2013 IEEE/RSJ International Conference on
Intelligent Robots and Systems (IROS)
November 3-7, 2013. Tokyo, Japan

SDF Tracker: A Parallel Algorithm for On-line Pose Estimation and Scene Reconstruction From Depth Images

Daniel R. Canelhas, Todor Stoyanov, Achim J. Lilienthal
Center of Applied Autonomous Sensor Systems (AASS), Örebro University, Sweden

Abstract— **Ego-motion estimation and environment mapping are two recurring problems in the field of robotics. In this work we propose a simple on-line method for tracking the pose of a depth camera in six degrees of freedom and simultaneously maintaining an updated 3D map, represented as a truncated signed distance function. The distance function representation implicitly encodes surfaces in 3D-space and is used directly to define a cost function for accurate registration of new data. The proposed algorithm is highly parallel and achieves good accuracy compared to state of the art methods. It is suitable for reconstructing single household items, workspace environments and small rooms at near real-time rates, making it practical for use on modern CPU hardware.**

I. Introduction

Work has been done in recent years to leverage the power of parallel hardware architectures to produce practical real-time solutions for detailed and accurate scene reconstruction (e.g. [1]). These methods have the potential to open new doors to robotic applications in which carefully planned interaction with the physical environment is necessary. The benefits gained from the use of parallel computational hardware depend on the data structure used to represent the environment and the methods used to update and maintain it. This paper presents a highly parallel algorithm for camera pose-estimation and scene reconstruction, made possible by using a distance function as the representation of choice.

We implement a truncated signed distance function (TSDF) over a 3D space by implicit definition through discrete samples on a cubic lattice (also referred to as voxels). At any point in space, the function evaluates to a scalar value that represents the signed distance of that point to the nearest surface in the environment. Closer to a surface the absolute value of the distance decreases, while its sign indicates if a point is behind (negative) or in front (positive) of a surface. Thus the TSDF provides means for distinguishing between points inside and outside of objects. A TSDF facilitates accurate reconstruction of surfaces, provides a metric for the misalignment of new data respective to previously observed parts of the environment, and encodes the approximate gradient direction towards the closest surface. The TSDF representation is useful to more general robotic applications, since it can be used to quickly and accurately compute surface position, orientation and curvature, which may be relevant to critical tasks e.g., grasp planning, navigation, object detection and collision avoidance [2].

In this article, we propose a 6 DoF ego-motion estimation system that uses depth images as input to maintain an updated 3D map of the environment while simultaneously tracking the camera pose. The proposed algorithm is conceptually simple and trivially parallelized. The map is updated on-line and fuses multiple measurements into a single consistent model, averaging out noise associated with the input. We make available an open-source reference implementation [1] using OpenMP [2] as a package for the Robot Operating System (ROS) [3] and provide quantitative results on a well-known RGB-D (color and depth images) dataset with known ground truth [4].

This work is related to the Kinect Fusion algorithm proposed by Newcombe et al. [5] by operating in the same problem domain, i.e. that of simultaneous pose tracking of a structured light sensor and data fusion into a model represented as a TSDF. However, the core of our data registration algorithm is quite different. Our method registers data directly to the TSDF model, rather than using it to obtain denoised depth images from a virtual sensor. One of the benefits is that our method does not depend on explicit correspondences. New depth images are registered directly to the model, which is independent of viewpoint and does not suffer from occlusions inherent to perspective projections. The proposed algorithm, like other dense methods, uses all available data from each input depth image. It does so in a parallel fashion, allowing for accurate registration and high quality scene reconstruction at an update frequency of up to 16 Hz (depending on size and resolution of the reconstructed volume). The accuracy of the environment mapping and ego-motion estimation is comparable to current dense reconstruction methods employing commodity structured light sensors. We show that even though our algorithm may accumulate drift over time, it compares well to modern SLAM-based methods that operate on sparse features, while featuring near real-time perrformance on a system without a GPU.

The next section provides a review of the TSDF as a 3D representation, followed by details of the camera pose estimation algorithm. We present results on an RGB-D benchmark for hand-held SLAM algorithms and finalize with a few concluding remarks.

II. 3D TSDF Representation

The properties and methods related to the 3D TSDF representation are very well documented in the work of Curless and Levoy [6] and more recently reviewed in the

[1] http://code.google.com/p/oru-ros-pkg/. 2012

[2] the OpenMP Architecture Review Board, The OpenMP API specification for parallel programming, *http://www.openmp.org* (accessed 2012/09)

work of Newcombe et al. [5]. For convenience we will briefly describe the representation below.

The representation used in this work is a TSDF, based on projective measurements. The TSDF is a voxel-based data structure in which each cell in a 3D grid stores an approximate distance to the nearest surface. A 2D illustration is shown in Fig. 1 and a real example, following the same color convention, is given in Fig. 2. For any given voxel, its distance value is computed as the difference in depth between its coordinates and the coordinates of the nearest measured surface point. The comparison is made in the frame of reference of the camera. The resulting distance assumes positive values for voxels located in the free space observed in front of surfaces and negative values for voxels that are situated behind observed surfaces. The difference in sign is then used to define a region of "zero distance", which implicitly represents the actual surface of objects. The distance values are truncated at pre-defined positive and negative limits, D_{max} and D_{min}, respectively. The truncation limits the impact of new observations in the environment to local changes. Furthermore, a true signed distance function cannot be computed with certainty from partial observations, since there is no information as to how far the negative side should extend on the unseen side of surfaces.

To accurately represent a surface using a TSDF, at least one non-truncated voxel is needed on both sides of a surface. In this non-truncated region it is then possible to interpolate between a negative (inside) and positive (outside) voxel to obtain an estimated location for the surface (represented by zero). However, in this work the TSDF is not only used to represent the surface, but also directly model the error of a given point with respect to the current model, and provide a gradient towards the surface. To estimate this gradient, our method typically requires the number of non-truncated voxels on either side to be larger than strictly needed for surface representation. The thickness of this non-truncated region needs to be related to the amount of physical camera movement that is expected between frames. It is important to note that the thickness on either side need not be symmetric and should preferably be kept small on the negative side, since it sets a limit for the minimum dimensions of reconstructed objects.

To improve the quality of the surface reconstruction and robustness to outliers, each voxel stores not only a signed distance, but also a weight W that corresponds to the certainty of the distance value in the cell. As been suggested by [5], [6] either a uniform weight can be used, producing a "rolling average", or the weight W can be related to an error model of the depth camera. In both cases, the update of the weight and distance values is as follows [6]:

$$D_n(\boldsymbol{x})_{n+1} = \frac{D_n(\boldsymbol{x})W_n(\boldsymbol{x}) + \hat{D}_n(\boldsymbol{x})\hat{W}(\boldsymbol{x})}{W(\boldsymbol{x})_n + \hat{W}(\boldsymbol{x})}, \quad (1)$$

$$W(\boldsymbol{x})_{n+1} = min(W(\boldsymbol{x})_n + \hat{W}(\boldsymbol{x}), W_{max}), \quad (2)$$

where $D_n(\boldsymbol{x})_{n+1}$ is the updated distance at location $\boldsymbol{x}$ based on the estimate $\hat{D}_n(\boldsymbol{x})$. The weight $W(\boldsymbol{x})_n$ is the accumulated sum (up to a limit W_{max}) of weights $\hat{W}(\boldsymbol{x})$, associated with the distances estimated for a particular position in the voxel grid. Limiting the value of W allows for the model to change in order to represent new configurations of the environment and react robustly to dynamic elements. The volume update can be efficiently done in parallel, since no dependence is assumed between voxels.

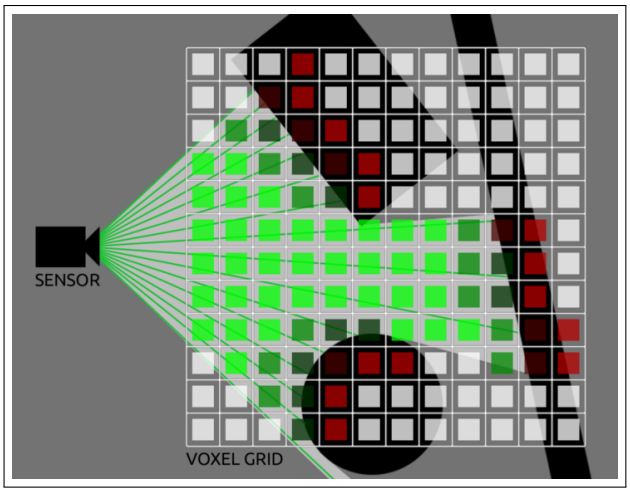


Fig. 1. This figure illustrates how a TSDF is generated from a depth image. The coordinates of each voxel are projected into the image plane and their depth (or "z-value") relative to the camera is compared with that of the nearest pixel in the depth image. The result of this comparison is written into the voxel. Positive (green) values are truncated to a pre-defined maximum. Negative (red) values less than a pre-defined minimum are not stored. White cells are those that have not yet been observed by the sensor.

III. Estimation of Camera Pose

A. Notation

Formally, a depth image is a scalar function $z_n(\boldsymbol{M})$ that assigns a depth to all pixels of each of the $n = 0 \dots N$ depth images in a video stream. The domain of z_n is defined on the 2D image plane $\boldsymbol{M} \in \mathbb{R}^2$. Formally, $z_n : \boldsymbol{M} \to \mathbb{R}_+$.

Given $z_n(\boldsymbol{M})$, and knowing the parameters of the depth camera, 3D surface points can be computed. Let $\boldsymbol{s}_n(\boldsymbol{M})$ be the projection of a depth image pixel into 3D. Formally, $\boldsymbol{s}_n : \mathbb{R}^2 \times \mathbb{R} \to \mathbb{R}^3$,

$$\boldsymbol{s}_n(\boldsymbol{m}) = \begin{bmatrix} \frac{m_1 - c_x}{f_x} z_n(\boldsymbol{m}) \\ \frac{m_2 - c_y}{f_y} z_n(\boldsymbol{m}) \\ z_n(\boldsymbol{m}) \end{bmatrix}, \quad (3)$$

where $\boldsymbol{m} = (m_1, m_2) \in \boldsymbol{M}$ represents an image pixel and $c_x, c_y, f_x, f_y \in \mathbb{R}$ represent the principal point and focal lengths of a pinhole camera model. The definition of a 3D rigid-body transformation, $\boldsymbol{T} \in \mathbb{R}^{4\times4} \in \mathbb{SE}(3)$ used in registration of points is,

$$\boldsymbol{T} = \begin{bmatrix} \boldsymbol{R} & \boldsymbol{t} \\ \boldsymbol{0}^T & 1 \end{bmatrix} \quad (4)$$

where $\boldsymbol{R} \in \mathbb{R}^{3\times3} \in \mathbb{SO}(3)$ is a 3D rotation matrix, $\boldsymbol{t} \in \mathbb{R}^3$ is a translation vector and $\boldsymbol{0}^T$ is a zero vector of appropriate dimensions. Close to $\boldsymbol{T} = \boldsymbol{I}$, with $\boldsymbol{I}$ being a 4×4 Identity matrix, we can accurately represent $\boldsymbol{T}$ as $\boldsymbol{T} = \boldsymbol{e}^{\Delta t \hat{\boldsymbol{\xi}}}$, where $\boldsymbol{e}$ is the matrix exponential function, Δt is a duration of time

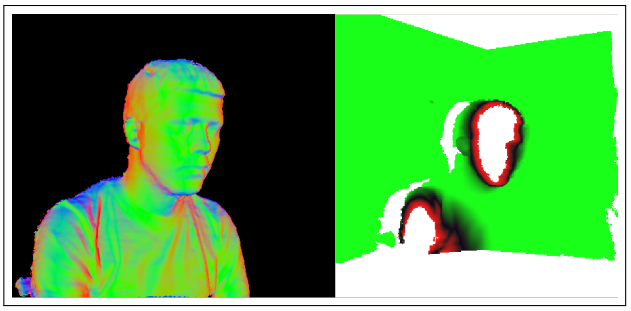


Fig. 2. Left: Raycasted 3D reconstruction with color corresponding to surface orientation. Right: 2D slice through the corresponding TSDF volume, red/green indicate negative/positive sign, brightness indicates absolute values (brighter is higher). Unseen areas are marked as white, following the same convention as in Fig. 1.

(regarded here as unitary) and $\boldsymbol{\xi} \in \mathbb{R}^6 \in \mathfrak{se}_3$ is a vector representing angular and linear velocities, i.e.

$$\boldsymbol{\xi} = \begin{bmatrix} \omega_1 & \omega_2 & \omega_3 & v_1 & v_2 & v_3 \end{bmatrix}^T \tag{5}$$

and $\hat{\boldsymbol{\xi}} \in \mathbb{R}^{4\times4}$ is the matrix,

$$\hat{\boldsymbol{\xi}} = \begin{bmatrix} 0 & -\xi_3 & \xi_2 & \xi_4 \\ \xi_3 & 0 & -\xi_1 & \xi_5 \\ -\xi_2 & \xi_1 & 0 & \xi_6 \\ 0 & 0 & 0 & 0 \end{bmatrix}. \tag{6}$$

With these definitions, $\boldsymbol{T}$ can be expressed as a function of $\boldsymbol{\xi}$, as long as it remains in the neighbourhood of $\boldsymbol{\xi} = \mathbf{0}$.

B. Objective Function

In order to define an appropriate objective function, we first examine a measure of error as the sum of squared distances between corresponding points in 3D, from two consecutive images, i.e.,

$$E = \sum_i^M \|\boldsymbol{s}_n(\boldsymbol{m}_i) - \boldsymbol{s}_{n+1}(\boldsymbol{m}_j)\|^2. \tag{7}$$

such that for every i we assume that there is a mapping to j denoting the corresponding pixels in the subsequent image frame. To bring these points into alignment, a transformation is applied to one of the two sets. For convenience we will use a dot to denote a vector written in homogeneous coordinates, e.g., $\dot{\boldsymbol{u}} = \left[\begin{smallmatrix} \boldsymbol{u} \\ 1 \end{smallmatrix}\right]$, defining an optimization objective as,

$$\min_{\boldsymbol{\xi}} \sum_i^M \|\dot{\boldsymbol{s}}_n(\boldsymbol{m}_i) - \boldsymbol{T}(\boldsymbol{\xi})\dot{\boldsymbol{s}}_{n+1}(\boldsymbol{m}_j)\|^2. \tag{8}$$

Let $D_n(\boldsymbol{x})$ be the TSDF, generated from the past n depth images, as described in Sec. II (for this work updating of $D_n(\boldsymbol{x})$ is done through a simple rolling average over past measurements). Using, $D_n(\boldsymbol{x})$, the above objective can be approximated as,

$$\min_{\boldsymbol{\xi}} \sum_i^M \|D_n(\boldsymbol{T}(\boldsymbol{\xi})\dot{\boldsymbol{s}}_{n+1}(\boldsymbol{m}_i))\|^2. \tag{9}$$

In other words, we are minimizing the sum of squared point-to-model distances, over the space of transformations of the points relative to the model. Note that we no longer require an explicit mapping between i and j.

C. Solution

We obtain the solution of the optimization problem defined in (9) by linearizing the objective function by means of a first-order power-series approximation around $\boldsymbol{\xi} = \mathbf{0}$, i.e.,

$$\begin{aligned} &D_n(\boldsymbol{T}(\boldsymbol{\xi})\dot{\boldsymbol{s}}_{n+1}(\boldsymbol{m}_i)) \approx \\ &D_n(\boldsymbol{T}(\boldsymbol{\xi})\dot{\boldsymbol{s}}_{n+1}(\boldsymbol{m}_i))|_{\boldsymbol{\xi}=\mathbf{0}} + \\ &\nabla_{\boldsymbol{\xi}} D_n(\boldsymbol{T}(\boldsymbol{\xi})\dot{\boldsymbol{s}}_{n+1}(\boldsymbol{m}_i))|_{\boldsymbol{\xi}=\mathbf{0}}\boldsymbol{\xi}. \end{aligned} \tag{10}$$

The assumption that we are close to $\boldsymbol{\xi} = \mathbf{0}$ holds as long as the inter-frame movement is small, which is true, granted that the camera speed or the time taken to process each depth image is not excessive. Noting that $\boldsymbol{T} = \boldsymbol{I}$ for $\boldsymbol{\xi} = \mathbf{0}$, Eq. (10) simplifies to,

$$\begin{aligned} &D_n(\boldsymbol{T}(\boldsymbol{\xi})\dot{\boldsymbol{s}}_{n+1}(\boldsymbol{m}_i)) \approx \\ &D_n(\dot{\boldsymbol{s}}_{n+1}(\boldsymbol{m}_i)) + \nabla_{\boldsymbol{\xi}} D_n(\dot{\boldsymbol{s}}_{n+1}(\boldsymbol{m}_i))\boldsymbol{\xi}. \end{aligned} \tag{11}$$

In (11) we identify the summand to the Jacobian matrix as,

$$\boldsymbol{J}(\boldsymbol{m}_i) = \nabla_{\boldsymbol{\xi}} D_n(\dot{\boldsymbol{s}}_{n+1}(\boldsymbol{m}_i)), \tag{12}$$

where $\boldsymbol{J}(\boldsymbol{m}_i) \in \mathbb{R}^{1\times6}$ and return to (9) to plug the new definitions back into the objective.

$$\min_{\boldsymbol{\xi}} \sum_i^M \|D_n(\dot{\boldsymbol{s}}_{n+1}(\boldsymbol{m}_i)) + \boldsymbol{J}(\boldsymbol{m}_i)\boldsymbol{\xi}\|^2. \tag{13}$$

Expanding the square and simplifying the algebraic expression we end up with,

$$\begin{aligned} &\min_{\boldsymbol{\xi}} \sum_i^M \boldsymbol{\xi}^T \boldsymbol{J}(\boldsymbol{m}_i)^T \boldsymbol{J}(\boldsymbol{m}_i)\boldsymbol{\xi} + \\ &2\boldsymbol{\xi}^T \boldsymbol{J}(\boldsymbol{m}_i)^T D_n(\dot{\boldsymbol{s}}_{n+1}(\boldsymbol{m}_i)) + D_n(\dot{\boldsymbol{s}}_{n+1}(\boldsymbol{m}_i))^2. \end{aligned} \tag{14}$$

Carrying out the sum over the pixels in $\boldsymbol{M}$ for the terms dependent on $\boldsymbol{\xi}$, we can define the following matrix and vector, respectively,

$$\boldsymbol{H} = \sum_i^M \boldsymbol{J}(\boldsymbol{m}_i)^T \boldsymbol{J}(\boldsymbol{m}_i), \tag{15}$$

$$\boldsymbol{g} = \sum_i^M \boldsymbol{J}(\boldsymbol{m}_i)^T D_n(\dot{\boldsymbol{s}}_{n+1}(\boldsymbol{m}_i)). \tag{16}$$

Differentiating (14) with respect to $\boldsymbol{\xi}$ and equating the result to zero, we find the least-squares solution $\boldsymbol{\xi}^\star$ by,

$$\boldsymbol{\xi}^\star = -\boldsymbol{H}^{-1}\boldsymbol{g}. \tag{17}$$

The set of points are transformed by $\boldsymbol{T}(\boldsymbol{\xi}^\star)$ and the process is repeated for several iterations. The estimated transform represents the incremental change between the current pose and the previous frame. Since the parametrization of $\boldsymbol{T}(\boldsymbol{\xi})$ is only valid locally around $\boldsymbol{\xi} = \mathbf{0}$ we reset $\boldsymbol{\xi}$ to zero and pre-transform each newly arrived depth image with a global transformation matrix $\bar{\boldsymbol{T}}_{n+1} = \boldsymbol{T}(\boldsymbol{\xi}^\star)\bar{\boldsymbol{T}}_n$, with $\bar{\boldsymbol{T}}_0 = \boldsymbol{I}$, representing the pose change since the beginning of tracking.

Though it is not strictly necessary for convergence, we iterate using a coarse-to-fine sub-sampling on the input depth image, as this provides significant speed-ups in the

registration by providing a fast initial alignment [7]. A fixed number of iterations are performed on each level of detail, or until the change in the optimization parameter falls below some pre-defined threshold.

To improve the basin of convergence for the solution, we scale the contribution of each measurement, based on a weighing function. This produces the standard iteratively reweighed least-squares algorithm, changing (15) and (16) to

$$\boldsymbol{H}_w = \sum_i^M w(D_n(\dot{\boldsymbol{s}}_{n+1}(\boldsymbol{m}_i)))\boldsymbol{J}(\boldsymbol{m}_i)^T\boldsymbol{J}(\boldsymbol{m}_i), \quad (18)$$

$$\boldsymbol{g}_w = \sum_i^M w(D_n(\dot{\boldsymbol{s}}_{n+1}(\boldsymbol{m}_i)))\boldsymbol{J}(\boldsymbol{m}_i)^T D_n(\dot{\boldsymbol{s}}_{n+1}(\boldsymbol{m}_i)). \quad (19)$$

The weighing function $w(x)$ characterizes an *M-estimator* [8] for which a sensible choice is the Huber estimator [9]. It can be defined as,

$$w(x) = \begin{cases} 1.0 & \text{if } |x| <= k \\ \frac{k}{|x|} & \text{otherwise} \end{cases}, \quad (20)$$

where k is a small constant (e.g. a tenth of the size of one voxel). It may also be beneficial to extend this weighing function to include an error model of the sensor, e.g. reducing the influence of measurements made at greater distance.

Lastly, we note that adding a small cost related to the norm of the parameter vector $\boldsymbol{\xi}$ itself (a regularizer), has benefits in situations where the solution would otherwise tend to oscillate around the minimum or lead to very inaccurate results. This typically happens when the solution is not adequately constrained by the geometry seen in the environment. Adding $\boldsymbol{\xi}^T\boldsymbol{W}\boldsymbol{\xi}$ into (12), where $\boldsymbol{W} \in \mathbb{R}^{6\times6}$ is a diagonal matrix attributing cost to the norm of each individual component of $\boldsymbol{\xi}$, results in,

$$\boldsymbol{H}_{w,r} = \sum_i^M \boldsymbol{W} + w(D_n(\dot{\boldsymbol{s}}_{n+1}(\boldsymbol{m}_i)))\boldsymbol{J}(\boldsymbol{m}_i)^T\boldsymbol{J}(\boldsymbol{m}_i), \quad (21)$$

where $\boldsymbol{W} = \alpha\boldsymbol{I}$, $\boldsymbol{I}$ being the 6×6 Identity matrix and α a linearly increasing function of the number of the current solution iteration. Large values for α will tend to dampen the amount of change made to the parameter vector at each iteration and such dampening is only interesting once the solution is close to optimal. To summarize, a high-level overview of the system is provided in Fig. 3.

D. Limitations

A general downside of dense volumetric scene representations are the large memory requirements for storage at high resolution. Recent work (e.g. [10],[11]) allows the reconstructed volume to move, providing pose estimation across larger distances, but maintaining the dense surface reconstruction around the current sensor location. Other approaches, based on non-uniform divisions of space (rather than a fixed-size cubical lattice) have been proposed (e.g. [12]) and can also be used to increase the represented space, without additional memory use.

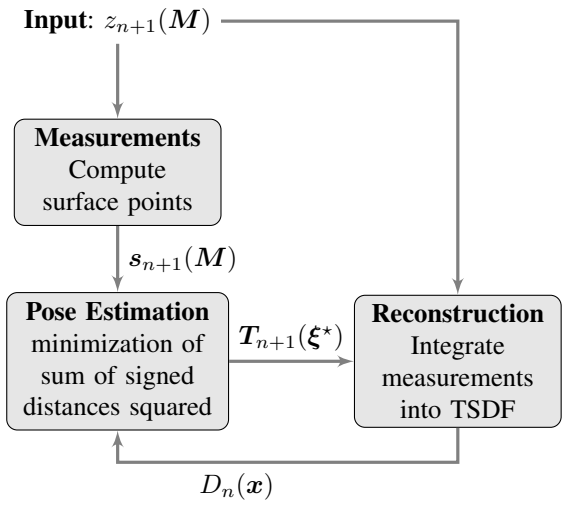


Fig. 3. Main components of the system, and information flow

Since our method uses the TSDF to represent alignment error, it needs a distance function that is truncated at larger values than methods that simply use the TSDF as a scene representation. The surface points measured in the second frame need to be within a region where numerical derivatives can be computed (see Eq.(12)) given the motion between two consecutive video frames. Since the truncation occurs at larger values, the method is unable to reconstruct details as fine as what would be possible using separate representations for tracking and mapping.

Apart from these limitations, the proposed method may lose track of the pose if the assumption that we are close to the solution does not hold or if the currently viewed geometry does not offer enough variation to constrain all six degrees of freedom, causing $\boldsymbol{H}$ to be close to singular.

IV. Related Work

The presented method is a fast approach to solving the implied registration problem of camera-tracking. It is similar to the approach presented by Fitzgibbon as Fast ICP using the distance transform (FICP) [9]. However, the full Euclidean distance transform (EDT) used by FICP is too costly to compute for the on-line scenario in the scope of this work. Due to the distance values being unsigned, surface extraction also becomes less straightforward. A signed version of the EDT is similarly costly, but provides additional challenges when faced with partial observations [13]. Both for sake of speed, and due to incomplete observations of the environment, we avoid computing the Signed EDT, and instead proceed as detailed in Sec. II. As an analogy, one can think of the presented method as pre-computing an approximate closest-point distance throughout the entire volume, making evaluations as simple as accessing a look-up table, without the need for matching corresponding points explicitly. The Kinect Fusion algorithm to which we compare our results, solves the camera tracking problem by maintaining an updated TSDF from which it can generate virtual depth images by means of ray-casting from a virtual sensor. Using the depth images from the virtual camera and the actual sensor,

TABLE I

COMPARATIVE RESULTS, THE PROPOSED ALGORITHM (OURS), AN OPEN-SOURCE IMPLEMENTATION OF THE KINECT FUSION ALGORITHM (PCL-KINFU), FEATURE-BASED NORMAL DISTRIBUTIONS TRANSFORM REGISTRATION (NDT-F) AND RGB-D SLAM

		Abs err. [m]		Rel err. [m]		Rel err. [deg]	
Dataset	Algorithm	RMS	max	RMS	max	RMS	max
rgbd_dataset_freiburg1_xyz (trajectory length: 7.112 m)	Ours	0.014	0.036	**0.003**	**0.012**	0.472	1.810
	PCL-KinFu	0.023	0.070	0.004	0.056	0.474	1.738
	RGB-D SLAM	0.014	**0.035**	0.006	0.021	**0.353**	**1.633**
	NDT-F	0.068	0.125	0.014	0.228	0.844	11.137
rgbd_dataset_freiburg1_desk (trajectory length: 9.263 m)	Ours	0.033	0.079	**0.007**	**0.051**	0.759	**3.336**
	PCL-KinFu	0.073	0.256	0.020	0.272	2.003	28.317
	RGB-D SLAM	**0.026**	0.079	0.012	0.063	**0.731**	6.855
	NDT-F	0.072	0.122	0.019	0.176	1.405	15.358
rgbd_dataset_freiburg1_desk2 (trajectory length: 10.161 m)	Ours	0.230	0.378	0.019	0.246	1.080	9.785
	PCL-KinFu	0.102	0.312	0.020	0.194	1.795	28.660
	RGB-D SLAM	**0.043**	**0.183**	0.018	0.218	**1.067**	9.632
	NDT-F	0.114	0.249	0.018	**0.107**	1.219	**9.241**
rgbd_dataset_freiburg1_floor (trajectory length: 12.569 m)	Ours	0.984	2.573	0.050	0.462	2.085	35.172
	PCL-KinFu	0.918	1.936	0.035	0.435	1.718	25.175
	RGB-D SLAM	**0.035**	**0.085**	**0.004**	**0.027**	**0.292**	**1.929**
	NDT-F	0.269	0.556	0.025	0.327	0.888	10.728

TABLE II

PARAMETERS USED DURING EVALUATION

Parameter	value
Voxels	$320 \times 320 \times 320$
Voxel size [m]	0.03
truncation (positive) [m]	0.1
truncation (negative) [m]	-0.06
Huber constant k [m]	0.003
Iteration levels	3
Downsampling / level	$\times 4, \times 2, \times 1$
Iterations / level	$12, 6, 2$
Regularization α	$0.001 \times$ iteration_count
Stopping condition (interrupts current level)	$\Delta\|\boldsymbol{\xi}\| < 0.0001$

a multi-scale point-to-plane ICP method is used to compute a transformation between them. The update of the TSDF is achieved in the same way as in this work. Interesting to note, is that the Kinect Fusion algorithm could be modified to work just as well with a different model representation (as long as a dense depth image can be generated from it). This is quite different from our method, in which the model representation and the cost function are inseparable.

V. RESULTS

To evaluate the accuracy of the proposed algorithm in an off-line setting, we tested on a subset of the RGB-D datasets from the "hand-held SLAM" category, provided by the CVPR group at the Technical University of Munich [4]. The parameters and constants introduced in Sec. III are listed in Table II with the values used throughout the benchmark. We evaluate the accuracy of our algorithm by computing the absolute trajectory error, relative position error and relative orientation error as indicated in [4]. We compare the results to those of Kinect Fusion (as implemented by the Point-Cloud Library[3] [14]) and to two algorithms that use depth information in conjunction with visual features, namely RGB-D SLAM [15], and feature-based 3D Normal Distributions Transform (NDT-F) [16].

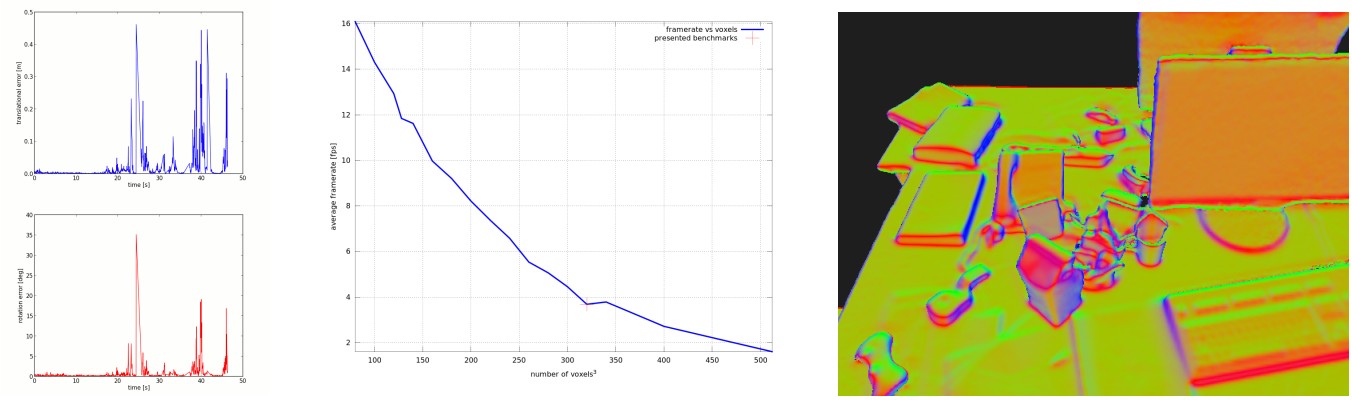
[3]Open Source Kinect Fusion implementation, PCL: http://svn.pointclouds.org/pcl/trunk, accessed: Sept. 2012.

As can be seen in Table I, our method achieves similar performance to the tested visual feature-based methods, in spite of tracking from depth alone and not performing any pose-graph optimization. Our method slightly outperforms the reference implementation of Kinect Fusion on the "desk" and "xyz" datasets but fares slightly worse on the "desk2" and "floor" sets. It is interesting to note that RGB-D SLAM typically has a larger inter-frame error but, due to global optimization and loop-closures, achieves a smaller absolute error. Compared to the results obtained using NDT-F around visual features, we generally achieve better results. This is expected, since NDT-F is a frame-to-frame method and can therefore not take advantage of a map with reduced noise. A notable exception to these statements is the rather poor performance of the proposed algorithm on the "floor" sequence. A closer look at the relative errors in pose estimation for that data set, shown in Fig. 4(a) reveals large spikes both in the rotational and translational error, occurring after ca. 25s. Looking at the input data for which this error is produced, we note that it is indeed a representative example for one of the failure modes discussed in Sec. III-D (namely, $\boldsymbol{H}$ close to singular due to lack of features). The Kinect Fusion algorithm produces similar behaviour, for the same reasons.

The run-time of our algorithm, as measured on an Intel Core i7-3740QM (2.70 GHz, 4 cores) CPU depends on the number of voxels used and is presented in Fig. 4(b). Performance scales up with the number of cores available on the system and their speed. The algorithm spends the majority of its time evaluating the TSDF which requires fetching eight floating-point values from memory, performing a tri-linear interpolation between them and casting the result to a double-precision value. This operation is needed to compute the error associated with a surface point, but also to compute the derivative of the error relative to position.

VI. CONCLUSIONS

We have presented a camera-tracking method that uses the 3D scene representation directly as a cost function to perform 6 DoF alignment of 3D surface points. The trajectories



(a) (b) (c)

Fig. 4. Fig. 4(a): Relative errors — translation (top) and orientation (bottom). A large spike is seen in both graphs at ca. 25s, caused by an insufficiently constrained solution for the depth-based algorithms. Fig. 4(b): The time required to process each frame increases with the number of voxels. Fig. 4(c): Example reconstruction from the fb1_xyz dataset, using 400^3 voxels with $0.006m$ edge length.

estimated by our method, on a well-known dataset, are comparable to those of current state of the art methods, including algorithms which, in addition to depth also employ visual features. Our main contribution lies in a direct model-based approach to on-line registration, as opposed to generating virtual sensor data from a model and performing registration in a sensor-centric frame of reference. This allows high quality surface reconstruction (e.g. Fig. 4(c)) and accurate pose estimation to be done on systems without a GPU. However, in lack of standardized implementations for both CPU and GPU architectures, practical comparisons to other algorithms in terms of computational complexity are difficult make. In addition, we make our implementation freely available to the benefit of the robotics community for use and verification of our results.

In future work, alternative objective functions, based on the point-to-plane [17] metric could be used for better convergence, at a modest increase in computational cost. Embedding information about local texture into the TSDF may provide additional improvements for situations where depth alone is insufficient to successfully estimate the sensor motion.

ACKNOWLEDGMENT

D.R.C. thanks R. A. Newcombe, S. Lovegrove and J. Sturm for explanations, inspiration and support in the course of this work. This work has partly been supported by the European Commission under contract number FP7-ICT-270350 (RobLog).

REFERENCES

[1] R. Newcombe, S. Lovegrove, and A. Davison, "Dtam: Dense tracking and mapping in real-time," in *Computer Vision (ICCV), 2011 IEEE Int. Conf. on*. IEEE, 2011, pp. 2320–2327.

[2] A. Fuhrmann, G. Sobottka, and C. Gross, "Abstract distance fields for rapid collision detection in physically based modeling," in *Int. Conf. on Computer Graphics and Vision (Graphicon)*, 2003, pp. 58–65.

[3] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, 2009.

[4] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of rgb-d slam systems," in *Proc. of the IEEE Int. Conf. on Intelligent Robot Systems (IROS)*, 2012.

[5] R. Newcombe, A. Davison, S. Izadi, P. Kohli, O. Hilliges, J. Shotton, D. Molyneaux, S. Hodges, D. Kim, and A. Fitzgibbon, "Kinectfusion: Real-time dense surface mapping and tracking," in *Mixed and Augmented Reality (ISMAR), 10th IEEE Int. Symp. on*, 2011, pp. 127–136.

[6] B. Curless and M. Levoy, "A volumetric method for building complex models from range images," in *Proc. of the 23rd conf. on Computer graphics and interactive techniques*. ACM, 1996, pp. 303–312.

[7] D. Ricao Canelhas, "Scene representation, registration and objectdetection in a truncated signed distance functionrepresentation of 3d space," Master's thesis, Örebro University, 2012.

[8] Z. Zhang, "Parameter estimation techniques: A tutorial with application to conic fitting," *Image and vision Computing*, vol. 15, no. 1, pp. 59–76, 1997.

[9] A. Fitzgibbon, "Robust registration of 2d and 3d point sets," *Image and Vision Computing*, vol. 21, no. 13, pp. 1145–1153, 2003.

[10] T. Whelan, M. Kaess, M. Fallon, H. Johannsson, J. Leonard, and J. McDonald, "Kintinuous: Spatially extended kinectfusion," in *Proc. of RSS workshop on RGB-D: Advanced Reasoning with Depth Cameras*, 2012.

[11] H. Roth and M. Vona, "Moving volume kinectfusion," in *British Machine Vision Conf.(BMVC),(Surrey, UK)*, 2012.

[12] S. Frisken, R. Perry, A. Rockwood, and T. Jones, "Adaptively sampled distance fields: A general representation of shape for computer graphics," in *Proc. of the 27th conf. on Computer graphics and interactive techniques*, 2000, pp. 249–254.

[13] P. Mullen, F. De Goes, M. Desbrun, D. Cohen-Steiner, and P. Alliez, "Signing the unsigned: Robust surface reconstruction from raw pointsets," in *Comp. Graphics Forum*, vol. 29, 2010, pp. 1733–1741.

[14] R. Rusu and S. Cousins, "3d is here: Point cloud library (pcl)," in *Robotics and Automation (ICRA), IEEE Int. Conf. on*, 2011, pp. 1–4.

[15] N. Engelhard, F. Endres, J. Hess, J. Sturm, and W. Burgard, "Real-time 3d visual slam with a hand-held rgb-d camera," in *Proc. of the RGB-D Workshop on 3D Perception in Robotics at Euron Forum*, 2011.

[16] H. Andreasson and T. Stoyanov, "Real time registration of rgb-d data using local visual features and 3d-ndt registration," in *SPME Workshop at Int. Conf. on Robotics and Automation (ICRA)*, 2012.

[17] C. Yang and G. Medioni, "Object modelling by registration of multiple range images," *Image and vision computing*, vol. 10, no. 3, pp. 145–155, 1992.