Real-Time Camera Tracking and 3D Reconstruction Using Signed Distance Functions

Erik Bylow*, Jürgen Sturm†, Christian Kerl†, Fredrik Kahl*
and Daniel Cremers†
*Center for Mathematical Sciences, Lund University, Lund, Sweden
Email: erikb@maths.lth.se, fredrik@maths.lth.se
† Department of Computer Science, Technical University of Munich, Garching, Germany
Email: juergen.sturm@in.tum.de, christian.kerl@in.tum.de, cremers@in.tum.de

***Abstract*—The ability to quickly acquire 3D models is an essential capability needed in many disciplines including robotics, computer vision, geodesy, and architecture. In this paper we present a novel method for real-time camera tracking and 3D reconstruction of static indoor environments using an RGB-D sensor. We show that by representing the geometry with a signed distance function (SDF), the camera pose can be efficiently estimated by directly minimizing the error of the depth images on the SDF. As the SDF contains the distances to the surface for each voxel, the pose optimization can be carried out extremely fast. By iteratively estimating the camera poses and integrating the RGB-D data in the voxel grid, a detailed reconstruction of an indoor environment can be achieved. We present reconstructions of several rooms using a hand-held sensor and from onboard an autonomous quadrocopter. Our extensive evaluation on publicly available benchmark data shows that our approach is more accurate and robust than the iterated closest point algorithm (ICP) used by KinectFusion, and yields often a comparable accuracy at much higher speed to feature-based bundle adjustment methods such as RGB-D SLAM for up to medium-sized scenes.**

Fig. 1: Reconstruction of a room with a handheld sensor using our approach. Camera tracking and reconstruction runs in real-time on a laptop and provides live feedback to the user. Top row: living room. Bottom row: study room.

I. Introduction

3D simultaneous localization and mapping (SLAM) is a highly active research area as it is a pre-requisite for many robotic tasks such as localization, navigation, exploration, and path planning. To be truly useful, such systems require the fast and accurate estimation of the robot pose and the scene geometry. The straightforward acquisition of 3D models is also of interest for augmented reality applications including computer games, home decoration, and refurbishment measures. For example, designers and architectures can benefit from this technology as it allows them to cost-effectively scan a room or a piece of furniture before starting the actual work.

Two examples of 3D models acquired with our approach are shown in Figure 1. Our scanning equipment consists of a handheld Asus Xtion Pro Live sensor and a laptop with a Quadro GPU from Nvidia. The laptop provides a live view of the reconstructed model. Both models were acquired in real-time from 1000 images (approx. 30 seconds). As can be seen in the figure, the resulting models are highly detailed and provide absolute metric information about the scene which is useful for a large variety of subsequent tasks.

This work has partially been supported by the DFG under contract number FO 180/17-1 in the Mapping on Demand (MOD) project.

Structure from motion (SfM) techniques from computer vision typically use images from a moving camera. By extracting and matching visual features across several views, bundle adjustment can be used to estimate both the camera poses and a sparse 3D model of the feature points [13, 2]. Furthermore, several methods have been proposed to compute dense depth maps from image data [10, 21]. The advent of depth sensors like the Microsoft Kinect opened new possibilities for approaches on dense 3D reconstruction, as they directly output dense depth images. Accordingly, we investigate in this paper how dense depth images can be efficiently registered and integrated, and how the scene geometry and texture can be represented to facilitate this endeavor.

Recently, Newcombe et al. [18] have presented impressive results by using signed distance functions (SDFs) to represent the scene geometry and the iterated closest point (ICP) algorithm for camera tracking. Whelan et al. [24] extended this approach with a rolling reconstruction volume and color fusion, and evaluated alternative methods for visual odometry estimation. However, pure visual odometry is inherently subject to significant drift, so that the registration with respect to a world model is in our point of view preferable.

The contribution of this work is a novel method for esti-

mating the camera motion directly based on the SDF. The key insight behind our approach is that the SDF already encodes the distance of each voxel to the surface. As a result, we do not need explicit data association or downsampling as in ICP to achieve real-time performance. We present an extensive evaluation of our approach on public benchmark datasets. In comparison to previous methods, we found that our approach yields more accurate and more robust tracking than ICP-based KinectFusion (as implemented in the point cloud library [1]) and it is often comparable to the feature-based RGB-D SLAM [8]. Furthermore, we demonstrate that our approach is stable enough to use it for position control of an autonomous quadrocopter.

II. Related Work

Simultaneous localization and mapping refers to both the estimation of the camera pose and mapping of the environment. This requires a suitable representation of the scene geometry, and the choice of this representation strongly influences the efficiency of pose estimation and map optimization.

Laser-based localization and mapping approaches often use scan matching or the iterated closest point algorithm (ICP) [4] to estimate the motion between frames. Graph SLAM methods use these motion estimates as input to construct and optimize a pose graph [15]. Typically, these methods render a joint map only after pose graph optimization, and this map is generally not used for further pose optimization. The resulting maps are often represented as occupancy grid maps or octrees [25] and are therefore well suited for robot localization or path planning. Henry et al. [9] were the first to apply the Graph SLAM approach to RGB-D data using a combination of visual features and ICP. A similar system was recently presented by Endres et al. [8] and extensively evaluated on a public benchmark [22]. In this paper, we compare the performance of our approach to the RGB-D SLAM system and demonstrate that we often achieve a comparable tracking performance at a higher frame-rate. Newcombe et al. [18] recently demonstrated with their famous KinectFusion approach that dense reconstruction is possible in real-time by using a Microsoft Kinect sensor. To represent the geometry, Newcombe et al. employ a signed distance function (SDF) [7] and use ICP in a coarse-to-fine manner to estimate the camera motion. For each image, the algorithm first renders a point cloud from the SDF at the previous pose using ray tracing and subsequently aligns this with the next depth image. Point correspondences are found using projective data association [5] and the point-to-plane distance. As the original implementation is not available and no benchmark evaluation is provided, we compare our approach to the KinFu open-source implementation as available in the point cloud library [1]. We show in this paper that our approach outperforms KinFu in terms of speed and accuracy.

While ICP only minimizes the error on point clouds, several approaches have recently appeared that minimize the photometric error [20, 12] or combinations of both [23], however without subsequent 3D reconstruction. Whelan et al. [24] recently integrated these methods with the KinectFusion approach and demonstrated that superior tracking performance can be achieved, however without evaluating the global consistency of the resulting model.

Canelhas [6] developed in his master's thesis an approach for camera tracking similar to ours. However, his focus lies more on object detection and recognition in an SDF, and no thorough evaluation of the accuracy was performed. Kubacki et al. [14] showed how an SDF can be used to estimate the camera pose, however, only on synthetic data and without a comparative evaluation. Recently, Ren and Reid [19] demonstrated SDF-based object tracking where they assume a known object model. Based on this, they generate a SDF that they subsequently use for object tracking. By using a robust cost function, robustness to fast camera motions and partial occlusions can be achieved.

In this paper, we describe (1) a direct approach to camera tracking on SDFs, (2) present a thorough evaluation of SDF-based tracking and mapping on public benchmarks, and (3) compare the tracking performance to existing real-time solutions. We study the influence of alternative distance metrics, weighting functions and different camera motions on a large number of different scenes. Furthermore, we demonstrate that our approach is directly applicable to position control of a quadrocopter and the automatic 3D reconstruction of rooms.

III. Notation and Preliminaries

We denote a 3D point as $\mathbf{x} \in \mathbb{R}^3$. Further, we denote the rotation of the camera as $R \in SO(3)$ and the translation as $\mathbf{t} \in \mathbb{R}^3$. At each time step, an RGB-D camera outputs a color and a depth image, to which we refer to by the functions

$$I_{RGB} : \mathbb{R}^2 \to \mathbb{R}^3 \text{ and } I_d : \mathbb{R}^2 \to \mathbb{R}. \tag{1}$$

We assume that the depth image is already registered to the color image, so that pixels correspond one-to-one.

We assume the pinhole camera model with intrinsic parameters f_x, f_y, c_x and c_y corresponding to the focal length and the optical center. According to this model, a 3D point $\mathbf{x} = (x, y, z)^\top$ is projected onto the image plane by

$$\pi(x, y, z) = \left(\frac{f_x x}{z} + c_x, \frac{f_y y}{z} + c_y \right)^\top \tag{2}$$

and we can reconstruct the 3D point corresponding to a pixel $(i, j)^\top \in \mathbb{R}^2$ with depth $z = I_d(i, j)$ by

$$\rho(i, j, z) = \left(\frac{(i - c_x)z}{f_x}, \frac{(j - c_y)z}{f_y}, z \right)^\top. \tag{3}$$

IV. Approach

We represent the geometry using a signed distance function stored in a voxel grid. We follow an iterative approach where we first estimate the camera pose given the previous SDF, and then update the SDF based on the newly computed camera pose. Note that we optimize the camera pose directly on the SDF, while KinectFusion [18] first generates a synthetic depth images that it subsequently aligns to the current depth image using ICP.

A. Camera Tracking

In this part, we present how we estimate the camera motion given an SDF and a depth image. For now, we assume that we already have a representation of the geometry acquired from the previous depth image given by the SDF

$$\psi : \mathbb{R}^3 \rightarrow \mathbb{R}. \tag{4}$$

This function returns for any point $\mathbf{x} \in \mathbb{R}^3$ the signed distance from $\mathbf{x}$ to the surface. The idea is now to use the SDF to construct an error metric that describes how well a depth image I_d fits to the SDF.

For each pixel (i, j), we have its depth $z = I_d(i, j)$. Given this, we can reconstruct the corresponding 3D point $\mathbf{x}_{ij}$ in the local coordinate system of the camera by (3). We can transform this point to the global coordinate frame using

$$\mathbf{x}_{ij}^G = R\mathbf{x}_{ij} + \mathbf{t}, \tag{5}$$

and now query the SDF to read out its distance from the surface. Given that the SDF and the camera pose is correct, the reported value should then be zero.

We seek to find the camera rotation R and translation $\mathbf{t}$ such that all reprojected points from the depth image lie as close as possible to the zero-crossing in the SDF (=surface), i.e.,

$$\{\mathbf{x} \mid \psi(\mathbf{x}) = 0\}. \tag{6}$$

This idea is illustrated in Figure 2.

By assuming Gaussian noise in the depth measurements of the camera and that all pixels are independent and identically distributed, the likelihood of observing a depth image I_d from camera pose $R, \mathbf{t}$ becomes

$$p(I_d \mid R, \mathbf{t}) \propto \prod_{i,j} \exp(-\psi(R\mathbf{x}_{ij} + \mathbf{t})^2). \tag{7}$$

Our goal is now to find the camera pose $R^*, \mathbf{t}^*$ that maximizes this likelihood, i.e.,

$$(R^*, \mathbf{t}^*) = \arg\max_{R,\mathbf{t}} p(I_d \mid R, \mathbf{t}). \tag{8}$$

To simplify subsequent computations, we take the negative logarithm and define the error function

$$E(R, \mathbf{t}) = \sum_{i,j} \psi(R\mathbf{x}_{ij} + \mathbf{t})^2, \tag{9}$$

where i, j iterate over all pixels in the depth image (in our case 640×480). Remember that in an SDF, all points on the surface have a distance of zero. Therefore, in the noise free case, our error function would return zero as all points would exactly reproject onto the surface. In practice, due to noise, the error function will never be exactly zero. Moreover, not all pixels in the depth image are defined (for example, due to missing values because of occlusions or poor reflectance). Furthermore, the voxel grid underlying the SDF has a finite size and may also have missing values. Therefore, we count the number of evaluated pixels n and normalize the error accordingly.

To minimize this error function we use the Lie algebra representation of rigid-body motion as described in Ma et al.

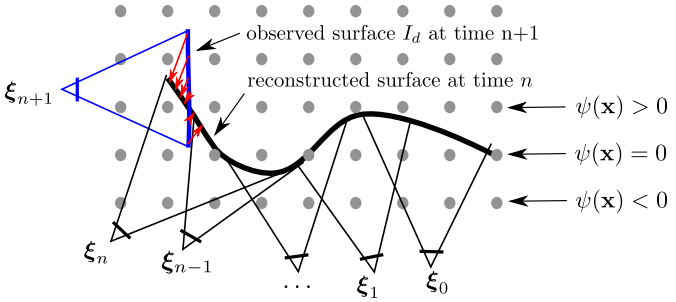

Fig. 2: Our goal is to find the camera pose $\boldsymbol{\xi}_{n+1}$ such that the SDF values between the reprojected 3D points is minimized. The SDF is constructed from the first n depth images and corresponding camera poses $\boldsymbol{\xi}_1, \ldots, \boldsymbol{\xi}_n$.

[17]. A rigid-body motion can be described with the 6-dimensional twist coordinates

$$\boldsymbol{\xi} = (\omega_1, \omega_2, \omega_3, v_1, v_2, v_3), \tag{10}$$

where (v_1, v_2, v_3) refer to the translational and $(\omega_1, \omega_2, \omega_3)$ to the rotational components. The advantage of the Lie algebra is that it is a minimal representation, i.e., it has only six degrees of freedom. Using this notation, we can rewrite (9) as

$$E(\boldsymbol{\xi}) = \sum_{i,j} \psi_{ij}(\boldsymbol{\xi})^2, \tag{11}$$

where

$$\psi_{ij}(\boldsymbol{\xi}) = \psi(R\mathbf{x}_{ij} + \mathbf{t}). \tag{12}$$

Our goal is now to find the twist $\boldsymbol{\xi}$ that minimizes this function. To solve this, we apply the Gauss-Newton method for nonlinear minimization. We start by linearizing ψ around our initial pose estimate $\boldsymbol{\xi}^{(0)}$ that we set to the estimated previous camera pose $\boldsymbol{\xi}_n$ of time step n,

$$\psi(\boldsymbol{\xi}) \approx \psi(\boldsymbol{\xi}^{(k)}) + \nabla\psi(\boldsymbol{\xi}^{(k)})^\top(\boldsymbol{\xi} - \boldsymbol{\xi}^{(k)}), \tag{13}$$

where $\nabla\psi(\boldsymbol{\xi}^{(k)})$ is the derivative of the signed distance function evaluated at $\boldsymbol{\xi}^{(k)}$ and k is the iteration step. Plugging this into (11) gives us a quadratic form that approximates the original error function, i.e.,

$$E_{\text{approx}}(\boldsymbol{\xi}) = \sum_{i,j} (\psi_{ij}(\boldsymbol{\xi}^{(k)}) + \nabla\psi_{ij}(\boldsymbol{\xi}^{(k)})^\top(\boldsymbol{\xi} - \boldsymbol{\xi}^{(k)}))^2. \tag{14}$$

We can efficiently minimize this equation by putting its derivative to zero, i.e.,

$$\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\xi}} E_{\text{approx}}(\boldsymbol{\xi}) = 0, \tag{15}$$

which, after some calculations, becomes

$$\sum_{i,j} \psi_{ij}(\boldsymbol{\xi}^{(k)})\nabla\psi_{ij}(\boldsymbol{\xi}^{(k)}) + \nabla\psi_{ij}(\boldsymbol{\xi}^{(k)})\nabla\psi_{ij}(\boldsymbol{\xi}^{(k)})^\top(\boldsymbol{\xi} - \boldsymbol{\xi}^{(k)}) = 0. \tag{16}$$

Here $\psi_{ij}(\boldsymbol{\xi}^{(k)})\nabla\psi_{ij}(\boldsymbol{\xi}^{(k)})$ gives a 6-dimensional vector $\mathbf{b}_{ij}$ for each pixel and $\nabla\psi_{ij}(\boldsymbol{\xi}^{(k)})\nabla\psi_{ij}(\boldsymbol{\xi}^{(k)})^\top$ gives a 6×6-dimensional matrix A_{ij} for each pixel. By defining

$$A := \sum_{i,j} \nabla\psi_{ij}(\boldsymbol{\xi}^{(k)})\nabla\psi_{ij}(\boldsymbol{\xi}^{(k)})^\top \in \mathbb{R}^{6\times 6}, \tag{17}$$

$$\mathbf{b} := \sum_{i,j} \psi_{ij}(\boldsymbol{\xi}^{(k)})\nabla\psi_{ij}(\boldsymbol{\xi}^{(k)}) \in \mathbb{R}^{6\times 1}, \tag{18}$$

we can rewrite (16) as

$$\mathbf{b} + A\boldsymbol{\xi} - A\boldsymbol{\xi}^{(k)} = 0. \tag{19}$$

From this, we can compute the camera pose that minimizes the linearized error as

$$\boldsymbol{\xi}^{(k+1)} = \boldsymbol{\xi}^{(k)} - A^{-1}\mathbf{b}, \tag{20}$$

which can quickly be computed as A is only a 6×6 matrix. Based on this new estimate, we re-linearize the original error function (11) around $\boldsymbol{\xi}$, and solve iteratively (20) until convergence. We stop when either the change $\|\boldsymbol{\xi}^{(k+1)} - \boldsymbol{\xi}^{(k)}\|_\infty$ is small enough or when a certain number of iterations is exceeded. Upon convergence, we assign the final pose estimate to the current camera $\boldsymbol{\xi}_{n+1}$.

In order to obtain real-time performance, we computed all vectors $\mathbf{b}_{ij}$ and matrices A_{ij} in parallel on the GPU since they are independent of each other.

B. Representation of the SDF

As detailed by Curless and Levoy [7], we represent the SDF using a discrete voxel grid of resolution m. We allocate two grids in memory, where one stores the averaged distances, and the second one stores the sum of all weights, i.e.,

$$D : [0,\dots,m-1]^3 \mapsto \mathbb{R}, \tag{21}$$

$$W : [0,\dots,m-1]^3 \mapsto \mathbb{R}. \tag{22}$$

Keeping track of the weight in each cell allows us to handle occlusion and sensor uncertainty appropriately, as we will detail in the next subsection. Given a reconstruction volume of dimension width × height × depth, a world point $\mathbf{x} = (x,y,z)^\top \in \mathbb{R}^3$ is mapped to voxel coordinates

$$\begin{pmatrix} i \\ j \\ k \end{pmatrix} = m \begin{pmatrix} x/\text{width} + 0.5 \\ y/\text{height} + 0.5 \\ z/\text{depth} + 0.5 \end{pmatrix}. \tag{23}$$

Since $(i,j,k)^\top$ is generally non-integer, we determine the signed distance value $\psi(\mathbf{x})$ by tri-linear interpolation between the values of its eight integer neighbors.

C. Distance and Weighting Functions

To integrate a new depth image, we need to determine which voxels have been observed by the depth camera and update their distances accordingly. While Curless and Levoy [7] originally proposed to perform ray casting from the sensor, we follow the opposite approach similar to [18] by projecting each voxel onto the image plane. This has the advantage that every voxel is visited exactly once, which is hard to ensure in the ray casting approach. As the operation that has to be carried out for each voxel is independent of its neighbors, this process can be easily parallelized on the GPU. In contrast to previous work, we present in the following several alternative distance and weighting functions and study their influence on the performance.

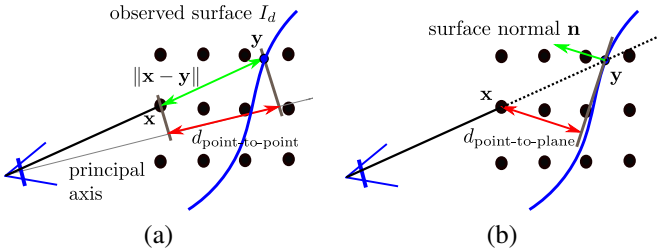

Fig. 3: Visualization of the projective point-to-point distance (a) and the point-to-plane distance (b). Note that computing the true distance is computationally involved.

To implement this strategy, we need to compute the distance of each voxel to the observed surface as represented by the current depth image. Note that computing the true distance of each voxel is time consuming, as it requires the computation of all shortest paths over the entire volume. Even efficient algorithms such as Fast Marching [3] are not well suited for real-time applications. Therefore, we resort instead to an approximation based on the projective distance. We investigated both the projective point-to-point and point-to-plane metric as illustrated in Figure 3 with more details given in the next two subsections.

1) Projective Point-To-Point: For each vertex we have its global (center) coordinates $\mathbf{x}^G$. Given the pose of the current camera $R, \mathbf{t}$, we can transfer these coordinates in the local coordinate frame of the camera as

$$\mathbf{x} = (x,y,z)^\top = R^\top(\mathbf{x}^G - \mathbf{t}). \tag{24}$$

According to our camera model (2), this point gets projected to the pixel

$$(i,j)^\top = \pi(\mathbf{x}) \tag{25}$$

in the image. We define then the projective point-to-point distance as the difference of the depth of the voxel and the observed depth at $(i,j)^\top$, i.e.,

$$d_{\text{point-to-point}}(\mathbf{x}) := z - I_d(i,j). \tag{26}$$

Note that this distance is signed, i.e., negative values are assigned to voxels in front of the observed surface, and positive values to voxels behind.

2) Projective Point-To-Plane: Our motivation for the investigating the point-to-plane metric was that the point-to-point metric gets increasingly inaccurate the less the viewing angle is orthogonal to the surface (see Figure 3 b). This can be resolved by using the point-to-plane distance under the assumption that the observed surface is locally planar.

As a first step, we apply a bilateral filter to the depth image and compute the normals for all pixels. Given a voxel $\mathbf{x}$, we

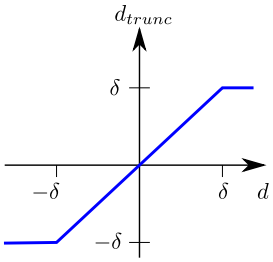

Fig. 4: We truncate large estimated distances to limit the influence of approximation errors and noise.

compute its corresponding pixel coordinates (i, j) and read out the observed surface normal $\mathbf{n}(i, j)$. The point-to-plane distance can then be computed as

$$d_{\text{point-to-plane}}(\mathbf{x}) := (\mathbf{y} - \mathbf{x})^\top \mathbf{n}(i, j). \tag{27}$$

Again, note that this distance is signed, i.e., negative values refer to voxels in front of the surface and positive values to voxels behind.

3) Truncation and Weighting: As projective distance metrics are only approximations of the true distance function, they can be highly erroneous, especially when the estimated distances are large. However, for tracking and reconstruction, we are in particular interested that a small band around the zero-crossing is accurately estimated in the SDF. Therefore, we truncate the projected distances d and apply a weighting term that blends out large distances. For truncation, we use

$$d_{trunc} = \begin{cases} -\delta & \text{if} \quad d < -\delta \\ d & \text{if} \quad |d| \leq \delta \\ \delta & \text{if} \quad d > \delta \end{cases} \tag{28}$$

The truncation function is also illustrated in Figure 4. Note that due to this truncation, the gradient of our SDF will be zero in regions that are far away from the estimated surface.

Furthermore, we employ a weighting function to give higher weights to voxels in front of the observed surface and lower weights to voxels behind. Depending on the observation model of the depth sensor, different weighting functions can be used. In this work, we study six different weighting functions as depicted in Figure 5: The constant weight trivially assumes a constant weight for all voxels, i.e.,

$$w_{\text{const}}(d) = 1. \tag{29}$$

This is suitable for distance sensors that can (deeply) penetrate objects, such as a radar. The linear weight, as proposed by Curless and Levoy [7] and used in KinectFusion, assigns a constant weight to all voxels up to a certain penetration depth ϵ. After this depth, the weight linearly decreases to zero at penetration depth δ:

$$w_{\text{lin}}(d) = \begin{cases} 1 & \text{if} \quad d < \epsilon \\ \frac{\delta - d}{\delta - \epsilon} & \text{if} \quad d \geq \epsilon \text{ and } d \leq \delta. \\ 0 & \text{if} \quad d > \delta \end{cases} \tag{30}$$

The linear model expresses a certain prior for the minimum depth δ of objects in the scene and a linear prior afterwards. In contrast to this, we propose an exponential weighting function motivated by a Gaussian noise model of depth measurements, i.e.,

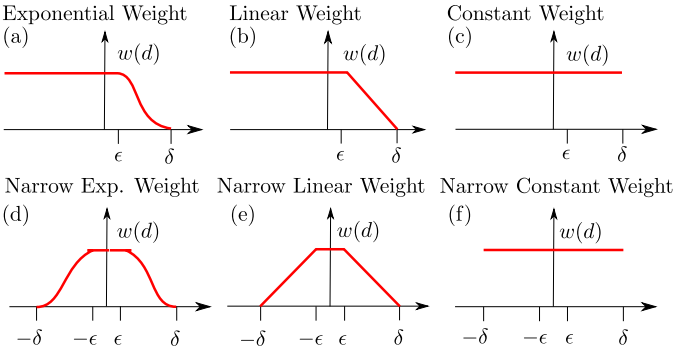

Fig. 5: We evaluated six different weighting functions for data fusion.

$$w_{\text{exp}}(d) = \begin{cases} 1 & \text{if} \quad d < \epsilon \\ e^{-\sigma(d-\epsilon)^2} & \text{if} \quad d \geq \epsilon \text{ and } d \leq \delta. \\ 0 & \text{if} \quad d > \delta \end{cases} \tag{31}$$

As we will show in our experimental evaluation, this weighting function leads to a slightly higher robustness than the linear weight function.

D. Data Fusion and 3D Reconstruction

Given a sequence of (approximate) distance measurements and the weights for a particular voxel cell $\bar{\mathbf{x}} = (i, j, k)$, our goal is to fuse all of these measurements to obtain the best possible estimate for $\psi(\mathbf{x})$. As this estimation process can be carried out independently for each voxel, we drop the index $\bar{\mathbf{x}}$ in the remainder of this subsection. For this, we follow the approach of Curless and Levoy [7], that we briefly summarize here.

Under the assumption that all distance measurements are normally distributed, this estimation problem can be formulated as follows. We seek the distance ψ that maximizes the observation likelihood

$$p(d_1, w_1, \ldots, d_n, w_n \mid \psi) \propto \prod_{i=1}^{n} \exp\left(-\frac{1}{2} w_i (\psi - d_i)^2\right), \tag{32}$$

where d_i and w_i refer to the observed truncated distances and weights in frame i, respectively. After taking the negative logarithm, we obtain a quadratic error function

$$L(\psi) = \sum_{i=1}^{n} \frac{1}{2} w_i (\psi - d_i)^2 \tag{33}$$

that we aim to minimize. By putting the derivative of $L(\psi)$ to zero we obtain

$$\psi = \frac{\sum_{i=1}^{n} w_i d_i}{\sum_{i=1}^{n} w_i}, \tag{34}$$

which means that the optimal ψ is the weighted average of all measurements. Therefore, we can calculate the estimated

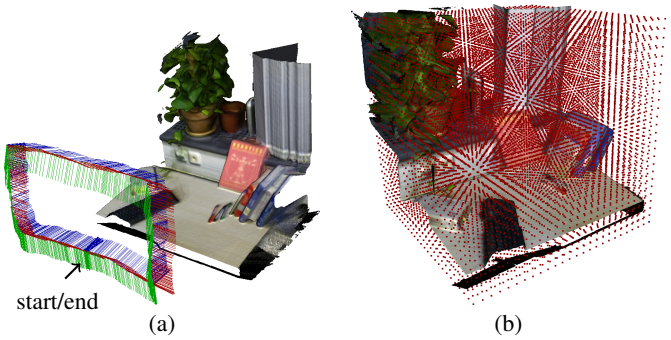

Fig. 6: On small work spaces, our method is nearly drift-free. (a) 3D reconstruction and the estimated camera trajectory of a small office scene. (b) Visualization of the (downsampled) voxel grid underlying the reconstruction volume ($m = 256$).

signed distance of each voxel as a running weighted average, i.e.,

$$D \leftarrow \frac{WD + w_{n+1}d_{n+1}}{W + w_{n+1}} \tag{35}$$

$$W \leftarrow W + w_{n+1}. \tag{36}$$

Note that this computation has to be carried our for each voxel. As each voxel does not depend on its neighbors, this update can easily be computed in parallel for all voxels on the GPU.

E. Meshing and Colorization

The SDF encodes the 3D geometry of the scene. As the surface is located at the zero crossing in the signed distance function, we apply a straight-forward implementation of the marching cubes algorithm [16] to extract the corresponding triangle mesh.

Next to the estimation of the geometry, we also estimate a color texture of the scene similar to [24]. We represent this texture using an additional voxel grid consisting of three channels R, G, B for the color and one additional channel for the color weights W_c. The computation can be carried out in parallel during the update step of the SDF as described in Section IV-D. Given that a voxel is sufficiently close to the surface, i.e., $\|d\| < \epsilon$, we retrieve the observed color

$$(r, g, b)^\top = I_{RGB}(i, j) \tag{37}$$

from the RGB image and update the color estimate as the running average

$$R \leftarrow \frac{W_c R + w_c^{n+1} r}{W_c + w_c^{n+1}} \tag{38}$$

$$G \leftarrow \frac{W_c G + w_c^{n+1} g}{W_c + w_c^{n+1}} \tag{39}$$

$$B \leftarrow \frac{W_c B + w_c^{n+1} b}{W_c + w_c^{n+1}} \tag{40}$$

where w_c^{n+1} is the weight for the new measurement. As weight, we use

$$w_c^{n+1} = w_{n+1} \cos\theta, \tag{41}$$

where θ is the angle between the ray and the principal axis to give more weight to pixels whose normal is pointing towards the camera. While extracting the triangle mesh using marching cubes, we query the color grid to compute colors for vertices using tri-linear interpolation.

V. Results

In this section we present both qualitative results of 3D reconstructions from live-data and quantitative results on the TUM RGB-D benchmark [22]. We compare the performance our algorithm to KinFu [1] and RGB-D SLAM [8], and study the influence of the truncation parameter and alternative weighting functions on the tracking performance. Finally, we analyze the robustness of our approach with respect to fast camera movements and provide runtime measurements.

A. Qualitative Results

Figures 1 and 6 show several live reconstructions of rooms using our algorithm at a grid resolution of $m = 512$ and $m = 256$, respectively. The resulting reconstruction is highly detailed and metrically accurate, so that it could for example be used by architects and interior designers for planning and visualization tasks.

We observe that our method is almost drift-free for small scenes, as can be seen in Figure 6a, where we started and ended a rectangular camera motion at the same spot. Moreover, fine details such as the cover of the RSS proceedings appear sharply.

Furthermore, we used our approach for 3D reconstruction from an autonomous quadrocopter (see Figure 7) equipped with an RGB-D camera. Note that tracking and reconstruction were carried out in real-time on an external ground station with GPU support. The estimated pose was directly used for position control. This demonstrates that our technique is applicable for the navigation of quadrocopters and other robots.

The video provided as supplemental material further illustrates our experimental setups and provides additional views of the reconstructed models.

B. Benchmark Evaluation

We also evaluated our approach on the TUM RGB-D benchmark [22]. As comparison we used the KinFu implementation [1] and RGB-D SLAM [8]. For the ICP-step of KinFu, we explored a large range of different parameter settings and selected the one with the highest performance.

In this evaluation, we chose $\delta = 0.3$ m and $\epsilon = 0.025$ m. The results are given in Table I. Our approach clearly outperforms KinFu which diverges in particular on the faster sequences. We believe that this is due to the fact that KinFu looses much valuable information because of the down-projection of the SDF to a synthetic depth image prior to camera tracking. In particular for large camera motions, the synthetic image is taken from a substantially different viewpoint so that the alignment process is more difficult. For our algorithm, we found that the point-to-point metric provides

TABLE I: The root-mean square absolute trajectory error for KinFu and our method for different resolutions, metrics and datasets. Also the result for RGB-D SLAM are presented.

Method	Res.	Teddy	F1 Desk	F1 Desk2	F3 Household	F1 Floor	F1 360	F1 Room	F1 Plant	F1 RPY	F1 XYZ
KinFu	256	0.156 m	0.057m	0.420 m	0.064 m	Failed	0.913 m	Failed	0.598 m	0.133 m	0.026 m
KinFu	512	0.337 m	0.068 m	0.635 m	0.061 m	Failed	0.591 m	0.304 m	0.281 m	0.081 m	0.025 m
Point-To-Plane	256	**0.072 m**	0.087 m	0.078 m	0.053 m	0.811 m	0.533 m	0.163 m	0.047 m	0.047 m	0.029 m
Point-To-Plane	512	0.101 m	0.059 m	0.623 m	0.053 m	0.640 m	0.206 m	0.105 m	**0.041 m**	**0.042 m**	0.026 m
Point-To-Point	256	0.086 m	0.038 m	**0.061 m**	**0.039 m**	0.641 m	0.420 m	0.121 m	0.047 m	0.047 m	**0.021 m**
Point-To-Point	512	0.080 m	**0.035 m**	0.062 m	0.040 m	**0.567 m**	**0.119 m**	**0.078 m**	0.043 m	**0.042 m**	0.023 m
RGB-D SLAM		0.111 m	0.026 m	0.043 m	0.059 m	0.035 m	0.071 m	0.101 m	0.061 m	0.029 m	0.013 m

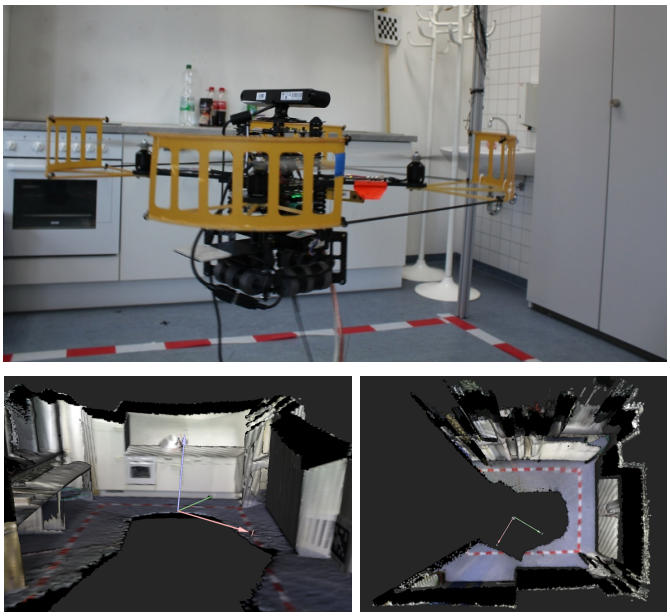

Fig. 7: 3D reconstruction using an autonomous quadrocopter. Top: AscTec Pelican platform used. Bottom: Resulting 3D reconstruction of the room computed in real-time on the ground station.

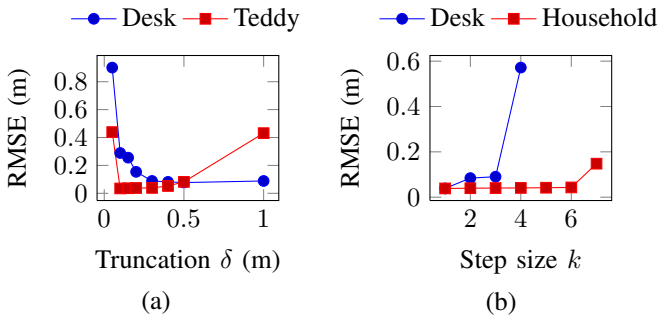

Fig. 8: (a) The choice of the truncation parameter δ depends on the average depth of the objects in the scene. For typical indoor scenes, $\delta = 0.3$ m is a good choice. (b) RMSE when using only every k-th frame (to emulate faster camera motions).

TABLE II: Evaluation of alternative weighting functions.

Dataset	F1 Teddy		F1 Desk	
	RMSE	Max	RMSE	Max
Exp. Weight	0.088 m	**0.213 m**	**0.038 m**	0.088 m
Linear Weight	**0.083 m**	0.285 m	**0.038 m**	0.089 m
Constant Weight	0.093 m	0.242 m	0.040 m	0.089 m
Narrow Exp.	0.170 m	0.414 m	**0.038 m**	**0.083 m**
Narrow Linear	0.382 m	0.688 m	0.044 m	0.085 m
Narrow Constant	0.379 m	0.694 m	0.044 m	0.209 m

better results on most sequences. In comparison to RGB-D SLAM, we achieve often a similar performance in terms of accuracy but require six to eight times less computation time. As our approach only uses structure for tracking, it fails in cases where only co-planar surfaces are visible, such as a wall. KinFu suffers the same limitation, while RGB-D SLAM uses the texture. It would be interesting to additionally exploit the color information during tracking [12].

C. Parameter Study

We investigated the influence of the truncation parameter δ and ϵ on the Teddy and Desk sequence. As Figure 8a shows, choosing δ too small results in poor tracking. This is expected since then the band where the gradient is non-zero is very narrow. The upper limit clearly depends on the scene and encodes a prior of the average object depth. For typical office scenes, $\delta = 0.3$ m is a reasonable choice.

Furthermore, we evaluated the weighting functions proposed in Section IV-C, Table II gives the results. We found that the exponential weighting function leads to more robust tracking, although the linear weight is slightly more accurate. Clearly, the narrow weighting functions yield inferior tracking results.

D. Robustness Evaluation

We also evaluated the robustness of our method with respect to the camera speed. For this, we tracked only every k-th frame on the Desk and Household sequences. The results are given in Figure 8b. The performance quickly decreases if more than three images are left out in the fast Desk sequence (0.4 m/s), while our approach yields decent results for up to six skipped frames for the slower Household sequence (0.25 m/s). The RMSE for every sixth image is 4.3 cm which still outperforms KinFu with 6.1 cm when using every image.

E. Runtime and Memory Consumption

For a resolution of $m = 256$, our approach consumes on average around 23 ms and thus runs easily in real-time time on 30 fps RGB-D data. In comparison, KinFu consumes on the same hardware and at the same resolution 20 ms per frame, while RGB-D SLAM requires around 100–250 ms. All three algorithms make extensive use of the GPU. Based on this

evaluation, we conclude that our algorithm runs approximately at the same speed as KinFu and six times faster than RGB-D SLAM.

In more detail, our approach requires 19.4 ms for pose optimization and 3.7 ms for data fusion at $m = 256$, while for $m = 512$, it takes 31.1 ms for pose optimization and 21.6 ms for data fusion. Note that in theory, the complexity of pose optimization solely depends on the size of the input images, while the complexity of data fusion depends cubically on the resolution of the volume.

For $m = 256$, our approach requires 128 MB of RAM on the GPU for the SDF and 256 MB for the color grid; for $m = 512$, it requires 1 GB for the SDF and 2 GB for the color grid.

VI. Conclusion

In this paper we presented a novel approach to directly estimate the camera movement using a signed distance function. Our method allows the quick acquisition of textured 3D models that can be used for real-time robot navigation. By evaluating our method on a public RGB-D benchmark, we found that it outperforms ICP-based methods such as KinFu and, at least on medium-sized scenes, often obtains a comparable performance with bundle adjustment methods such as RGB-D SLAM at a significantly reduced computational effort. In the future, we plan to include color information in camera tracking and investigate methods that allow a more efficient representation of the 3D geometry. For larger geometries, the combination of our method with a SLAM solver like [15, 11] would be interesting.

References

[1] KinectFusion Implementation in the Point Cloud Library (PCL). http://svn.pointclouds.org/pcl/trunk/.

[2] S. Agarwal, N. Snavely, I. Simon, S.M. Seitz, and R.Szeliski. Building rome in a day. In *ICCV*, 2009.

[3] J. A. Bærentzen. On the implementation of fast marching methods for 3D lattices. Technical report, Technical University of Denmark, 2001.

[4] P.J. Besl and N.D. McKay. A method for registration of 3-D shapes. *IEEE Trans. Pattern Anal. Mach. Intell.*, 14 (2):239–256, 1992.

[5] G. Blais and M.D. Levine. Registering multiview range data to create 3D computer objects. *IEEE Trans. Pattern Anal. Mach. Intell.*, 17:820–824, 1993.

[6] D. Canelhas. Scene representation, registration and object detection in a truncated signed distance function representation of 3d space. Master's thesis, 2012.

[7] B. Curless and M. Levoy. A volumetric method for building complex models from range images. In *SIGGRAPH*, 1996.

[8] F. Endres, J. Hess, N. Engelhard, J. Sturm, D. Cremers, and W. Burgard. An evaluation of the RGB-D SLAM system. In *ICRA*, May 2012.

[9] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox. RGB-D mapping: Using depth cameras for dense 3D modeling of indoor environments. 2010.

[10] H. Hirschmüller. Accurate and efficient stereo processing by semi-global matching and mutual information. In *CVPR*, 2005.

[11] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. Leonard, and Frank Dellaert. iSAM2: Incremental smoothing and mapping with fluid relinearization and incremental variable reordering. In *ICRA*, 2011.

[12] C. Kerl, J. Sturm, and D. Cremers. Robust odometry estimation for rgb-d cameras. In *ICRA*, 2013.

[13] G. Klein and D. Murray. Parallel tracking and mapping for small AR workspaces. In *ISMAR*, 2007.

[14] D.B Kubacki, H.Q. Bui, S.D. Babacan, and M.N. Do. Registration and integration of multiple depth images using signed distance function. In *SPIE, Computational Imaging X*, 2012.

[15] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. g2o: A general framework for graph optimization. In *ICRA*, 2011.

[16] William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. *Computer Graphics*, 21(4):163–169, 1987.

[17] Y. Ma, S. Soatto, J. Kosecka, and S. Sastry. *An Invitation to 3D Vision: From Images to Geometric Models*. Springer Verlag, 2003.

[18] R.A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A.J. Davison, P. Kohli, J. Shotton, S. Hodges, and A.W. Fitzgibbon. KinectFusion: Real-time dense surface mapping and tracking. In *ISMAR*, pages 127–136, 2011.

[19] C.Y. Ren and I. Reid. A unified energy minimization framework for model fitting in depth. In *Workshop on Consumer Depth Cameras for Computer Vision (CDC4CV) at ECCV*, 2012.

[20] F. Steinbrücker, J. Sturm, and D. Cremers. Real-time visual odometry from dense rgb-d images. In *Workshop on Live Dense Reconstruction with Moving Cameras at ICCV*, 2011.

[21] J. Stühmer, S. Gumhold, and D. Cremers. Real-time dense geometry from a handheld camera. In *Pattern Recognition (Proc. DAGM)*, Darmstadt, Germany, 2010.

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

[23] T.M. Tykkälä, C. Audras, and A.I Comport. Direct iterative closest point for real-time visual odometry. In *Workshop on Computer Vision in Vehicle Technology at ICCV*, 2011.

[24] T. Whelan, H. Johannsson, M. Kaess, J.J. Leonard, and J.B. McDonald. Robust real-time visual odometry for dense RGB-D mapping. In *IEEE Intl. Conf. on Robotics and Automation, ICRA*, Karlsruhe, Germany, May 2013.

[25] K.M. Wurm, A. Hornung, M. Bennewitz, C. Stachniss, and W. Burgard. OctoMap: A probabilistic, flexible, and compact 3D map representation for robotic systems. In *ICRA*, 2010.