Semi-Dense Visual Odometry for AR on a Smartphone

Thomas Schöps* Jakob Engel† Daniel Cremers‡

Technische Universität München

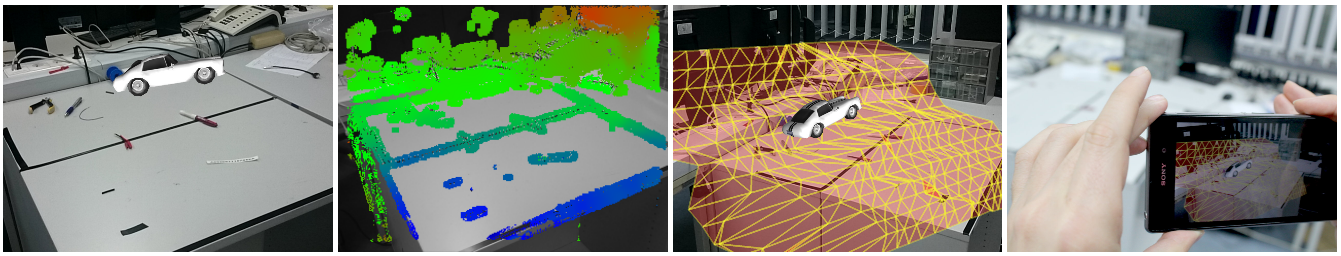

Figure 1: From left to right: AR demo application with simulated car. Corresponding estimated semi-dense depth map. Estimated dense collision mesh, fixed and shown from a different perspective. Photo of running system. The attached video shows the system in action.

Abstract

We present a direct monocular visual odometry system which runs in real-time on a smartphone. Being a direct method, it tracks and maps on the images themselves instead of extracted features such as keypoints. New images are tracked using direct image alignment, while geometry is represented in the form of a semi-dense depth map. Depth is estimated by filtering over many small-baseline, pixel-wise stereo comparisons. This leads to significantly less outliers and allows to map and use all image regions with sufficient gradient, including edges. We show how a simple world model for AR applications can be derived from semi-dense depth maps, and demonstrate the practical applicability in the context of an AR application in which simulated objects can collide with real geometry.

Keywords: Semi-Dense, Direct Visual Odometry, Tracking, Mapping, AR, Mobile Devices, 3D Reconstruction, NEON

1 Introduction

Estimating the movement of a monocular camera and the 3D structure of the environment is amongst the most prominent challenges in computer vision. Commonly referred to as monocular SLAM or structure from motion, it is a key enabler for many augmented reality applications: only if the precise pose of the camera is available in real-time, virtual objects can be rendered into the scene as if they were part of it. Further, knowledge about the geometry of the scene allows virtual objects to interact with it: in an augmented reality game, game characters can collide with, be occluded by or be placed on top of real obstacles. To assist with furnishing or redecorating a room, a piece of furniture could be reconstructed from a video taken by a smartphone, and virtually rendered into different locations in the room. Figure 1 shows an example AR application realized on top of our direct Visual Odometry (VO) system.

Apart from marker based methods [24, 23, 6] – which allow for precise and fast camera pose estimation at the cost of having to manually place one or more physical markers into the scene – state-of-the-art monocular SLAM methods generally operate on features. While this allows to estimate the camera movement in real-time on mobile platforms [12, 15], the resulting feature based maps hardly provide sufficient information about the 3D geometry of the scene for physical interaction.

At the same time, recent advances in computer vision have shown the high potential of *direct* methods for monocular SLAM [16, 5, 7, 17]: instead of operating on features, these methods perform both tracking and mapping directly on the image intensity values. Fundamentally different from feature based methods, direct methods not only allow for fast, sub-pixel accurate camera tracking, but also provide substantially more information about the 3D structure of the environment, are less susceptible to outliers, and more robust in environments with little texture [5].

1.1 Related Work

In this section we give an overview over existing monocular SLAM and VO methods, divided into *feature based* and *direct* methods. While there exists a large number of feature based methods for mobile phones, existing direct methods are computationally expensive and require a powerful GPU to run in real-time. Figure 2 summarizes the main differences between feature based and direct methods.

Feature Based. The basic idea behind features is to split the overall problem – estimating geometric information from images – into two separate, sequential steps: First, a set of feature observations is extracted from the image, typically independently of one another. This can be done using a large variety of methods, including different corner detectors and descriptors, as well as fast matching methods and outlier detection schemes like RANSAC. Second, camera position and scene geometry are computed as a function of these feature observations only. Again, there exists a variety of methods to do this, including bundle adjustment based approaches [11] or filtering based approaches [3, 14].

While decoupling image based (photometric) estimation from subsequent geometric estimation simplifies the overall problem, it comes with an important limitation: *Only information that conforms to the feature type and parametrization can be used.* In particular, when using keypoints, information contained in edges is discarded.

Today, there are several keypoint based monocular VO and SLAM methods which run in real-time on mobile devices [14, 12]. In order to obtain a denser 3D reconstruction, one approach is to

*e-mail: schoepst@in.tum.de
†e-mail: engelj@in.tum.de
‡e-mail: cremers@tum.de

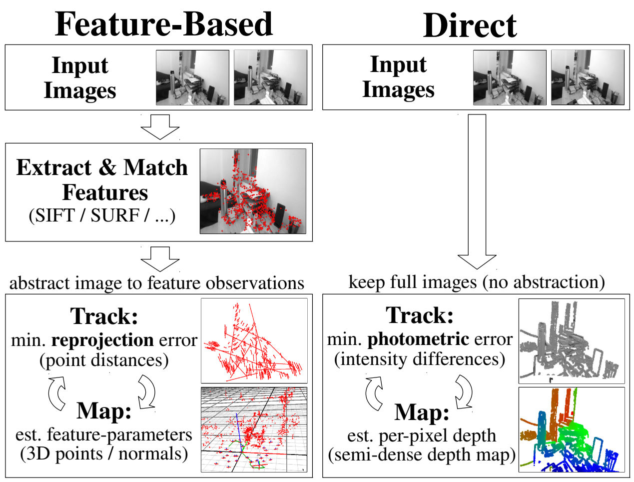

Figure 2: Feature based methods abstract images to feature observations and discard all other information. In contrast, the proposed direct approach maps and tracks directly on image intensities: this allows to (1) use all information, including e.g. edges and (2) directly obtain rich, semi-dense information about the geometry of the scene.

perform two-frame or multi-frame stereo on selected frames, where the camera pose is obtained from a full feature based SLAM system running in the background [22]. However – even though the computed dense depth maps are often more accurate and precise than the feature based map, they cannot directly be fed back into the SLAM system running in the background, thereby discarding valuable information.

Direct. Direct approaches circumvent these limitations by directly optimizing the camera poses and scene geometry on the raw images. *This allows to use all information in the image*, leading to higher accuracy and robustness in particular in indoor environments with only few features. Early direct or semi-direct approaches were based on scene representations by sets of planar patches: [19] presents such a system which simultaneously estimates the motion, scene structure and illumination. [8] also combines tracking and reconstruction and especially discusses local optimums in the error function.

Images are tracked by direct minimization of the per-pixel photometric error (see Sec. 2.1), which is well established for tracking RGB-D or stereo sensors [10, 1]. In a monocular setting, the required per-pixel depth values are in turn computed from stereo on previous frames: in [5], a pixel-wise filtering formulation was proposed, which fuses information from many small-baseline stereo comparisons. This approach allows to obtain accurate and precise semi-dense depth maps in real-time on a CPU. It has recently been extended to LSD-SLAM, a large-scale direct monocular SLAM system [4] including loop-closures. Another approach is to compute fully dense depth maps using a variational formulation [20, 16, 17], which however is computationally demanding and requires a state-of-the-art GPU to run in real-time. A common feature of direct methods is their inherent parallelism: as many operations are defined on a per-pixel basis, they can ideally be parallelized using GPGPUs or SIMD (Single Instruction Multiple Data) instructions, achieving considerable speed-ups in practice.

1.2 Contributions and Outline

In this paper we present a direct monocular VO system based on [5] which runs in real-time on a smartphone. In addition to accurately computing the camera pose at over 30 Hz, the proposed method provides rich information about the environment in the form of a

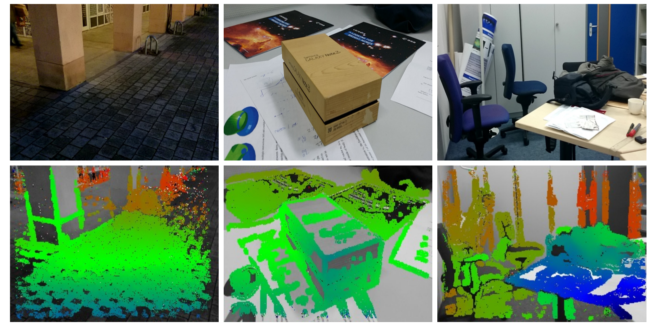

Figure 3: Examples of semi-dense depth maps estimated in real-time on a smartphone. See also the attached video.

semi-dense depth map of the currently visible scene. In particular we (1) describe modifications required to run the algorithm in real-time on a smartphone, and (2) propose a method to derive a dense world model suitable for basic physical interaction of simulated objects with the real world. We demonstrate the capabilities of the proposed approach with a simple augmented reality game, in which a simulated car drives through the environment, and can collide with real obstacles in the scene.

The paper is organized as follows: We describe the proposed semi-dense, direct VO method in Sec. 2. In particular, in Sec. 2.3, we describe the steps required for real-time performance on a smartphone. Following this, we show how collision meshes are computed from the semi-dense depth map in Sec. 3, and how they are used in a simple AR application. Finally, we qualitatively evaluate the resulting system in different environments in Sec. 4.

2 Semi-Dense Direct Visual Odometry

The proposed monocular VO algorithm does not use features at any stage of the algorithm, but instead directly operates on the raw intensity images: The map is represented as a *semi-dense inverse depth map*, which contains a Gaussian probability distribution (mean and variance) for the inverse depth of a subset of pixels, hence "semi-dense". An example is shown in Fig. 3: pixels that have a depth hypothesis are shown in color (encoding the depth).

The whole system is divided into two parts (see Fig. 4), running in parallel: **tracking** and **mapping**. In Sec. 2.1, we describe tracking using direct image alignment. In Sec. 2.2, we present the mapping part which simultaneously estimates and propagates the depth map. The system is closely based on the approach by Engel et al. [5] for real-time operation on a consumer laptop.

Notation. We represent an image as function $I\colon \Omega \to \mathbb{R}$. Similarly, we represent the inverse depth map and inverse depth variance map as functions $D\colon \Omega_D \to \mathbb{R}^+$ and $V\colon \Omega_D \to \mathbb{R}^+$, where Ω_D contains all pixels which have a valid depth hypothesis. Note that D and V denote mean and variance of the *inverse* depth, as this approximates the uncertainty of stereo much better than assuming a Gaussian-distributed depth.

Initialization. We initialize the map with random depth values and large variance for the first frame. When moving the camera slowly and in parallel to the image plane, the algorithm (running normally) typically locks onto a consistent depth configuration and quickly converges to a valid map. This is in contrast to [5], in which a keypoint based initializer was used. While the process is successful in most cases, we observed one distinct failure case which results in an inverse estimate of both the depth map and the camera motion, which is further discussed in [8]. A numerical evaluation of the initialization success and convergence rate is given in Sec. 4.

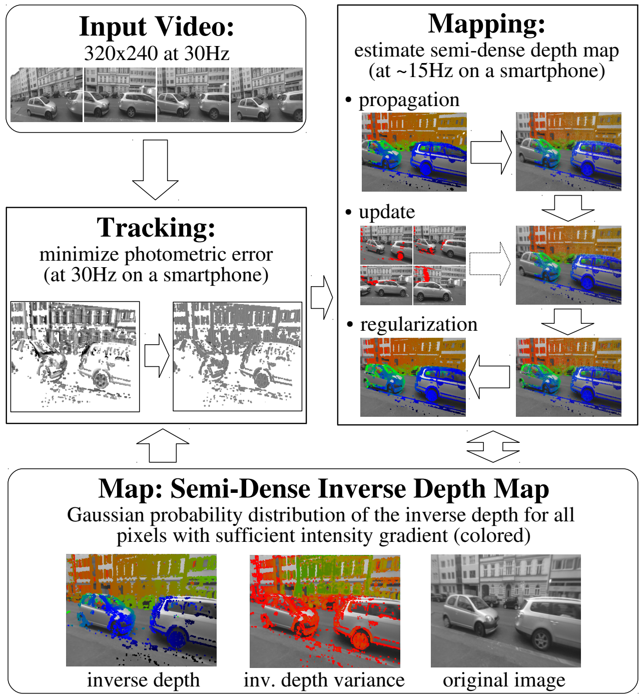

Figure 4: Semi-Dense Visual Odometry: Tracking and mapping are performed in parallel, operating on a semi-dense inverse depth map as central data structure. It contains an inverse depth hypothesis for all pixels close to sufficiently strong image gradient. It is continuously propagated to new frames, updated with new stereo observations and spatially regularized. More details on the respective steps are given in Sec. 2.

2.1 Tracking

The pose of new frames is estimated using direct image alignment: given the current map $\{I_M, D_M, V_M\}$, the relative pose $\boldsymbol{\xi} \in \mathrm{SE}(3)$ of a new frame I is obtained by directly minimizing the photometric error

$$E(\boldsymbol{\xi}) := \sum_{\mathbf{x} \in \Omega_{D_M}} \left\| I_M(\mathbf{x}) - I(\omega(\mathbf{x}, D_M(\mathbf{x}), \boldsymbol{\xi})) \right\|_\delta , \qquad (1)$$

where $\omega\colon \Omega_{D_M} \times \mathbb{R} \times \mathrm{SE}(3) \to \Omega$ projects a point from the reference image into the new frame, and $\| \cdot \|_\delta$ is the Huber norm to account for outliers. Global brightness changes due to auto-shutter typically have little effect, as we only use image regions with strong gradient. The minimum is computed using iteratively re-weighted Levenberg-Marquardt minimization, as described in [4].

Image Pyramid. To handle larger inter-frame motions, we use a pyramid scheme: Each new frame is first tracked on a very low resolution image and depth map, the tracked pose is then used as initialization for the next higher resolution. Depth maps are down-sampled by factors of two, using a weighted average of the *inverse* depth. To account for the strong correlation between neighbouring pixels, we average the information (inverse variance), giving

$$D_{l+1}(\mathbf{x}) := \frac{\sum_{\mathbf{x}' \in \Omega_\mathbf{x}} \frac{D_l(\mathbf{x}')}{V_l(\mathbf{x}')}}{\sum_{\mathbf{x}' \in \Omega_\mathbf{x}} \frac{1}{V_l(\mathbf{x}')}} \qquad (2)$$

$$V_{l+1}(\mathbf{x}) := \frac{|\Omega_\mathbf{x}|}{\sum_{\mathbf{x}' \in \Omega_\mathbf{x}} \frac{1}{V_l(\mathbf{x}')}} \qquad (3)$$

where l is the index of the pyramid level. $\Omega_\mathbf{x}$ denotes the set of valid pixels (i.e. with depth value) contained in pixel $\mathbf{x}$ at the next higher resolution.

Note that a pixel at low resolution has an associated depth hypothesis if at least one of the contained high-resolution pixels has a depth hypothesis. This strategy automatically lets the semi-dense depth maps become denser on lower resolutions (see Fig. 5), leading to more robust low-resolution tracking results without affecting the accuracy of the final result on the highest resolution. Averaging the inverse depth effectively averages the optical flow, causing this strategy to work well for minimizing the photometric error (1). If used for reconstruction purposes however, it will create undesired points between the front and back surface around depth discontinuities.

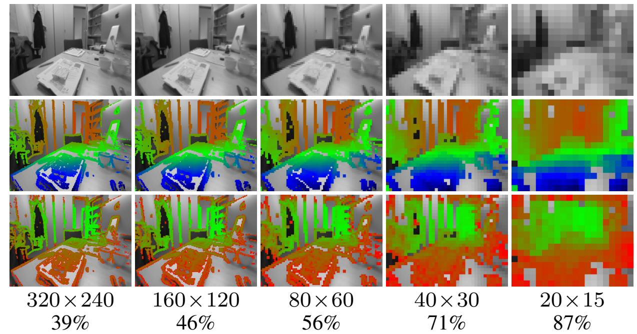

Figure 5: Image Pyramid. Top: intensity image I_l, middle: color-coded inverse depth D_l, bottom: inverse depth variance V_l. The given percentage corresponds to the density, i.e., the percentage of pixels that have a depth value. The described down-sampling strategy causes the depth maps to become significantly denser on higher pyramid levels.

2.2 Mapping

Depth maps are estimated by filtering over small-baseline pixel-wise stereo comparisons, interleaved with spatial regularization and propagation to a new frame, as first proposed in [5]. Each mapping iteration consists of three main steps:

1. **Propagation**: Depth hypotheses are projected into the most recently tracked frame, giving an initialization for the new depth map (prediction in an extended Kalman filter (EKF)).

2. **Update**: New depth measurements are obtained from a large number of pixel-wise stereo comparisons with previous frames, and merged into the existing depth map by filtering (observation in an EKF). We use the propagated prior hypothesis to constrain the search interval, greatly accelerating the search and reducing the probability for false observations in repetitive image regions.

   Stereo is only performed for a subset of suitable pixels, that is pixels where the expected accuracy is sufficiently high. This depends on the intensity gradient at that point as well as the camera motion, and is efficiently determined as proposed in [5]. In particular, regions with little image gradient are never updated as no accurate stereo measurements can be obtained. Ω_D contains all pixels that have a depth hypothesis, either propagated from previous frames, or observed in that frame.

3. **Regularization**: In a last step, the depth map is spatially regularized and outliers are removed.

Mapping runs in a continuous loop, in each iteration propagating the depth map to the most recently tracked frame, potentially skipping some frames. The runtime of one iteration varies in practice, as it depends on the density of the current depth map and the camera motion; an experimental runtime evaluation is given in Sec. 4.

2.3 Implementation on Mobile Phones

Current smartphone cameras have a rolling shutter, which introduces systematic distortions and, during quick motion, can have strong effects on the accuracy of stereo observations. While there exist methods to correctly model this in an off-line reconstruction setting [18], or to approximate it in real-time [9, 13], we found that ignoring the rolling shutter still gives very good results in practice, and significantly saves computational time.

All experiments are conducted on a Sony Xperia Z1, which is equipped with a 2.3 GHz quad-core CPU. While the processing power of mobile devices has increased rapidly in the last years, mobile processors based on the ARM architecture are generally still much slower than their desktop counterparts; we currently do not use GPU or DSP features for tracking or mapping. In order to achieve real-time performance, i.e. tracking with at least 30 fps under these conditions, two steps were crucial: (1) separation of mapping and tracking resolution and choice of a suitable compromise, and (2) NEON optimization of computation-heavy algorithmic steps.

Image Resolution. While current desktop CPUs easily allow for real-time operation at VGA resolution (640×480), our mobile implementation performs mapping at 320×240. As tracking performance is crucial for a smooth AR experience, we further reduce the maximum resolution used for tracking down to 160×120. While this greatly reduces the computation time, the effect on accuracy is relatively small (see Sec. 4). This can be explained by the sub-pixel accuracy of direct image alignment: In practice, inaccuracies from motion blur, rolling shutter and other model violations (e.g. reflections, occlusions, specular highlights, etc.) dominate the error.

NEON Parallelization. Many parts of the tracking stage are well suited for optimization using SIMD parallelization. We use NEON instructions, which offer this functionality on ARM processors, leading to greatly improved performance and thereby being a vital step in achieving real-time performance on mobile processors. There are two algorithmic steps in tracking which particularly benefit from NEON optimization:

(1) Calculating the approximated Hessian $\mathbf{H}$ and gradient $\mathbf{g}$ of the error required for building the linear system to compute the pose increment $\boldsymbol{\Delta\xi}$, that is

$$\underbrace{\sum_{\mathbf{x}\in\Omega_D} \mathbf{J}_\mathbf{x}^T\mathbf{J}_\mathbf{x} w_\mathbf{x}}_{\mathbf{H}} \cdot \boldsymbol{\Delta\xi} = \underbrace{\sum_{\mathbf{x}\in\Omega_D} \mathbf{J}_\mathbf{x}^T r_\mathbf{x} w_\mathbf{x}}_{\mathbf{g}}, \tag{4}$$

where $r_\mathbf{x}$, $\mathbf{J}_\mathbf{x}$ and $w_\mathbf{x}$ are, for one pixel $\mathbf{x}$, the pixel's residual, its Jacobian with respect to the pose update, and the computed Huber weight. Using NEON optimization, four elements in the sum – which goes over all pixels which have depth – can be processed at once, resulting in a significant speed-up.

(2) Calculating the weights and residual sum: again, four pixels can be processed at the same time. In addition, NEON offers fast inverse approximations, which help to reduce processing time.

Both the above steps are required in every Levenberg-Marquardt iteration, thereby making up a large part of overall tracking performance. Details to the runtime of our implementation, with and without NEON acceleration, are given in Sec. 4.

3 Augmented Reality Application

We demonstrate a simple AR game using the computed semi-dense depth maps, in which a simulated car can be driven through the environment. For this, we construct a low-resolution collision mesh from the semi-dense depth map, which is used for real-time physics simulation with the free Bullet library [2].

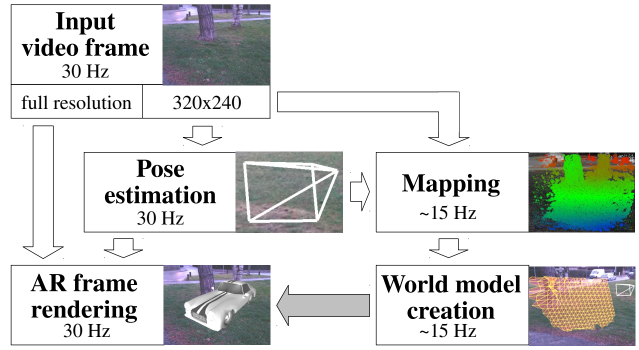

Figure 7: AR Processing Pipeline: Camera images are retrieved and displayed directly at full resolution. Simultaneously, a downsampled (320×240) version is computed and processed in the VO pipeline to allow real-time tracking and mapping. The estimated pose of that frame, as well as the generated world model are then used to render virtual objects into the scene. The world model is updated asynchronously at a lower frequency. All components run in parallel.

For this, we assume that the scene has a well-defined ground plane, which we estimate with the help of the IMU. The full processing pipeline for augmented reality is shown in Fig. 7.

3.1 Collision Mesh Generation

We first compute a fully dense low-resolution (15×20) depth map using a variational in-painting approach. As data term for valid pixels we use the hypothesis from the corresponding level of the semi-dense depth map. Additionally, to cover up large unconstrained regions, we assume that pixels that do not have a depth hypothesis lie on the estimated ground plane π. As regularizer we use the Huber norm

$$\|\mathbf{x}\|_\delta := \begin{cases} \frac{\|\mathbf{x}\|_2^2}{2\delta} & \text{if } \|\mathbf{x}\|_2 < \delta \\ \|\mathbf{x}\|_1 - \frac{\delta}{2} & \text{otherwise} \end{cases} \tag{5}$$

of the inverse depth gradient. The Huber norm is a combination of a quadratic regularizer favouring smooth surfaces, and the total variation (TV), which allows sharp transitions at occluding edges. The combined energy to be minimized with respect to the resulting inverse depth map u is hence given by

$$\begin{aligned} E(u) :=& \int_{\Omega_D} \frac{(u(\mathbf{x}) - D(\mathbf{x}))^2}{V(\mathbf{x})} d\mathbf{x} \\ &+ \int_{\Omega\setminus\Omega_D} \frac{(u(\mathbf{x}) - \pi(\mathbf{x}))^2}{V_\pi} d\mathbf{x} \\ &+ \alpha \int_\Omega \|\nabla u\|_\delta d\mathbf{x}, \end{aligned} \tag{6}$$

where $\pi(\mathbf{x})$ denotes the inverse depth of pixel $\mathbf{x}$ assuming it lies on the estimated ground plane, while V_π and α are parameters of the energy functional. This is a convex energy, and on the used resolution can be minimized globally and quickly using gradient descent. Fig. 6 shows some results. Afterwards, a triangle mesh is generated from the resulting depth map by interpreting the depth pixels as corners of a regular triangle grid. Some examples in different scenes are shown in Fig. 9. A desirable effect of this approach is that the collision meshes naturally have a higher resolution in close-by than in far-away regions, where the un-projected mesh vertices are more tightly spaced and at the same time the depth map is more accurate.

3.2 Ground plane estimation

We estimate the ground plane normal by low-pass filtering accelerometer measurements which are available on all modern

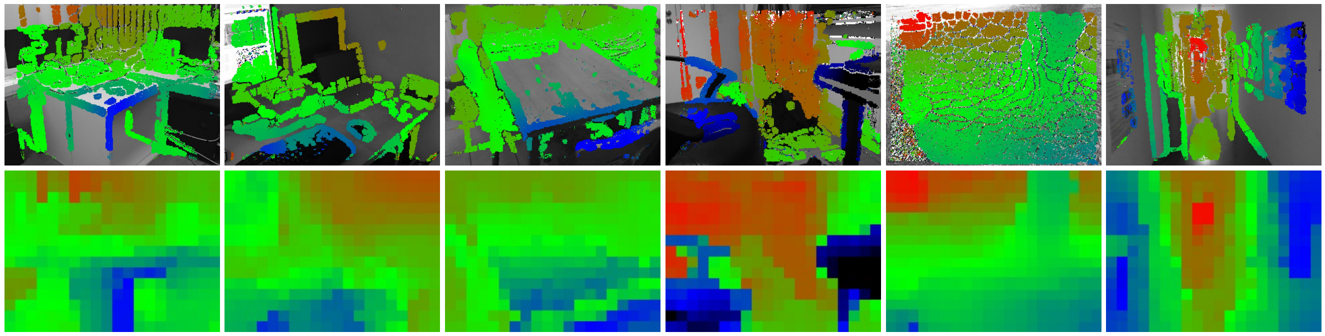

Figure 6: Variational Inpainting: the top row shows a number of full-resolution depth maps from live operation, the bottom row shows the computed low-resolution, regularized version used to build the collision mesh.

Table 1: Tracking accuracy (as drift per second) for two sequences from the TUM RGB-D benchmark [21] at different resolutions. For comparison, we also include results obtained from PTAM [11]. The sequences contain significant rolling shutter and motion blur effects, which reduce the tracking accuracy of PTAM significantly. Accuracy of direct tracking degrades only very little with decreasing image resolution.

			fr2/xyz		fr2/desk	
method	mapping	tracking	(cm/s)	(deg/s)	(cm/s)	(deg/s)
PTAM	640×480	640×480	8.2	3.2	fail	fail
ours	640×480	640×480	0.50	0.31	2.2	0.96
ours	640×480	320×240	0.58	0.32	3.6	1.25
ours	320×240	320×240	0.58	0.32	3.3	0.96
ours	320×240	160×120	0.62	0.33	4.9	1.38
ours	160×120	160×120	0.68	0.37	fail	fail
ours	160×120	80×60	1.58	0.71	fail	fail

Table 2: Performance of tracking and mapping for the first 600 frames of the fr2/desk sequence, on a Sony Xperia Z1 (mean and standard deviation, in ms per iteration). The given resolution is the mapping resolution, tracking is done with one pyramid level higher as the last level. Mapping is not NEON-optimized; the frames were played back with 30 fps. Note that tracking and mapping run in parallel.

	C++ 320×240	ASM with NEON 640×480	ASM with NEON 320×240
Tracking	30.7 (±11.2)	39.2 (±15.9)	14.7 (±6.8)
Mapping	46.6 (±35.4)	184.4 (±123.3)	52.6 (±37.7)

smartphones, giving the direction of gravity. To determine the plane height, we search for the lowest height which is supported by a certain minimum number of depth map samples. The maximum height of all supporting samples is then taken as ground plane: this assures that small bumps, caused by inaccurate height estimates of individual samples, are covered up with a smooth ground surface to drive on.

4 RESULTS

Initialization. We evaluate the success rate of random initialization by running our system on many subsequences of the fr2/desk sequence of the TUM RGB-D benchmark [21]. Note that this includes subsequences with all types of motion, in particular strong rotation or forward-translation, which are ill-conditioned for initialization – causing the initialization to fail more often than in a hand-held case.

A run is classified as successful if the mean relative depth error after 3 seconds is at most 16%, and final relative translational drift (computed over 15 frames) is at most 60%. We observed that the success rate strongly depends on the movement speed of the camera: if the camera does not move sufficiently fast, the depth filters become over-confident, and get stuck at wrong values – this however can be avoided by only mapping on a subset of frames, e.g. every 4th frame. Overall, the measured initialization success rate with this configuration is 67%. Figure 8 shows the evolution of the relative translational drift as well as the mean relative depth error over the first 10 s of all successful runs. Note that these results are obtained using resolutions as employed on the smartphone, and some of the sequences contain strong motion blur and rolling shutter artifacts.

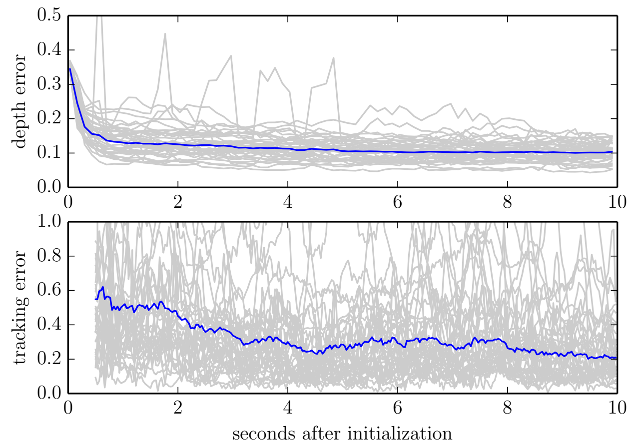

Figure 8: Initialization evaluation: development of errors in successful random initialization runs (average in blue, all samples in gray), evaluated on subsequences of the TUM RGB-D benchmark fr2/desk sequence. The depth error shows the relative deviation from the ground truth depth, after choosing the optimal scale. The tracking error shows the translational drift per 0.5 seconds, relative to the traveled distance. The largest reduction in depth error happens already within the first 0.5 seconds; the tracking error requires longer to stabilize.

Accuracy. We numerically evaluate the tracking accuracy of the proposed approach for different resolutions using the TUM RGB-D benchmark. To be independent from initialization issues and to obtain the correct scale, we use the very first depth image for initialization, while for the remainder of the sequences only the provided intensity images are used. Table 1 shows the results. Notably, the accuracy only changes very little with decreasing image resolution, allowing smooth yet accurate operation on a smartphone.

Speed. With NEON optimizations at a resolution of 320×240, our system is able to track the camera pose with usually well more than

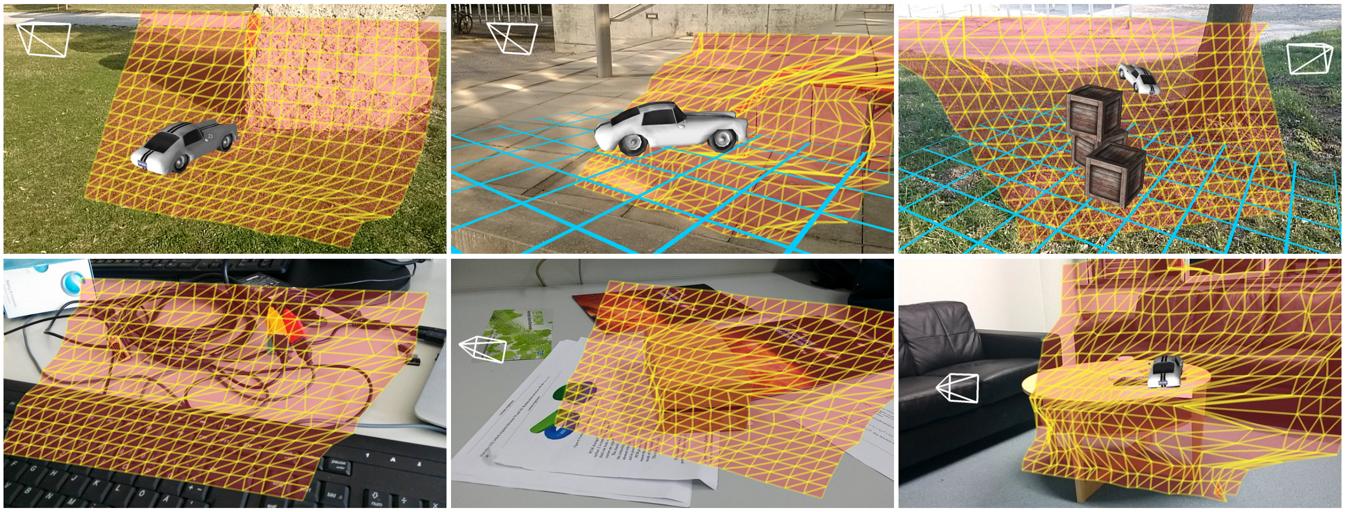

Figure 9: Qualitative evaluation of the system in different, challenging scenes. The collision mesh is fixed and shown from a different perspective, together with its original viewport, augmented objects and the ground plane. The screenshots are taken by the smartphone during live operation.

30 Hz on current-generation smartphones. See Table 2 for timing values measured on a Sony Xperia Z1.

Qualitative Results. We extensively tested the system in real-time operation, Fig. 9 shows some examples of augmented scenes. A full sequence is shown in the attached video.

5 Conclusion

The presented direct monocular visual odometry algorithm is able to operate in real-time on a modern smartphone, with tracking rates of well above 30 Hz at a mapping resolution of 320×240. It operates fully without features; instead it is based on direct image alignment for tracking, and semi-dense depth estimation by pixel-wise filtering over many small-baseline stereo comparisons for mapping. This allows to use much more information in the images (including e.g. edges) and reduces the number of outliers drastically. In addition to accurately and robustly estimating the camera pose, the estimated semi-dense depth maps can be used to build a physical world model for AR with little additional computational effort. We demonstrated this with a small example application.

As future work, using a more sophisticated regularizer for depth map in-painting (e.g. total generalized variation) will eliminate the need for estimating a ground plane. At the same time, more sophisticated minimization schemes as e.g. in [20] will allow world modeling on higher resolutions. Integrating the recent extension towards full, large-scale direct monocular SLAM (LSD-SLAM) [4] will allow to merge collision meshes and scale the method to larger environments.

References

[1] A. Comport, E. Malis, and P. Rives. Accurate quadri-focal tracking for robust 3D visual odometry. In *ICRA*, 2007.

[2] E. Coumans et al. Bullet physics library, `http://bulletphysics.org`.

[3] A. Davison, I. Reid, N. Molton, and O. Stasse. MonoSLAM: Real-time single camera SLAM. *Trans. on Pattern Analysis and Machine Intelligence (TPAMI)*, 29, 2007.

[4] J. Engel, T. Schöps, and D. Cremers. LSD-SLAM: Large-scale direct monocular SLAM. In *ECCV*, 2014.

[5] J. Engel, J. Sturm, and D. Cremers. Semi-dense visual odometry for a monocular camera. In *ICCV*, 2013.

[6] M. Fiala. ARTag, a fiducial marker system using digital techniques. In *CVPR*, 2005.

[7] C. Forster, M. Pizzoli, and D. Scaramuzza. SVO: Fast semi-direct monocular visual odometry. In *ICRA*, 2014.

[8] O. Kahler and J. Denzler. Tracking and reconstruction in a combined optimization approach. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 34(2):387–401, 2012.

[9] A. Karpenko, D. Jacobs, J. Baek, and M. Levoy. Digital video stabilization and rolling shutter correction using gyroscopes. *CSTR*, 1:2, 2011.

[10] C. Kerl, J. Sturm, and D. Cremers. Dense visual SLAM for RGB-D cameras. In *Intelligent Robot Systems (IROS)*, 2013.

[11] G. Klein and D. Murray. Parallel tracking and mapping for small AR workspaces. In *Mixed and Augmented Reality (ISMAR)*, 2007.

[12] G. Klein and D. Murray. Parallel tracking and mapping on a camera phone. In *Mixed and Augmented Reality (ISMAR)*, 2009.

[13] M. Li, B. Kim, and A. Mourikis. Real-time motion estimation on a cellphone using inertial sensing and a rolling-shutter camera. In *ICRA*, 2013.

[14] M. Li and A. Mourikis. High-precision, consistent EKF-based visual-inertial odometry. *International Journal of Robotics Research*, 32:690–711, 2013.

[15] A. Mourikis and S. Roumeliotis. A multi-state constraint Kalman filter for vision-aided inertial navigation. In *IEEE International Conference on Robotics and Automation*, 2007.

[16] R. Newcombe, S. Lovegrove, and A. Davison. DTAM: Dense tracking and mapping in real-time. In *ICCV*, 2011.

[17] M. Pizzoli, C. Forster, and D. Scaramuzza. REMODE: Probabilistic, monocular dense reconstruction in real time. In *ICRA*, 2014.

[18] O. Saurer, K. Köser, J.-Y. Bouguet, and M. Pollefeys. Rolling shutter stereo. In *ICCV*, 2013.

[19] G. Silveira, E. Malis, and P. Rives. An efficient direct approach to visual SLAM. *Robotics, IEEE Transactions on*, 24(5):969–979, 2008.

[20] J. Stuehmer, S. Gumhold, and D. Cremers. Real-time dense geometry from a handheld camera. In *Pattern Recognition (DAGM)*, 2010.

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

[22] P. Tanskanen, K. Kolev, L. Meier, F. C. Paulsen, O. Saurer, and M. Pollefeys. Live metric 3D reconstruction on mobile phones. In *ICCV*, 2013.

[23] D. Wagner, T. Langlotz, and D. Schmalstieg. Robust and unobtrusive marker tracking on mobile phones. In *Mixed and Augmented Reality (ISMAR)*, 2008.

[24] D. Wagner and D. Schmalstieg. ARToolKitPlus for pose tracking on mobile devices. In *Proceedings of 12th Computer Vision Winter Workshop (CVWW'07)*, 2007.