

SDF-Tracking

Leonardo Maximilian Freiherr von Lerchenfeld

Chair for Computer Vision & Artificial Intelligence - Technical University of Munich

Abstract

Quickly generating 3D models of an indoor environment is very useful in robotics, computer vision and architecture. This report is about SDF-Tracking, which is a method for real-time camera tracking and 3D reconstruction. In this method, the surface of the environment is represented with a signed distance function (SDF) in a voxel grid. The surface is built from data fusion of depth images taken by a RGB-D sensor. To integrate the images into a 3D model, the camera poses are iteratively estimated. The tracking of the camera is solved with a novel approach: point-to-volume mapping. The evaluation of the data on publicly available benchmark shows that SDF-Tracking is as accurate as RGB-D SLAM, which uses feature-based bundle adjustment, and it is as fast as KinectFusion, which uses the iterated closest point (ICP) algorithm for point clouds.

1 Introduction and Problem Motivation

Simultaneous localization and mapping (SLAM) is a very powerful method for robots, nevertheless, it is very important that the geometry of the scene is accurate and the estimation of the pose is fast. To achieve this, Newcombe et al. [8] presented a method called KinectFusion, which uses signed distance functions (SDFs) to represent the scene geometry and the iterated closest point (ICP) algorithm for camera tracking.

This report describes SDF-Tracking, which is a method introduced by Bylow et al. in their paper *Real-Time Camera Tracking and 3D Reconstruction Using Signed Distance Functions* [2]. The novelty of their work is that the camera motion is directly estimated on the SDF instead of downsampling the SDF into a virtual camera view and therefore, the resulting 3D models are more accurate than those of KinectFusion. An example of a 3D model acquired with SDF-Tracking is illustrated in Figure 1, showing a study room. As the scanning of the room is very fast and detailed, not only robots but also architects profit from this method.

An Asus Xtion Pro Live sensor is used to scan a room. RGB-D sensors provide depth values for most pixels of an image. RGB-D sensors are active sensors because they not only receive light but also emit Infra-Red (IR) light. RGB-D sensors emit structured light into the scene and receive the reflections of the scene. According to the difference between the emitted and received structure, the depth values are calculated by the sensor itself through a triangulation process. (Larger shifts correspond to positions further from the sensor and conversely.) IR light is also sent by the sun and therefore depth images of outdoor environments are distorted. To reconstruct the 3D model, many images are integrated. In the example of Figure 1, the sensor, which records at 30 fps, provided roughly 1000 images. The model is built in approximately 30 seconds. To achieve real-time performance, a laptop with a Quadro GPU from Nvidia is used.



Figure 1: Using SDF-Tracking, a study room is reconstructed in real-time with a handheld sensor and a laptop with a Quadro GPU [2].

Section 2 describes related work such as RGB-D SLAM and KinectFusion. In Section 3, some functions are introduced that are used in the approach in Section 4. Section 5 shows the results of SDF-Tracking. Finally, a summary of the scientific contribution of SDF-Tracking and suggestions for further improvement are given in Section 6.

2 Related Work

Localization and mapping require a suitable representation of the scene geometry. For robot localization or path planning, occupancy grids or octrees are well suited [11]. 3D occupancy grid maps are used to represent the environment in RGB-D SLAM [4]. The first step of RGB-D SLAM is to extract visual features from the incoming color images. With these features, the transformation between two frames and the 3D model are then determined through bundle adjustment.

RGB-D SLAM performs well [4], but it requires on average 350 ms per frame computed with a quad-core CPU [4] or 100-250 ms per frame computed with a Quadro GPU [2]. Thus, with the sensor recording at 30 fps, RGB-D SLAM is not applicable in real-time. Newcombe et al. [8], however, demonstrated that 3D reconstruction is possible in real-time with KinectFusion.

KinectFusion estimates the pose with ICP on the predicted surface and the current sensor measurement. The surface is predicted by rendering a point cloud from the SDF into a virtual camera. Consequently, the surface from the SDF itself is not used but downsampled. Then, the ICP algorithm minimizes the error on the point clouds from the virtual camera and the depth image. Kerl et al. [5] minimize the photometric error for odometry estimation, however, without subsequent 3D reconstruction. Whelan et al. [10] combined this method with the KinectFusion approach and achieved superior tracking performance and the reconstruction of large scenes. Unfortunately, in their method, it is not possible to return with the camera to a previous location.

SDF-Tracking addresses this problem by using a voxel grid to represent the geometry. Furthermore, to achieve superior tracking performance than KinectFusion, SDF-Tracking tracks the camera pose directly on the SDF. In particular, the error between each voxel and the estimated 3D model is minimized. This camera tracking is the novelty of this approach and it is further called point-to-volume mapping. A comparison in tracking between SDF-Tracking, Kinect Fusion and RGB-D SLAM is given in Table 1. A detailed description of the camera tracking in SDF-Tracking is given in Section 4.5.

RGB-D SLAM	KinectFusion	SDF-Tracking
Features are extracted and in the next frame, the pairwise 6D transformation of the features is determined through bundle adjustment .	To predict the surface, it is rendered into a virtual camera. With the ICP algorithm, the transformation between the previous and the current camera frame is determined.	The pose transformation is determined through point-to-volume mapping , which minimizes the error between each voxel and the estimated 3D model.

Table 1: Comparison in Tracking between RGB-D SLAM, KinectFusion, and SDF-Tracking

3 Notation and Preliminaries

A 3D point is denoted as $\mathbf{x} \in \mathbb{R}^3$. The rotation of the camera is defined as $R \in SO(3)$ and the translation as $\mathbf{t} \in \mathbb{R}^3$. To extract the color and depth information from the pixels, the following functions are used:

$$I_{RGB} : \mathbb{R}^2 \rightarrow \mathbb{R}^3 \quad (1)$$

$$I_d : \mathbb{R}^2 \rightarrow \mathbb{R} \quad (2)$$

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

$$\pi(x, y, z) = \left(\frac{f_x x}{z} + c_x, \frac{f_y y}{z} + c_y \right)^\top \quad (3)$$

Conversely, a 3D point can be reconstructed from a pixel $(i, j)^\top \in \mathbb{R}^2$ with depth $z = I_d(i, j)$ by

$$\rho(i, j, z) = \left(\frac{(i - c_x)z}{f_x}, \frac{(j - c_y)z}{f_y}, z \right) \quad (4)$$

4 Approach

SDF-Tracking consists of five components, which are visualized in Figure 2 and which are described in each subsection in detail. In the following, the components are briefly summarized:

1. Representation of the SDF: The scene geometry is represented in voxel grids, where each voxel stores the distance to the surface of the scene.
2. Distance and Weighting Functions: In this section, two different distance functions are presented, which quickly estimate the distance between a voxel and the surface. Additionally, a truncation and a weighting function are applied on the distances in order to achieve higher robustness.
3. Data Fusion and 3D Reconstruction: All distance measurements are fused together to obtain the optimal SDF $\psi(\mathbf{x})$.
4. Color and Mesh: The surface points are obtained from the SDF. These points are then meshed together to obtain the surface. Afterwards, the surface is colored.
5. Camera Tracking: The transformation between two frames is determined such that the measured surface lies as close as possible to the estimated surface.

Note that in the original paper [2], camera tracking is described before SDF modeling. This is changed in this report because it is easier to understand camera tracking if one knows how the SDF is built.

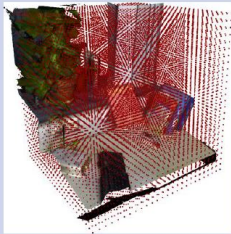
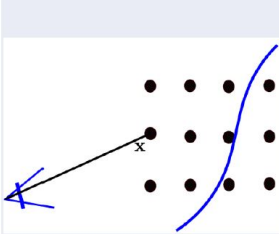
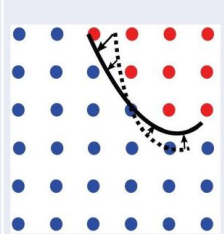
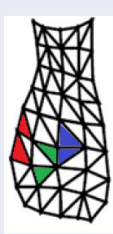
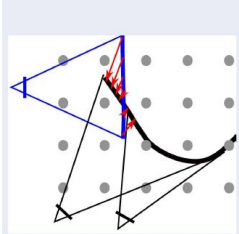
Representation of the SDF	Distance and Weighting Functions	Data Fusion and 3D Reconstruction	Color and Mesh	Camera Tracking
Voxel grids	Projective distance Weighting functions	Optimal SDF ψ	Surface Color	Rotation R Translation \mathbf{t}
				

Figure 2: Visualization of the approach

4.1 Representation of the SDF

The SDF $\psi(\mathbf{x})$ returns for any point $\mathbf{x} \in \mathbb{R}^3$ the signed distance from \mathbf{x} to the surface. The SDF is represented in a discrete voxel grid of resolution m . The voxel grid consists of six channels: averaged distance (D), sum of all weights for the distance (W), color weights (W_c), and red- (R), green- (G), and blue-values (B). The allocation of the grid in memory is then, e.g., for distance:

$$D : [0, \dots, m-1]^3 \rightarrow \mathbb{R} \quad (5)$$

To reconstruct a volume, e.g., a room, of dimensions *width* x *height* x *depth*, a world point $\mathbf{x} = (x, y, z)^\top \in \mathbb{R}^3$ is mapped to voxel coordinates by

$$\begin{pmatrix} i \\ j \\ k \end{pmatrix} = m \begin{pmatrix} x/\text{width} + 0.5 \\ y/\text{height} + 0.5 \\ z/\text{depth} + 0.5 \end{pmatrix} \quad (6)$$

In the following section, it is described how the distances, which are signed, between voxels and the surface are estimated. The surface is where the distances have a value of zero. However, it is very unlikely that voxels have a value of exactly zero because they might have instead values very near to zero. Thus, one obtains a point of the exact surface position by tri-linear interpolation of eight neighboring voxels if they have positive and negative values.

4.2 Distance and Weighting Functions

To integrate a new depth image into the 3D model, it is necessary to know which voxels have been observed by the RGB-D sensor. This can be done either by projecting each pixel onto the voxel grid or in the opposite direction by projecting each voxel onto the image plane. The latter is preferred because it ensures that each voxel is visited exactly once. Another advantage is that the voxels are independent of their neighbors, hence, the process can be parallelized on the GPU. This operation requires that the distance of each voxel to the observed surface has to be computed. Computing the true distance of a voxel is time consuming as it requires the computation of the shortest path over the whole volume. Consequently, for a real-time application, the true distance cannot be computed.

Therefore, projective distances are computed. In this approach, two metrics are investigated: Point-To-Point and Point-To-Plane. In KinectFusion, the Point-To-Plane metric is used. In Section 5, the two metrics are evaluated and compared. Both return distances that are signed, i.e., negative values are assigned to voxels in front of the observed surface and positive values to voxels behind.

4.2.1 Projective Point-To-Point

Given the pose of the camera R, \mathbf{t} , a world point \mathbf{x}^G can be transformed into camera coordinates with

$$\mathbf{x} = (x, y, z)^\top = R^\top (\mathbf{x}^G - \mathbf{t}). \quad (7)$$

According to the camera model presented in Section 3, this point can be projected onto pixel coordinates by

$$(i, j)^\top = \pi(\mathbf{x}) \quad (8)$$

The Projective Point-To-Point distance is then defined as

$$d_{\text{point-to-point}}(\mathbf{x}) := z - I_d(i, j) \quad (9)$$

In this procedure, the projective distance from \mathbf{x} to the surface is determined. First, the global coordinates are transformed into the local camera coordinates. Second, the coordinates are further transformed into the pixel coordinates resulting in a depth value z . Finally, the observed depth value for this pixel is subtracted from the calculated depth value z . As this distance is only an approximation, it depends heavily on the orientation of the camera and becomes less accurate the less the viewing angle is orthogonal to the surface. This problem is considered in the Point-To-Plane metric.

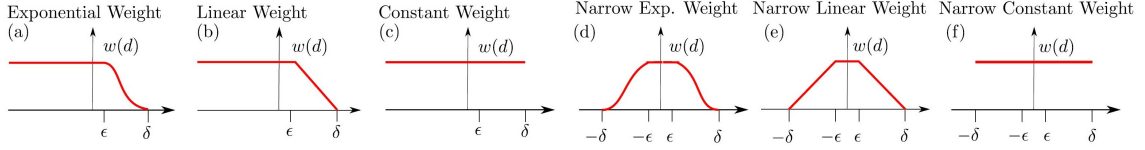


Figure 3: Six different weighting functions for the distances [2]

4.2.2 Projective Point-To-Plane

First, a bilateral filter is applied to the depth image in order to smooth it. (Note that bilateral filters replace the intensity of each pixel with a weighted average of intensity values from nearby pixels while preserving sharp edges.) Second, the normals for all pixels are computed.

Given a voxel \mathbf{x} , the corresponding pixel coordinates (i, j) can be computed and the observed surface normal can be read out with $\mathbf{n}(i, j)$. A point \mathbf{y} is defined as the point on the surface that lies on the ray from the camera to the voxel \mathbf{x} . The Point-To-Plane distance is then, the distance between the voxel \mathbf{x} and the point \mathbf{y} , in the direction of the normal:

$$d_{point-to-plane}(\mathbf{x}) := (\mathbf{y} - \mathbf{x})^\top \mathbf{n}(i, j) \quad (10)$$

4.2.3 Truncation

Both distance metrics are only approximations. For large distances, the error can be very significant. Nevertheless, for reconstruction (Section 4.3) and tracking (Section 4.5), only the region around the zero-crossing is interesting. To make tracking and reconstruction more robust, large distances are truncated as follows:

$$d_{trunc} = \begin{cases} -\delta & \text{if } d < -\delta \\ d & \text{if } |d| \leq \delta \\ \delta & \text{if } d > \delta \end{cases} \quad (11)$$

4.2.4 Weighting Functions

In addition to truncation, tracking and reconstruction can be made more robust by applying a weighting function on the observed distances. Three different weighting functions are considered - constant, linear, and exponential - each with a narrowed version as well, for a total of six variations. You can see the different weighting functions in Figure 3. All the functions assign a weight of zero for $d > \delta$ because behind the surface the distances of voxels are difficult to estimate. Therefore, these distances will not be considered. In the narrowed version, a weight of zero is also assigned for $d < -\delta$, since only a small band around the zero crossing is interesting for tracking and reconstruction.

A constant weight in the region between $-\delta < d < \delta$ can be considered as applying no weight at all. This works for sensors that can deeply penetrate objects such as a radar. However, the constant weighting functions is not the optimal one for RGB-D sensors. In KinectFusion, a linear weighting function is used. Moreover, an exponential weighting function shall be evaluated. The exponential function is motivated by a Gaussian distribution. In Section 5, you will read that the exponential weighting function leads to robust results.

4.3 Data Fusion and 3D Reconstruction

After taking a sequence of measurements, they need to be fused together to obtain the best estimate for the SDF $\psi(\mathbf{x})$. Assuming that all distance measurements are normally distributed, the problem can be formulated as determining the distance ψ that maximizes the observation likelihood of the measured distances:

$$p(d_1, w_1, \dots, d_n, w_n | \psi) \propto \prod_{i=1}^n \exp\left(-\frac{1}{2} w_i (\psi - d_i)^2\right) \quad (12)$$

Luckily, monotonic functions preserve critical points, hence, one can apply the logarithm. Instead of maximizing the likelihood, one can also minimize the negative likelihood. Thus, one obtains the following quadratic error function:

$$L(\psi) = \sum_{i=1}^n \frac{1}{2} w_i (\psi - d_i)^2 \quad (13)$$

To obtain the optimal solution, the error function L can be derived with ψ and then set equal to zero.

$$\frac{\partial L}{\partial \psi} = 0 = \sum_{i=1}^n \frac{1}{2} w_i (2\psi - 2d_i) \quad (14)$$

$$\sum_{i=1}^n w_i \psi = \sum_{i=1}^n w_i d_i \quad (15)$$

$$\psi = \frac{\sum_{i=1}^n w_i d_i}{\sum_{i=1}^n w_i} \quad (16)$$

As a result, the optimal ψ is the weighted average of all measurements. Therefore, the distance of each voxel can be calculated as a running weighted average, i.e.,

$$D \leftarrow \frac{WD + w_{n+1}d_{n+1}}{W + w_{n+1}} \quad (17)$$

$$W \leftarrow W + w_{n+1} \quad (18)$$

This computation needs to be done for each voxel. Fortunately, the voxels do not depend on their neighbors, so the update can be computed in parallel on the GPU.

4.4 Color and Mesh

One can calculate the pixel coordinates from the global coordinates of the voxels and then retrieve the observed color by

$$(r, g, b)^\top = I_{RGB}(i, j) \quad (19)$$

The color updates can be formulated in a running average as well, for instance, green-values (G) can be updated with

$$G \leftarrow \frac{W_c G + w_c^{n+1} g}{W_c + w_c^{n+1}} \quad (20)$$

$$w_c^{n+1} = w_{n+1} \cos \theta \quad (21)$$

In the previous equation, w_c^{n+1} is the weight for the new measurement and θ is the angle between the ray and the principal axis. As a weighting function, the cosine function is an adequate candidate, so weights are higher for those pixels whose normal is pointing towards the camera.

The surface is located at the zero crossing in the SDF. To extract the corresponding triangle mesh, a straight forward implementation of the marching cubes algorithm [6] can be applied to obtain the geometry of the scene, i.e., a cube containing eight neighboring voxels “marches” through the entire voxel grid and extracts a point by tri-linear interpolation each time the distances of the eight voxels have both negative values (in front of the surface) and positive values (behind the surface).

Inspiration for the color texture is given in [10]. From a voxel that is sufficiently close to the surface, the color information is retrieved from the color channels: R, G, B . The color of the surface is then computed using tri-linear interpolation. Figure 4 illustrates how an object is firsts converted into a point cloud, then meshed into a surface, and finally colored.

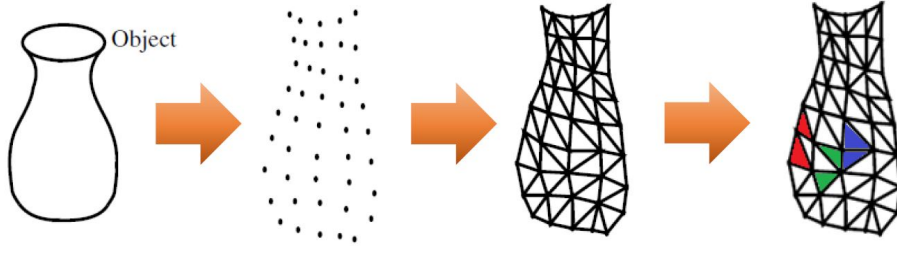


Figure 4: From an object to a point cloud to a triangle mesh [3] to a colored surface

4.5 Camera Tracking

This part presents how the transformation of the camera between two frames is estimated. The RGB-D sensor provides for each pixel (i, j) its depth $z = I_d(i, j)$. Then, the camera coordinates can be calculated with $\rho(i, j, z)$ and these coordinates can be further transformed into global coordinates for each pixel (i, j) using

$$x_{ij}^G = R\mathbf{x}_{ij} + \mathbf{t} \quad (22)$$

Now the camera pose should be determined such that all projected points from the depth image lie as close as possible to the estimated surface (= zero-crossing in the SDF). This idea is illustrated in Figure 5, where the blue line is the surface observed from the RGB-D sensor and that should fit as good as possible to the estimated surface, which is illustrated by the bold black line.

With the assumption that the depth measurements have a Gaussian noise and that all pixels are independent and identically distributed, the likelihood of observing the surface given a camera pose R, \mathbf{t} becomes

$$p(I_d|R, \mathbf{t}) \propto \prod_{i,j} \exp(-\psi(R\mathbf{x}_{ij} + \mathbf{t})^2) \quad (23)$$

Again, we can apply the logarithm. Then, instead of maximizing the likelihood, the negative likelihood can be minimized. Consequently, the error function is defined as

$$E(R, \mathbf{t}) = \sum_{ij} \psi(R\mathbf{x}_{ij} + \mathbf{t})^2 = \sum_{ij} \psi_{ij}^2 \quad (24)$$

In practice, the error will never be zero due to noise and due to the fact that not all pixels in the depth image are defined.

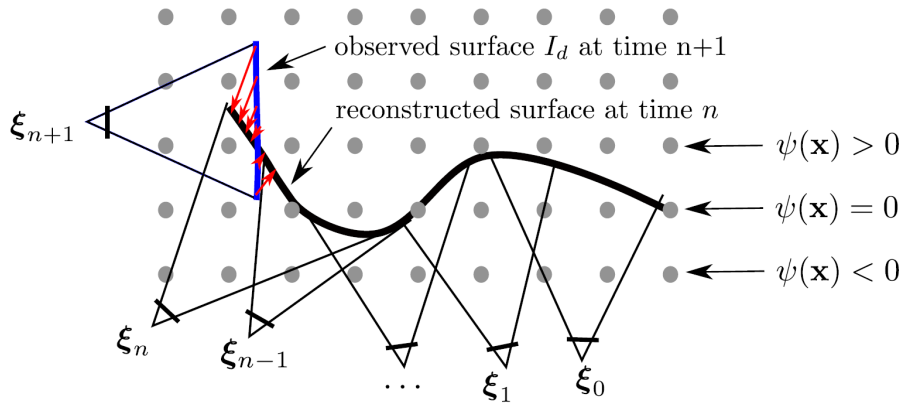


Figure 5: Camera Tracking: Determine the homogenous transformation between two camera frames such that the observed surface lies as close as possible to the estimated surface [2]

A rigid-body motion can be described with a rotation matrix R and a translational vector \mathbf{t} or with twist coordinates [7]. In the twist coordinates ξ , \mathbf{v} is referred as the linear component and ω as the angular component. The homogeneous transformation matrix is then defined as

$$\begin{bmatrix} R & \mathbf{t} \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} e^{\hat{\omega}} & (I - e^{\hat{\omega}})\hat{\omega}v + \omega\omega^\top v \\ 0 & 1 \end{bmatrix} \quad (25)$$

In the following, twist coordinates are used because that is a minimal representation of the rigid-body motion, i.e., it has only six degrees of freedom ($v_1, v_2, v_3, \omega_1, \omega_2, \omega_3$). Moreover, they facilitate optimization because then, it must not be ensured that R is a special orthogonal matrix. Now the error function is formulated as

$$E(\xi) = \sum_{i,j} \psi_{ij}(\xi)^2 \quad (26)$$

The optimal solution is found by deriving the equation and setting it to zero. However, this becomes quite complex for a 6D vector. For this reason, the Gauss-Newton method is applied. First, the Taylor approximation is used and then, all non-linear terms are removed. Note that this approximation only holds if the curvature is close to zero.

$$\psi(\xi) \approx \psi(\xi^{(k)}) + \nabla\psi(\xi^{(k)})^\top (\xi - \xi^{(k)}) \quad (27)$$

In the previous equation, $\xi^{(k)}$ describes ξ at the k -th iteration step. Fortunately, now $\psi(\xi)$ is 1D, so it can easily be derived and set to zero to obtain the optimal solution iteratively.

$$\nabla\psi_{ij} = \left(\frac{d\psi_{ij}(\xi)}{d\omega_1}, \frac{d\psi_{ij}(\xi)}{d\omega_2}, \frac{d\psi_{ij}(\xi)}{d\omega_3}, \frac{d\psi_{ij}(\xi)}{dv_1}, \frac{d\psi_{ij}(\xi)}{dv_2}, \frac{d\psi_{ij}(\xi)}{dv_3} \right)^\top \quad (28)$$

$$E_{approx}(\xi) = \sum_{i,j} \left(\psi_{ij}(\xi^{(k)}) + \nabla\psi_{ij}(\xi^{(k)})^\top (\xi - \xi^{(k)}) \right)^2 \quad (29)$$

$$\frac{d}{d\xi} E_{approx}(\xi) = \sum_{i,j} 2\psi_{ij}(\xi^{(k)})\nabla\psi_{ij}(\xi^{(k)}) + 2 \left(\nabla\psi_{ij}(\xi^{(k)})\nabla\psi_{ij}(\xi^{(k)})^\top (\xi - \xi^{(k)}) \right) = 0 \quad (30)$$

To make this equation more compact, a matrix \mathbf{A} and a vector \mathbf{b} are defined for every pixel according to

$$\mathbf{A}_{ij} := \nabla\psi_{ij}(\xi^{(k)})\nabla\psi_{ij}(\xi^{(k)})^\top \in \mathbb{R}^{6 \times 6} \quad (31)$$

$$\mathbf{b}_{ij} := \psi_{ij}(\xi^{(k)})\nabla\psi_{ij}(\xi^{(k)}) \in \mathbb{R}^{6 \times 1} \quad (32)$$

$$(33)$$

Summing up over all pixels, Equation 30 becomes

$$\frac{d}{d\xi} E_{approx}(\xi) = \mathbf{b} + \mathbf{A}(\xi - \xi^{(k)}) = 0 \quad (34)$$

$$\mathbf{A}\xi = \mathbf{A}\xi^{(k)} - \mathbf{b} \quad (35)$$

The camera pose that minimizes the linearized error is formulated as

$$\xi^{(k+1)} = \xi^{(k)} - \mathbf{A}^{-1}\mathbf{b} \quad (36)$$

This iteration is repeated until either the change $\|\xi^{k+1} - \xi^{(k)}\|_\infty$ is small enough or a certain number of iterations is reached. In order to obtain real-time performance, all vectors \mathbf{b}_{ij} and matrices \mathbf{A}_{ij} are computed in parallel on the GPU since they are independent of each other.

To sum up, after some iterations, the vector ξ is calculated that describes the transformation between two camera frames in twist coordinates. Consequently, the camera is tracked within the reconstructed 3D model.

5 Results

In this section, the results of SDF-Tracking are presented and compared to KinectFusion and RGB-D SLAM on the TUM RGB-D benchmark [9]. Furthermore, the optimal parameters for the truncation as well as the distance and weighting functions are determined. Finally, the memory and runtime consumption are evaluated.

In Figure 1, you see that the reconstructed 3D model is highly detailed. In Table 2, RGB-D SLAM, KinectFusion, and two versions of SDF-Tracking, each using a different distance metric, are compared. SDF-Tracking is more accurate than RGB-D SLAM for small scenes. Using a higher resolution, e.g., $m = 512$, it can also be more accurate for larger scenes. (You can see all the results in the original paper [2].) SDF-Tracking clearly outperforms KinectFusion in terms of accuracy and robustness. Probably, this is due to the fact that KinectFusion projects the SDF down to a virtual camera view for tracking. Another result from Table 2 is that the Point-To-Point metric is more accurate than the Point-To-Plane metric, which is used by KinectFusion.

Method	Teddy	F1 Desk	F1 Floor	F1 360	F1 Room	F1 Plant
KinectFusion	0.156 m	0.057 m	Failed	0.913 m	Failed	0.598 m
Point-To-Plane	0.072 m	0.087 m	0.811 m	0.533 m	0.163 m	0.047 m
Point-To-Point	0.086 m	0.038 m	0.641 m	0.420 m	0.121 m	0.047 m
RGB-D SLAM	0.111 m	0.026 m	0.035 m	0.071 m	0.101 m	0.061 m

Table 2: Comparison of RGB-D SLAM, KinectFusion and SDF-Tracking (using a resolution of $m = 512$ in each dimension) in the root mean squared error on the TUM RGB-D Dataset [9]

The results of Table 2 are created by using a truncation of $\delta = 0.3$ m and $\epsilon = 0.025$ m. Unfortunately, it is not mentioned in the original paper [2] which weighting function is used. However, in another evaluation, which compares only the weighting functions, the exponential weight leads to more robust tracking (meaning that the error is usually low), although the linear weight is slightly more accurate (meaning that normally this error is lower than the one of the exponential weight, but it can be much higher some times). The constant weight leads, as expected, to worse tracking results and the narrow weighting functions usually lead to worse tracking results as well.

For a resolution of $m = 256$, SDF-Tracking consumes on average around 23 ms and thus, runs easily in real-time on 30 fps. In comparison, KinectFusion consumes at the same resolution on the same hardware 20 ms per frame and RGB-D SLAM requires around 100 - 250 ms. Nevertheless, for larger scenes RGB-D SLAM is more accurate than SDF-Tracking in a resolution of $m = 256$, so it might be desired to use a resolution of $m = 512$. However, for a resolution of $m = 512$, SDF-Tracking takes 53 ms per frame for pose optimization and data fusion and thus does not run in real-time on 30 fps. Fortunately, SDF-Tracking is very robust, hence, every third image can be used without losing too much accuracy. Note that this statement is very vague because it is not mentioned how fast the camera moves.

The memory consumption grows cubically with the size of the resolution. For a resolution of $m = 256$, SDF-Tracking 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 the double amount for the color grid. KinectFusion also uses a voxel grid to represent the SDF but it does not save the color information (which is possible though), so the memory consumption of KinectFusion is one third of SDF-Tracking. RGB-D SLAM in contrast, uses an octree representation, which requires less memory than the other two methods.

6 Conclusion

In the original paper of SDF-Tracking [2], Bylow et al. mentioned that SDF-Tracking can also be used for controlling the position of an autonomous quadcopter. In this report, the control part was left out because it is not sufficiently described how position control can be ensured. However, it is worthwhile to mention it here since SDF-Tracking has the potential to contribute another direct benefit in the field of robotics.

SDF-Tracking is an approach to estimate the camera motion directly on the signed distance function. This method allows to generate a 3D model of an indoor environment in real-time. The evaluation on a public RGB-D benchmark shows that SDF-Tracking, which involves point-to-volume mapping, clearly outperforms ICP-based methods such as KinectFusion in accuracy and robustness. Moreover, it has a comparable performance to bundle adjustment methods such as RGB-D SLAM, while it is much faster than RGB-D SLAM. However, the memory consumption in SDF-Tracking is enormous, especially for saving the color grid. The color grid requires double the amount of memory, although it is not used for tracking itself. This was further improved by Bylow et al. in [1] in order to make the tracking more robust. Nevertheless, the memory consumption is significant, hence, this approach cannot be used for the reconstruction of large geometries.

References

- [1] E. Bylow, C. Olsson, and F. Kahl. Robust camera tracking by combining color and depth measurements. In *ICPR*, 2014.
- [2] E. Bylow, J. Sturm, C. Kerl, F. Kahl, and D. Cremers. Real-time camera tracking and 3d reconstruction using signed distance functions. In *RSS*, 2013.
- [3] B. Curless and M. Levoy. A volumetric method for building complex models from range images. In *SIGGRAPH*, 1996.
- [4] F. Endres, J. Hess, N. Engelhard, J. Sturm, D. Cremers, , and W. Burgard. An evaluation of the rgb-d slam system. In *ICRA*, 2012.
- [5] C. Kerl, J. Sturm, and D. Cremers. Robust odometry estimation for rgb-d cameras. In *ICRA*, 2013.
- [6] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *Computer Graphics*, page 21(4):163–169, 1987.
- [7] Y. Ma, S. Soatto, J. Kosecka, and S. Sastry. *An Invitation to 3D Vision: From Images to Geometric Models*. Springer Verlag, 2003.
- [8] 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*,, page 127–136, 2011.
- [9] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A benchmark for the evaluation of rgb-d slam systems. In *IROS*, 2012.
- [10] T. Whelan, H. Johannsson, M. Kaess, J.J. Leonard, and J.B. McDonald. Robust real-time visual odometry for dense rgb-d mapping. In *ICRA*, 2013.
- [11] 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.