2018 IEEE Winter Conference on Applications of Computer Vision

Tracking an RGB-D Camera on Mobile Devices Using an Improved Frame-to-Frame Pose Estimation Method

Jaepung An* Jaehyun Lee† Jiman Jeong† Insung Ihm*
*Department of Computer Science and Engineering
Sogang University, Korea
†TmaxOS
Korea
{ajp5050,ihm}@sogang.ac.kr {jaehyun_lee,jiman_jeong}@tmax.co.kr

Abstract

The simple frame-to-frame tracking used for dense visual odometry is computationally efficient, but regarded as rather numerically unstable, easily entailing a rapid accumulation of pose estimation errors. In this paper, we show that a cost-efficient extension of the frame-to-frame tracking can significantly improve the accuracy of estimated camera poses. In particular, we propose to use a multi-level pose error correction scheme in which the camera poses are re-estimated only when necessary against a few adaptively selected reference frames. Unlike the recent successful camera tracking methods that mostly rely on the extra computing time and/or memory space for performing global pose optimization and/or keeping accumulated models, the extended frame-to-frame tracking requires to keep only a few recent frames to improve the accuracy. Thus, the resulting visual odometry scheme is lightweight in terms of both time and space complexity, offering a compact implementation on mobile devices, which do not still have sufficient computing power to run such complicated methods.

1. Introduction

The effective estimation of 6-DOF camera poses and reconstruction of a 3D world from a sequence of 2D images captured by a moving camera has a wide range of computer vision and graphics applications, including robotics, virtual and augmented reality, 3D games, and 3D scanning. With the availability of consumer-grade RGB-D cameras such as the Microsoft Kinect sensor, direct dense methods, which estimate the motion and shape parameters directly from raw image data, have recently attracted a great deal of research interest because of their real-time applicability in robust camera tracking and dense map reconstruction.

The basic element of the direct dense visual localization and mapping is the optimization model which, derived from pixel-wise constraints, allows an estimate of rigid-body motion between two time frames. For effective pose estimation, several different forms of error models to formulate a cost function were proposed independently in 2011. Newcombe et al. [11] used only geometric information from input depth images to build an effective iterative closest point (ICP) model, while Steinbrücker et al. [14] and Audras et al. [1] minimized a cost function based on photometric error. Whereas, Tykkälä et al. [17] used both geometric and photometric information from the RGB-D image to build a bi-objective cost function. Since then, several variants of optimization models have been developed to improve the accuracy of pose estimation. Except for the KinectFusion method [11], the initial direct dense methods were applied to the framework of frame-to-frame tracking that estimates the camera poses by repeatedly registering the current frame against the last frame. While efficient computationally, the frame-to-frame approach usually suffers from substantial drift due to the numerical instability caused mainly by the low precision of consumer-level RGB-D cameras. In particular, the errors and noises in their depth measurements are one of the main sources that hinder a stable numerical solution of the pose estimation model.

In order to develop a more stable pose estimation method, the KinectFusion system [11] adopted a frame-to-model tracking approach that registers every new depth measurement against an incrementally accumulated dense scene geometry, represented in a volumetric truncated signed distance field. By using higher-quality depth images that are extracted on the fly from the fully up-to-date 3D geometry, it was shown that the drift of the camera can decrease markedly while constructing smooth dense 3D models in real-time using a highly parallel PC GPU implementation. A variant of the frame-to-model tracking was presented by Keller et al. [9], in which aligned depth images were incrementally fused into a surfel-based model, instead of a 3D volume grid, offering a relatively more efficient memory implementation. While producing more accurate pose estimates than the frame-to-frame tracking, the frame-to-model tracking techniques must manipulate

978-1-5386-4886-5/18/$31.00 ©2018 IEEE
DOI 10.1109/WACV.2018.00130

the incrementally updated 3D models during camera tracking, whether they are stored via a volumetric signed distance field or a surfel-based point cloud. This inevitably increases the time and space complexity of the camera tracking method, often making the resulting implementation inefficient on low-end platforms with limited computational resources such as mobile devices.

In this paper, we show that a cost-efficient extension of the simple, drift-prone, frame-to-frame tracking can improve the accuracy of pose estimation markedly. The proposed multi-level pose error correction scheme decreases the estimation errors by re-estimating the camera poses on the fly only when necessary against a few reference frames that are selected adaptively according to the camera motion. It differs from the visual odometry techniques, such as [10], which estimate camera poses with respect to adaptively switched keyframes but without any error correction.

Our method is based on the observation that supplying a better initial guess for the rigid-body motion between two frames results in a numerically more stable cost-function optimization. By using high-quality initial guesses derived from available pose estimates, we show that our error correction scheme produces a significantly enhanced camera trajectory. The extra cost additionally required for the computationally cheap frame-to-frame tracking is quite low in terms of both computation time and memory space, thus enabling an efficient implementation on such low-end platforms as mobile devices. We describe our experience with this technique and demonstrate how it can be applied to developing an effective mobile visual odometry system.

2. Related work

A variant of the frame-to-model tracking algorithm of Newcombe et al. [11] was presented by Bylow et al. [2], where the distances of back-projected 3D points to scene geometry were directly evaluated using the signed distance field. While effective, these frame-to-model tracking approaches needed to manipulate a (truncated) 3D signed distance field during the camera tracking, incurring substantial memory overhead usually on the GPU. To resolve the limitation caused by a fixed, regular 3D volume grid, Roth and Vona [13] and Whelan et al. [20] proposed the use of a dynamically varying regular grid, allowing the camera to move freely in an unbounded extended area. A more memory-efficient, hierarchical volume grid structure was coupled with the GPU-based pipeline of the KinectFusion by Chen et al. [3] in an attempt to support large-scale reconstructions without sacrificing the fine details of the reconstructed 3D models. Nießner et al. [12] employed a spatial hashing technique for the memory-efficient representation of the volumetric truncated signed distance field, which, because of its simple structure, allowed higher frame rates for 3D reconstruction via the GPU. In an effort to achieve higher memory efficiency without trading off the pose estimation accuracy, several surfel-based techniques were also proposed to represent the dynamically fused scene geometry, for example, [9,15,21], for which efficient point manipulation and splatting algorithms were needed.

Whichever base tracking method is used to register captured images, small pose estimation errors eventually accumulate in the trajectory over time, demanding periodic optimizations of the estimated camera trajectory and maps for global consistency, which usually involves the nontrivial computations of keyframe selection, loop-closure detection, and global optimization. Several algorithms suitable for low-cost RGB-D cameras have been proposed. For instance, Endres et al. [5], Henry et al. [7], and Dai et al. [4] extracted sparse image features from the RGB-D measurements to estimate spatial relations between keyframes, from which a pose graph was incrementally built and optimized. A surfel-based matching likelihood measure was explored by Stückler and Behnke [15] to infer spatial relations between the nodes of a pose graph. Whereas, Kerl et al. [10] presented an entropy-based method to both select keyframes and detect loop closures for the pose graph construction. Whelan et al. [19] combined the pose graph optimization framework with non-rigid dense map deformation for efficient map correction. Whelan et al. [21] also proposed a more map-centric dense method for building globally consistent maps by frequent refinement of surfel-based models through non-rigid surface deformation.

In spite of the recent advances in mobile computing technology, implementing the above methods on mobile platforms is still quite challenging, and very few studies addressed this issue. Kähler et al. [8] presented an efficient mobile implementation, based on the voxel block hashing technique [12], achieving high frame rates and tracking accuracy on mobile devices with an IMU sensor.

3. Improved frame-to-frame pose estimation

In this section, we first explain our multi-level adaptive error correction scheme which, as an inherently frame-to-frame method, enables to implement an efficient camera tracking system on mobile devices with limited computing power and memory capability. Consider a live input stream produced by a moving RGB-D camera, where each frame at time i ($i = 0,1,2,\cdots$) provides an augmented image $F_i = (I_i, D_i)$ that consists of an intensity image $I_i(\mathbf{u})$ and a depth map $D_i(\mathbf{u})$, respectively seen through every pixel $\mathbf{u} \in U \subseteq \mathbb{R}^2$. We assume that the intensity image has been aligned to the depth map, although the reverse would also be possible. The basic operation in the frame-to-frame camera tracking is, for two given augmented images F_i and F_j from two different, not necessarily consecutive, time steps i and j ($i > j$), to estimate the rigid transformation $T_{i,j} \in \mathbb{SE}(3)$ that maps the camera coordinate frame at time i to that at

time j by registering F_i against F_j. Once $T_{i,j}$ is known, the ith camera's pose in the *global space* is computed as $T_{i,0} = T_{j,0}T_{i,j}$, where the global space is set to the camera space of the zeroth frame as usual.

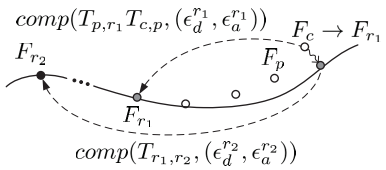
We denote by the function $T_{i,j} = MOTION(F_i, F_j, T_{init})$ the process of estimating the relative rigid body motion from time i to time j through the dense, direct alignment of both intensity and depth images, where T_{init}, an initial guess for $T_{i,j}$, is provided as a function parameter. While any feasible method (such as the one presented in, for instance, [1, 11, 14]) may be applied to implement the motion estimation function, we use a slight variant of the method by Tykkälä et al. [17], in which the weights in the iteratively re-weighted least squares formulation are computed based on the t-distribution as proposed by Kerl et al. [10].

3.1. Basic idea and algorithm

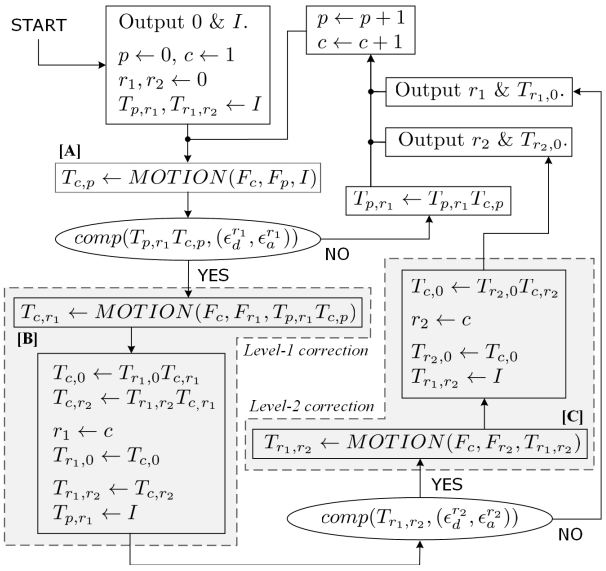
It should be emphasized that selecting a good initial guess for the unknown rigid-body motion greatly influences the accuracy and stability of the iterative pose estimation process. Usually, the camera pose from the last frame is chosen as the initial guess for the current frame (i.e. the identity matrix I is used as the initial guess for the relative motion), there being no obvious better alternative. Combined with any possible deficiency in the captured images, supplying a poor initial value to the optimization algorithm often leads to an inaccurate approximation.

Observe that, despite the use of the null motion as the initial guess, the estimated camera poses for the next few frames will be fairly close to reality. Therefore, they may be utilized to figure out better initial guesses in later rounds of the pose estimation computation to improve the accuracy of the camera poses. Figure 1(a) illustrates the basic idea of our adaptive pose error correction method, where the relative rigid transformation $T_{c,p}$ from the current frame F_c to the previous frame F_p is to be estimated by a function call $MOTION(F_c, F_p, I)$ (see the box **[A]** in Figure 1(b)). Here, F_{r_1} is the frame that has been set the last time as a *(level-one) reference frame*. After each camera pose is estimated for the current frame, it is checked if there has been a significant rigid-body motion since the last reference frame. That is, we investigate if the accumulated rigid transformation $T_{p,r_1}T_{c,p}$ from F_c to F_{r_1} is *large enough* to attempt a pose error correction. This query is done by a boolean function $comp(T_{p,r_1}T_{c,p}, (\varepsilon_d^{r_1}, \varepsilon_a^{r_1}))$ against a given pair of motion thresholds $(\varepsilon_d^{r_1}, \varepsilon_a^{r_1})$, whose implementation shall be explained shortly.

If that is the case, in an effort to reduce the error in the pose estimate of the current frame F_c, we re-estimate it with respect to the reference frame F_{r_1} using the $T_{p,r_1}T_{c,p}$ as an initial guess of the relative rigid-body motion T_{c,r_1} (see the box **[B]** in Figure 1(b)). After the camera pose for F_c is re-estimated, it becomes the next level-one reference frame



(a) Error correction against reference frames



(b) Flowchart of the two-level pose error correction algorithm

Figure 1. Adaptive camera pose correction scheme. Every time the rigid-body motion from the current frame F_c to the last frame F_p is estimated, it is tested if a level-one error correction is necessary by investigating the (approximate) rigid-body motion from F_c to the level-one reference frame F_{r_1} (figure (a)). This adaptive pose error correction idea may easily be extended to a multi-level technique, where a two-level algorithm is summarized in figure (b).

in subsequent frame-to-frame camera tracking. As will be shown later, the effect of the pose error correction with a more elaborate initial guess is evident, tending to refrain the camera from drifting over time from the real trajectory.

In hope to suppress the camera drift as much as possible at little additional expense, we apply another level of pose error correction, in which the new level-one reference frame is further compared to a level-two reference frame. That is, when a current frame becomes a level-one reference frame F_{r_1} after the error correction, the accumulated relative rigid-body motion T_{r_1,r_2} between F_{r_1} and the previously set level-two reference frame F_{r_2} is investigated against looser motion thresholds $(\varepsilon_d^{r_2}, \varepsilon_a^{r_2})$ if there has been sufficient motion. If there is, an additional pose estimation is performed between F_{r_1} and F_{r_2} with the better initial motion matrix T_{r_1,r_2} (see the box **[C]** in Figure 1(b)), hoping to enhance further the accuracy of the estimated pose of F_{r_1}. Once this is done, F_{r_1} also becomes a new level-two reference frame F_{r_2}, and the next round of pose estimation starts. As will

be shown, this simple second-level computation often has an effect of correcting the pose estimation errors between two frames of longer distances, enhancing the overall performance of the camera tracking.

The resulting two-level pose error correction algorithm is shown in Figure 1(b), where the poses of the reference frames ($T_{r_1,0}$ or $T_{r_2,0}$ if a level-two reference frame is found) are output with the respective frame numbers as a result of camera tracking. Whereas the reference frames work as keyframes for improving the pose estimates, our method differs from the conventional keyframe techniques in many ways. Contrary to those methods, our 'memoryless scheme' maintains only the last reference frame, one frame for each correction level, that is updated adaptively during camera tracking. Neither expensive loop-closure detection nor global optimization is performed, but the pose error correction is made at only a small additional cost when it appears to be necessary by comparing the current frame with the reference frames.

Figure 2 gives us a clue to how the idea of adaptive error correction actually works. Although, sometimes, the level-two error correction mechanism ("L2-EC") erroneously pushed the poses away from the exact trajectory, in the adverse situation that the erroneous pose estimated for the last reference frame is assumed correct and the exact pose to be estimated for the next frame is unknown, it effectively suppressed the tendency of the original frame-to-frame tracking method ("Naïve") toward the rapid accumulation of both translational and rotational errors, eventually leading to a significant improvement in the camera tracking result. Also, the error correction had the effect of temporal filtering of the live RGB-D stream so that the frames with sufficiently close camera poses were culled automatically, enabling efficient generation of 3D point set data.

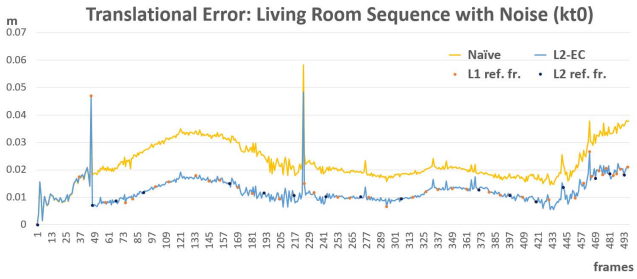


Figure 2. Adaptive pose error correction. The translational errors generated from a living room sequence of the ICL-NUIM benchmark dataset [6] are displayed for the first 495 frames. The adaptively selected reference frames are marked in dots.

3.2. Adaptive selection of reference frames

A simple and effective way to adaptively choose the intervals between the reference frames is to reflect the actual motion of the camera whereby an additional pose error correction is executed if predicted camera pose deviates markedly from that of the last reference frame. Let $T_{c,r} = T(R_{c,r}, \mathbf{t}_{c,r})$ be the accumulated relative rigid transformation from a current frame F_c to the last reference frame F_r, where $R_{c,r} \in \mathbb{SO}(3)$ and $\mathbf{t}_{c,r} \in \mathbb{R}^3$ are the rotational and translational components, respectively. Given distance and angle thresholds ε_d and ε_a, respectively, we regard that an error correction is needed if either the translational distance $||\mathbf{t}_{c,r}||_2$ or the rotation angle $\cos^{-1}\frac{\mathbf{tr}(R_{c,r})-1}{2}$ exceeds the respective threshold, where $\mathbf{tr}(\cdot)$ denotes the trace of a square matrix. Therefore, in our current implementation, the function call $comp(T(R,\mathbf{t}),(\varepsilon_d,\varepsilon_a))$ in Figure 1(b) returns YES if and only if at least one of the following conditions is met: $||\mathbf{t}||_2 > \varepsilon_d$ and $\mathbf{tr}(R) < \varepsilon_a^*$ with $\varepsilon_a^* = 2\cos\varepsilon_a + 1$.

3.3. Error analysis using benchmark datasets

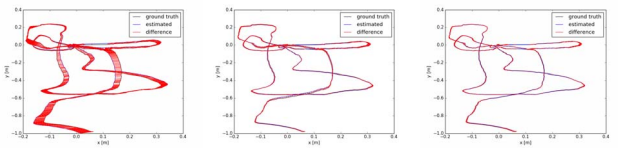
We evaluated the accuracy of the presented approach using two benchmark datasets: the ICL-NUIM dataset [6], which provides RGB-D sequences with and without simulated sensor noise for synthetic scenes, and the TUM dataset [16], which is composed of real RGB-D sequences captured by a Microsoft Kinect sensor. Table 1 summarizes the performance statistics measured on the PC platform where the simple frame-to-frame tracking ("Naïve") was compared to the two enhanced versions ("L1-EC" and "L2-EC") whose pose error correction were applied up to level one and two, respectively. For all these tests, the motion thresholds for our adaptive error correction were set to $(\varepsilon_d^{r_1}, \varepsilon_a^{r_1}) = (30\,mm, 7^\circ)$ for the first level and $(\varepsilon_d^{r_2}, \varepsilon_a^{r_2}) = (120\,mm, 12^\circ)$ for the second. For depth filtering, which will be explained in the next section, the angular thresholds for the normal- and contour-based depth removal were $\varepsilon_n = 28^\circ$ and $\varepsilon_g = 7^\circ$, respectively. Furthermore, the relative gain for the depth component in the cost function was equally set to $\lambda = 1.8$ for the tested sequences.

As clearly indicated by the measurement of the absolute trajectory root-mean-square error metric in meters ("ATE RMSE"), the pose estimation errors of the simple frame-to-frame tracking ("Naïve") decreased significantly when they were adaptively corrected on the fly up to level one ("L1-EC") or level two ("L2-EC"), indicating that the idea of multi-level error correction worked solidly (refer to Figure 3 to see an example of how nicely the errors decreased through error correction). The extra temporal overhead of correcting the pose estimation errors can be revealed by comparing the measures of how many times the basic frame-to-frame registration was performed, i.e. how many times the function $MOTION(\cdot)$ was called. Here, the differences in the numbers between the simple tracking ("Naïve") and the level-one error correction scheme ("L1-EC") show the additional costs, which reveal how many level-one reference frames were chosen during the camera tracking. On

		ICL-NUIM benchmark dataset [6]				TUM benchmark dataset [16]			
		kt0 (1,509)	kt1 (966)	kt2 (881)	kt3 (1,241)	fr1/desk (573)	fr1/desk2 (620)	fr1/xyz (792)	fr1/room (1,352)
Naïve	ATE RMSE (m)	0.1212	0.1714	0.4653	0.1451	0.1526	0.1470	0.0258	0.2157
	motion est.s	1,508	965	880	1,240	572	619	791	1,351
	ref. fr.s (L1/L2)	-	-	-	-	-	-	-	-
L1-EC	ATE RMSE (m)	0.0116	0.1671	0.4564	0.1258	0.0836	0.1117	0.0345	0.2142
	motion est.s	1,812	1,095	1,240	1,534	896	935	1,045	2,125
	ref. fr.s (L1/L2)	304/0	130/0	360/0	294/0	324/0	316/0	254/0	774/0
L2-EC	ATE RMSE (m)	**0.0071** (0.0063)	0.0964 (0.0925)	0.4531 (0.4176)	0.1128 (0.1105)	0.0452 (0.0449)	0.0799 (0.0678)	0.0210 (0.0207)	0.1616 (0.1540)
	motion est.s	1,895	1,111	1,339	1,609	1,123	1,127	1,102	2,402
	ref. fr.s (L1/L2)	305/82	116/30	359/100	292/77	323/288	334/174	258/53	764/287
KinFu [11]		-	-	-	-	0.057	0.420	0.026	0.313
RGB-D [14]		0.3603	0.5951	0.2931	0.8524	0.023	**0.043**	0.014	0.084
ICP [20]		0.0724	**0.0054**	**0.0104**	0.3554	-	-	-	-
ICP+RGB-D [18]		0.3936	0.0214	0.1289	0.8640	-	-	-	-
RGB-D SLAM [5]		0.026	0.008	0.018	0.433	-	-	-	-
MRSMap [15]		0.204	0.228	0.189	1.090	0.043	0.049	0.013	0.069
DVO SLAM [10]		0.104	0.029	0.191	0.152	0.021	0.046	**0.011**	**0.053**
VolumeFusion [19]		-	-	-	-	0.037	0.071	0.017	0.075
ElasticFusion [21]		0.009	0.009	0.014	**0.106**	**0.020**	0.048	**0.011**	0.068

Table 1. Performance statistics for the RGB-D sequences in two benchmark datasets, whose numbers of input frames are given in parentheses. The ATE RMSE results of three variations of our local approach were generated on the PC, and then compared to the state-of-the-art techniques, whose figures were collected from [6], [21], and [10]. The ATE RMSE values in parentheses of the level-two error correction were measured only for the selected reference frames. The total number of frame-to-frame motion estimations applied by each variation ("motion est.s") were counted along with the numbers of selected reference frames at each level ("ref. fr.s (L1/L2)"), which imply the temporal overhead of correcting pose errors. The extra spatial overhead was just to keep one or two last reference frames during camera tracking. Whereas our low-cost method, which guarantees a compact implementation on mobile platforms, dose not handle the jerky camera motion and intractable lighting condition, often found in the TUM benchmark, as effectively as the previous methods, it still provides acceptable camera tracking results for usual smooth camera motions, which will be discussed in the result section.

one hand, the differences with the level-two scheme ("L2-EC") include the extra costs incurred by performing it.



(a) Naïve (b) L1-EC (c) L2-EC

Figure 3. Trajectories on the kt0 sequences assuming noise. Pose errors between our estimate and ground truth are shown in red.

For the four living room sequences with the sensor noise from the ICL-NUIM benchmark (kt0 to kt3), the level-two correction (the level-one correction as well) produced quite competitive and sometimes more accurate results compared with the ATE RMSE results of the state-of-the-art techniques. We find that our solution demanded roughly 1.30 and 1.58 (for the L1 correction scheme) and 1.44 and 1.78 (for the L2 correction scheme) times more tracking computations, respectively, than the simple frame-to-frame tracking, which is quite acceptable considering the increased precision in the pose estimation and no extra need for memory space.

Compared with the ICL-NUIM benchmark dataset, the sequences from the TUM benchmark often included fast and jerky camera motions in addition to sudden changes of lighting conditions on specular surfaces. Therefore, this dataset was quite challenging for frame-to-frame tracking methods like ours because, in some portions of the sequences, the camera moved very fast at a high angular velocity, which often resulted in motion blurs and abrupt view changes in the captured images. The jerky camera movement made both registration between successive frames and error correction between neighboring frames very ineffective. As shown in the table, our approach did not perform better than the compared state-of-the-art techniques in the tested TUM sequences. Despite the unfavorable tracking situations, however, we observe that our method effectively showed the capability of keeping the pose errors to quite

lower levels than those for simple frame-to-frame tracking.

In summary, we observe that our method, which is a local method with low computational burden designed primarily for mobile devices, may not cope with the abrupt camera motions and challenging lighting conditions as effectively as the state-of-the art methods. However, if the user tends to move the camera somewhat carefully, our method adopting the adaptive error correction can also be quite useful for manipulating such usual smooth camera motions, as demonstrated with the sequences from the ICL-NUIM benchmark. More importantly, our method requires basically no spatial overhead because only one last reference frame per correction level needed to be kept during camera tracking. Thus, unlike to those previous methods that perform the frame-to-model tracking and/or some kind of global optimization, demanding nontrivial additional computational resources, our camera tracking scheme guarantees an efficient implementation particularly on mobile platforms.

4. Developing a mobile visual odometry system

4.1. Overall system design

To show the effectiveness of the presented low-cost pose estimation method, we have implemented a visual odometry system that is fully optimized for mobile platforms. Figure 4 illustrates the step-by-step procedure of our camera tracking system in which the adaptive error correction scheme is implemented into the "Adaptive EC" module. To evaluate our mobile system, we generated test datasets by first performing camera tracking on a Lenovo Phab 2 Pro smartphone using the Tango C API and then dumping the captured live RGB-D streams into files.

As shown in the system flowchart, each major task was implemented as an independent parallel thread, which allowed to fully exploit the multithreading capability of the recent mobile processors. Each thread has a dedicated FIFO queue that keeps the frame data from preceding stage. Since the threads were designed to always work if their queues have frames to process, the overall system was run naturally in a parallel pipeline. Note that the life time of each RGB-D frame in input streams differs to each other in our multi-level pose error correction scheme because several different kinds of frames, classified as "current," "previous," "level-one reference," and "level-two reference," and so on, may exist in the system at the same time. So we had to develop an efficient way of managing the collection of valid image data. In our system, the "Frame Pool" module was designed to keep a reference count for each frame in the storage, indicating the number of parallel threads that need the corresponding frame. Then, by updating the reference counts when needed and periodically examining them, it was possible to safely remove unnecessary frames from the system.

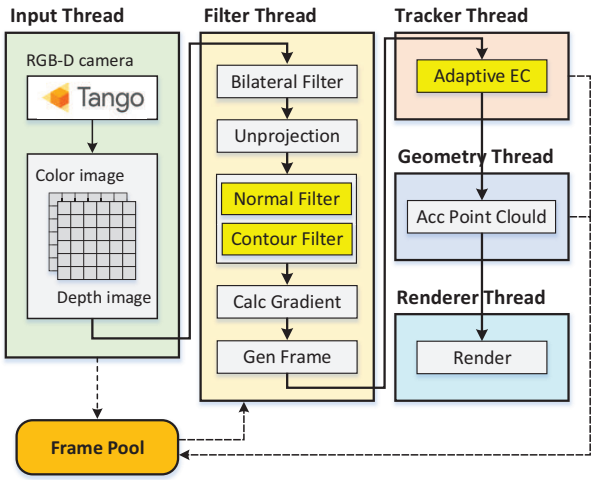
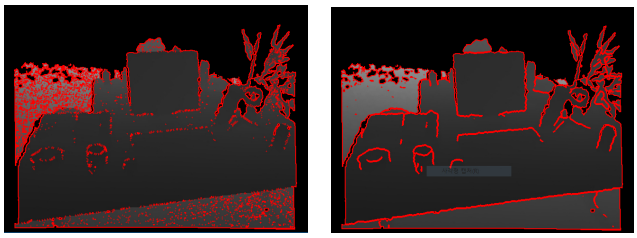


Figure 4. System architecture of our mobile visual odometry system. The "Adaptive EC" module is the core component that performs the 6-DOF pose estimation based on the presented multi-level pose error correction scheme. In addition to the conventional bilateral filter, we also included two depth filters in the "Normal Filter" and "Contour Filter" modules, respectively, to enhance the stability of the mobile camera tracking system.

4.2. Removal of unreliable depth values

Despite the bilateral filtering on perceived raw depth data, the resulting depth map $\bar{D}_i(\mathbf{u})$ usually contains faulty depth values which often undermine significantly the robustness of the pose estimation. Thus we applied two extra depth value removal steps in the filtering stage, respectively called *normal filtering* and *contour filtering*, that aim to detect possibly troublesome pixels in the depth image and remove them from consideration in the pose estimation (and surface reconstruction) (see Figure 4 again). First, we observed that the reliability in the normal vector $\mathbf{n}_i(\mathbf{u})$ at $\mathbf{p}_i(\mathbf{u})$, estimated from $\bar{D}_i(\mathbf{u})$ by applying a divided-difference scheme to (noise-prone) back-projected points, is often a good indicator of confidence in the captured depth values around the pixel $\mathbf{u}$. To check this, two unit normal approximations were computed in two different divided-difference directions: $\mathbf{n}_i^p(\mathbf{u})$ in the principal axis direction and $\mathbf{n}_i^d(\mathbf{u})$ in the diagonal direction. Then, if the angle between $\mathbf{n}_i^p(\mathbf{u})$ and $\mathbf{n}_i^d(\mathbf{u})$ was greater than a given angular threshold ε_n, we assumed that the captured depth values around $\mathbf{u}$ are not sufficiently dependable, and culled out the depth pixel from the subsequent computation (see Figure 5(a)). For a surviving pixel $\mathbf{u}$, their average direction was used as $\mathbf{n}_i(\mathbf{u})$, enabling the use of more reliable, smoothed normal vectors in the point cloud generation.

The second depth value removal technique is to get rid of those pixels around which depth values change suddenly. When a pixel is transformed during the pose estimation from source to target frame through a warping process, the



(a) Normal-based removal (b) Contour-based removal

Figure 5. The effects of two extra depth filters. Figures (a) and (b) show depth images of a frame, where unreliable pixels that were removed as a result of the respective filters are colored in red.

inevitable roundoff errors cause the resulting pixel location to be moved, which, although slight, may entail a significant change in the depth value. In order to decrease the possibility of incorrect depth values to be supplied in the optimization computation, we also removed the possibly problematic pixels in the areas with high depth gradient magnitude. In this process, instead of approximating the depth gradient in the 2D pixel space, we found out through experiments that estimating the depth change in the 3D camera space is more intuitive and effective in culling out such pixels: for each pixel $\mathbf{u} = (u_x, u_y)$, if the angle made by the vector $(\mathbf{p}_i(\mathbf{u}_{-1,0}) - \mathbf{p}_i(\mathbf{u})) \times (\mathbf{p}_i(\mathbf{u}_{0,-1}) - \mathbf{p}_i(\mathbf{u}))$ and the view vector $\mathbf{p}_i(\mathbf{u})$ is in the range of $[90° - \varepsilon_g, 90° + \varepsilon_g]$ for another angular threshold ε_g, it was assumed that the depth value undergoes a drastic depth transition at $\mathbf{u}$, and, for a safety reason, the pixels in the 3×3 pixel area centered at $\mathbf{u}$ were removed from the subsequent process (see Figure 5(b)).

5. Results

To evaluate how effectively the adaptive error correction scheme combines with frame-to-frame tracking on mobile platforms, the visual odometry system was implemented on two smartphones: the Lenovo Phab 2 Pro using the Qualcomm Snapdragon 652 chipset with the Qualcomm Adreno 510 GPU, and the Samsung Galaxy S8 using the Samsung Exynos 8895 chipset with the ARM Mali-G71 MP20 GPU. As a Google Tango-enabled device, the Lenovo smartphone has a depth sensor, allowing the generation of live RGB-D streams at 320×180 pixels.

We first tested the mobile version with the ICL-NUIM and TUM benchmarks and confirmed that the ATE RMSE results behaved quite similar to those produced on the PC, trying to keep the pose errors to levels much lower than those from the simple frame-to-frame tracking. However, the achieved frame rates were quite low for those streams of 640×480 images. Then, we generated several RGB-D sequences of 320×180 images on the Lenovo phone using the C API provided by the Google Tango SDK where, to enable precise comparison between different methods, each sequence was saved in a file. For these real-scene datasets, a proper application of the additional depth removal filters was especially important because the mobile depth sensor usually generated less accurate depth maps.

Figures 6 (a) to (d) show four RGB-D sequences whose runtime performance is reported in Table 2. In this test, the iterative pose estimation process proceeded in multi-scale fashion, in which the RGB-D images formed by sampling every other eight, four, and then two pixels were used to progressively refine estimated camera poses. Overall, our visual odometry system took about 25 to 30 ms per input frame on average for camera tracking on the Samsung phone when the level-two error correction mechanism was applied. The figures in the "motion est.s" rows give a clue to the extra overhead of correcting the pose estimation errors produced by a simple frame-to-frame tracking. We find that our solution demanded roughly 1.11 to 1.27 times more tracking computations on average than the faulty naïve frame-to-frame tracking, which is quite acceptable considering the increased precision in the pose estimation. As clearly implied by the timings in the last three rows in Table 2, the times taken by the two parallel, filtering and tracking threads were effectively overlapped through the time steps in the pipelined architecture of our camera tracking system (please compare their sums to the total times).

Note that the size of the point sets created on the fly as a result of 3D reconstruction ("generated pt.s") grew progres-

		Venus (500)	Table (170)	Office (219)	Room (400)
Naïve	motion est.s	499	169	218	399
	ref. fr.s(L1/L2)	-	-	-	-
	Tracker	17.39 ms	16.64 ms	17.70 ms	18.36 ms
	generated pt.s	14,073 K	3,982 K	7,395 K	13,712 K
L1-EC	motion est.s	604	248	322	631
	ref. fr.s(L1/L2)	105/0	79/0	104/0	232/0
	Tracker	18.81 ms	18.32 ms	20.43 ms	19.78 ms
	generated pt.s	2,994 K	1,875 K	3,519 K	7,890 K
L2-EC	motion est.s	637	282	375	741
	ref. fr.s(L1/L2)	105/33	79/34	109/48	234/109
	Tracker	19.39 ms	18.48 ms	21.64 ms	23.30 ms
	generated pt.s	3,016 K	1,875 K	3,641 K	7,970 K
L2-EC	Filter	22.49 ms	19.25 ms	21.01 ms	22.65 ms
	Tracker	19.39 ms	18.48 ms	21.64 ms	23.30 ms
	Total	**28.20 ms**	**24.51 ms**	**25.89 ms**	**30.69 ms**

Table 2. Performance statistics on the Samsung Galaxy S8 smartphone. The applied control parameters were $\varepsilon_n = 45°$, $\varepsilon_g = 20°$, $(\varepsilon_d^{r_1}, \varepsilon_a^{r_1}) = (30\,\text{mm}, 3°)$, $(\varepsilon_d^{r_2}, \varepsilon_a^{r_2}) = (70\,\text{mm}, 7°)$, and $\lambda = 1.8$. "Filter" and "Tracker" respectively indicate the average times per frame taken by the corresponding threads in our visual odometry system illustrated in Figure 4. Also, "Total" shows the total average times per frame, excluding the rendering time which varied with the number of accumulated points. All computations except for the Filter thread, which was accelerated using the OpenCL API, were done by the mobile CPU alone without GPU assistance.

(a) Venus (b) Table (c) Office (d) Room

(e) "Naïve" (f) "L2-EC" (g) "Naïve" (h) "L2-EC"

(i) Tango API (j) Ours ("L2-EC") (k) Tango API (l) Ours ("L2-EC")

Figure 6. Camera tracking on the mobile platform. Figures (a) to (d) show four test sequences captured by a Lenovo Phab 2 Pro smartphone, which generated RGB-D streams at 320×180 pixels. Figures (e) and (f), and (g) and (h) respectively compare parts of reconstructed surfaces, represented in point sets, where the 3D reconstruction quality markedly improved via our adaptive pose error correction scheme. Figures (i) and (j), and (k) and (l) also compare parts of the surfaces by our method with those produced on the Lenovo Phab 2 Pro smartphone using the Tango C API. A direct comparison of the pose estimation times between these two methods was not possible because we could not directly call the Tango C API function to initiate the pose estimation process against the RGB-D streams loaded from files.

sively; this is often burdensome for mobile devices to handle. Often, the mobile phone crashed due to excessive memory usage if the frame-to-frame tracking method without error correction ("Naïve") tried to collect all back-projected 3D points. (Therefore, the point sizes had to be measured on the PC for some troublesome cases.) Our approaches ("L1-EC"/"L2-EC") were able to automatically filter dense live RGB-D streams via adaptively selected reference frames, preventing the point set from growing excessively.

Figures 6 (e) to (h) offer qualitative comparisons between the simple and enhanced tracking techniques. In these typical examples, we find that the 3D reconstruction quality was markedly improved in several regions around which the simple frame-to-frame tracking produced poor reconstruction. Figures 6 (i) to (l) also compare the point sets generated by our method to those generated using the standard method from the Google Tango SDK. In general, the camera tracking method performed a stable pose estimation on the Lenovo phone. However, we observe that our adaptive error correction technique often generated point clouds that are visually more detailed as shown in these figures.

6. Conclusion

In this paper, we presented an adaptive pose error correction scheme for tracking a low-cost RGB-D camera, and showed that it can be used to significantly enhance the accuracy of the drift-prone frame-to-frame camera tracking method without seriously harming the computational efficiency. The proposed camera tracking model is simple and efficient enough to allow a compact implementation on mobile devices, and was demonstrated to produce quite acceptable tracking results on the tested smartphones, especially for usual smooth camera motions.

Acknowledgements

This research was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (No. NRF-2017R1D1A1B03029625).

References

[1] C. Audras, A. Comport, M. Meilland, and P. Rives. Real-time dense appearance-based SLAM for RGB-D sensors. In *Aust. Conf. on Robotics and Automation (ACRA)*, 2011.

[2] E. Bylow, J. Sturm, C. Kerl, F. Kahl, and D. Cremers. Real-time camera tracking and 3D reconstruction using signed distance functions. In *Robotics: Sci. & Syst. (RSS)*, 2013.

[3] J. Chen, D. Bautembach, and S. Izadi. Scalable real-time volumetric surface reconstruction. *ACM Trans. Graph. (ACM SIGGRAPH 2013)*, 32(4):Article No. 113, 2013.

[4] A. Dai, M. Nießner, M. Zollöfer, S. Izadi, and C. Theobalt. BundleFusion: real-time globally consistent 3D reconstruction using on-the-fly surface re-integration. *ACM Trans. Graph. (TOG)*, 2017.

[5] F. Endres, J. Hess, N. Engelhard, J. Sturm, D. Cremers, and W. Burgard. An evalution of the RGB-D SLAM system. In *2012 IEEE Int. Conf. Robotics Automation (ICRA)*, pages 1691–1696, 2012.

[6] A. Handa, T. Whelan, J. McDonald, and A. Davison. A benchmark for RGB-D visual odometry, 3D reconstruction and SLAM. In *IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2014.

[7] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox. RGB-D mapping: using Kinect-style depth cameras for dense 3D modeling of indoor environments. *Int. J. Robotics Research*, 31(5):647–663, 2012.

[8] O. Kähler, V. Prisacariu, C. Ren, X. Sun, and D. Torr, P. Murray. Very high frame rate volumetric integration of depth images on mobile devices. *IEEE Trans. Vis. Comput. Graphics*, 21(11):1241–1250, 2015.

[9] M. Keller, D. Lefloch, M. Lambers, S. Izadi, T. Weyrich, and A. Kolb. Real-time 3D reconstruction in dynamic scenes using point-based fusion. In *Proc. Int. Conf. on 3D Vision (3DV)*, pages 1–8, 2013.

[10] C. Kerl, J. Sturm, and D. Cremers. Dense visual SLAM for RGB-D cameras. In *2013 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 2100–2106, 2013.

[11] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohli, J. Shotton, S. Hodges, and A. Fitzgibbon. KinectFusion: Real-time dense surface mapping and tracking. In *Proc. 10th IEEE Int. Symp. Mixed Augmented Reality*, pages 127–136, 2011.

[12] M. Nießner, M. Zollhöfer, S. Izadi, and M. Stamminger. Real-time 3D reconstruction at scale using voxel hashing. *ACM Trans. Graph. (ACM SIGGRAPH Asia 2013)*, 32(6):Article No. 169, 2013.

[13] H. Roth and M. Vona. Moving volume KinectFusion. In *Proc. British Machine Vision Conf.*, pages 112.1–112.11, 2012.

[14] F. Steinbrücker, J. Sturm, and D. Cremers. Real-time visual odometry from dense RGB-D images. In *IEEE Intl. Conf. on Computer Vision Workshops (ICCV Workshops)*, pages 719–722, 2011.

[15] J. Stückler and S. Behnke. Integrating depth and color cues for dense multi-resolution scene mapping using RGB-D cameras. In *IEEE Intl. Conf. on Multisensor Fusion and Information Integration (MFI)*, 2012.

[16] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A benchmark for the evaluation of RGB-D SLAM systems. In *Intl. Conf. on Intelligent Robot Systems (IROS)*, 2012.

[17] T. Tykkälä, C. Audras, and A. Comport. Direct iterative closest point for real-time visual odometry. In *IEEE Intl. Conf. on Computer Vision Workshops (ICCV Workshops)*, pages 2050–2056, 2011.

[18] T. Whelan, H. Johannsson, M. Kaess, J. Leonard, and J. McDonald. Robust real-time visual odometry for dense RGB-D mapping. In *2013 IEEE Int. Conf. Robotics and Automation*, pages 5724–5731, 2013.

[19] T. Whelan, M. Kaess, H. Johannsson, M. Fallon, J. Leonard, and J. Mcdonald. Real-time large-scale dense RGB-D SLAM with volumetric fusion. *Int. J. Robotics Research*, 34(4-5):598–626, 2015.

[20] T. Whelan, J. McDonald, M. Kaess, M. Fallon, H. Johannsson, and J. Leonard. Kintinuous: Spatially extended KinectFusion. In *Proc. RSS Workshop on RGB-D: Advanced Reasoning with Depth Cameras*, 2012.

[21] T. Whelan, R. Salas-Moreno, B. Glocker, J. Davison, and S. Leutenegger. ElasticFusion: Real-time dense SLAM and light source estimation. *The Intl. J. of Robotics Research*, 35(14):1697–1716, 2016.