**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 re-decorating 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...
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 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 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 $I: \Omega \rightarrow \mathbb{R}$. Similarly, we represent the inverse depth map and inverse depth variance map as functions $D: \Omega_D \rightarrow \mathbb{R}^+$ and $V: \Omega_D \rightarrow \mathbb{R}^+$, where $\Omega_D$ contains all pixels which have a valid depth hypothesis. We approximate the uncertainty of stereo much more 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 inaccurate 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.
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 \( \xi \in \text{SE}(3) \) of a new frame \( I \) is obtained by directly minimizing the photometric error

\[
E(\xi) := \sum_{x \in \Omega_{D_M}} \|I_M(x) - I(\omega(x, D_M(x), \xi))\|_\delta,
\]

where \( \omega: \Omega_{D_M} \times \mathbb{R} \times \text{SE}(3) \rightarrow \Omega \) 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 downsampled by factors of two, using a weighted average of 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.

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. \( \Omega_D \) 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 well suited for optimization using SIMD parallelization. We use 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 $\Delta \mathbf{X}$, that is

$$
\sum_{x \in \Omega_D} \mathbf{J}_x^T (\mathbf{r}_x \mathbf{w}_x) \cdot \Delta \mathbf{X} = \sum_{x \in \Omega_D} \mathbf{J}_x^T \mathbf{g}_x \mathbf{w}_x,
$$

(4)

where $\mathbf{r}_x$, $\mathbf{J}_x$ and $\mathbf{w}_x$ are, for one pixel $x$, 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].

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 $\pi$. As regularizer we use the Huber norm

$$
\| \mathbf{x} \|_\delta := \begin{cases} 
\frac{\| \mathbf{x} \|^2}{2\delta^2} & \text{if } \| \mathbf{x} \| < \delta \\
\frac{1}{2} & \text{otherwise}
\end{cases}
$$

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

$$
E(u) := \int_{\Omega_D} \frac{(u(x) - D(x))^2}{V(x)} \, dx + \int_{\Omega \setminus \Omega_D} \frac{(u(x) - \pi(x))^2}{V(x)} \, dx + \alpha \int_{\Omega} \| \nabla u \|_\delta \, dx,
$$

(6)

where $\pi(x)$ denotes the inverse depth of pixel $x$ assuming it lies on the estimated ground plane, while $V_\pi$ and $\alpha$ 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
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.

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
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 $320 \times 240$. 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 to semi-dense visual odometry (e.g. total generalized variation) will eliminate the need for estimating collision meshes and scale the method to larger environments.

**References**


