Robotics and Autonomous Systems, 162:104337, Elsevier, April 2023.
https://doi.org/10.1016/j.robot.2022.104337

Rendering the Directional TSDF for Tracking and Multi-Sensor Registration with Point-To-Plane Scale ICP

Malte Splietker and Sven Behnke

***Abstract*— Dense real-time tracking and mapping from RGB-D images is an important tool for many robotic applications, such as navigation and manipulation. The recently presented Directional Truncated Signed Distance Function (DTSDF) is an augmentation of the regular TSDF that shows potential for more coherent maps and improved tracking performance. In this work, we present methods for rendering depth- and color images from the DTSDF, making it a true drop-in replacement for the regular TSDF in established trackers. We evaluate the algorithm on well-established datasets and observe that our method improves tracking performance and increases re-usability of mapped scenes. Furthermore, we add color integration which notably improves color-correctness at adjacent surfaces. Our novel formulation of combined ICP with frame-to-keyframe photometric error minimization further improves tracking results. Lastly, we introduce Sim(3) point-to-plane ICP for refining pose priors in a multi-sensor scenario with different scale factors.**

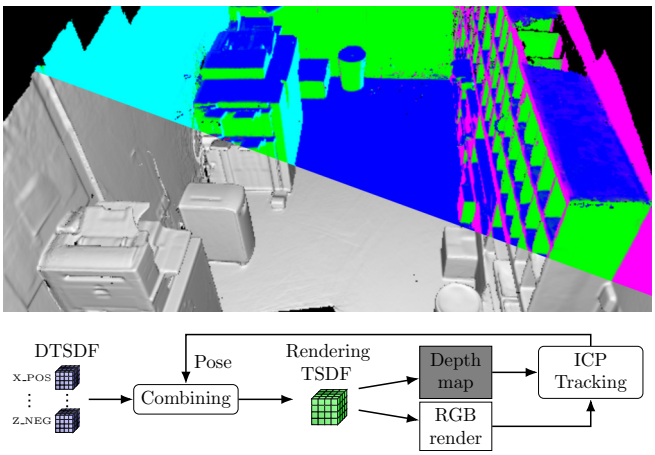


Figure 1: The top part shows a cut view of the reconstructed Stanford copyroom scene. The lower left half is the rendered depth; upper right shows the directions involved in rendering in different colors, mixed whenever multiple directions contributed. The bottom part shows the pipeline for rendering and tracking with the DTSDF.

1 Intoduction

Since its first appearance in KinectFusion [1], GPU accelerated TSDF algorithms have become a de-facto standard in scene reconstruction from depth images, leveraging inexpensive sensors and massive parallel processing on GPUs for good real-time performance. By modeling the closest distance to the next surface with a signed distance function (SDF), geometry can be reconstructed by finding the zero-transition from positive (i.e., in front of the surface) to negative (i.e., behind the surface) values. In practice, this function is obtained by fusing measurements into a regular grid, the so called voxels, and interpolating between them. The necessity to store both front- and backside of the surface implies, however, that there is a minimum thickness of objects that can be represented. Especially with thin objects, integration of new measurements might interfere with and contradict old data belonging to a different surface, leading to a corrupted model. We have explored this issue and introduced the concept of the Directional Truncated Signed Distance Function (DTSDF) in our previous work [2]. The DTSDF[1] uses six TSDF volumes, one for each positive and negative coordinate axis, to store surface sections with different orientations. We proposed a method for fusing depth images into this data structure and for extracting meshes with a modified marching cubes algorithm. The latter is, however, not applicable for real-time tracking applications.

Instead, in this work we propose methods for rendering virtual camera views, which allows to use the standard ICP algorithm for real-time sensor motion tracking. Moreover, we introduce color integration into the DTSDF, which helps preserving color details of adjacent object surfaces, especially along sharp edges. With these additions, the DTSDF becomes a true replacement for the regular TSDF with only minor modifications to the fusion and rendering algorithms. We showcase and evaluate our method on well-known datasets and deduce, that the DTSDF has advantages in tracking certain types of sequences and is better at preserving the overall map for later reuse.

Throughout the related work the ICP algorithm has been implemented with some variations, some of which we discuss in Section 5. Especially, we investigate the use of photometric ICP with the TSDF and found, that there are different variations of frame-to-frame and frame-to-model photometric ICP, but no frame-to-keyframe. We re-formulate the error function accordingly and conduct a case study which indicates, that frame-to-keyframe is favorable in most sequences.

As part of the PhenoRob Cluster of Excellence [2] we are involved in a project to create 3D reconstructions of crops captured by a robot on the field. One of the challenges is to align the multiple point clouds from high-accuracy laser depth- and RGB stereo sensors. Even given an offline pose prior of the sensors, each scan requires pose refinement, due to the limited rigidity of the robot's chassis. The computed depth from the stereo-pairs is also affected by geometric changes, so in Section 6 we showcase our method by extending the point-to-plane **SE**(3) ICP algorithm to the **Sim**(3) Group, which jointly optimizes translation, rotation and scale.

2 Related Work

In 3D reconstruction and SLAM feature-based, sparse, and dense methods are distinguished. Our work belongs to the category of dense methods that describe closed surfaces, separate objects, and even free space [3].

Research focus in this field has shifted in recent years towards learned representations. Occupancy networks [4] learn a binary descriptor describing the occupancy of space, i.e., whether a point lies inside an object or not. In DeepSDF by Park et

[1] Code available at `https://github.com/AIS-Bonn/DirectionalTSDF`

[2] Cluster of Excellence PhenoRob – Robotics and Phenotyping for Sustainable Crop Production `https://www.phenorob.de`

al. [5], a representation is learned, which like our work allows querying the signed distance to the closest object for arbitrary points in space. Neural Radiance Fields (NeRF) [6, 7] use deep networks to regress density and color. While these approaches show impressive results and use less memory to store the model or even enable scene completion, they have some shortcomings. The limited model size results in a lack of detail for large scenes. Also training and inference times, though improving lately, are still not applicable for real-time applications. Hence, the classic TSDF fusion algorithms are still state-of-the-art in live mapping scenarios.

Since its first occurrence, the TSDF fusion algorithm has seen widespread use cases and is mostly used in its original form without changes to the representation. There have been attempts to augment it, though. Dong et al. [8] created a hybrid data structure, combining the TSDF with probabilistic surfels. Multiple overlapping TSDF sub-volumes are used in pose graphs for large-scale SLAM, enabling re-aligning parts of the map on loop closure detection for consistency [9, 10, 11]. This approach is similar to the DTSDF in the way it maintains several overlapping representations, but does not fix the TSDF's inability to represent thin objects observed from opposite sides within the same volume. While this may not be an issue for some applications, object scans and walk-around type scenes profit from this capability.

Zhang et al. [12] give an overview of current RGB-D SLAM algorithms. The typical method for localizing the sensor pose in the TSDF is frame-to-model geometric ICP [1], where the current depth image is registered against a point cloud rendered from the TSDF at the previous position. It uses the point-to-plane metric and Gauss Newton for minimizing the sum of squared errors. There are modified versions, such as the extended ICP tracker [13], which uses the Huber-norm and has advanced outlier detection. Nguyen et al. [14] model the depth dependent noise of the RGB-D sensor in a weighted ICP scheme. Xia et al. [15] use a simplified weight based on the inverse quadratic depth. But the regular ICP is still most commonly used.

Photometric ICP instead uses a registration loss that is based on the per-pixel photometric error of RGB images [16]. This is helpful for preventing drift in geometrically ambiguous scenes, e.g., textured planar surfaces. To take advantage of both, color and depth, combined ICP optimizes both error functions simultaneously [17, 9, 10, 18].

Common for most TSDF ICP implementations is, that they register the current input depth- and color images against point clouds and color images rendered from the TSDF (frame-to-render). Some combined ICP variants use frame-to-render for geometric- and frame-to-frame for photometric registration [10, 13]. Moreover, there is the direct volume matching line of algorithms that directly performs registration within the SDF. Point-to-SDF [19, 20, 21] and SDF-to-SDF [22] approaches can be distinguished. Millane et al. [23] recently proposed a method for extracting and matching local features directly on the SDF. Model-less RGB-D SLAM systems like DVO [17] or ORB-SLAM [24] often use keyframes combined with a pose graph. Even without graph optimization, the use of keyframes has an advantage over frame-to-frame tracking which tends to accumulate drift faster. To our knowledge, this has not been used in combination with the TSDF yet. Section 5.2 discusses and compares the different option choices.

Further hybrid approaches utilize other tracking sources. BundleFusion [18] is an advanced method that combines ICP error minimization and visual SIFT features in a global bundle adjustment, and then de- and reintegrates parts of scene to keep the overall representation consistent.

The goal of this work is to make DTSDF usable as replacement or supplement for the regular TSDF. To be able to profit from established tracking methods without further modifications we

- introduce color fusion into the DTSDF,
- present an efficient method for generating rendered views of the DTSDF,
- use these rendered views to track sensor motion with the combined ICP algorithm,
- derive a formulation for combined ICP with keyframes for the photometric error,
- Implement $\mathbf{Sim}(3)$ point-to-plane ICP for jointly optimizing pose and scales in a multi-sensor setup.

3 Fusion and Weights

Formally, the signed distance function $\Phi : \mathbb{R}^3 \longrightarrow (d, w_d, \mathbf{c}, w_c)$ maps an arbitrary point in space to a tuple comprising signed distance to the closest surface d, distance weight w_d, RGB color $\mathbf{c}$ and color weight w_c, where the weights represent the confidence of the integrated information. The surfaces of the environment are determined by finding zero-transition of Φ, i.e., finding the subset in $\mathbb{R}^3$ where the signed distance turns from positive (in front of surface) to negative (behind surface) values. As reconstruction only requires information close to the actual surface, the TSDF only maps points within a truncation band τ around the actual surface. In practice, the TSDF is stored as a evenly-spaced grid of voxels and the signed distance, color, and weights in Φ for an arbitrary point in $\mathbb{R}^3$ are estimated by linear interpolation between tuples stored in the neighboring voxels.

The directional TSDF [2] $\Phi_{\text{dir}}(\mathbf{p}) = (\Phi^D(\mathbf{p}))_{D \in \text{Directions}}$ extends this representation by mapping a point to multiple signed distance functions – one for each direction $\{X^+, X^-, Y^+, Y^-, Z^+, Z^-\}$ – corresponding to the positive and negative coordinate axes $\mathbf{v} = \{(1,0,0)^\intercal, \cdots, (0,0,-1)^\intercal\}$. Observed depth points are assigned to those directions D that fulfill

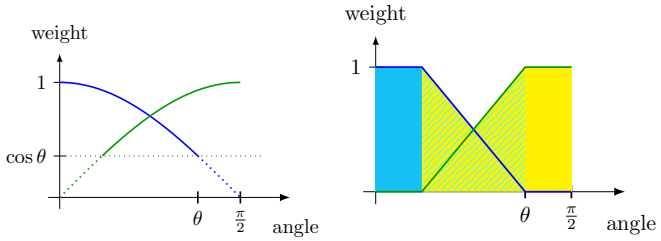
$$\arccos\langle \mathbf{n}, \mathbf{v}^D \rangle < \theta \tag{1}$$

for depth normal $\mathbf{n}$ and angular threshold $\theta \in (\pi/4, \pi/2]$, i.e., the angular difference between surface normal and direction vector is smaller than threshold θ. The range for θ is chosen, such that an overlap between neighboring directions is guaranteed and every surface point matches at least one and at most three directions.

Fusion is the process of integrating new observations into the voxels as weighted cumulative moving average, where weights denote the certainty of the added information. For every voxel and associated depth point, a fusion weight is calculated. While there is no definite weighting scheme, most implementations use a combination of factors to compensate for measurement noise and for uncertainty by down-weighting voxels behind the surface [1, 25, 14]. In addition to these factors, the fusion weight in [2] includes a direction factor $\hat{w}^D_{\text{dir}} = \langle \mathbf{n}, \mathbf{v}^D \rangle$ to blend surfaces which are represented by multiple directions (c.f. Eq. (1)) over the whole span of $[0, \pi/2]$. Then, all $\hat{w}^D_{\text{dir}}$ are explicitly compared to threshold $\cos(\theta)$ and fused, if smaller. Instead, we propose the membership function

$$w^D_{\text{dir}}(\mathbf{n}) = \min\left(\max\left(\frac{1 - \arccos\langle \mathbf{n}, \mathbf{v}^D \rangle}{2\theta - \frac{\pi}{4}}, 0\right), 1\right) \tag{2}$$

which has multiple advantages, as illustrated in Figure 2. Firstly, for angles larger than θ the weight becomes zero, so explicit thresholding becomes superfluous. In the exclusive area, where the point belongs to exactly one direction, the weight is one.



(a) Old direction weight from [2] **(b)** New direction weight w^D_{dir}

Figure 2: Direction weight for angle $\arccos\langle \mathbf{n}, \mathbf{v}^D \rangle$ between surface normal $\mathbf{n}$ and two neighboring direction vectors (e.g., $\mathbf{v}^{X^+}$ and $\mathbf{v}^{Y^+}$) indicated by the blue and green lines, respectively. The angular threshold θ determines the width of exclusive (solid) and overlapping (hatched) areas of the two directions.

The blending area, where a point belongs to multiple directions, now blends linearly on the full $[0, 1]$ range, whereas the old function had an effective range of $[\cos\theta, 1]$. Just like in the previous work [2], the combined depth fusion weight for fusing a point into direction D is

$$w_d = w_{\text{depth}} \cdot w_{\text{angle}} \cdot w^D_{\text{dir}}. \tag{3}$$

The weight function parameters were omitted for improved readability.

Color is fused analogous to distances, but with a different weight. Again, there are different variants throughout literature. Dryanovski et al. [26] use the same constant weight for depth and color to save computation time. Whelan et al. [10] use angle between depth normal and view ray to downweight steep observation angles, which Bylow et al. [25] use in combination with the depth weight. This factor is also included in our depth weight. We argue that using the depth fusion weight for colors is important, as the uncertainty is reflected in the choice of voxels associated with pixels. Let $\mathbf{x} \in \mathbb{R}^3$ be the voxel location and $\mathbf{p} \in \mathbb{R}^3$ the depth point with associated color that is fused into the voxel. Then our color fusion weight is

$$w_c = w_d \left(1 - \min\left(1, \frac{||\mathbf{p} - \mathbf{x}||}{\tau}\right)\right), \tag{4}$$

where w_d is the depth fusion weight. The factor in parenthesis reduces the confidence for voxels further away from the surface, as multiple colors from various observations may blend together here. For depth fusion we use the point-to-plane metric, which mitigates this issue, but there is no equivalent for colors, as their information is only accurate right at the surface.

Free space, that is space between camera and observed surface, is not explicitly mapped to save memory. Nonetheless, due to noise, sensor error, or dynamic objects it can happen that spurious measurements are mapped in space, that is unoccupied in reality, and it is important to remove these artifacts. When the computed distance Eq. (5) is larger than the truncation range τ, the voxel is located in free space and updated with a SDF value of 1 and a constant weight. No directional weight is used in this case, as the goal is to carve everything in free space. Special care has to be taken at depth discontinuities: carving can corrupt voxels of edges, because aliasing and small tracking inaccuracies associate the voxel with a more distant surface. Therefore, carving is only applied if there is no depth difference of more than τ in a radius of two pixels to the associated depth pixel. To free up memory, voxels that are erroneously allocated but become free space by repeated carving can be recycled in an asynchronous process, as has been done in [27].

For depth point $\mathbf{p}$ with normal $\mathbf{n}$ and truncation range τ, the signed distance to voxel position $\mathbf{x}$ is computed with the point-to-plane metric

$$d = \frac{1}{\tau}\langle \mathbf{p} - \mathbf{x}, \mathbf{n} \rangle. \tag{5}$$

For convenience, the distance is normalized and clamped to $[-1, 1]$. The point-to-plane metric helps keeping the SDF consistent with varying observation angles as opposed to the point-to-point metric [25].

While in our previous work [2] we explored ray casting similarly to Klingensmith et al. [28] for fusing individual depth pixels along the view- or normal direction, this method often shows bad results with noisy real-world data. For tracking applications, voxel projection, like in the original KinectFusion, has proven more robust. During voxel projection fusion, every allocated voxel within the current view frustum is projected into the current camera frame, associated with a depth (and color) pixel, and updated with the respective values and weights.

4 DTSDF Raycast Rendering

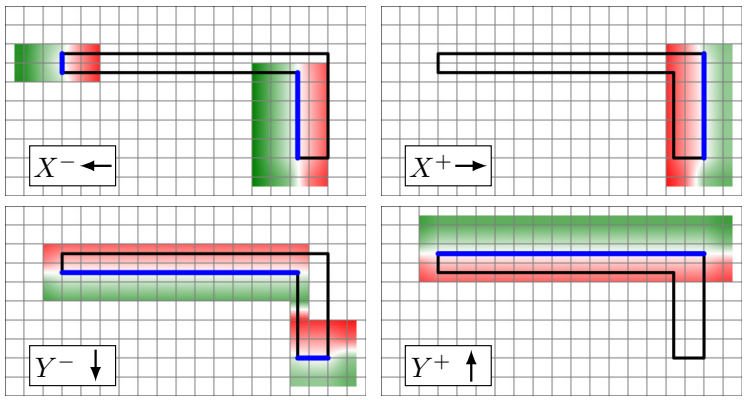
Rendering real-time views of the model from arbitrary positions is useful for visualization and also tracking. Instead of developing specialized tracking methods for the DTSDF, our approach is to render a map of depth points and use known and tested ICP-based algorithms [1, 13, 17, 9] to register input depth images.

The rendering process involves casting a ray per pixel of the virtual depth camera and extracting the iso-surface, i.e., the first transition from positive to negative SDF values. This involves probing the TSDF along the ray at regular intervals, until the distance turns negative and then multiple small steps, determined by the interpolated SDF value, to minimize the absolute distance value. Similar to the meshing presented in [2] the question is: how to combine up to six SDF values from partially overlapping directions?

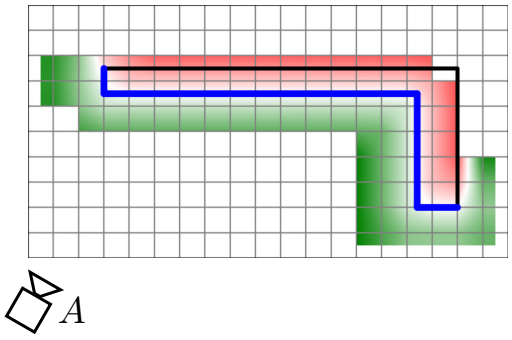
By its mathematical definition, the signed distance function can represent any given object. In other words, the six directions could, given a fine enough resolution, be combined into a single, conflict-free TSDF. But in practice, the combination is not straight forward: overlapping free and occupied space from different volumes has to be combined in accordance with the orientation of mapped surfaces, while considering corner cases, real-world noise and imperfections. Also, the practical use is limited. Ray-cast rendering relies on the width of the truncation range for finding zero-transitions, which for thin objects can be easily missed. Instead, we made an important observation:

Lemma 1. *For a DTSDF and a fixed camera position, a combined conflict-free regular TSDF can be computed.*

The basic intuition behind this lemma is, that surfaces the camera perceives from the backside are not relevant from a given position. Figure 3 visualizes this idea as a 2D example, omitting the z-axis: a thin, L-shaped object is represented by TSDFs for the directions X^-, X^+, Y^- and Y^+ as depicted in Figure 3a. For the given resolution, a complete representation by a regular TSDF is not possible, as the negative distance for the inside of the object cannot be stored. However, given a camera pose A, a conflict-free combination is possible, as depicted in Figure 3b. Only the blue surfaces are visible and negative SDF values from backside surfaces are omitted. Positive values are still important, as they prevent rendering surfaces in free space, like the overhanging zero-transitions of individual directions. In comparison, even doubling the resolution barely enables visualization, as the inside of the object is very thin. This is especially problematic, when the depth noise and truncation range exceed the thickness.



(a) TSDFs of directions X^+, X^-, Y^+, Y^- for the scene. Blue indicates surfaces, that fulfill $w^D_{\text{dir}}(\mathbf{n}) > 0$ (c.f. Eq. (2)), i.e., surfaces that are represented by direction D.



(b) combined TSDF from view point A. The blue line indicates the surfaces that are visible from A.

Figure 3: 2D example of computing the combined TSDF. The black outline represents the mapped object, green/red gradients correspond to positive/negative SDF values and the grid denotes the voxel grid.

The challenge is, for each point in $p \in \mathbb{R}^3$ to decide whether it is free space (SDF > 0) or occupied (SDF ≤ 0) and which directions to combine. We developed the following algorithm:

Algorithm 1 Compute Combined TSDF.

procedure CombinedTSDF($\mathbf{p}, \mathbf{c}$)
  $\mathbf{r} \leftarrow \frac{\mathbf{p}-\mathbf{c}}{||\mathbf{p}-\mathbf{c}||}$ ▷ view ray
  freespace $\leftarrow 0$
  if $\exists D : \Phi^D(\mathbf{p}) > 0$ **then**
    $\mathbf{q} \leftarrow$ first zero-transition from $\mathbf{p}$ in direction $-\mathbf{r}$ in Φ^D
    if $\mathbf{q} = \emptyset$ **then**
      freespace $\leftarrow 1$
    else if $\nexists \widetilde{D} : \Phi^{\widetilde{D}}(\mathbf{q}) < 0$ **and** $\langle \mathbf{n}^{\widetilde{D}}(\mathbf{q}), -\mathbf{r} \rangle > 0$ **then**
      freespace $\leftarrow 1$
    end if
  end if
  if freespace $= 1$ **then return** weighted sum of free space directions
  else return weighted sum of occupied space directions
  end if
end procedure

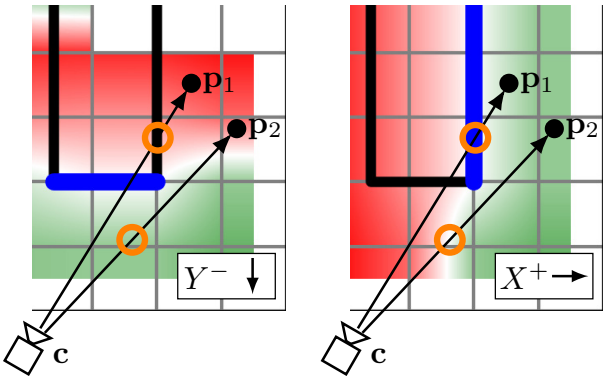


Figure 4: Visualization of view-point dependent free-space detection. From the camera's perspective point $\mathbf{p}_1$ is hidden behind the surface. The zero-transition (circles) in X^+ found by ray-casting from $\mathbf{p}_1$ towards $\mathbf{c}$ lies inside the occupied space of Y^-, so there is no free space. The first zero-transition from $\mathbf{p}_2$, on the other hand, lies within the free space of Y^-, so it does not conflict and $\mathbf{p}_2$ is marked as free space. The same holds, if there is no zero transition for $\mathbf{p}_2$ (i.e., $q = \emptyset$).

The input values are the point to look up $\mathbf{p}$ and camera position $\mathbf{c}$. If there is a direction D, for which $\mathbf{p}$ lies in free space it has to be checked, whether it conflicts with other directions under consideration of the cameras position: if this free space in D lies within the occupied space of a visible surface from another direction $\widetilde{D}$ and, moreover, the surface point q on the ray between $\mathbf{p}$ and $\mathbf{c}$ in $\widetilde{D}$ also lies within the occupied space, $\mathbf{p}$ cannot be free space. Figure 4 illustrates explain this rule with a positive and negative example, each. After determining whether $\mathbf{p}$ is free space or occupied space, all congruent directions are combined as weighted sum using the fused voxel weights.

There are some additional considerations for implementing the algorithm, including

- ignoring information from points with invalid gradients w.r.t. the direction vector, as this is not supposed to be represented by that direction.

- If point $\mathbf{p}$ is close to a surface and one direction indicates free space, one occupied space, the algorithm would only consider free space, instead of a weighted combination, even when both directions map the same surface.

- In reality there are many TSDF entries with low weight, which can erroneously induce free space.

While in theory and on synthetic data fused from ground truth poses the algorithm performs well, real-world usage is very limited. Due to sensor noise and slight tracking errors, fused data often leads false free space identification, resulting in dents and holes in the rendered surfaces — especially near corners. Instead, we propose a simple weighting scheme as approximation, which performs better in practice:

For a point $\mathbf{x} \in \mathbb{R}^3$ and direction D, let $\mathbf{n}^D$ be the normalized SDF gradient $(\partial\Phi^D/\partial x, \partial\Phi^D/\partial y, \partial\Phi^D/\partial z)^\intercal$ at $\mathbf{x}$, w_d^D the stored distance weight and $\mathbf{r}$ the normalized view ray from camera center to $\mathbf{x}$. Then the combination weight for direction D is defined as

$$w^D_{\text{comb}} = w^D_{\text{dir}}(\mathbf{n}^D) \cdot \langle \mathbf{n}^D, -\mathbf{r} \rangle \cdot w^D_d . \tag{6}$$

The first factor in Eq. (6) ensures that only gradients that actually comply with the direction they are stored in are used, with according weights to blend between directions. The second factor ensures that only directions with eligible surfaces are used, which is the main reason for using the DTSDF. The

approximation is not perfect and certain constellations work only under the premise, that all direction's SDF weights are similar. On the other hand, it has shown to be more robust in practice than the algorithm proposed above.

These per-direction weights can be used to directly look up the combined SDF value at any point in space as weighted sum, but ray-casting becomes very slow, because many TSDF lookups and memory reads have to be performed, especially for the gradient computation. The massive parallel computation also results in many cache misses, so the algorithm becomes memory bound. As suggested by Lemma 1, a view-dependent combined TSDF can be pre-computed by calculating the combined SDF for every voxel in the view frustum. This combined TSDF can then be used like a regular TSDF, but only for ray-casting from the pose used during combination. As a bonus, this opens up yet another class of tracking algorithms: the direct volume matching type, that perform registration directly within the SDF [19, 20, 22].

To always use the most recent observations, all voxels that received new information during fusion also need to be updated in the combined TSDF. But for static scenes this is not always necessary. Instead, we use conditional combination. Only meeting one of the following criteria triggers an update of the combined TSDF:

$$\text{framesSinceStart} < 5, \qquad \textit{boot up} \tag{7}$$

$$\text{framesSinceLastUpdate} > 50, \qquad \textit{stale state} \tag{8}$$

$$\|\text{pose} - \text{lastPose}\|_{\text{translation}} > 0.05\,\text{m}, \qquad \textit{translation} \tag{9}$$

$$\|\text{pose} - \text{lastPose}\|_{\text{angle}} > 0.05\frac{\pi}{2}. \qquad \textit{rotation} \tag{10}$$

The boot up condition Eq. (7) ensures, that during the first frames where the map is still uncertain, always the most recent data is used for tracking. Eq. (8) enforces regular updates in case the camera does not move. Eq. (9, 10) are a relaxation of Lemma 1, that states minor changes in the camera pose don't change the combined TSDF, similar to small-motion assumption on which the data association for ICP is based [1]. We experimentally chose the thresholds relatively small, so as not to violate the underlying assumption. A more thorough investigation on the impact of these limits would be interesting. On the tested sequences, the update is triggered on average around every third frame. By also selecting voxels slightly beyond the camera frustum ($\pm 1/8$ image size), motions of the camera will not leave the scope of the combined TSDF before triggering a recalculation. Voxels that receive data for the first time are always directly added to the combined TSDF.

To prevent empty voxels in the absence of gradients in all directions (e.g., at edges), the weights

$$w^D_{\text{noGrad}} = w^D \cdot \langle \mathbf{v}^D, -\mathbf{r} \rangle \tag{11}$$

are used instead.

5 ICP Tracking

The geometric Iterative Closest Point algorithm optimizes the pose difference $\Delta T \in \mathbb{R}^{4\times4}$ between two point clouds, or in our case a point cloud and a depth map. As introduced by Newcombe et al. [1], the new depth frame is registered against a point cloud rendered from the previous pose estimate, starting with an estimate of $\Delta T = I_4$. This frame-to-render as compared to direct frame-to-model registration might seem to be an unnecessary intermediate step, but saves time on data lookups and the data is often already available, if the algorithm is running live with a visualization. The algorithm uses projective data association to project each point from the depth frame at time $t+1$ into the frame rendered from the estimated pose at time t according to the current estimate of ΔT.

The rigid body transform ΔT is element of the Lie-group $\mathbf{SE}(3)$. Optimization takes place in the accompanying Lie-algebra $\mathfrak{se}(3)$. The 6-vector $\xi = (\nu, \omega) \in \mathfrak{se}(3)$ with translation and rotation components $\nu, \omega \in \mathbb{R}^3$ is a minimal representation for the rigid body transform and the exponential map $\exp : \mathfrak{se}(3) \to \mathbf{SE}(3)$ converts from algebra to group elements. We adopt the notation from Blanco [29], who published a good review on the $\mathbf{SE}(3)$, including the most important derivatives. For improved readability, we omit conversions from and to homogeneous coordinates and use group elements of $\mathbf{SE}(3)$ synonymous with the respective transformation matrices $\in \mathbb{R}^{4\times4}$. Let $p_i \in \mathbb{R}^3$ be the i-th point of the depth frame and $q_i, n_i \in \mathbb{R}^3$ the associated point and normal in the rendered scene. Then the weighted geometric ICP formulation is

$$E_{\text{geom}} = \sum_i w_i \underbrace{\langle q_i - \exp(\xi)\Delta T p_i \mid n_i \rangle}_{=:r_i}{}^2. \tag{12}$$

The error is minimized using the Gauss-Newton method, where linearization around $\xi = 0$ allows to derive the Jacobian

$$J_i = \left.\frac{\partial r_i}{\partial \xi}\right|_{\xi=0} = \begin{pmatrix} -n_i & -\Delta T p_i \times n_i \end{pmatrix}^\intercal \in \mathbb{R}^6 \tag{13}$$

We had some serious issues with the repeatability of our experiments due atomic operations in the CUDA kernels: while computing the Hessian matrices during ICP with a three-level reduction pyramid, the resulting summands of the individual blocks get atomically added together. Floating point addition is, however, not commutative on computation hardware and we've seen deviation of our tracking results in the order of 1%. Therefore we changed summation to a fully-deterministic hierarchical scheme, with no noticeable increase in computation time.

$$H = 2\sum_i w_i \sum_{j,k} J_{i,j} J_{i,k} \in \mathbb{R}^{6\times6} \tag{14}$$

$$g = 2\sum_i w_i J_i^\intercal r_i \in \mathbb{R}^6 \tag{15}$$

The equation system

$$H\hat{\xi} = -g \tag{16}$$

can be solved efficiently using Cholesky decomposition, yielding solution $\hat{\xi}$ that is used to update the current pose estimate for the next iteration:

$$\Delta T' = \exp(\hat{\xi})\Delta T. \tag{17}$$

The algorithm iterates until the step size $\|\hat{\xi}\|_2$ falls below a certain threshold or a maximum number of iterations is reached.

5.1 Weighted ICP

The error formulation in Eq. (12) contains an optional per-pixel weight factor w_i. Surprisingly, while most TSDF implementations use some form of weighting during fusion, especially regarding the increasing unreliability with larger depth (c.f. Figure 5), only few implementations apply weights during tracking, when it is most crucial. Xia et al. [15] apply the following weight function for each depth value z and valid depth range $z \in [z_{\min}, z_{\max}]$:

$$w_{\text{Xia}} = \frac{\frac{1}{z^2} - \frac{1}{z^2_{\max}}}{\frac{1}{z^2_{\min}} - \frac{1}{z^2_{\max}}}. \tag{18}$$

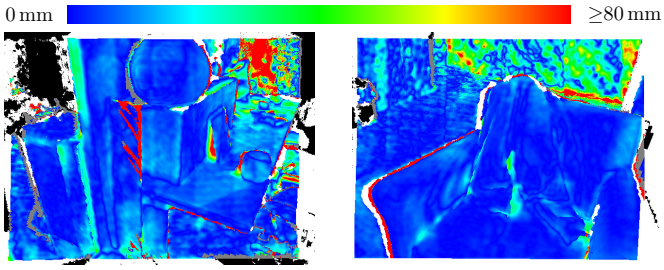


Figure 5: Depth noise examples, error increases from blue to red.

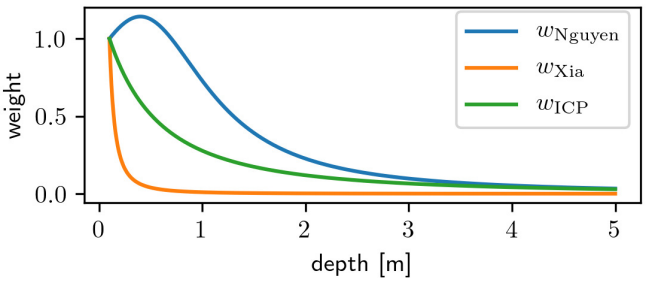


Figure 6: Visualization of depth-dependent ICP weights used in [14] (w_{Nguyen}), [15] (w_{Xia}) and proposed (w_{ICP}) for depth range $[0.1, 6]$ m.

Nguyen et al. [14] developed a more accurate, specific noise model for the Kinect v1 camera by measuring a flat panel at set distances and angles and fitting a function. They include an additional factor if the angle θ between surface normal to camera view ray exceeds 60°:

$$w_{\text{Nguyen}} = \frac{\sigma(z_{\min}, 0)}{\sigma(z, \theta)}, \tag{19}$$

$$\sigma(z, \theta) = 0.0012 + 0.0019 * (z - 0.4)^2 \left[+ \frac{0.0001}{\sqrt{z}} \frac{\theta^2}{\left(\frac{\pi}{2} - \theta\right)^2} \right]. \tag{20}$$

The summand in squared brackets is only applied for $\theta > 60°$. It is easy to spot that, apart from the angular factors, both weight functions use the inverse quadratic depth, but w_{Nguyen} has additional shaping parameters. As visualized in Figure 6 for a depth range of $[0.1, 6]$, the weight function w_{Xia} puts very high emphasis on close observations, so everything closer than 0.5 m far outweighs other measurements. This creates problems in certain scenes, where objects sweep through the frame at a very close distance, e.g., it leads to tracking failure in the *ICL deer walk* sequence. With constant ICP weights, noisy sequences often fail completely (for instance, the noise-augmented sequences of the Zhou dataset, which have somewhat overexaggerated noise levels compared to real-world sensors). While of course desirable, it is impractical to calibrate a specific noise model for every sensor. Therefore, we propose another depth weight, which has similar characteristics to the model by Nguyen et al., but does not rely on any magic numbers:

$$w_{\text{ICP}} = \frac{1}{(z + (1 - z_{\min}))^2}. \tag{21}$$

The function stays within $[0, 1]$ and its asymptotic behavior still considers distant points to a reasonable degree.

As proposed before by Prisacariu et al. [13], we also performed experiments with the Huber loss as cost function, that theoretically should make tracking more robust against outliers, but found it to generally reduce tracking performance. Similar findings were reported by Bellekens et al. [30]. Therefore, we kept the standard quadratic error function.

5.2 Photometric ICP

Photometric ICP, that minimizes the photometric error between pixels of two RGB frames to determine the relative pose difference, has been used in RGB-D tracking for some time [16] and was successfully adapted to the TSDF as well [10, 9, 31].

Setting aside the discussion of how to incorporate RGB data into tracking, a more fundamental question is which data to actually compare. While geometric ICP relies on rendering a point cloud from the TSDF, for photometric ICP there are two directions throughout literature: frame-to-frame (`f2f`) tracking [10, 13], where the frame at time t is registered against the frame at time $t-1$, and frame-to-render (`f2r`) [31, 9, 15], where the current frame is tracked against a point cloud rendered from the TSDF. The authors of [19, 21] perform frame-to-model tracking, but the same discussion as above holds. Frame-to-render has the advantage of tracking against a consistent model, thus it should be more robust against drift. On the other hand, because of the way colors are fused into the TSDF (without other augmentations), the contained photometric information is subject to blur, lighting and exposure changes, error accumulation etc. Moreover, the detail and sharpness directly correlate to the voxel size. Therefore, it would be plausible, that with increasing voxel sizes, the frame-to-render tracking deteriorates. Frame-to-frame tracking, on the other hand, has the advantage of retaining the resolution of the input images, which for modern sensors (e.g., Intel RealSense) usually exceeds the depth image resolution. Whelan et al. [10] argue, that very distant points that are useful for rotational constraining are usually not represented in the TSDF and also the TSDF resolution limits the effectiveness, though this argument only holds to a certain extent, as data association is performed by projecting depth points from the current image pair to the previous color image, so it is still bounded by the maximum perceivable depth of the sensor. A disadvantage, besides the obvious drift, is the individual image quality which is subject to blur, rolling shutter, and depth-color synchronization issues.

Interestingly, to our knowledge, frame-to-keyframe (`f2kf`) tracking, where the current frame is registered against a selected keyframe from the past, has not been applied in photometric tracking with the TSDF yet. It has been successfully implemented in model-less graph-based algorithms [17, 24, 18], effectively reducing drift, so we want to investigate its use in combination with the TSDF. We discuss this approach more in Section 5.3.

Figure 7 displays a comparison of photometric error for the different reference methods. All error images are rendered from the same ground truth poses. Even given perfect poses from a noise-less artificial dataset (ICL deer firefly), the `f2f` and `f2kf` still show some error: a point from the current depth frame gets back projected into the reference color frame for data association and the photometric value is bilinearly interpolated, as the coordinates are not integer. The `f2kf` images (third column) have some missing pixels, as the warped keyframe is partially outside the current view. The second row contains footage from the real-world dataset TUM fr3. Especially for larger voxel sizes where many colors, lighting conditions and image exposures blend together, the rendered color image are very blurred. Therefore, many pixels fall below the minimum gradient threshold or above the maximum intensity threshold that prevent using pixels with unreliable information during optimization.

Only using photometric ICP does not provide reliable tracking,

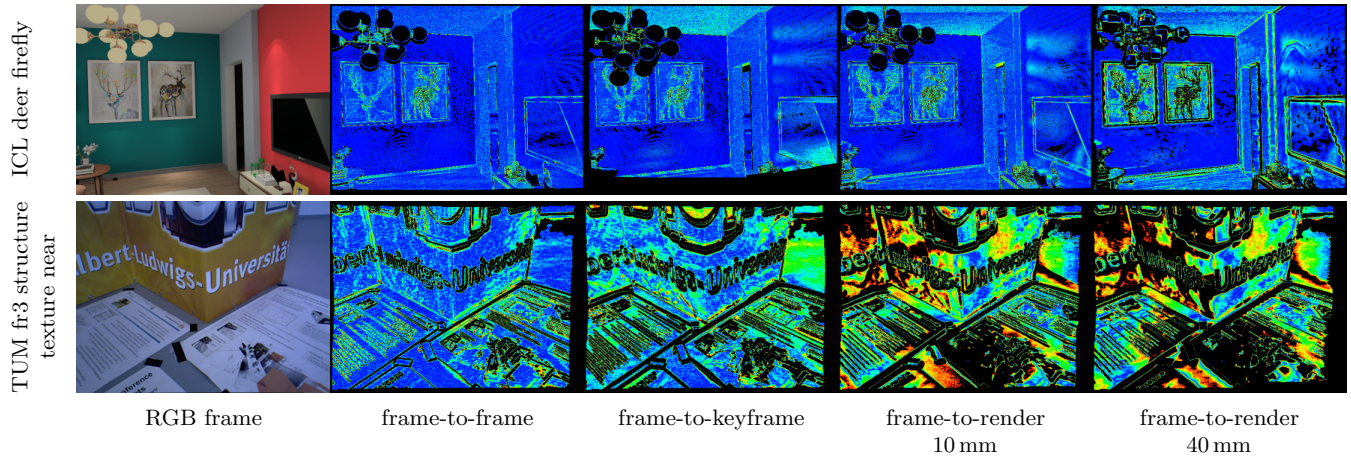


Figure 7: Photometric error comparison for different reference image types for photometric ICP. Photometric error from blue (no error) to red (intensity threshold).

so we use combined ICP as described in [10, 15], where the error functions are combined as

$$E = E_{\text{geom}} + \lambda E_{\text{photo}}. \tag{22}$$

Most combined ICP approaches use a fixed weight between photometric and geometric terms for the entire image pair, though its role is not clearly stated in all works: it is required to compensate the difference in metric between depth and intensity errors. A reoccurring constant weight throughout literature is 0.1, though there are some discrepancies whether the weight is squared and whether the kernel for gradient computation is normalized [9, 10, 15]. We used 0.1 without squaring and, upon inspection, the magnitudes of the geometric and photometric Hessian matrices were in the same order.

In their implementation Whelan et al. [10] additionally use the root mean square of the intensity difference to down-weight pixels with larger errors, but didn't find any good explanation. We apply the same per-pixel depth weight as before, as the data association of color pixels also depends on the quality of the depth.

5.3 Keyframe Photometric ICP

For combined ICP, The frame-to-frame and frame-to-render formulations use the fact that the model is rendered at the same location as the RGB reference image frame. With frame-to-keyframe ICP, the pose of the reference (key)frame at time t^{kf} differs from the pose of the rendered scene at time t. Let T^{RS} be the transform from rendered scene to reference frame and ΔT, just like above, the pose estimate from rendered scene to frame $t+1$ of the current ICP iteration (initialized with I_4). Then the error formulation becomes

$$E^{\text{kf}}_{\text{RGB}} = \sum_i \left\| \underbrace{I^{t+1}\left(\Pi(p_i)\right) - I^{t^{\text{kf}}}\left(\Pi\left(K \cdot T^{\text{RS}} \exp(\xi)\Delta T \cdot p_i\right)\right)}_{=:r(\xi,p_i)} \right\|^2. \tag{23}$$

with camera matrix K, projective function Π (Eq. (28)). Linearization around $\xi = 0$ gives

$$\frac{\partial r}{\partial \xi}(\xi, p_i) \approx r(0, p_i) + \frac{\partial r}{\partial \xi}(0, p_i) \cdot \xi \tag{24}$$

with the derivative being

$$\begin{aligned}
&\frac{\partial r}{\partial \xi}(0, p_i) \\
=&\frac{\partial I^{\text{kf}}}{\partial \Pi} \cdot \frac{\partial \Pi}{\partial K} \cdot \frac{\partial K}{\partial p_i} \cdot \frac{\partial T^{\text{RS}} \exp(\xi)\Delta T p_i}{\partial \xi} && (25)\\
=&\nabla I^{\text{kf}} \cdot \frac{\partial \Pi}{\partial a}\Big|_{a = K T^{\text{RS}} \Delta T p_i} \cdot K \cdot R(T^{\text{RS}}) \left[I_3 \mid -(\Delta T p_i)^\wedge\right]. && (26)
\end{aligned}$$

$R(\cdot)$ extracts the rotation matrix from the transform. The wedge operator for a vector $\nu = (x, y, z)^\intercal \in \mathbb{R}^3$ is defined as

$$\nu^\wedge = \begin{pmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{pmatrix}. \tag{27}$$

The derivative of the projection function is

$$\frac{\partial \Pi}{\partial (x, y, z)^\intercal} = \frac{\partial}{\partial (x, y, z)^\intercal} \begin{bmatrix} \frac{x}{z} \\ \frac{y}{z} \end{bmatrix} = \begin{bmatrix} \frac{1}{z} & 0 & \frac{x}{z^2} \\ 0 & \frac{1}{z} & -\frac{y}{z^2} \end{bmatrix}. \tag{28}$$

For the `f2f` scenario, the same formulation can be used with $T^{\text{RS}} = I_4$.

Keyframe Selection The choice of keyframes is the problem of finding a compromise between good coverage of the scene and using only as few views as necessary. Most approaches in the past used heuristics of different complexity to find suitable keyframes. The easiest way is to select a new keyframe at fixed intervals, as it is done in ORB-SLAM [24]. Other common heuristics are thresholding the change of angular and translational change [32, 33, 34]. Sucar et al. [35] compute a measure on how much of the current depth image is not yet represented by the map. Some methods also decide based on image quality, by selecting frames with low jitter, camera velocity [36] or when the exposure time changed too much [34]. The downside to these hand-crafted metrics is, that they require tuning of multiple parameters, possibly even depending on the scene. Entropy based selection schemes use the covariance of the Hessian matrix of the Gauss-Newton algorithm to determine when a new keyframe is due. Kerl et al. [17] compare the negative entropy of the keyframe and the current frame and if the ratio falls below a threshold, select a new keyframe. Because the average negative entropy levels can drift with the camera moving (especially they can also increase), Kuo et al. [37] apply a running average filter to the entropies for comparing. The main advantage of this method is that this only leaves one threshold as tunable parameter. In practice we found, though, that tuning these methods is not that

(a) Photoneo depth **(b)** DSLR RGB image

Figure 8: Example data from PhenoRob UGV dataset.

simple. High frequency fluctuations can cause fast change of keyframes, but using a more conservative threshold often results in keeping the keyframe for too long. Because of the running average adapting to the overall trend of negative entropies, the method proposed by Kuo et al. [37] would sometimes not select any new keyframes at all.

In our experiments we want to investigate, whether using keyframes can help mitigate drift and jitter. For comparability, it is important that all runs use the same keyframes, as the heuristics described above might select different keyframes based on the differences in tracking. Therefore, we chose a very simple scheme: every 10 frames a new keyframe is selected. All datasets used in our experiments have an image rate of 20-30 Hz, so this allows for large enough overlap, but is far enough apart to tell if there is any advantage or disadvantage compared to frame-to-frame tracking.

6 Multi-Sensor Sim3 Scale ICP

Part of our work in the PhenoRob Cluster of Excellence is to reconstruct crops from data collected on the field by a UGV. The UGV is an inverse U-shape constructed from aluminum struts, driving over the crops and taking a snapshot with 14 Nikon DSLR cameras (five stereo pairs, four monocular cameras) and five Photoneo PhoXi 3D scanners, producing high-accuracy depth maps. The robot stops over a crop patch, takes a snapshot with all sensors and then proceeds to the next patch. Figure 8 shows sample plant measurements.

One challenge is to fuse the data from different sensors, as there are slight depth discrepancies. Moreover, in this scenario the extrinsic parameters of the cameras change when the robot moves, because of its size and limited rigidity. Also, the stereo-pair baseline is used to convert disparity images to depth maps, so the depth might vary, based on small changes in the camera setup. To compensate for this while fusing the data, we extend the standard $\mathbf{SE}(3)$ ICP tracking, which estimates translation and rotation, to the similarity transform $\mathbf{Sim}(3)$, which jointly optimizes pose and scale. We apply this method to refine the depth scale and pose priors provided by the extrinsic calibration of the multiple sensors.

For the point-to-point scale ICP least-squares optimization, there exists a closed form solution of rotation, translation and scale, first described by Horn [38]. This method is, however, not applicable for partially similar point clouds. This problem was addressed by Sahillioglu et al. [39]. TEASER [40] is an algorithm that is highly robust to large amounts of outliers and non-zero-mean Gaussian noise. Du et al. [41] proposed an algorithm that optimizes the scales for each coordinate axis. LSD-SLAM [32] performs $\mathbf{Sim}(3)$ photometric ICP to unify the scale-uncertainty of monocular SLAM.

Chen and Medioni [42] state, that with unknown correspondences ICP with the point-to-plane metric converges quicker than point-to-point, so it is the more sensible choice in our case. Note, that without one-to-one point correspondences this is an ill-posed problem, as translation and scale optimization to a degree modify the same error gradient (in z-direction) and because of the projective data association, especially for environments with little geometric diversity like corridors etc., for a majority of points both parameters get optimized. In other words there are multiple solutions with similar low residuals, but different bias towards scale and translation optimization. In the extreme case of, for instance, perceiving only a plain a wall, optimizing either pose or scale will have identical results.

We use the same error formulations Eq. (12) and Eq. (23) as before, only with $\xi = (\nu, \omega, \sigma)^\intercal \in \mathfrak{sim}(3)$ instead. The additional 7th parameter represents the scale and the corresponding exponential map is

$$\exp : \quad \mathfrak{sim}(3) \to \mathbf{Sim}(3), \tag{29}$$

$$(\nu, \omega, \sigma)^\intercal \mapsto \begin{bmatrix} \exp(\sigma)\exp(\omega) & V\nu \\ 0 & 1 \end{bmatrix}. \tag{30}$$

For $A, D \in \mathbf{Sim}(3), p \in \mathbb{R}^3$ we require the derivative of the transposed point at $\xi = 0$:

$$\frac{\partial}{\partial \xi} A \oplus \exp(\xi) \oplus D \oplus p \Big|_{\xi=0} = R(A) \begin{bmatrix} I_3 & -(D \oplus p)^\wedge & D \oplus p \end{bmatrix} \tag{31}$$

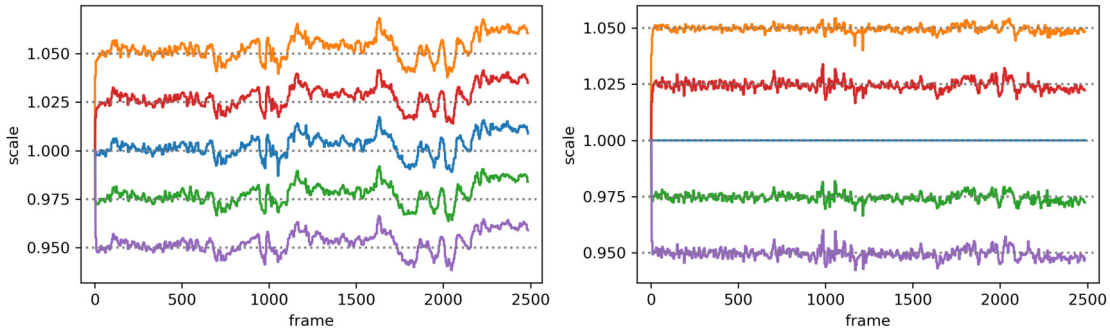
Eq. (31) can be directly inserted into Eq. (13) and Eq. (26) to get the respective Jacobians. The rest of the algorithm works analogously, with the minor difference, that the resulting Hessian is 7×7 instead of 6×6.

For camera intrinsics (f_x, f_y, c_x, c_y), image coordinates x, y, and depth value d, the depth reprojection function Eq. (32) is linear in d, so scaling the depth is equivalent to scaling the point in the camera frame:

$$\Pi^{-1}(x, y, d) = \left(d\frac{x - c_x}{f_x}, d\frac{y - c_y}{f_y}, d \right)^T. \tag{32}$$

This implies that instead of changing the entire TSDF pipeline to use $\mathbf{Sim}(3)$ poses, it is possible to just scale the input images with the given factor and use the corresponding $\mathbf{SE}(3)$ poses for everything else.

To test the behavior in a controlled manner, we transform the *fr3_long_office* sequence into an artificial five-sensor sequence, by scaling batches of five consecutive depth images with constant



(a) Unbounded **(b)** Drift compensation by anchoring.

Figure 9: Comparison of unbounded- and anchored scale estimation for five sensors. Gray dotted lines are the true scale factors. The colored lines correspond to the scale factors estimated for the individual sensors; the anchored sensor in the (b) is represented by the blue line.

factors $(1.0, 1.05, 0.975, 1.025, 0.95)$, such that the sequence consists of a repeated cycle of five different sensors. Just like with pose tracking, estimation of scale against the model accumulates drift over time, as Figure 9a shows. Noticeably, the drift happens in unison across the sensors. By anchoring the scale of one of the sensors to a fixed value $\hat{s_0}$ and compensating the other estimates, the drift is effectively prevented and absolute error of scale estimates is significantly lower, as can be seen in Figure 9b. Let $\hat{s_i}$ be the estimated scale of sensor $i > 0$ at time t. Then the compensated scale is computed as

$$s_i^t = \exp(\hat{s_i}^t) \exp(\hat{s_0})^{-1} = \hat{s_i}^t - \hat{s_0}. \tag{33}$$

Figure 8 depicts examples of typical data acquired by the robot on the field. Note, how the Photoneo sensor yields many missing measurements around the plants. These are caused by occlusions and wind moving the plants during scans, as the acquisition process takes 250 – 2500 ms. We use Kalibr [43] to determine intrinsic and extrinsic parameters of the RGB cameras and multiple ArUco markers on the side panels of the robot to align RGB and the Photoneo sensors. For generating depth from stereo, we have experimented with classical methods like Semi-Global Matching [44] and the hierarchical deep stereo matching method by Yang et al. [45], but have not yet generated satisfying results, i.e., the depth error is too large to reliably overlay multiple views. Instead, we use the multi-view stereo (MVS) pipeline COLMAP [46, 47] to generate cross-matched depth maps for all RGB cameras. Owing to the large overlap of camera views, a depth map is generated for each of the 14 cameras. Figure 10 shows a set of all 14 depth images that form a single scan.

While COLMAP already computes good poses, which don't require further refinement, MVS does not produce an absolute metric scale, so we use the depth of the high-accuracy Photoneo PhoXi sensors and register the MVS depth maps against them. The initial guess of the optimization procedure takes as input the scale estimate which is visually close to the Photoneo depth, and use the extrinsics from our offline calibration process, including the RGB sensor poses, so all sensors get registered. Figure 11 shows example reconstructions without alignment, with **SE**(3) and with **Sim**(3) ICP pose refinement in comparison. **Sim**(3) optimization produces the smallest error.

7 Implementation Details

Our implementation is based on InfiniTAM [48], with significant modifications. For optimized memory usage, the voxel hashing scheme introduced by Nießner et al. [31] is used. Voxels are allocated in blocks of $8 \times 8 \times 8$ only where required and a hash map is used for constant-time access. Among other changes, the implementation now uses the stdgpu library by Stotko [49] to replace several components, especially the original hash map, which could not allocate blocks with colliding hash values within the same iteration.

Unlike the previous DTSDF implementation presented in [2] where 6 separate TSDF volumes were used for the different directions, here only a single TSDF is utilized. The hash index is extended from $(x, y, z) \in \mathbb{Z}^3$ to $(x, y, z, D) \in \mathbb{Z}^3 \times$ Directions. This simplifies many functions and better utilizes the statically allocated memory on the GPU: For most scenes the DTSDF has an imbalance of direction-usage, which wastes a lot of memory in the old scheme. Resizing the volumes is an option, but requires additional overhead which can be simply avoided by the aforementioned modification.

As a proof of concept, we use the pipeline depicted in Figure 1 with the geometric and combined ICP tracker described in Section 5. We solve the optimization with the Levenberg-Marquardt (LM) algorithm. The originally implemented LM damping scheme, where the damping factor is multiplied/divided by 10 whenever the error increases/decreases was replaced by the scheme proposed by Madsen et al. [50], which promises better convergence while avoiding premature convergence towards local minima.

The time for computing the rendering TSDF is crucial for real-time usage of our method. It can be significantly sped up by taking advantage of the lookup positions being only integer voxel positions. Hence, no trilinear interpolation is required and the gradient can be computed with just looking up the SDF values stored in the 6 neighboring voxels. We further managed to halve the time by pre-caching the TSDF lookups for all voxels of the same block in shared memory. At the very most, for every direction there are the current block and its six neighboring blocks, so a total of 48 are looked up at the beginning.

Image preprocessing includes a depth filter. Especially the artificial datasets with noise augmentation contain an abundance of noisy pixels around object boundaries. If there are not at least two depth pixels in the direct neighborhood, that support the depth value, the pixel is discarded. Depth normals are approximated as cross product of neighboring pixels in x- and y direction and afterwards processed by a bilateral filter.

8 Evaluation

The datasets used in our evaluation are the Stanford 3D Scene Data (totempole, etc.) [51], ICL NUIM [52] (*lr* and *office*), Zhou [53], the TUM RGB-D benchmark [54] (*fr1* and *fr3*) as well as (new) ICL [55] (*deer* and *diamond*). For improved readability, we omit the dataset name where possible. Frame-to-

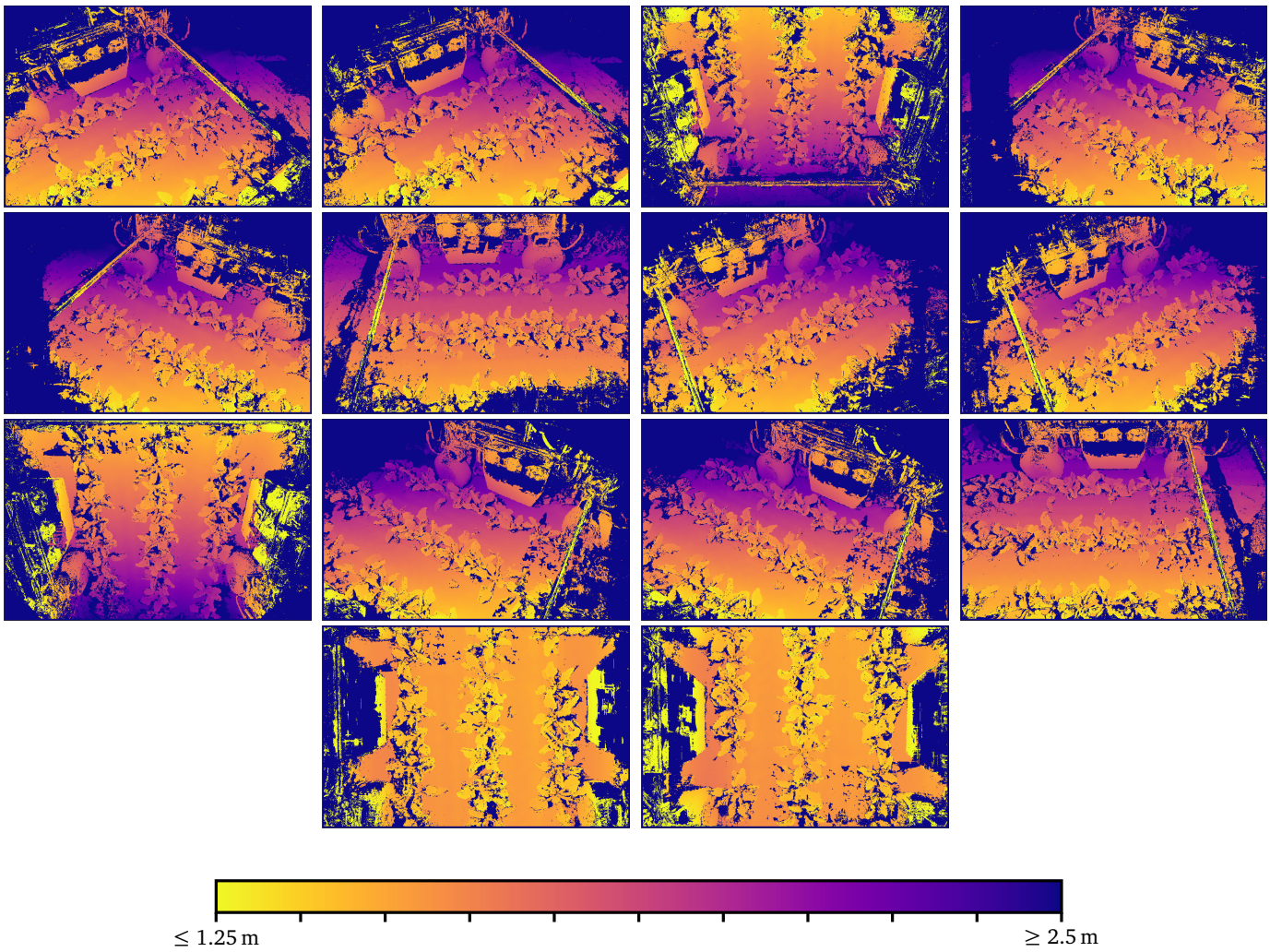
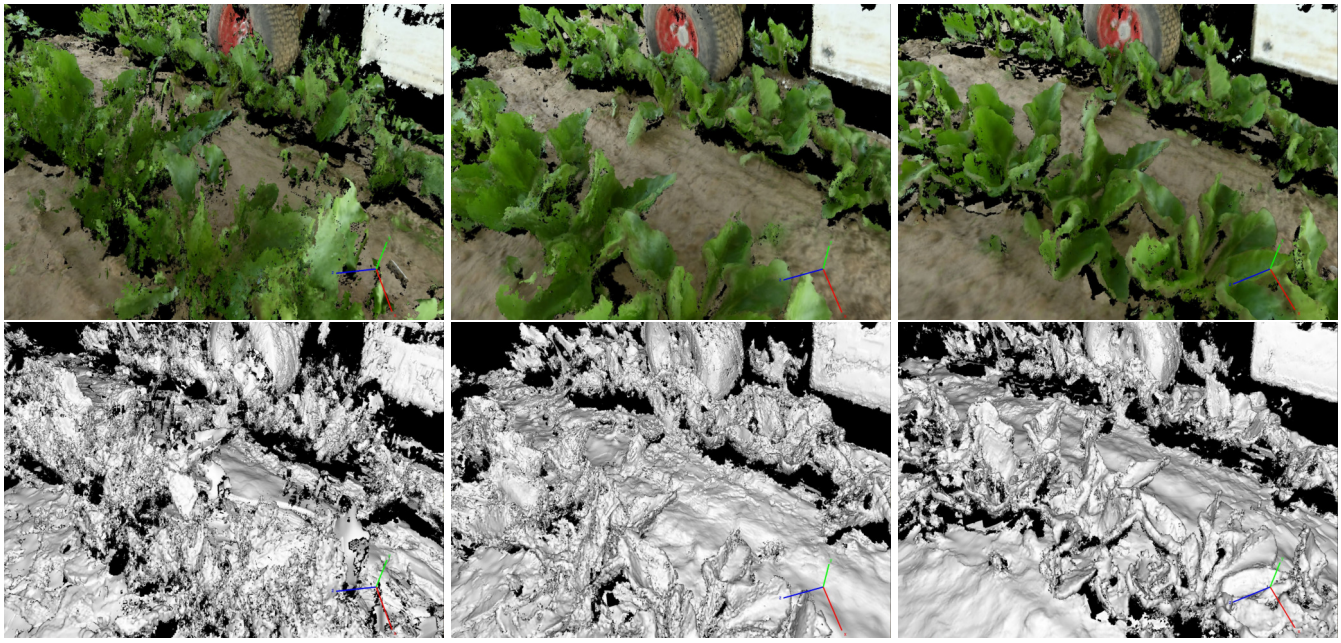


Figure 10: Example depth images generated by COLMAP.



(a) No pose refinement. **(b)** **SE**(3) pose refinement. **(c)** **Sim**(3) pose refinement.

Figure 11: PhenoRob UGV sugar beet reconstructions with and without pose refinement. Voxel size 2.5 mm.

frame, frame-to-keyframe, and frame-to-render are abbreviated as `f2f`, `f2kf` and `f2r`, respectively. SotA refers to the state-of-the-art regular TSDF.

Our evaluation compares and analyzes the impact of selected parameters on the results of the algorithm, which will be clearly stated for the experiments. All other parameters are consistent throughout all runs and were chosen based on experiments and values found in related work to give a good balance between convergence reliability and tracking quality:

- depth outlier filter as described in Section 7,
- bilateral depth filter ($\sigma_d = 5.0, \sigma_r = 0.025, \text{radius} = 5$),
- bilateral normal filter ($\sigma_d = 2.5, \sigma_r = 5.0, \text{radius} = 5$),
- ICP settings
  - termination condition minimum step size 10^{-6}m,
  - iteration upper bound 20 coarse / 50 fine,
  - depth outliers threshold 0.05 m coarse / 0.005 m fine and
  - intensity outlier threshold 0.175 coarse / 0.05 fine.

8.1 ICP Tracking

For comparing the tracking performance, we evaluate the regular TSDF (marked state-of-the-art, SotA) against the DTSDF by running scenes from the aforementioned datasets and comparing the tracking results against the provided ground-truth trajectory using the relative pose error (RPE) with a window size of 30 frames (1 s). Note that this study does not try to compare to complete SLAM algorithm with loop closure detection and correction, but showcases the performance of the DTSDF relative to the regular TSDF as an enhanced data structure. All settings are equal across both modes and the tracker uses the default geometric ICP algorithm.

As expected, tracking does benefit from the DTSDF in scenes, where the camera observes thin structure from different angles. Otherwise, there is no significant difference.

The first test on artificially generated sequences from the ICL NUIM and Zhou datasets is reported in Table 1 for different voxel sizes. Note that the noise-augmented sequences are being used. The tracking performance is similar for most sequences, which is likely due to the mapped environments, which are convex rooms where the regular TSDF does not display its issues. The Zhou office sequences, a scene of cluttered office desks scanned from different directions, provides an environment where the DTSDF actually has an advantage, which is reflected in the RPE. Table 2 does the same comparison on the turntable-like dataset used in the original DTSDF paper [2]. In those sequences the camera orbits around a center point and only the model is visible, which is challenging to track due to the details and high percentage of thin structure w.r.t. the whole scene. The RPE distinctly shows the strength of the DTSDF.

Table 3 shows the results with real-world scans from the TUM dataset, with very similar results. Large planar surfaces with sharp corners (structure notex sequences) seem to benefit from the DTSDF. Here, the measurements have to be considered with care, as the underlying ground-truth is not perfect.

8.2 Map Reusability

Overall, the tracking results show that the DTSDF generally only has an advantage in sequences, where the camera observes structure from different sides. In those sequences, the performance is significantly better, especially with increasing voxel size, as the problems of the regular TSDF increase as well. In all other sequences the performance is very similar, which is also due to the fact that the fusion process makes the locally visible area compliant: given enough observations from the current viewpoint, all conflicts in the representation will be evened out due to the running average (unless the conflicting side has been observed a long time, resulting in a high weight). In many cases this does not become apparent during tracking, but for the reusability of the overwritten parts of the completed map it is important to test these effects. To this end, we propose the geometric post-fusion per-frame error: after completing fusion of the entire sequence, for every estimated pose, a depth map is rendered again and compared to the corresponding input depth image by computing the pixel-wise mean absolute error (MAE). Figure 12 shows a side-by-side example of post-fusion error images, where the regular TSDF clearly shows more errors at corners and around thin structures. Figure 13 plots the MAE and shows that although the tracking performance is very

Table 1: Tracking RPE in mm, mean memory usage, and per-frame runtime of synthetic ICL NUIM [52] and Zhou [53] sequences for different voxel sizes.

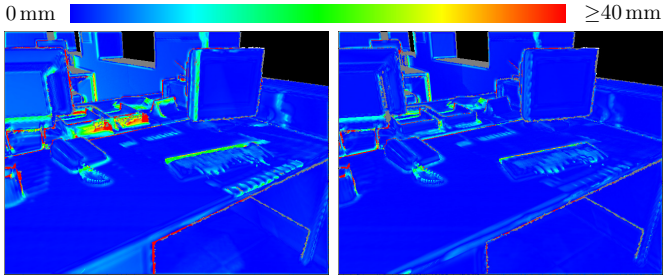
voxel size	5		10		20	
	SotA	DTSDF	SotA	DTSDF	SotA	DTSDF
lr kt0n	10.5	**10.3**	9.4	**9.3**	**9.5**	**9.5**
lr kt1n	**9.5**	**9.5**	9.9	**9.6**	9.7	**9.4**
lr kt2n	**15.3**	**15.3**	15.6	**15.5**	**15.5**	**15.5**
lr kt3n	15.6	**11.7**	29.1	**14.9**	**16.8**	87.4
office kt0n	**9.7**	**9.7**	**9.7**	**9.7**	**9.7**	9.8
office kt1n	**9.5**	**9.5**	**9.5**	**9.5**	**9.5**	**9.5**
office kt2n	**15.4**	15.5	**15.6**	16.0	**15.4**	**15.4**
office kt3n	11.0	**10.9**	11.3	**10.9**	11.2	**11.0**
Zhou lr1	1.5	**0.8**	1.3	**1.1**	2.4	**2.2**
Zhou lr2	0.8	**0.7**	1.1	**1.0**	2.3	**2.0**
Zhou office1	1.2	**1.0**	3.2	**1.3**	10.5	**3.1**
Zhou office2	1.0	**0.8**	3.6	**1.4**	14.8	**2.4**
∅ time [ms]	**8.4**	10.7	**6.4**	7.1	**5.7**	6.0
∅ mem [MB]	**1350**	1855	**326**	502	**75**	137

Table 2: Tracking RPE in mm, mean memory usage, and per-frame runtime of synthetic sequences rendered from Stanford 3D models [2] for different voxel sizes.

voxel size	5		10		20	
	SotA	DTSDF	SotA	DTSDF	SotA	DTSDF
armadillo	2.8	**2.7**	2.9	**2.5**	10.6	**5.6**
bunny	**2.0**	**2.0**	2.3	**1.9**	9.4	**4.8**
dragon	3.0	**2.9**	4.7	**3.4**	21.6	**10.6**
turbine blade	10.4	**6.7**	16.9	**10.1**	84.8	**13.5**
∅ time [ms]	**4.9**	**4.9**	4.8	**4.7**	**4.7**	**4.7**
∅ mem [MB]	**18**	41	**4**	13	**1**	4

Table 3: Tracking RPE in mm, mean memory usage, and per-frame runtime of TUM sequences [54] for different voxel sizes.

voxel size	5		10		20	
	SotA	DTSDF	SotA	DTSDF	SotA	DTSDF
desk1	62.9	**59.5**	63.7	**58.0**	66.6	**60.6**
long office	24.3	**24.1**	26.8	**25.2**	**25.5**	25.8
structure notex far	12.1	**12.0**	12.2	**12.1**	12.1	**12.0**
structure notex near	15.3	**15.1**	15.3	**15.2**	**15.4**	**15.4**
∅ time [ms]	**7.0**	10.3	**5.9**	6.6	**5.6**	5.9
∅ mem [MB]	**670**	1419	**132**	332	**28**	84



regular TSDF DTSDF

Figure 12: Visualization of post-fusion error of frame 330 in sequence *Zhou office 2*.

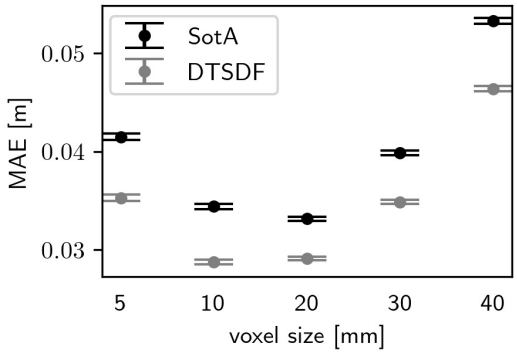


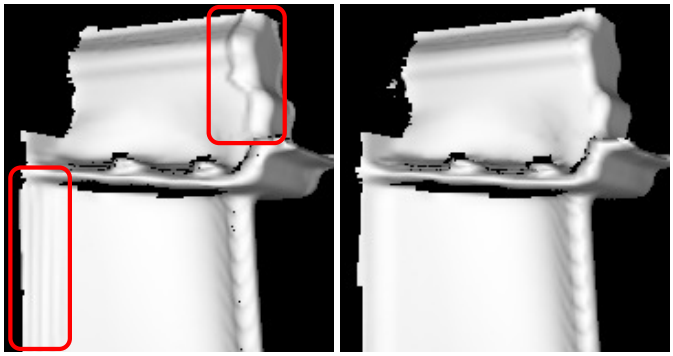
Figure 13: Post-fusion MAE (dot) and 95% confidence intervals (bars) on example dataset *TUM fr3 long office*.

Table 4: Difference of photometric MAE between SotA and DTSDF (in %, numbers smaller zero mean DTSDF is better) for different voxel sizes averaged over all sequences in the datasets.

	voxel size [mm]				
dataset	5	10	20	30	40
Zhou	4.3	-4.4	-4.2	-4.0	-4.1
TUM_fr1	-2.1	-0.7	-0.7	-0.7	-0.5
TUM_fr3	-1.2	-1.3	-1.6	-1.8	-2.1
ICL_NUIM	-0.2	-2.3	-2.1	-2.3	-1.4
ICL	-2.5	-2.3	-2.2	-2.0	-2.0

Table 5: MAE (in mm) for different voxel sizes and datasets.

dataset	mode	voxel size [mm]				
		5	10	20	30	40
SUN copyroom	SoTA	23.0	**20.9**	24.8	54.4	41.8
	DTSDF	**20.7**	30.5	**21.7**	**25.9**	**30.0**
SUN lounge	SoTA	16.6	16.9	24.1	33.2	45.7
	DTSDF	**14.9**	**15.7**	**21.8**	**30.0**	**39.2**
ICL NUIM lr kt1n	SoTA	13.2	48.4	99.0	104.5	117.7
	DTSDF	**6.8**	**14.9**	**37.4**	**81.6**	**76.6**
ICL NUIM office kt3	SoTA	**2.0**	**2.2**	**3.6**	**5.5**	**7.3**
	DTSDF	**2.0**	**2.2**	3.7	5.6	**7.3**
Zhou office2	SoTA	5.5	9.0	21.9	52.7	–
	DTSDF	**5.1**	**7.2**	**13.8**	**22.2**	**36.2**
turbine blade	SoTA	2.6	4.9	13.5	25.0	34.8
	DTSDF	**2.3**	**3.1**	**5.7**	**10.5**	**17.2**
TUM fr1 desk1	SoTA	26.7	23.9	25.9	31.9	39.3
	DTSDF	**25.2**	**22.3**	**24.6**	**29.2**	**36.0**
TUM fr3 long office	SoTA	41.5	34.4	33.2	39.8	53.3
	DTSDF	**35.3**	**28.8**	**29.1**	**34.9**	**46.4**



(a) regular TSDF **(b)** DTSDF

Figure 14: Qualitative comparison of regular TSDF and DTSDF on turntable-style sequences. The lower left rectangle highlights artifacts from data fused from the backside. The upper right rectangle shows artifacts resulting from fusion conflicts between right- and front side.

similar, the DTSDF is better at retaining the map.

Table 5 lists the geometric post-fusion MAE of a selection of sequences (a complete list can be found in the Appendix Table 8). One can observe that in most cases there is not too much difference, especially in concave rooms like the ICL sequences. The effect usually only affects small parts of the model, like a corner or a computer monitor. Consequently, for the mean error over the whole image the effect is not that significant, but visible nonetheless. Object-scanning type sequence with the camera orbiting around objects generally seem to profit from the DTSDF more (c.f. Figure 14). Decreasing the voxel size certainly does mitigate some of these issues for the SotA, but ultimately the effect highlighted by the lower left rectangle remains for thin surfaces. Also, as Table 1, 3 show, halving the voxel size instead of using the DTSDF will require more memory and computation time.

Regarding the improvements of color fusion and rendering, Figure 19 gives a good example the DTSDF's advantage in color separation. While in the regular TSDF (Figure 19a) the colors blend because of fusion from two surfaces into the same voxels, the DTSDF retains different colors across edges (Figure 19c). Figure 19d shows which directions contribute to which rendered pixel. To quantify these improvements, we compute the photometric post-fusion per-frame MAE, analogue to the geometric error. The results, averaged over the datasets, are presented in Table 4. The effect is most visible at specific locations, but even the averaged error improves.

8.3 Photometric ICP

In this subsection, we analyze the impact of photometric ICP and especially compare the three reference modes `f2f`, `f2kf` and `f2r`. Again, all comparisons use the RMSE RPE.

In Table 7 we analyze over all datasets, whether the DTSDF or the regular TSDF has better tracking by means of a win-loss-tie table. We use a 0.1% tie rate, so if the RPEs differ by no more than 0.1% it's considered a tie. The results clearly show, that the DTSDF is the overall winner and in combination with photometric tracking, the gap is even higher. In the Zhou dataset, the DTSDF outperforms the SotA in most sequences. Especially in the noisy sequences, the regular TSDF fails completely at larger voxel sizes. Some sequences also confirm our hypothesis that the tracking performance of `f2r` degrades more with increased voxel size than the other variants. In Figure 15, the gap between `f2r` and the other modes increases with growing voxel size.

Table 6: Comparison of ICP results of photometric modes `f2f`, `f2kf` and `f2r` for different voxel sizes. The mode win column shows how many of the individual sequences are dominated by which mode. The rightmost three columns show the mean of the RPE ratios between `f2f`, `f2kf` and `f2r` w.r.t. geometric ICP (smaller is better).

voxel size [mm]	TSDF mode	mode win `f2f`–`f2kf`–`f2r`	mean $\frac{\text{RPE(f2f)}}{\text{RPE(geom)}}$	mean $\frac{\text{RPE(f2kf)}}{\text{RPE(geom)}}$	mean $\frac{\text{RPE(f2r)}}{\text{RPE(geom)}}$
5	SotA	9–**14**–10	1.512	1.412	**1.029**
	DTSDF	6–**14**–13	1.322	1.083	**1.006**
10	SotA	4–**16**–13	1.053	**0.990**	1.015
	DTSDF	5–**18**–10	1.102	**0.886**	1.028
20	SotA	6–**16**–11	0.976	**0.894**	1.117
	DTSDF	7–**19**–7	1.023	**0.914**	1.293
30	SotA	6–**17**–10	0.939	**0.905**	0.965
	DTSDF	7–**17**–9	0.927	**0.907**	1.109
40	SotA	5–**17**–9 [3]	1.027	**0.989**	1.329
	DTSDF	6–**21**–5 [3]	1.166	**0.931**	1.096

Table 7: RPE Win-loss-tie table for different datasets (format: SotA–DTSDF–tie).

dataset	geometric ICP	combined ICP `f2f`	combined ICP `f2kf`	combined ICP `f2r`
Zhou	1–**19**–0	1–**19**–0	1–**18**–0	4–**16**–0
TUM fr3	3–**21**–1	4–**19**–2	7–**18**–0	8–**17**–0
ICL NUIM	**32**–31–17	35–**36**–9	**34**–**34**–12	**42**–33–5
ICL	11–**26**–3	12–**26**–2	12–**26**–2	15–**23**–2
total	47–**97**–21	52-**100**-13	54–**96**–15	69–**89**–7

It is not straightforward to answer, which of the three reference methods is the definite winner, as the performance of the overall system is rated, which depends on many factors and small differences in one module can have a huge impact on the overall trajectory. Table 6 gives, however, a good indication: throughout all voxel sizes, the frame-to-keyframe method wins in the majority of sequences. In terms of the mean improvement over geometric ICP. Here, again, `f2kf` is the winner in most cases. Note, how `f2r` on average actually decreases tracking performance for voxel sizes larger than 5 mm, though individual sequences do profit from it. This shows that frame-to-keyframe offers good improvements over the other methods — even with the simple selection scheme of every 10th frame. Frame-to-frame is a good default choice, as it is often not far behind the frame-to-keyframe tracking and, thus, no attention to keyframe selection has to be paid. The drift typically experienced with frame-to-frame tracking is not so prominent, likely because of the combined ICP, i.e., the geometric frame-to-model tracking prevents excessive drift. Frame-to-render seems to be the overall worst, especially at lower resolutions. Without proper color equalization and lighting compensation the color inside the TSDF becomes unusable for tracking, as can be observed in Figure 7.

8.4 Runtime and Memory Consumption

All experiments were performed on an Intel i7-8700K with 3.70GHz and a GeForce RTX 3090. The CPU part of the code runs entirely on a single core. By reducing the changes to rendering the DTSDF while keeping the rest of the pipeline

[3] Tracking failed for some sequences at this resolution.

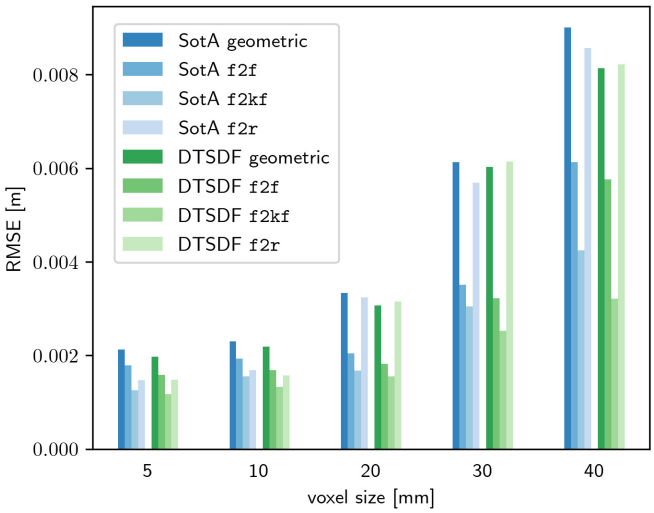


Figure 15: RPE of different tracking modes for sequence *ICL diamond walk*.

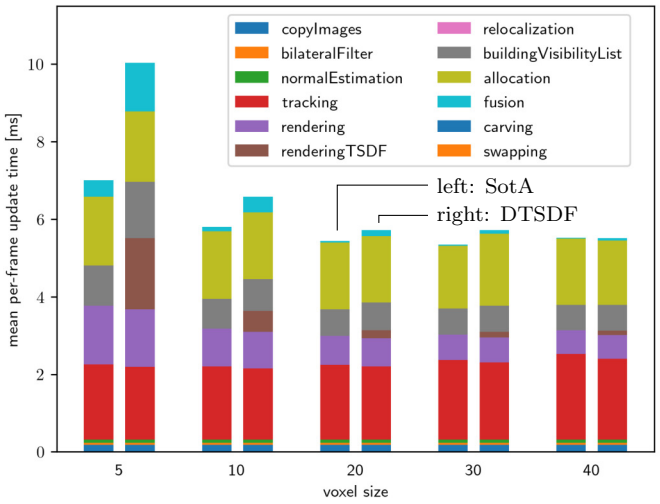


Figure 16: Mean per-frame update time comparison between state-of-the-art and DTSDF on SUN totempole sequence for different voxels sizes. Conditional combination is activated.

original, the runtime only differs in allocation, fusion and rendering. Figure 16 breaks down and compares runtimes for different voxels sizes. One can observe that the additional overhead is quite small. Note that for this example conditional combination was activated, and a rendering TSDF was computed for 35-40% of the frames with the conditions specified in Eq. (7)-(10).

As expected, the memory usage of the DTSDF is higher, as surfaces can overlap in up to three directions. In Figure 17 the ratio of additional memory required by the DTSDF w.r.t. the regular TSDF is displayed for ICL NUIM scenes. For smaller voxels, the amount for extra memory is quite small. With increasing voxel size, the ratio increases as blocks are allocated in chunks of $8 \times 8 \times 8$ voxels and more surfaces with different orientations fall into the same block, though the actual number of blocks is significantly smaller.

Figure 18 plots the memory ratio for various sequences of the datasets fused with 5 mm voxel size. It is also noticeable, that synthetic datasets (ICL, Zhou) use less memory than real ones. This is probably noise-related, as the depth-noise augmented ICL sequences also have a higher ratio, which suggests that with a more conservative allocation scheme memory can be saved. At the moment even for stray measurements blocks are allocated, as long as they have a valid normal. As surfaces can be fused

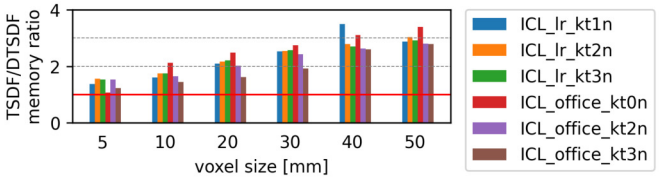
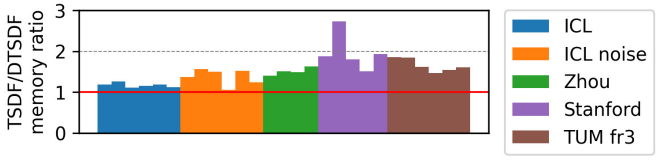
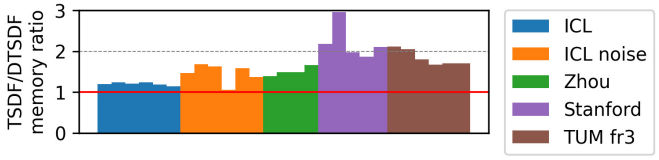


Figure 17: Ratio of allocated memory for DTSDF w.r.t. regular TSDF for different voxel sizes and scenes form the ICL NUIM dataset [52].



(a) best case: floor/walls aligned with coordinate axes.



(b) worst case: floor/walls 45° to all axes.

Figure 18: Ratio of allocated memory for DTSDF w.r.t. regular TSDF (red line) for various sequences of different datasets and 5 mm voxel size.

into up to three directions, determined by Eq. (2), an interesting question is how the alignment of the map coordinate frame to the scene affects memory usage. For this, we pre-computed initial poses for each sequence by identifying the largest planes of the sequence. In the best case scenario (Figure 18a), the coordinate axes are parallel; in the worst case (Figure 18b), tilted 45° to the identified planes. Noticeably, the alignment does have an effect, but not very significant. This is, however, highly dependent on the scenes and voxel sizes, because it induces only a one-time cost. Automating this process at the initialization phase of mapping is recommended.

Overall for the majority of scenes the DTSDF requires around 1.5 to 2 times as much memory as the regular TSDF.

9 Conclusion

In this work, we introduced the tools to use the DTSDF as a drop-in replacement for regular TSDF for mapping, tracking, and visualization applications. The ability to simply extract a regular TSDF for a given pose enables using it for a variety of tasks and with many algorithms that have been developed over the years.

We have shown that the DTSDF has advantages over the state-of-the-art for the majority of sequences, both qualitatively and quantitatively. Moreover, with the post-fusion MAE metric we showed that, while the regular TSDF is usable for local maps, reusability and revisiting of mapped places becomes problematic, if conflicting information from different surfaces corrupts the model. With reasonable memory and computation overhead, better results and a more consistent map can be obtained by the proposed DTSDF method. Our investigation and derivation of frame-to-keyframe photometric ICP has shown that it has clear benefits over frame-to-frame and frame-to-render tracking. **Sim**(3) pose refinement shows promising first results for the crop reconstruction project.

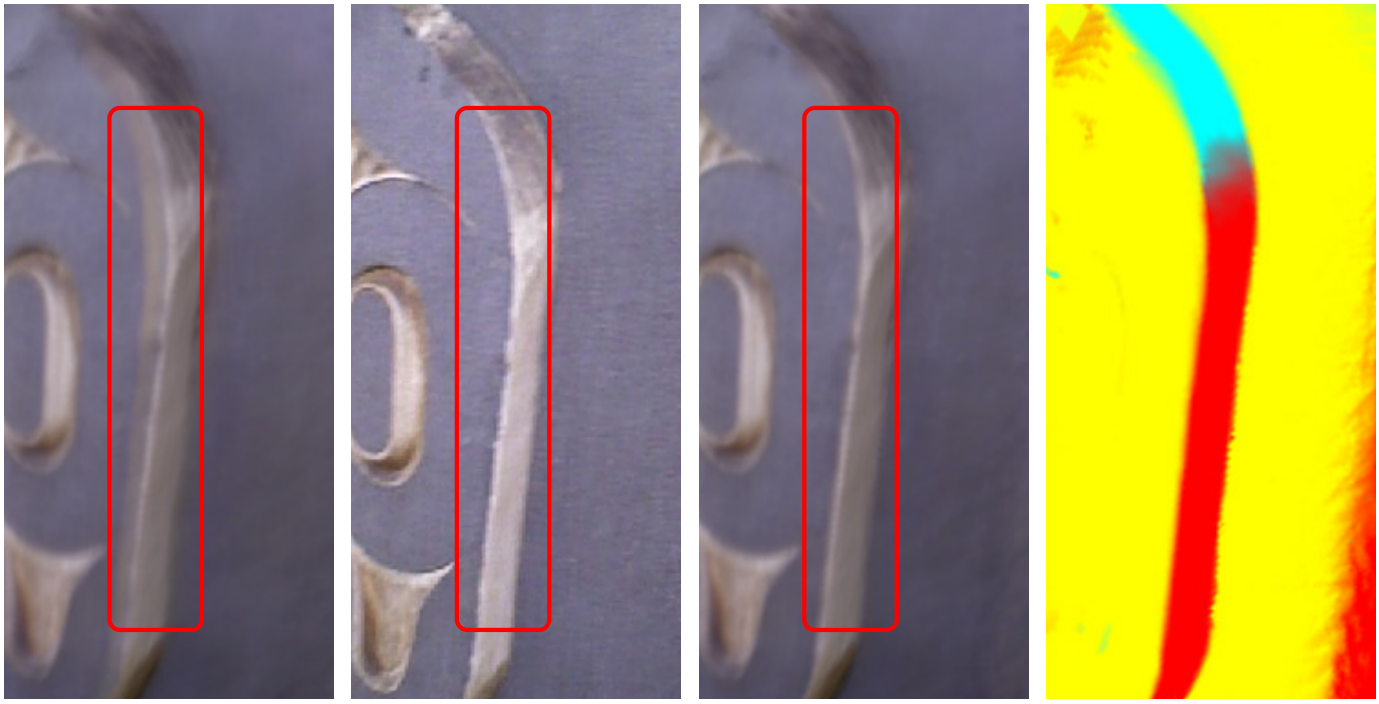
Acknowledgments

This work has been partially funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – EXC 2070 – 390732324 – PhenoRob.

Appendix

Table 8: MAE (in mm) of state-of-the-art and DTSDF compared for different voxel sizes and datasets.

dataset	mode	voxel size [mm]				
		5	10	20	30	40
SUN burghers	SoTA	**103.2**	**21.5**	19.9	27.4	41.0
	DTSDF	–	22.2	**18.3**	**24.3**	**33.9**
SUN copyroom	SoTA	23.0	**20.9**	24.8	54.4	41.8
	DTSDF	**20.7**	30.5	**21.7**	**25.9**	**30.0**
SUN cactusgarden	SoTA	**28.8**	**27.2**	**37.5**	52.8	69.9
	DTSDF	29.8	29.1	39.7	**50.9**	**59.5**
SUN lounge	SoTA	16.6	16.9	24.1	33.2	45.7
	DTSDF	**14.9**	**15.7**	**21.8**	**30.0**	**39.2**
SUN stonewall	SoTA	52.7	**22.1**	62.7	**33.5**	111.8
	DTSDF	**26.1**	60.8	**20.7**	66.6	**72.6**
SUN totempole	SoTA	**10.2**	9.9	12.5	17.4	24.1
	DTSDF	10.4	**9.5**	**10.9**	**14.5**	**19.2**
armadillo	SoTA	3.6	4.0	10.2	20.0	32.5
	DTSDF	**3.5**	**3.6**	**6.2**	**11.1**	**17.8**
bunny	SoTA	2.1	2.5	8.4	17.7	27.5
	DTSDF	**1.9**	**2.2**	**4.7**	**9.1**	**14.6**
dragon	SoTA	4.4	5.5	14.3	25.8	34.1
	DTSDF	**4.1**	**4.4**	**8.8**	**16.1**	**23.0**
turbine blade	SoTA	2.6	4.9	13.5	25.0	34.8
	DTSDF	**2.3**	**3.1**	**5.7**	**10.5**	**17.2**
Zhou lr1	SoTA	3.8	3.2	6.2	11.1	16.0
	DTSDF	**2.7**	**3.1**	**5.7**	**10.3**	**14.9**
Zhou lr2	SoTA	3.8	4.6	9.0	14.3	19.8
	DTSDF	**3.6**	**4.3**	**7.8**	**13.0**	**18.3**
Zhou office1	SoTA	4.4	6.5	18.3	56.4	58.5
	DTSDF	**4.2**	**5.4**	**10.1**	**17.4**	**26.2**
Zhou office2	SoTA	5.5	9.0	21.9	52.7	–
	DTSDF	**5.1**	**7.2**	**13.8**	**22.2**	**36.2**
ICL NUIM lr kt0	SoTA	**2.3**	**2.9**	6.1	7.5	**9.5**
	DTSDF	2.6	**2.9**	**5.8**	**7.4**	**9.5**
ICL NUIM lr kt1	SoTA	**2.1**	**2.1**	2.8	4.6	6.2
	DTSDF	**2.1**	**2.1**	**2.7**	**4.4**	**6.0**
ICL NUIM lr kt2	SoTA	**3.4**	**3.6**	**5.3**	8.5	12.3
	DTSDF	**3.4**	3.7	**5.3**	**8.3**	**12.0**
ICL NUIM lr kt3	SoTA	**71.9**	**31.5**	**41.5**	234.4	**25.5**
	DTSDF	89.8	83.1	69.0	**152.7**	124.0
ICL NUIM lr kt0n	SoTA	6.6	7.1	8.5	12.1	13.0
	DTSDF	**6.3**	**6.5**	**8.4**	**10.1**	**11.8**
ICL NUIM lr kt1n	SoTA	13.2	48.4	99.0	104.5	117.7
	DTSDF	**6.8**	**14.9**	**37.4**	**81.6**	**76.6**
ICL NUIM lr kt2n	SoTA	22.1	47.4	87.6	94.9	95.7
	DTSDF	**15.8**	**21.2**	**44.5**	**50.4**	**72.9**
ICL NUIM lr kt3n	SoTA	24.3	79.9	**134.5**	**53.2**	–
	DTSDF	**20.3**	**20.0**	138.9	97.2	**62.2**
ICL NUIM office kt0	SoTA	**2.4**	**2.6**	3.7	5.7	**6.9**
	DTSDF	**2.4**	**2.6**	**3.6**	**5.6**	**6.9**
ICL NUIM office kt1	SoTA	**1.1**	**1.2**	**1.8**	**2.6**	**4.0**
	DTSDF	**1.1**	**1.2**	**1.8**	**2.6**	4.1
ICL NUIM office kt2	SoTA	**2.6**	42.3	3.9	5.9	70.6
	DTSDF	2.7	**2.7**	**3.7**	**5.8**	**61.6**
ICL NUIM office kt3n	SoTA	459.8	376.5	258.3	159.2	**147.0**
	DTSDF	**83.9**	**5.8**	**6.3**	**154.3**	159.5



(a) TSDF **(b)** Input RGB **(c)** DTSDF **(d)** Contributing directions

Figure 19: Color bleeding effect on Stanford totempole sequence.

Table 8: (Continued)

dataset	mode	voxel size [mm]				
		5	10	20	30	40
TUM fr3 long office	SoTA	41.5	34.4	33.2	39.8	53.3
	DTSDF	**35.3**	**28.8**	**29.1**	**34.9**	**46.4**
TUM fr3 structure texture far	SoTA	35.9	**13.5**	12.4	15.1	17.8
	DTSDF	**21.9**	13.7	**11.5**	**12.7**	**14.1**
TUM fr3 structure texture near	SoTA	10.6	9.9	111.6	12.2	16.6
	DTSDF	**8.9**	**8.6**	**9.8**	**10.2**	**13.2**
TUM fr3 structure notexture far	SoTA	15.5	**10.7**	11.2	13.5	18.0
	DTSDF	**14.4**	11.7	**10.8**	**12.0**	**17.3**
TUM fr3 structure notexture near	SoTA	4.4	4.4	4.8	6.0	7.3
	DTSDF	**3.8**	**3.7**	**3.9**	**4.3**	**5.0**
TUM fr1 desk1	SoTA	26.7	23.9	25.9	31.9	39.3
	DTSDF	**25.2**	**22.3**	**24.6**	**29.2**	**36.0**

References

[1] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, A. Fitzgibbon, KinectFusion: Real-time dense surface mapping and tracking, in: IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR), 2011, pp. 127–136.

[2] M. Splietker, S. Behnke, Directional TSDF: Modeling surface orientation for coherent meshes, in: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2019, pp. 1727–1734.

[3] H. Oleynikova, Z. Taylor, M. Fehr, R. Siegwart, J. Nieto, Voxblox: Incremental 3D Euclidean signed distance fields for on-board MAV planning, in: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2017, pp. 1366–1373.

[4] L. Mescheder, M. Oechsle, M. Niemeyer, S. Nowozin, A. Geiger, Occupancy networks: Learning 3D reconstruction in function space, in: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2019, pp. 4455–4465.

[5] J. J. Park, P. Florence, J. Straub, R. Newcombe, S. Lovegrove, DeepSDF: Learning continuous signed distance functions for shape representation, in: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2019, pp. 165–174.

[6] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, R. Ng, NeRF: Representing scenes as neural radiance fields for view synthesis, Communications of the ACM 65 (1) (2021) 99–106.

[7] D. Azinović, R. Martin-Brualla, D. B. Goldman, M. Nießner, J. Thies, Neural RGB-D surface reconstruction, in: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2022, pp. 6290–6301.

[8] W. Dong, Q. Wang, X. Wang, H. Zha, PSDF Fusion: Probabilistic signed distance function for on-the-fly 3D data fusion and scene reconstruction, in: European Conference on Computer Vision (ECCV), 2018, pp. 701–717.

[9] P. Henry, D. Fox, A. Bhowmik, R. Mongia, Patch volumes: Multiple fusion volumes for consistent RGB-D modeling, in: RSS Workshop on RGB-D: Advanced reasoning with depth cameras, Berlin, Germany, 2013.

[10] T. Whelan, M. Kaess, H. Johannsson, M. Fallon, J. J. Leonard, J. McDonald, Real-time large-scale dense RGB-D SLAM with volumetric fusion, The International Journal of Robotics Research 34 (4-5) (2015) 598–626.

[11] A. Millane, Z. Taylor, H. Oleynikova, J. Nieto, R. Siegwart, C. Cadena, C-blox: A scalable and consistent TSDF-based

dense mapping approach, in: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2018, pp. 995–1002.

[12] S. Zhang, L. Zheng, W. Tao, Survey and evaluation of RGB-D SLAM, IEEE Access 9 (2021) 21367–21387.

[13] V. A. Prisacariu, O. Kähler, S. Golodetz, M. Sapienza, T. Cavallari, P. H. Torr, D. W. Murray, InfiniTAM v3: A framework for large-scale 3D reconstruction with loop closure (2017). arXiv:1708.00783.

[14] C. V. Nguyen, S. Izadi, D. Lovell, Modeling Kinect sensor noise for improved 3D reconstruction and tracking, in: International Conference on 3D Imaging, Modeling, Processing, Visualization and Transmission (3DIMPVT), 2012, pp. 524–530.

[15] J. K. Zhengyu Xia, Joohee Kim, Real-time 3D reconstruction using a combination of point-based and volumetric fusion, in: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2018, pp. 8449–8455.

[16] F. Steinbrücker, J. Sturm, D. Cremers, Real-time visual odometry from dense RGB-D images, in: Proc. IEEE Int. Conf. Computer Vision Workshops (ICCV Workshops), 2011, pp. 719–722.

[17] C. Kerl, J. Sturm, D. Cremers, Dense visual SLAM for RGB-D cameras, in: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2013, pp. 2100–2106.

[18] A. Dai, M. Nießner, M. Zollhöfer, S. Izadi, C. Theobalt, BundleFusion: Real-time globally consistent 3D reconstruction using on-the-fly surface reintegration, ACM Transactions on Graphics (TOG) 36 (3) (2017) 24.

[19] E. Bylow, C. Olsson, F. Kahl, Robust online 3D reconstruction combining a depth sensor and sparse feature points, in: International Conference on Pattern Recognition (ICPR), 2016, pp. 3709–3714.

[20] D. R. Canelhas, T. Stoyanov, A. J. Lilienthal, SDF Tracker: A parallel algorithm for on-line pose estimation and scene reconstruction from depth images, in: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2013, pp. 3671–3676.

[21] E. Palazzolo, J. Behley, P. Lottes, P. Giguère, C. Stachniss, ReFusion: 3D reconstruction in dynamic environments for RGB-D cameras exploiting residuals, in: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2019, pp. 7855–7862.

[22] M. Slavcheva, W. Kehl, N. Navab, S. Ilic, SDF-2-SDF registration for real-time 3D reconstruction from RGB-D data, International Journal of Computer Vision 126 (2018) 615–636.

[23] A. J. Millane, H. Oleynikova, C. Lanegger, J. Delmerico, J. Nieto, R. Siegwart, M. Pollefeys, C. Cadena Lerma, Freetures: Localization in signed distance function maps, IEEE Robotics and Automation Letters.

[24] R. Mur-Artal, J. M. M. Montiel, J. D. Tardos, ORB-SLAM: a versatile and accurate monocular SLAM system, IEEE Transactions on Robotics 31 (5) (2015) 1147–1163.

[25] E. Bylow, J. Sturm, C. Kerl, F. Kahl, D. Cremers, Real-time camera tracking and 3D reconstruction using signed distance functions., in: Robotics: Science and Systems (RSS), Vol. 2, 2013.

[26] I. Dryanovski, M. Klingensmith, S. S. Srinivasa, J. Xiao, Large-scale, real-time 3D scene reconstruction on a mobile device, Autonomous Robots 41 (2017) 1423–1445.

[27] W. Dong, J. Shi, W. Tang, X. Wang, H. Zha, An efficient volumetric mesh representation for real-time scene reconstruction using spatial hashing, in: IEEE International Conference on Robotics and Automation (ICRA), 2018, pp. 6323–6330.

[28] M. Klingensmith, I. Dryanovski, S. Srinivasa, J. Xiao, Chisel: Real time large scale 3D reconstruction onboard a mobile device using spatially hashed signed distance fields., in: Robotics: Science and Systems (RSS), Vol. 4, 2015.

[29] J.-L. Blanco, A tutorial on SE(3) transformation parameterizations and on-manifold optimization, University of Malaga, Tech. Rep 3 (2010) 6.

[30] B. Bellekens, V. Spruyt, R. Berkvens, M. Weyn, A survey of rigid 3D pointcloud registration algorithms, in: Fourth International Conference on Ambient Computing, Applications, Services and Technologies (AMBIENT), Rome, Italy, 2014, pp. 8–13.

[31] M. Nießner, M. Zollhöfer, S. Izadi, M. Stamminger, Real-time 3D reconstruction at scale using voxel hashing, ACM Transactions on Graphics (ToG) 32 (6) (2013) 169.

[32] J. Engel, T. Schöps, D. Cremers, LSD-SLAM: Large-scale direct monocular SLAM, in: European Conference on Computer Vision (ECCV), 2014, pp. 834–849.

[33] C. Forster, Z. Zhang, M. Gassner, M. Werlberger, D. Scaramuzza, SVO: Semidirect visual odometry for monocular and multicamera systems, IEEE Transactions on Robotics 33 (2017) 249–265.

[34] J. Engel, V. Koltun, D. Cremers, Direct sparse odometry, IEEE Transactions on Pattern Analysis and Machine Intelligence 40 (2018) 611–625.

[35] E. Sucar, S. Liu, J. Ortiz, A. Davison, iMAP: Implicit mapping and positioning in real-time, in: IEEE International Conference on Computer Vision (ICCV), 2021, pp. 6229–6238.

[36] L. Yang, Q. Yan, Y. Fu, C. Xiao, Surface reconstruction via fusing sparse-sequence of depth images, IEEE Transactions on Visualization and Computer Graphics 24 (2) (2017) 1190–1203.

[37] J. Kuo, M. Muglikar, Z. Zhang, D. Scaramuzza, Redesigning SLAM for arbitrary multi-camera systems, in: IEEE International Conference on Robotics and Automation (ICRA), 2020, pp. 2116–2122.

[38] B. K. P. Horn, Closed-form solution of absolute orientation using unit quaternions, J. Opt. Soc. Am. A 4 (4) (1987) 629–642.

[39] Y. Sahillioglu, L. Kavan, Scale-adaptive ICP, Graph. Model. 116 (2021) 101113.

[40] H. Yang, J. Shi, L. Carlone, TEASER: Fast and certifiable point cloud registration, IEEE Transactions on Robotics 37 (2) (2021) 314–333.

[41] S. Du, N. Zheng, L. Xiong, S. Ying, J. Xue, Scaling iterative closest point algorithm for registration of m–d point sets, Journal of Visual Communication and Image Representation 21 (5-6) (2010) 442–452.

[42] Y. Chen, G. Medioni, Object modelling by registration of multiple range images, Image and Vision Computing 10 (3) (1992) 145–155.

[43] P. Furgale, J. Rehder, R. Siegwart, Unified temporal and spatial calibration for multi-sensor systems, in: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2013, pp. 1280–1286.

[44] H. Hirschmuller, Accurate and efficient stereo processing by semi-global matching and mutual information, in: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Vol. 2, 2005, pp. 807–814.

[45] G. Yang, J. Manela, M. Happold, D. Ramanan, Hierarchical deep stereo matching on high-resolution images, in: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2019, pp. 5515–5524.

[46] J. L. Schönberger, J.-M. Frahm, Structure-from-motion revisited, in: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 4104–4113.

[47] J. L. Schönberger, E. Zheng, M. Pollefeys, J.-M. Frahm, Pixelwise view selection for unstructured multi-view stereo, in: European Conference on Computer Vision (ECCV), 2016, pp. 501–518.

[48] O. Kähler, V. Prisacariu, J. Valentin, D. Murray, Hierarchical voxel block hashing for efficient integration of depth images, IEEE Robotics and Automation Letters 1 (1) (2015) 192–197.

[49] P. Stotko, stdgpu: Efficient STL-like data structures on the GPU (2019). `arXiv:1908.05936`.

[50] K. Madsen, H. Nielsen, O. Tingleff, Methods for non-linear least squares problems (2nd ed.) (01 2004).

[51] Q.-Y. Zhou, V. Koltun, Dense scene reconstruction with points of interest, ACM Transactions on Graphics 32 (4) (2013) 112.

[52] A. Handa, T. Whelan, J. McDonald, A. J. Davison, A benchmark for RGB-D visual odometry, 3D reconstruction and SLAM, in: IEEE International Conference on Robotics and Automation (ICRA), 2014, pp. 1524–1531.

[53] S. Choi, Q.-Y. Zhou, V. Koltun, Robust reconstruction of indoor scenes, in: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2015, pp. 5556–5565.

[54] J. Sturm, N. Engelhard, F. Endres, W. Burgard, D. Cremers, A benchmark for the evaluation of RGB-D SLAM systems, in: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2012, pp. 573–580.

[55] S. Saeedi, E. D. Carvalho, W. Li, D. Tzoumanikas, S. Leutenegger, P. H. Kelly, A. J. Davison, Characterizing visual localization and mapping datasets, in: IEEE International Conference on Robotics and Automation (ICRA), 2019, pp. 6699–6705.