to keep the map consistent. Fig. 4 shows the general principle of our SLAM back-end working in parallel with and extending PTAM: PTAM's tracking thread is operating at camera rate, computing a pose estimate for each incoming image. Its mapping thread becomes active whenever there is a new keyframe and will optimize the relatively small local map of the few latest keyframes using bundle adjustment.

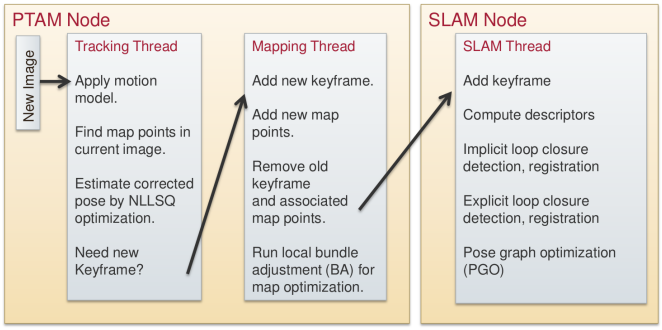


Fig. 4. Functional overview of PTAM front-end and SLAM back-end

The SLAM back-end waits for keyframes to be deleted from PTAM and adds these to its own map. Whereas PTAM and the camera driver are run as nodelets (modules with separate threads within the same process) to minimize overhead when transferring images, the SLAM back-end runs in its own process at a lower priority, so it does not slow down tracking. For each incoming keyframe, it will try to perform the following steps:

A. Keyframe Conversion

The back-end uses a keyframe representation similar to PTAM: A keyframe consists of a scale-space image pyramid with FAST corners computed on every level. What is changed is that map points are now stored within their source keyframe and that we rely on local descriptors instead of small image patches for matching.

PTAM uses a global representation for map points: All map point positions are stored in a global reference frame. This is impractical for pose graph optimization, so we instead store map points relative to their source keyframe, i.e. in which they were observed for the first time. This means that map point positions will be adjusted implicitly during pose graph optimization.

Finding map points within an image in PTAM relies on comparing warped templates and minimizing their zero-mean sum of squared differences (ZMSSD). This is feasible for tracking as in PTAM, where the image region in which a map point is searched for is small, but not for wide-baseline matching as required when closing loops. We thus compute BRIEF descriptors [13] for all corners and map points. We choose BRIEF because it is fast and we rely on neither scale invariance (which is achieved using a scale-space pyramid) nor rotation invariance (since we are using a forward-looking camera, flying close to hovering at all times) in a descriptor.

B. Retry Registration for Tracking Failures

Tracking in PTAM can sometimes fail for various reasons, resulting in inaccurate motion model edges between two keyframes when it had to reset its map. If this is the case, we try to register these keyframes again using the descriptor-based matching and registration described in Sect.V-E.

C. Implicit Loop Closure Detection

A trivial method of finding promising loop closure candidates is considering old keyframes whose pose estimates are close to the current keyframe. We ignore keyframes that are either too close in time to the current keyframe (since we expect their drift to be negligible) or geometrically too far way (since matching is unlikely to succeed). Among the remaining keyframes, we find the one with the most map points visible within the image of the current keyframe by projecting map points according to current pose estimates. If we can successfully register both keyframes (see Sect. V-E), we add a loop closure edge to the pose graph.

D. Explicit Loop Closure Detection

For longer loops, drift might be too big to find loops based on geometrical closeness of keyframe pose estimates. This is why we also consider loop closure candidates based on appearance, i.e. similarity of keyframes instead.

We use a hierarchical bag-of-words approach [14] to retrieve the previous keyframe which is most similar to the current one. Again, if we succeed in registering both keyframes, we add a loop closure edge.

E. Keyframe-to-Keyframe Registration

Attempts to register a pair of keyframes (A, B) start by matching map points of keyframe B to corners of A. Using these matches, we can compute a pose estimate by solving the PnP problem. We do this by applying Gao's solution to the P3P problem within a preemptive RANSAC scheme to identify inliers and arrive at a rough estimate for the relative pose ${}^{A}T_{B}$. In a second step, we estimate the inverse relative pose ${}^{B}T_{A}$ by matching map points of A to corners of B. If there were enough inliers in both cases and both relative poses agree, we refine the relative pose using nonlinear optimization, minimizing the 2D reprojection error of all map point/corner pairs that were inliers, and insert an edge between both keyframes into the pose graph.

We also considered matching all corners to all corners and running full bundle adjustment on all matches. This proved to be much slower than the method described above, though, since it involves matching many more feature descriptors (there are typically more than ten times as many corners as map points), applying a slower five point algorithm within a RANSAC scheme, and finally optimizing both relative keyframe pose and map point positions. We do not want to add new map points or change their relative positions once they are transferred to the back-end, so this increased effort would not be of much use.

F. Pose Graph Optimization

1) Keyframe-Keyframe Edges: The final pose graph consists of several binary edges between keyframes that were either computed by visual odometry, by blindly applying the motion model if visual odometry failed, or by keyframe-keyframe registration within the backend. We assign the same constant weights to VO and KF-KF edges and a much lower weight to motion model edges, since their relative poses are very uncertain.

2) Ground Plane Edges: If we found a ground plane in the original image that was made a keyframe by PTAM, this is an incomplete absolute pose measurement we should also use in pose graph optimization. We utilize this information by adding special unary edges with the following two reprojection errors for height and attitude:

$$e_{att} = \operatorname{acos}\left(\left({}^{C_i}\boldsymbol{R}_W \cdot {}^{W}\boldsymbol{up}\right)^T \cdot {}^{C_i}\boldsymbol{up}_{meas}\right)$$
$$e_h = h_{meas} - h_{exp}$$

The first element is on the angle between measured and expected up direction, the second element is the error in height. Both the expected up direction ${}^{C_i}\boldsymbol{up}$ and height origin h_{exp} can be found in the third row of the homogeneous transformation matrix that corresponds to the current pose estimate of the keyframe pose.

$$\left({}^{W}\boldsymbol{T}_{C_i}\right)_3 = \begin{pmatrix} {}^{C_i}\boldsymbol{up}^T & h_{exp} \end{pmatrix}$$

3) Implementation: The actual optimization of our pose graph is implemented using g2o [15], which supports deriving custom SLAM edges and thus allows us to easily integrate ground plane measurements.

VI. NAVIGATION

We implemented two navigation schemes to prove the vision-based autonomy of our MAV. The first is a semi-autonomous navigation scheme using drift-corrected visual odometry poses, the second is a fully autonomous waypoint navigation system using SLAM poses.

A. Semi-Autonomous Navigation

We call this method semi-autonomous navigation, since the operator interacts with the MAV during its flight, remotely modifying the desired pose of the MAV. The MAV will try to reach and hold the latest desired pose using a PD position controller, which is provided with the drift-corrected visual odometry pose as its input.

For semi-autonomous navigation, relying on pose estimates by drift-corrected visual odometry alone is perfectly fine: Since the human operator specifies desired poses relative to the MAVs current pose, drift in x, y and yaw is automatically corrected by the operator. The SLAM back-end in this case is only responsible for producing consistent maps.

B. Fully Autonomous Waypoint-Following

Our second navigation scheme is fully autonomous waypoint-following using SLAM pose estimates. Here we want the MAV to follow a predefined path given as a list of waypoints, with drift in x and y corrected by loop closures whenever possible.

Using the SLAM pose estimate to directly control the MAVs pose, however, can lead to instability during flight, since it can exhibit large jumps after closed loops, which would result in extreme control outputs. We instead implement the following navigation scheme using both SLAM and visual odometry pose estimates:

When trying to reach a waypoint w provided in SLAM coordinates ${}^{S}w$, we compute and update its drift-corrected visual odometry coordinates ${}^{VO}w$ at a constant rate. We limit the distance between the MAV's current pose and ${}^{VO}w$ to prevent extreme control output: If ${}^{VO}w$ is too far away, we choose the admissible pose closest to ${}^{VO}w$ instead.

VII. Evaluation on Benchmark Dataset

We evaluate the system described above and its individual building blocks using the TUM RGBD benchmark dataset [16], which contains several log files of image streams captured with an RGBD camera including its ground truth pose estimates obtained from an external tracking system. We choose four logs that correspond to a handheld-slam (all) scenario and ideally contain clearly visible ground planes.

A. Ground Plane Detection

We first compare the accuracy of the various ground plane detection techniques described in Sect. III based on the ground truth data included in the file *fr3_long_office_household*. The results can be seen in table I, where *RANSAC* denotes the common preemptive RANSAC method using an inlier-outlier scheme, *I/O/W RANSAC* is preemptive RANSAC using our Inlier/Outlier/Worse Outlier Model, I/O/W + PCA is the above followed up with a refinement step using the PCA method and *I/O/W + ROB. PCA* uses our robust refinement method. We compute height error Δh and attitude error $\Delta\alpha$ which is the angle between actual and expected up direction.

	$\Delta\alpha$ in [°]		Δh in [cm]	
	MAE	RMSE	MAE	RMSE
RANSAC	25.25	33.22	33.35	35.38
I/O/W RANSAC	1.22	1.52	2.60	3.38
I/O/W + PCA	0.60	0.71	0.92	1.12
I/O/W + ROB. PCA	0.58	0.68	0.56	0.73

TABLE I

Ground plane accuracy of methods described in Sect. III.

We can clearly see that each extension successively improves the result. Plane estimates using basic RANSAC are more than one order of magnitude worse compared to all other methods because it often detects other planar surfaces (e.g. the surface of a desk) instead of the actual ground plane. This problem is solved in successive methods by penalizing outliers below the ground plane with the inlier/outlier/worse outlier model.

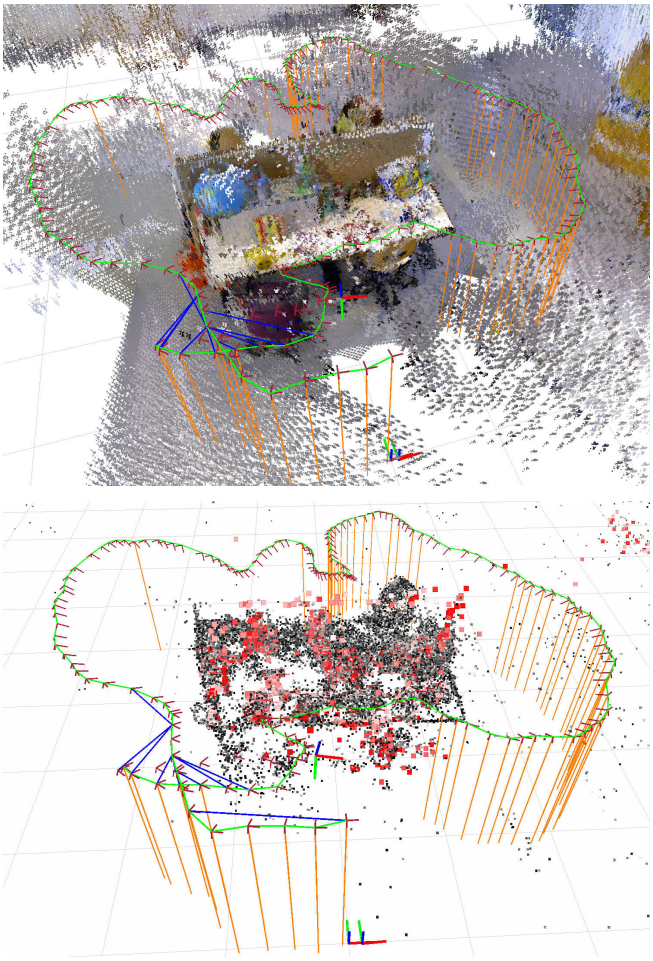


Fig. 5. Point cloud reconstruction of the fr3_long_office_household map and SLAM graph (top) and actual map (bottom) with points currently used by visual odometry (red) and the SLAM back-end (gray).

B. Drift-Corrected Odometry

We also explicitly investigate drift in height and attitude using visual odometry (VO) and drift-corrected visual odometry (DC-CVO) by considering the final error in attitude and height, as can be seen in table II. Even though VO alone is rather accurate already, DC-VO offers a significant improvement in both drift in height and attitude.

	VO		DC-VO	
	Δh [cm]	$\Delta\alpha$[°]	Δh [cm]	$\Delta\alpha$[°]
fr2_desk	4.58	3.96	2.44	1.66
fr3_office	7.71	1.18	0.82	0.13

TABLE II

Final error in height and orientation.

C. VO, DC-VO, and SLAM

Finally, we compare the accuracy of visual odometry, drift-corrected visual odometry, and the SLAM back-end using the

RGBD benchmark tool[2], which reports the drift in relative position error (RPE) per time and the absolute trajectory error (ATE). The results are shown in table III. Ground truth data for the last entry is not publicly available but obtained the results from the online evaluation service. We consider the pose estimates obtained at camera rate for VO and DC-VO and keyframe poses for SLAM.

	VO		DC-VO		SLAM
	ATE [cm]	RPE [cm/s]	ATE [cm]	RPE [cm/s]	ATE [cm]
fr1_room	12.67	4.48	n/a	n/a	8.16
fr2_desk	7.69	1.83	7.58	1.62	8.44
fr3_office	4.18	1.17	3.88	1.46	2.10
fr3_office(v)	2.51	1.01	3.29	1.19	3.19

TABLE III

VISUAL ODOMETRY ACCURACY ON VARIOUS DATASETS AS REPORTED BY THE RGBD BENCHMARK TOOL.

Note that the errors reported by the benchmark tool sometimes increase for DC-VO. The reasons are twofold: Since our drift-correction step deliberately corrupts the relative pose in order to improve the absolute pose estimate, we can expect RPE to increase. ATE is computed by the benchmark tool after aligning evaluated and reference trajectory using a rigid body transform that minimizes ATE in a least-squares sense. This alignment often removes most position drift already. Disabling this functionality leads to the expected results that differ significantly. We decided to report the results obtained by the standard evaluation, however, to avoid confusion when comparing with other publications.

VIII. EXPERIMENTS WITH AUTONOMOUS MAV

A. Fully Autonomous Flight of our MAV using SLAM poses

We first demonstrate that the proposed system enables our MAV to fly completely autonomously when following a set of predefined waypoints using the method described in Sect. VI-B. We do this by letting the MAV fly the same rectangle three times in a row within our laboratory. A screenshot of the visualization containing a part of the built map can be seen in Fig. 6, or in much more details videos available online.[3] The current pose estimate is depicted using the simple quadrotor model. The red arrow is the currently active waypoint, whereas the big coordinate frame is the currently active set point for position control. We also show keyframes (small red frames), visual odometry edges (green), loop closure edges (blue) and ground plane edges (orange). Within the dense point cloud reconstruction, one can see a few very small red points, which are the map points currently used by PTAM.

[2] https://vision.in.tum.de/data/datasets/rgbd-dataset/tools

[3] Videos are available at: http://www.cogsys.cs.uni-tuebingen.de/mitarb/scherer/dctam/

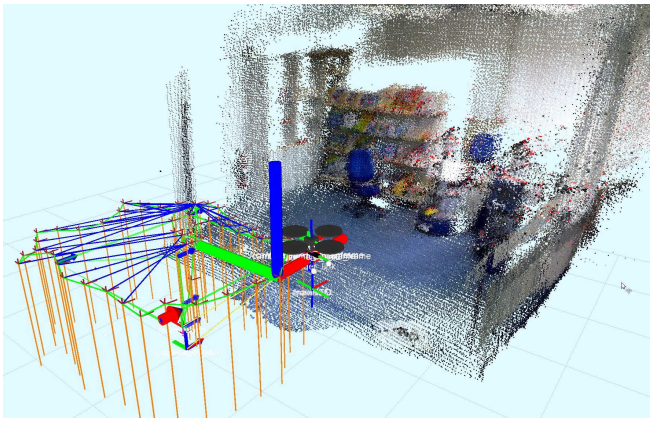


Fig. 6. Visualization while our MAV is flying fully autonomously by following a predefined path within our robot laboratory.

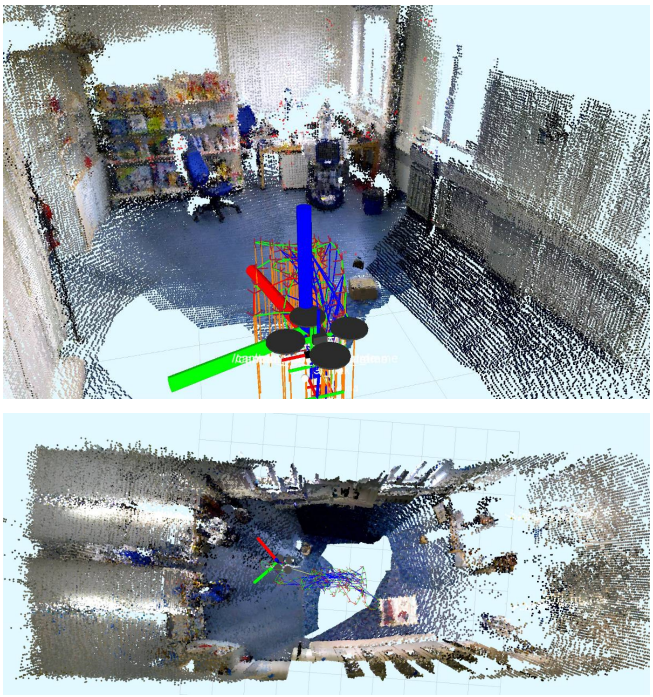


Fig. 7. Visualization while our MAV is navigating semi-autonomously, i.e. autonomously flying to waypoints that are modified by an operator in flight, (top) and overview of the reconstructed room at the end of the flight (bottom).

B. Semi-Autonomous Flight

We also show a semi-autonomous flight in order to demonstrate that we can also map slightly larger environments using on-board processing alone. We rely on semi-autonomous operation as described in Sect. VI-A in this case. A visualization of the map built by the MAV in flight and the final map of the room is shown in Fig. 7. Again, please refer to the accompanying video for more images.

C. Evaluation: Processing Time

We evaluate the processing time required for individual steps of both tracking and the SLAM back-end while our MAV is flying semi-automatically through our laboratory.

The measured times are shown in Table IV. During this ca. 5 minutes long flight, it created a map consisting of 307 keyframes in total. Fast tracking is imperative to enable autonomous flight, especially when using only a basic PD controller. It is obvious that our combination of ground plane detection and tracking using PTAM can conveniently be computed onboard at camera rate (30 Hz), even though there are now three major threads competing for two cores of the CPU. Image preparation includes converting the original RGB image to grayscale, which is still required.

step	time [ms]	step	time [ms]
image prep.	1.55 ± 0.84	KF conversion	20.34 ± 8.96
ground plane	0.96 ± 0.75	loop detection	48.71 ± 27.42
tracking	22.17 ± 13.42	registration	59.68 ± 26.16
total	24.79 ± 13.85	PGO iteration	5.62 ± 4.16

TABLE IV

COMPUTATION TIMES REQUIRED BY STEPS FOR TRACKING (LEFT) AND THE SLAM BACK-END (RIGHT).

IX. CONCLUSIONS & FUTURE WORK

A. Conclusions

We present a drift-corrected software system for efficient on-board visual odometry and SLAM, aimed at but not limited to being used by MAVs. We demonstrate its accuracy on parts of the TUM RGBD benchmark dataset and show that this system enables our MAV to fly both semi-autonomously with relative set points chosen by a human operator and fully autonomously following predefined paths of waypoints.

B. Future Work

Future work will focus on completely autonomous exploration of indoor environments. The major missing step towards this goal is efficiently converting keyframe-based maps to representations that are more useful for path planning and exploration than point clouds, e.g. occupancy grid maps. A promising and efficient method was recently shown in [17], a suitable method of handling loop closures when building occupancy grid maps, preferably without having to rebuild the whole occupancy map, however, still remains to be investigated.

We currently assume to get at least a few depth measurements for each keyframe, as is the case for both RGBD and stereo cameras, in our modified version of PTAM. The back-end, however, never uses these depth measurements but relies only on triangulated 3D map points instead. We are considering extending our back-end to also support monocular SLAM by converting both keyframe to keyframe registration and pose graph optimization to work on the scale-drift aware Lie group SIM(3) instead of SE(3) as proposed in [18].

REFERENCES

[1] A. Bachrach, S. Prentice, R. He, P. Henry, A. S. Huang, M. Krainin, D. Maturana, D. Fox, and N. Roy, "Estimation, planning, and mapping for autonomous flight using an rgb-d camera in gps-denied environments," *The International Journal of Robotics Research*, vol. 31, no. 11, pp. 1320–1343, 2012.

[2] K. Schmid, T. Tomic, F. Ruess, H. Hirschmuller, and M. Suppa, "Stereo vision based indoor/outdoor navigation for flying robots," in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, Nov 2013, pp. 3955–3962.

[3] S. A. Scherer and A. Zell, "Efficient Onboard RGBD-SLAM for Fully Autonomous MAVs," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2013)*, Tokyo Big Sight, Japan, November 2013.

[4] 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.

[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] 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.

[7] 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. [Online]. Available: http://www.springerlink.com/content/x7p122p320v3q027/

[8] J. Shi and C. Tomasi, "Good features to track," in *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR '94., 1994 IEEE Computer Society Conference on*, 1994, pp. 593–600.

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

[10] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[11] N. Campbell, "Robust procedures in multivariate analysis i: Robust covariance estimation," *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, vol. 29, pp. 231–237, 1980.

[12] J. W. Weingarten, G. Gruener, and R. Siegwart, "Probabilistic plane fitting in 3d and an application to robotic mapping," in *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, 2004, pp. 927–932.

[13] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "Brief: Binary robust independent elementary features," in *Computer Vision–ECCV 2010*. Springer, 2010, pp. 778–792.

[14] D. Galvez-Lopez and J. D. Tardos, "Bags of binary words for fast place recognition in image sequences," *IEEE Transactions on Robotics*, vol. 28, no. 5, pp. 1188–1197, October 2012.

[15] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, "g2o: A general framework for graph optimization," in *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, Shanghai, China, May 2011.

[16] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A Benchmark for the Evaluation of RGB-D SLAM Systems," in *Proc. of the International Conference on Intelligent Robot Systems (IROS)*, Oct. 2012.

[17] K. Schauwecker and A. Zell, "Robust and efficient volumetric occupancy mapping with an application to stereo vision," in *In IEEE International Conference on Robotics and Automation (ICRA)*, 2014.

[18] H. Strasdat, J. Montiel, and A. J. Davison, "Scale drift-aware large scale monocular slam." in *Robotics: Science and Systems*, vol. 1, no. 2, 2010, p. 4.