Using Depth in Visual Simultaneous Localisation and Mapping

Sebastian A. Scherer, Daniel Dube and Andreas Zell

***Abstract*— We present a method of utilizing depth information as provided by RGBD sensors for robust real-time visual simultaneous localisation and mapping (SLAM) by augmenting monocular visual SLAM to take into account depth data. This is implemented based on the feely available software "Parallel Tracking and Mapping" by Georg Klein. Our modifications allow PTAM to be used as a 6D visual SLAM system even without any additional information about odometry or from an inertial measurement unit.**

I. Introduction

A. RGBD Sensors

RGBD sensors are cameras that provide a regular color (RGB) image augmented by depth measurements (D) for a large number of image pixels. Early research on RGBD sensors was based on combining color images with depth measurements from time-of-flight cameras or laser rangefinders. These usually produce only few depth measurements compared to the number of pixels within the RGB image.

With the introduction of Microsoft's Kinect, cheap and lightweight 3D sensors with high resolution became widely available and research on RGBD sensors greatly increased. The depth sensing system of Microsoft's Kinect was developed by and licensed from PrimeSense, whose reference design is very similar and was available to a small group of developers even before the release of the Kinect.

Both sensors provide frames at VGA resolution (640×480 pixels). If we want the depth image to be aligned pixel-by-pixel to its color counterpart, we need to calibrate both cameras and transform all points in the depth image into RGB pixel coordinates, as the center of projection of both cameras can never be at the same position. Fortunately, the Kinect can do this transformation in Hardware already.

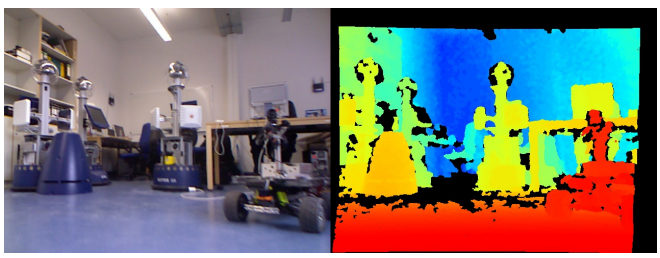

Fig. 1. Example RGBD frame captured using a Kinect. It consists of a regular color (RGB) image (shown left) and a depth image (shown right, visualized as a heatmap). Note that the latter is transformed such that each position in both RGB and depth image correspond to the same ray in 3D.

S.A. Scherer, D. Dube and A. Zell are with the Chair of Cognitive Systems, Computer Science Department, University of Tübingen, Sand 1, D-72076 Tübingen, Germany {sebastian.scherer, daniel.dube, andreas.zell}@uni-tuebingen.de

B. Visual SLAM

There has been a large amount of work on visual simultaneous localisation and mapping already. The problem of inferring camera poses and mapping its environment given only images is also known as "structure and motion" in the computer vision literature and as the problem of photogrammetry, which dates back more than a century. The main difference to visual SLAM in robotics is the need for real-time algorithms: An autonomous robot needs to know where it is while driving though an unknown environment.

Work on real-time visual SLAM started to spread after Andrew Davison's work on MonoSLAM in 2003 ([4]), which used an Extended Kalman Filter (EKF) to track image features and the camera pose. Due to the computational complexity of the EKF, however, it can only track a very limited number of features at the same time. Nistér in [10] demonstrated that Visual Odometry (VO) can be computed efficiently even by matching large numbers of interest points in successive frames.

C. SLAM using RGBD Sensors

When using RGBD sensors, a whole new class of SLAM approaches become available if we assume that there is always a large number of depth measurements. Henry et al. [7] match image features between successive RGB frames, use depth to determine their 3D positions in each camera frame, and estimate the transform between both frames by aligning both sets of points, e.g. using one of the algorithms described in [5]. After that they refine this transform by running Iterative Closest Point (ICP) on both full point clouds.

If it is always possible to extract the full 6D transform between successive frames, the SLAM task can be divided into a frontend estimating transforms between frames and a backend optimizing the pose graph only, as in [6].

D. Our Approach

In constrast to the approaches mentioned before, we do not want to rely on having depth measurements at almost all pixel positions for two main reasons:

First, even modern RGBD sensors produce dense depth images only under certain constrained conditions: They are limited by their minimum and maximum range, they have serious problems with glass and black material, and they cannot cope with direct sunlight. This is demonstrated in Fig. 2. We therefore want a SLAM system that integrates depth data, if available, but can always fall back to pure monocular SLAM using only the RGB image if there is no depth information available.

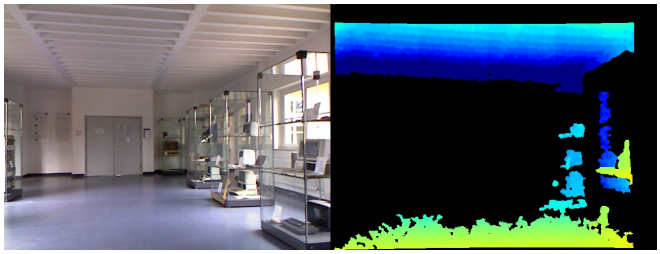

Fig. 2. Example RGBD frame illustrating why we should not rely on depth information too much: Depths higher than the sensors's operating range, (indirect) sunlight, and both reflective and opaque surfaces lead to a rather sparse depth image.

Secondly, we would like to integrate depth measurements from sensors that work under a wider range of environmental conditions but provide only very sparse depth information, e.g. 2D or 3D laser rangefinders, in future work. We therefore decided to start with an existing monocular SLAM system and augment it so it takes depth measurements into account for improved accuracy.

The main contribution of this paper is providing a technique to integrate depth measurements into an existing monocular visual SLAM system. This consists of several rather straightforward changes but also on a way to use depth measurements as additional constraints in bundle adjustment, which, to our knowledge, has not yet been explored.

II. Parallel Tracking and Mapping revisited

Parallel Tracking and Mapping is available in source code[1] and described in [8]. The original software builds keyframe-based maps from data acquired by a monocular camera and uses tracking to compute camera pose estimates relative to the map at high frequencies. This is achieved by two threads running in parallel: One thread is responsible for tracking the camera relative to the map, a second thread integrates new keyframes into the map and refines it using bundle adjustment.

The map is stored keyframe-based and represents both camera poses and map points in a global coordinate system. It consists of an array of keyframes, an array of map points, and maps both from each keyframe to the map points seen in it and from each map point to all keyframes in which it was found. Keyframes in turn contain a pyramidal representation of a camera image, interest points for all levels of the image pyramid, and an estimate for the camera pose from which the image was captured. Map points are represented i 3D coordinates and associated with an image patch around the position in the keyframe where it was seen for the first time.

A. Tracking

A tracking thread tracks the camera position relative to the existing map of the environment using the following steps:

- Predict the camera pose using a simple damped constant-velocity motion model.
- Project all map points and keep only those that could be visible at the predicted pose.
- Warp patches associated with these map points to account for the new viewpoint.
- Try to find these patches at the locations of FAST corners close to the expected position by comparing zero-mean sum of squared differences of pixel intensities.
- Refine detections to sub-pixel positions.
- Update the predicted camera pose using Gauß-Newton, minimizing the reprojection error.
- Decide whether the current frame should be added to the map as a new keyframe.

B. Mapping

The second mapping thread running in parallel integrates keyframes whenever the tracking thread tells it to and spends the rest of the time refining the map. It operates in an infinite loop doing the following tasks:

1) Do local bundle adjustment, modifying the keyframes closest to the most recently added only.
2) Try to find recently added map points in previous keyframes.
3) Do global bundle adjustment, possibly modifying all but the initial keyframe.
4) Clean up the map: Remove measurements that are outliers, remove map points that are supported by too few measurements.
5) Add new keyframe, if available.

Steps 2) to 4) are interrupted as soon as a new keyframe should be added to the map.

C. Problems and Limitations

There are several problems and limitations that prevent PTAM from being widely used for visual SLAM for mobile robots.

1) Scale: Scale ambiguity is an inherent problem of monocular visual SLAM. Using only a monocular camera, we can infer the scale of the map and the camera motion only if we have prior information about one of these two. But even if we had a way of removing the global scale ambiguity, e.g. by starting looking at a scene with known dimensions, applying monocular visual SLAM systems on long trajectories often introduces considerable scale drift.

2) Need For Triangulation: Another problem arises from the fact that PTAM represents all map points as points in 3D space. A point seen in a single monocular image, however, can lie anywhere on a ray through the scene, so PTAM needs to triangulate any point before it can add it to the map. Because of this, PTAM cannot cope with pure rotational movements: It can track previously triangulated map points while rotating. But while they leave the field of view of the camera, it cannot add new map points and will finally lose track of the map. Requiring triangulation also wastes information: Each map point has to be found in two keyframes before it can be added to the map and finally be used for tracking a third keyframe. If the map point was added after the first detection, it could provide useful information for tracking the second keyframe already.

[1] http://www.robots.ox.ac.uk/~gk/PTAM/

3) Large-Scale Mapping: PTAM was originally implemented for augmented reality applications in small workspaces and never intended for large-scale mapping. It is therefore neither prepared to cope with loop closures nor with processing too big maps. This affects both tracking, as it iterates over all map point in each step, and mapping, which cannot complete the global bundle adjustment step for maps with too many keyframes and map points.

III. Using Depth in Parallel Tracking and Mapping

This section describes small changes we made to PTAM in order to efficiently employ depth information as provided by an RGBD sensor, i.e. depth measurements for a large ratio of image pixels. The changes decribed in this section are rather specific to the PTAM software. Our most important modification, using depth in bundle adjustment, can easily be integrated into all systems based on bundle adjustment and is therefore described in detail in Sec.IV.

A. Storing Depth Information

Up to now, our system considers depth information only at the position of interest points in the color image. We therefore store one depth value per interest point instead of the full depth image.

B. Initialization

The original implementation of PTAM requires an interactive initialization procedure: The user needs to manually select two frames viewing the same planar scene as an initial stereo pair, from which it computes the relative transform by homography estimation, triangulates corresponding points, and uses these as the initial map. This initialization step is trivial with depth information: We create the initial map with a single keyframe from the first RGBD frame and add map points for those interest points (FAST corners with non-maximum suppression), for which we have valid depth data available, as we can easily obtain the 3D position of any point using its image coordinates and its depth.

C. Adding Keyframes and Points

When a new keyframe is added, PTAM tries to triangulate interest points of the current keyframe using the previous one and add it as a new map point. If depth is available, we skip epipolar search and just add a new map point using measured depth to obtain a 3D position. If there is no valid depth measurement, we still use epipolar search to try and triangulate it.

IV. Bundle Adjustment

Bundle Adjustment (BA) is the iterative optimization of both camera poses and map points in order to minimize the reprojection error, i.e the distance between the expected and actually measured projected position of a map point within the image of the camera at a certain pose. An extensive overview over all facets of BA can be found in [13].

Using the notation of Klein in [8], BA minimizes the following objective function:

$$\{\{\mu_2\mu_N\},\{p_1p_M\}\} = \underset{\{\{\mu\},\{p\}\}}{\operatorname{argmin}} \sum_{i=1}^{N}\sum_{j\in S_i} \mathrm{Obj}\left(\frac{\|\mathbf{e}_{ji}\|}{\sigma ji},\sigma_T\right) \quad (1)$$

Where μ_i is the representation of the pose of camera i, p_j is the position of map point j, $\|\mathbf{e}_{ji}\|$ is the reprojection error of map point j in camera i, σ_{ji} its estimated standard deviation, σ_T is a robust standard deviation estimate across all reprojection errors based on their median (see [14]), and Obj is a robust objective function, e.g. Tukey's biweight function.

For regular BA, the reprojection error is the difference between the expected position $\hat{\mathbf{u}}$ of a map point within the image based on the current estimates of μ_i and $\mathbf{p}_j$ and the actual measured position $\mathbf{u}$.

$$\mathbf{e}_{ji} = \mathbf{u} - \hat{\mathbf{u}} \quad (2)$$

For the rest of this paper, will assume the 2D reprojection error to be measured in actual pixel positions. Minimizing it in normalized image coordinates is an alternative, but using pixel coordinates allows us to intuitively provide an estimate for its standard deviation σ_{ji} (see Sect. V-A).

A. Projection Model

PTAM models the position $\mathbf{p} \in \mathbb{R}^3$ of map points in world coordinates and camera poses $\mathbf{A} \in \mathbf{SE}(3)$ with the corresponding transformation matrix $\mathbf{T}(\mathbf{A}) \in \mathbb{R}^{n\times n}$ using their inverse pose, i.e. the position of point $\mathbf{p}$ relative to the camera at pose $\mathbf{A}$ is:

$$\mathbf{p_A} = \mathbf{T}(\mathbf{A}) \cdot \mathbf{p} = \mathbf{A} \oplus \mathbf{p} \quad (3)$$

Where the latter is the composition of a 6D pose and a 3D point using the notation as defined in [1].

The complete projection model used in PTAM is

$$\mathbf{u} = \mathbf{c}(\mathbf{h}(\mathbf{T}(\mathbf{A}) \cdot \mathbf{p})) \quad (4)$$

where:

- $\mathbf{u} \in \mathbb{R}^2$ is the projected pixel position of the point $\mathbf{T}(\mathbf{A})\cdot\mathbf{p}$ in camera coordinates,
- $\mathbf{h} : \mathbb{R}^3 \to \mathbb{R}^2$, $\left(p_x, p_y, p_z\right)^T \mapsto \left(\frac{p_x}{p_z}, \frac{p_x}{p_z}\right)^T$ is the basic perspective projection,
- $\mathbf{c} : \mathbb{R}^2 \to \mathbb{R}^2$ transforms normalized image coordinates to actual pixel coordinates, considering the camera calibration matrix and its distortion model.

B. Optimization

Minimizing Eq. (1) is usually performed by choosing Obj according to an M-estimator and treating the minimization as an iteratively reweighted (nonlinear) least squares problem. This assumes we can approximate the cumulative objective function by:

$$\sum_{i=1}^{N}\sum_{j\in S_i} \mathrm{Obj}\left(\frac{\|\mathbf{e}_{ji}\|}{\sigma ji},\sigma_T\right) = \mathbf{f}^T\mathbf{W}\mathbf{f} \quad (5)$$

Where the vector $\mathbf{f}$ consists of all reprojection errors stacked to form a vector and $\mathbf{W}$ ist a weight matrix weighing all measurements according to the value of the objective function. This can then be optimized e.g. using the Levenberg-Marquardt method. A detailed introduction would go beyond the scope of this paper but can be found in [11].

In order to compute optimization update steps efficiently, we need to know the Jacobians of the reprojection error function with respect to the estimated point position and estimated camera pose. The tutorial in [1] is an invaluable resource for hints on this. For the projection model above, the Jacobian $\mathbf{J_p}$ of the reprojection error of a single measurement of point $\mathbf{p}$ seen at pose $\mathbf{A}$ with respect to the point position can be computed using the chain rule as:

$$\mathbf{J_p} = \frac{\partial \mathbf{c}(\mathbf{h}(\mathbf{A} \oplus \mathbf{p}))}{\partial \mathbf{p}} \tag{6}$$

$$= \left.\frac{\partial \mathbf{c}(\mathbf{u})}{\partial \mathbf{u}}\right|_{\mathbf{u}=\mathbf{h}(\mathbf{A}\oplus\mathbf{p})} \cdot \left.\frac{\partial \mathbf{h}(\mathbf{p_A})}{\partial \mathbf{p_A}}\right|_{\mathbf{p_A}=\mathbf{A}\oplus\mathbf{p}} \cdot \frac{\mathbf{A}\oplus\mathbf{p}}{\partial \mathbf{p}} \tag{7}$$

The first factor in this matrix product is the Jacobian of the camera's intrinsic calibration model. The second term can easily be computed to be:

$$\mathbf{J}_h(p) = \left.\frac{\partial h(\mathbf{p_A})}{\partial \mathbf{p_A}}\right|_{p_A=\mathbf{A}\oplus\mathbf{p}} = \frac{1}{p_z^2}\begin{pmatrix} p_z & 0 & -1 \\ 0 & p_z & -1 \end{pmatrix} \tag{8}$$

The last term is simply:

$$\frac{\mathbf{A}\oplus\mathbf{p}}{\partial \mathbf{p}} = \mathbf{R_A} \tag{9}$$

i.e., the Rotation matrix associated with the pose $\mathbf{A}$ (see [1] Eq. 7.13). Similarly, the Jacobian $\mathbf{J_A}$ with respect to the camera pose is:

$$\mathbf{J}_\mu = \frac{\partial \mathbf{c}(\mathbf{h}(e^\epsilon \mathbf{A} \oplus \mathbf{p}))}{\partial \epsilon}$$

$$= \left.\frac{\partial \mathbf{c}(\mathbf{u})}{\partial \mathbf{u}}\right|_{\mathbf{u}=\mathbf{h}(\mathbf{A}\oplus\mathbf{p})} \cdot \left.\frac{\partial \mathbf{h}(\mathbf{p_A})}{\partial \mathbf{p_A}}\right|_{\mathbf{p_A}=\mathbf{A}\oplus\mathbf{p}} \cdot \frac{e^\epsilon \oplus \mathbf{A} \oplus \mathbf{p}}{\partial \epsilon}$$

Only the last factor is different from $\mathbf{J_p}$, i.e.:

$$\frac{e^\epsilon \oplus \mathbf{A} \oplus \mathbf{p}}{\partial \epsilon} = (\mathbf{I}_3 \quad -[\mathbf{A}\oplus\mathbf{p}]_\times) \tag{10}$$

C. Integrating Depth

If we can gather additional information about the environment (here: depth data), we should really use this additional information in BA for higher accuracy by adding more or different measurements to be considered when optimizing Eq. (1). Two alternatives quickly come to mind:

- Using the full 3D reprojection error, i.e. computing the difference between expected and measured position of a point in 3D instead of in 2D.
- Relying on the conventional 2D reprojection error with an additional constraint for the deviation of measured depth from the expected depth estimate.

It is important to realize that it does not matter whether we mix measurements of different units or magnitudes in Eq. (1) as long as we provide a reasonable estimate of the error σ_{ij} for each measurement.

D. BA with full 3D reprojection error

The idea of minimizing the full 3D reprojecting error is convincingly simple: According to the RGBD assumption, the full 3D position in camera coordinates of the point corresponding to each pixel with depth available can be computed from its normalized image coordinates $\mathbf{n}$ and depth d:

$$\mathbf{p_A} = \left(p_x, p_y, p_z\right)^T = d\left(n_x, n_y, 1\right)^T \tag{11}$$

Instead of minimizing the 2D reprojection error (c.f. Eq. (2)), we can now minimize the 3D reprojection error:

$$\mathbf{e}_{ji} = \mathbf{p} - \hat{\mathbf{p}} \tag{12}$$

This leads to much simpler Jacobians compared to the 2D case:

$$\mathbf{J_p} = \frac{e^\epsilon \oplus \mathbf{A} \oplus \mathbf{p}}{\partial \mathbf{p}} = \mathbf{R_A} \tag{13}$$

$$\mathbf{J_A} = \frac{e^\epsilon \oplus \mathbf{A} \oplus \mathbf{p}}{\partial \epsilon} = (\mathbf{I}_3 - [\mathbf{g}]_\times) \tag{14}$$

The problem with using this representation, however, is the fact that the 2D position of a measurement and its depth value both introduce different types of errors, which we cannot accurately acccount for by using only one single scale factor in Eq. (1). We might be able to fix this by considering the 3D covariance matrix. There is, however, a much more elegant solution:

E. BA with depth reprojection errors

A much cleaner way of integrating depth is adding one single 1D constraint per depth measurement in addition to regular 2D constraints derived from the RGB image alone.

$$d = (\mathbf{T}(\mathbf{A}) \cdot \mathbf{p})_z \tag{15}$$

This is equivalent to using only the last row of Eq. (12). The corresponding Jacobians each consist of the last rows of Eq. (13) and (14), respectively.

By adding separate measurements for 2D and depth, we can also treat their errors individually. Estimating these errors is described in Sect. V-A and V-B. In addition to modeling errors more accurately, we can also decide for each measurement individually whether it is an outlier or not: If the depth measurement at a certain position is completely off and should be discarded as an outlier, there is still a chance that the 2D measurement is correct and adds useful information to the optimization system.

V. Experiments

This section describes experiments conducted to model measurement noise and to evaluate the gain in accuracy by using the proposed way of integrating depth into monocular visual SLAM.

We did our experiments on data recorded on a MetraLabs SCITOS G5 rolling mobile robot with a differential drive mechanism, a SICK S300 laser rangefinder for 2D localisation, and a forward-looking Microsoft Kinect mounted on its top. It traveled ca. 106 meters along a long corridor and drove through a museum room with old computers behind

several glass cabinets. Long, weakly textured walls, reflecting surfaces, and a mixture of artificial light and sunlight shining through windows made this a challenging environment.

A. Modeling 2D Reprojection Errors

Klein and Murray in [8] already use observations with different measurement noise estimates: Even using sub-pixel refinement, we will expect the matching accuracy to be proportional to the length of a pixel in the image that was used for matching. This varies as map points can be found at different levels of the image pyramid, yet their measured image coordinates are given with respect to scale level 0. The measurement noise is thus modeled as:

$$\sigma_{ij} = k \cdot s_{ij} \tag{16}$$

Where s_{ij} is the scale of the pyramid level at which map point j was found in keyframe i and k is an unknown scale factor. The value of k is dropped in the original PTAM system as it is the same constant for all 2D measurements and applying a global scale factor to all measurements does not change the optimization outcome.

In our case, however, we do need to estimate k in order to combine 2D image point measurements with depth placing neither too much nor too low confidence on either type of measurement. We did this by running PTAM using only 2D measurements and estimating k from the distribution of all remaining reprojection errors, which yielded $k = 0.987$

B. Modeling Depth Errors

We used the Kinect sensor to record RGBD frames of a single large planar wall seen at distances ranging from 0.5 m to 3.5 m as we expected the error to depend on the true depth. We used the LO-RANSAC algorithm [3] to estimate a planar model of the wall for each of these frames and used this model to compute the depth error, i.e. the z component of the vector connecting the 3D point with the point at which its corresponding ray in the scene intersects the plane.

In a second step we sorted all depth errors by the true depth into bins of 10 cm width and calculated the standard deviation σ_b for each bin b. One can clearly see from Fig. 3 (sampled standard deviation) that the error is proportional to the depth squared, so we modeled the standard deviation by a second order polynomial function

$$\sigma(d) = a \cdot d^2 \tag{17}$$

where d is the depth of a reading. We estimated the parameter $a = 3.331 \cdot 10^{-3}$ by minimizing the relative error e_b of the model with respect to the standard deviations of the sampled bins:

$$e_b = \sum_b \frac{\sigma_b - \sigma(d_b)}{\sigma_b} \tag{18}$$

This noise model was measured using the easiest scenario possible for a 3D sensor and assuming that all depth measurements are statistically independent of each other. Due to the measurement method of the Kinect, we can not expect that this model is still valid for more complex or cluttered environments. However, we can expect that the noise can still be modeled by function (17) for complex scenarios but with a higher value for a.

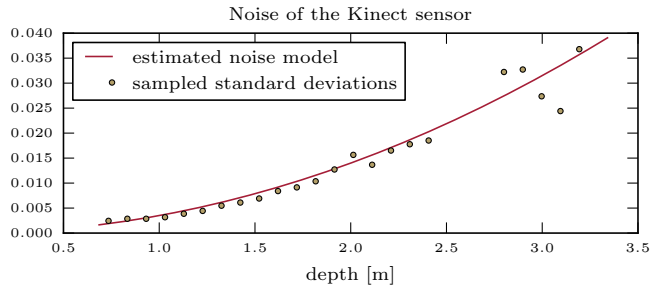

Fig. 3. Standard deviations of the Kinect sensor we determined experimentally for different distances and the model we estimated from these

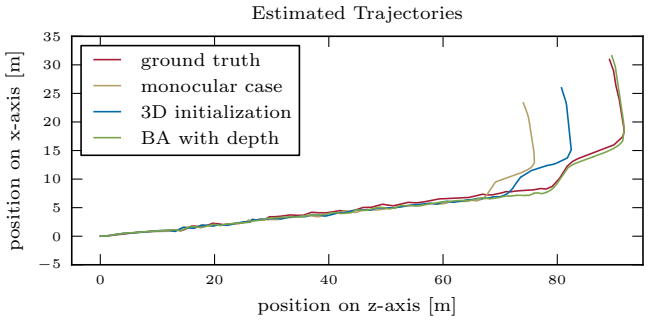

Fig. 4. Trajectories as estimated by laser localisation (red), monocular visual SLAM (yellow), monocular visual SLAM with 3D initialization (blue), and SLAM with depth constraints (green).

C. Evaluating SLAM accuracy

For accuracy evaluation, we modified our version of PTAM to run in batch-mode in a single thread to process every single frame recorded by the robot without randomly dropping frames due to thread timings or the hard disk not keeping up with the required bandwidth while playing back. Ground truth data is available from 2D laser-based localisation within an existing 2D map of this environment. Fig. 4 shows the trajectories estimated using different methods of integrating depth data. The yellow line shows the resulting trajectory of using regular monocular visual SLAM. Only depth measurements from the very first RGBD frame are used to initialize the map at the correct scale. If we did not do this, the resulting map and trajectory would be of arbitrary scale and we could not compare it to any other result. The blue line in Fig.4 shows the trajectory obtained by using depth information for initial 3D estimates of points before they are added to the map. If there is no depth information available for a prominent interest point, we still fall back to epipolar search. The green line, finally, combines using depth for initial 3D estimates and bundle adjustment using depth data. We can clearly see the scale drift when using monocular SLAM without additional information. In

TABLE I

LOCALIZATION ERROR AT THE END OF THE RUN.

	position error [m]	orientation error [°]
monocular VO	16.78	2.61
with 3D initialization	9.64	2.28
with depth constraint	2.13	3.30

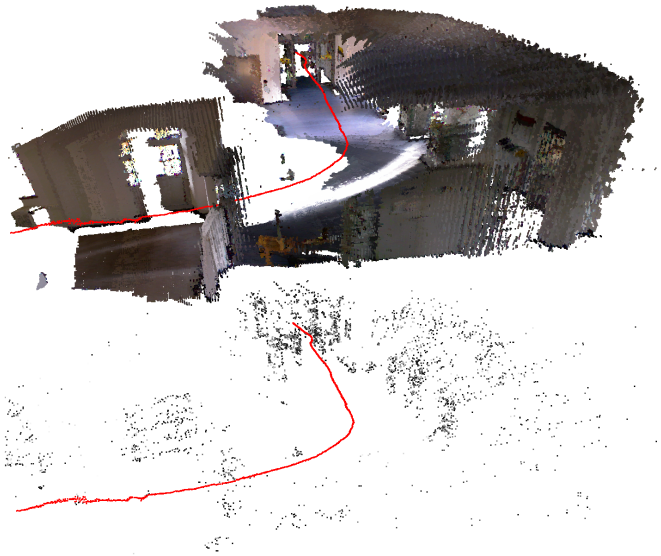

Fig. 5. A part of the environment (museum room) passed by our mobile robot while recording the evaluation dataset. Each red dot shows a pose estimate obtained after processing a single RGBD frame. Top: Full colored point clouds of some of the keyframes, transformed according to camera poses as reconstructed by bundle adjustment with depth information. Bottom: Same perspective, but only actual map points are drawn.

our experiments, PTAM generally underestimated scale. This could be due to the damped constant-velocity motion model, which always underestimates the motion of a robot moving at a constant speed. Tracking and bundle adjustment will then iteratively approach the true motion of the robot but rarely overshoot. Using depth for map point initialization improves the estimated trajectory. There is, however, still a a considerably large difference compared to the ground truth trajectory due to scale drift. Only by adding depth constraints to bundle adjustment we can finally get results very close to what we expect from laser-based localisation.

VI. Results

A. Conclusions

Even though modern 3D sensors are capable of providing dense RGBD frames in some environments, relying on this fact for SLAM in arbitrary environments is probably not a good idea. As a simple alternative, we demonstrate an easy way to utilize sparse depth information in a visual monocular SLAM system without negative impact on its computational complexity. We show that even using only these sparse depth measurements effectively removes the scale drift and improves the accuracy considerably from 16.78 meters down to 2.13 meters over a traveled distance of 106 meters. This is equivalent to a position error of 2.01%.

There are still some problems inherent to PTAM which we did not solve in this work: As the map grows larger in the number of keyframes (N) and the number of map points (M), its global bundle adjustment step will at some point fail to finish even a single iteration before it gets interrupted by new keyframes being added, whis is due to its runtime complexity of $O(N^3 + N^2M)$. This could be sped up by utilizing the sparse secondary structure of bundle adjustment as shown in [9] or by optimizing the global system using faster iterations that converge slower, e.g. by using conjugate gradient bundle adjustment ([2]).

Global bundle adjustment would still not scale arbitrarily, but using a combination of local bundle adjustment with global pose graph optimization or a relative representation as in [12] could solve this problem.

B. Future Work

Our future work will involve using different depth sensors that might provide a much smaller number of depth measurements, but work more reliably: We will try and utilize depth information from 2D or 3D laser rangefinders in visual SLAM. The hard part in this case, however, is inferring depth at pixel positions when the next depth measurement is a certain distance (in image coordinates) away.

We plan to deploy the system described in this paper for SLAM on micro aerial vehicles. Due to the fast tracking speed of PTAM even on constrained hardware we believe it can provide pose estimates fast enough to control these during autonomous flight.

References

[1] José-Luis Blanco. A tutorial on se(3) transformation parameterizations and on-manifold optimization. Technical report, University of Malaga, September 2010.

[2] Martin Byröd and Kalle Åström. Conjugate gradient bundle adjustment. In Kostas Daniilidis, Petros Maragos, and Nikos Paragios, editors, *ECCV (2)*, volume 6312 of *Lecture Notes in Computer Science*, pages 114–127. Springer, 2010.

[3] Ondřej Chum, Jiří Matas, and Štěpán Obdržálek. Enhancing ransac by generalized model optimization. In *Proc. of the Asian Conference on Computer Vision (ACCV)*, volume 2, pages 812–817, Seoul, Korea South, January 2004. Asian Federation of Computer Vision Societies.

[4] A.J. Davison. Real-time simultaneous localisation and mapping with a single camera. In *Proc. International Conference on Computer Vision, Nice*, October 2003.

[5] D. W. Eggert, A. Lorusso, and R. B. Fisher. Estimating 3-d rigid body transformations: a comparison of four major algorithms. *Mach. Vision Appl.*, 9:272–290, March 1997.

[6] N. Engelhard, F. Endres, J. Hess, J. Sturm, and W. Burgard. Real-time 3d visual slam with a hand-held camera. In *Proc. of the RGB-D Workshop on 3D Perception in Robotics at the European Robotics Forum*, Vasteras, Sweden, April 2011.

[7] Peter Henry, Michael Krainin, Evan Herbst, Xiaofeng Ren, and Dieter Fox. RGB-D Mapping: Using Depth Cameras for Dense 3D Modeling of Indoor Environments.

[8] 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.

[9] Kurt Konolige. Sparse sparse bundle adjustment. In *Proceedings of the British Machine Vision Conference*, pages 102.1–102.11. BMVA Press, 2010. doi:10.5244/C.24.102.

[10] David Nistér, Oleg Naroditsky, and James Bergen. Visual odometry for ground vehicle applications. *Journal of Field Robotics*, 23:2006, 2006.

[11] Jorge Nocedal and Stephen J. Wright. *Numerical optimization*. Springer, August 1999.

[12] Gabe Sibley, Christopher Mei, Ian Reid, and Paul Newman. Adaptive relative bundle adjustment. In *Robotics Science and Systems (RSS)*, Seattle, USA, June 2009.

[13] Bill Triggs, P. McLauchlan, Richard Hartley, and A. Fitzgibbon. Bundle adjustment – a modern synthesis. In B. Triggs, A. Zisserman, and R. Szeliski, editors, *Vision Algorithms: Theory and Practice*, volume 1883 of *Lecture Notes in Computer Science*, pages 298–372. Springer-Verlag, 2000.

[14] Zhengyou Zhang, Zhengyou Zhang, Programme Robotique, and Projet Robotvis. Parameter estimation techniques: A tutorial with application to conic fitting. *Image and Vision Computing*, 15:59–76, 1997.