Real-time large scale dense RGB-D SLAM with volumetric fusion

Journal:	*International Journal of Robotics Research*
Manuscript ID:	IJR-13-1804
Manuscript Type:	Robot Vision
Date Submitted by the Author:	31-Jul-2013
Complete List of Authors:	Whelan, Thomas; Computer Science Kaess, Michael; Massachusetts Institute of Technology, Johannsson, Hordur; MIT, CSAIL Fallon, Maurice; Massachusetts Institute of Technology, Computer Science and Artificial Intelligence Lab Leonard, John; Massachusetts Institute of Technology, Mechanical Engineering McDonald, John; Computer Science
Keyword:	Localization < Mobile and Distributed Robotics SLAM, Mapping < Mobile and Distributed Robotics SLAM, Visual Tracking < Sensing and Perception Computer Vision
Abstract:	We present a new SLAM system capable of producing high quality globally consistent surface reconstructions over hundreds of metres in real-time with only a cheap commodity RGB-D sensor. By using a fused volumetric surface reconstruction we achieve a much higher quality map over what would be achieved using raw RGB-D point clouds. In this paper we highlight three key techniques associated with applying a volumetric fusion-based mapping system to the SLAM problem in real-time. First, the use of a GPU-based 3D cyclical buffer trick to efficiently extend dense every frame volumetric fusion of depth maps to function over an unbounded spatial region. Second, overcoming camera pose estimation limitations in a wide variety of environments by combining both dense geometric and photometric camera pose constraints. Third, efficiently updating the dense map according to place recognition and subsequent loop closure constraints by the use of an "as-rigid-as-possible" space deformation. We present results on a wide variety of aspects of the system and show through evaluation on de facto standard RGB-D benchmarks that our system performs strongly in terms of trajectory estimation, map quality and computational performance.

Note: The following files were submitted by the author for peer review, but cannot be converted to PDF. You must view these files (e.g. movies) online.
ijrr1.m4v ijrr2.m4v

SCHOLARONE™
Manuscripts

For Peer Review

Real-time large scale dense RGB-D SLAM with volumetric fusion

Thomas Whelan, Michael Kaess, Hordur Johannsson, Maurice Fallon, John J. Leonard and John McDonald

DRAFT: July 29, 2013

Abstract

We present a new SLAM system capable of producing high quality globally consistent surface reconstructions over hundreds of metres in real-time with only a cheap commodity RGB-D sensor. By using a fused volumetric surface reconstruction we achieve a much higher quality map over what would be achieved using raw RGB-D point clouds. In this paper we highlight three key techniques associated with applying a volumetric fusion-based mapping system to the SLAM problem in real-time. First, the use of a GPU-based 3D cyclical buffer trick to efficiently extend dense every frame volumetric fusion of depth maps to function over an unbounded spatial region. Second, overcoming camera pose estimation limitations in a wide variety of environments by combining both dense geometric and photometric camera pose constraints. Third, efficiently updating the dense map according to place recognition and subsequent loop closure constraints by the use of an "as-rigid-as-possible" space deformation. We present results on a wide variety of aspects of the system and show through evaluation on *de facto* standard RGB-D benchmarks that our system performs strongly in terms of trajectory estimation, map quality and computational performance.

Keywords: volumetric fusion, camera pose estimation, dense methods, large scale, real-time, RGB-D, SLAM, GPU

For Peer Review

1 Introduction

The ability for a robot to create a map of an unknown environment and localise within that map is of extreme importance in intelligent autonomous operation. Simultaneous Localisation and Mapping (SLAM) has been one of the large focuses of robotics research over the last two decades, with 3D mapping becoming more and more popular within the last few years over traditional 2D laser scan SLAM. The recent explosion in full dense 3D SLAM is arguably a result of the release of the Microsoft Kinect commodity RGB-D sensor, which provides high quality depth sensing capabilities for a little over one hundred US dollars. Before the advent of the Kinect, 3D SLAM methods required either time of flight (TOF) sensors, 3D LIDAR scanners or stereo vision, which were typically either quite expensive or not suitable for fully mobile real-time operation if dense reconstruction was desired. Another recent technology which is often coupled with dense methods is General-Purpose computing on Graphics Processing Units (GPGPU) which exploits the massive parallelism available in GPU hardware to perform high speed and often real-time processing on entire images every frame. Being an affordable commodity technology, GPU-based programming is arguably another large enabler in recent dense SLAM research.

Many visual SLAM systems and 3D reconstruction systems (both offline and online) have been published in recent times that rely purely on RGB-D sensing capabilities because of the Kinect's low price and accuracy; Henry et al. (2012); Endres et al. (2012); Stückler and Behnke (2013). The KinectFusion algorithm of Newcombe et al. (2011) is one of the most notable RGB-D-based 3D reconstruction systems of recent times, which allows real-time volumetric dense reconstruction of a desk sized scene at sub-centimetre resolution. KinectFusion enables reconstructions of an unprecedented quality at real-time speeds but comes with a number of limitations, namely 1) restriction to a fixed small area in space; 2) reliance on geometric information alone for camera pose estimation and 3) no means of explicitly incorporating loop closures. These three limitations severely limit the applicability of KinectFusion to the large scale SLAM problem where it is desirable due to its real-time nature and very high surface reconstruction fidelity.

In this paper we present solutions to the three aforementioned limitations such that the system can be used in a full real-time large scale SLAM setting. We address the three limitations respectively by 1) representing the volumetric reconstruction data structure in memory with a rolling cyclical

Draft manuscript, July 29, 2013. Submitted to IJRR.

T. Whelan and J. McDonald are with the Department of Computer Science, National University of Ireland Maynooth, Co. Kildare, Ireland. thomas.j.whelan@nuim.ie, johnmcd@cs.nuim.ie.

M. Kaess, H. Johannsson, M. Fallon and J. Leonard are with the Massachusetts Institute of Technology (MIT), Cambridge, Massachusetts 02139, USA {kaess,hordurj,mfallon,jleonard}@mit.edu.

This work was presented in part at the Robotics Science and Systems RGB-D Workshop, Sydney, July 2012 (Whelan et al. (2012)), in part at the International Conference on Robotics and Automation, Karlsruhe, May 2013 (Whelan et al. (2013a)) and in part will appear at the International Conference on Intelligent Robots and Systems, Japan, November 2013 (Whelan et al. (2013b)).

buffer; 2) estimating a dense photometric camera constraint in conjunction with a dense geometric constraint and jointly optimising for a camera pose estimate and 3) optimising the dense map by means of a non-rigid space deformation parameterised by a loop closure constraint. Following we provide a discussion on the existing work related to the area of dense RGB-D SLAM and in the remainder of this paper describe our three main contributions along with our evaluation of the system.

1.1 Related Work

A large number of publications have been made over the last few years specifically using RGB-D data for camera pose estimation, dense mapping and full SLAM pipelines. One of the earliest RGB-D tracking and mapping systems uses FAST feature correspondences between frames for visual odometry and offloads dense point cloud map building to a post processing step utilising sparse bundle adjustment (SBA) for global consistency by minimizing feature reprojection error (Huang et al. (2011)). One of the first real-time dense RGB-D tracking and mapping systems estimates an image warping function with both geometric and photometric information to compute a camera pose estimate, however only relies on rigid reprojection for point cloud map reconstruction without using a method for global consistency (Audras et al. (2011)). Similar work on dense RGB-D camera tracking was done by Steinbruecker et al. (2011), also estimating an image warping function based on geometric and photometric information. Recent work by Kerl et al. (2013) presents a more robust dense photometrics-based RGB-D visual odometry system that proposes a t-distribution-based error model which more accurately matches the residual error between RGB-D frames in scenes that are not entirely static.

Henry et al. (2012) presented one of the first full SLAM systems based entirely upon RGB-D data, using visual feature matching with Generalised Iterative Closest Point (GICP) to build up a pose graph and following that an optimised surfel map of the area explored. The use of pose graph optimisation versus SBA is studied, minimising feature reprojection error in an offline rigid transformation framework. Visual feature correspondences are used in conjunction with pose graph optimisation in the RGB-D SLAM system of Endres et al. (2012). An octree-based volumetric representation is used to store the map, created by reprojecting all point measurements into the global frame. This map representation is provided by the OctoMap framework of Hornung et al. (2013), which includes the ability to take measurement uncertainties into account and implicitly represent free and occupied space while being space efficient. An explicit voxel volumetric occupancy representation is used by Pirker et al. (2011) in their GPSlam system which uses sparse visual feature correspondences for camera pose estimation. They make use of visual place recognition and sliding window bundle adjustment in a pose graph optimisation framework. To achieve global consistency the occupancy grid is "morphed" by a weighted average of the log-odds perceptions of each camera for each voxel. Stückler and Behnke (2013) register surfel maps together for camera pose estimation and store a multi-resolution surfel map in an octree, using pose graph optimisation for global consistency. After pose graph optimisation is complete a globally consistent map is created by fusing key views together. In recent work Hu et al. (2012) proposed a system that uses bundle adjustment in order to make use of pixels for which no valid depth exists, and Lee et al. (2012) presented a system which exploits GPU processing power for real-time camera tracking. Both systems produce an optimised map as a final step in the process.

A substantial number of derived works have been published recently after the advent of the KinectFusion system of Newcombe et al. (2011), mostly focused on extending the range of operation, with other related work on object recognition and motion planning (Karpathy et al. (2013); Wagner et al. (2013)). Recent work by Bylow et al. (2013) and Canelhas et al. (2013) directly tracks the camera pose against the accumulated volumetric model by exploiting the fact that the truncated signed distance function (TSDF) representation used by KinectFusion stores the signed distance to the closest surface at voxels near the surface. This avoids the need to raycast a vertex map each frame to perform camera pose estimation, which potentially discards information about the surface reconstruction.

Roth and Vona (2012) extend the operational range of KinectFusion by using a double buffering mechanism to map between volumetric models upon camera translation and rotation, using a voxel interpolation for the latter. However no method for recovering the map is listed. Zeng et al. (2012) replace the explicit voxel representation used by KinectFusion with an octree representation which allows mapping of areas up to 8m×8m×8m in size. However this method does increase the chance for drift within the map and provides no means of loop closure or map correction. Keller et al. (2013) present an extended fusion system made space efficient by using a point-based surfel representation, although lacking in drift correction or loop closure detection. Chen et al. (2013) present a novel hierarchical data structure that enables extremely space efficient volumetric fusion, using a streaming framework allowing effectively unbounded mapping range, limited only by available memory. However the system lacks any method for mitigating drift or enforcing global consistency.

An alternative approach to the modern SLAM problem is introduced by Salas-Moreno et al. (2013), whereby known objects are detected, tracked and mapped in real-time in a dense RGB-D framework. Pose graph optimisation is used to ensure global consistency on the level of camera poses and detected object positions. This does allow loop closure, however less influence is placed on a full scene reconstruction with only point cloud reprojections being used for mapped loop closure. Finally, recent work by Henry et al. (2013) uses multiple smaller "patch volumes" to segment the mapped space into a set of discrete TSDFs, each with a 6-degrees-of-freedom (6-

For Peer Review

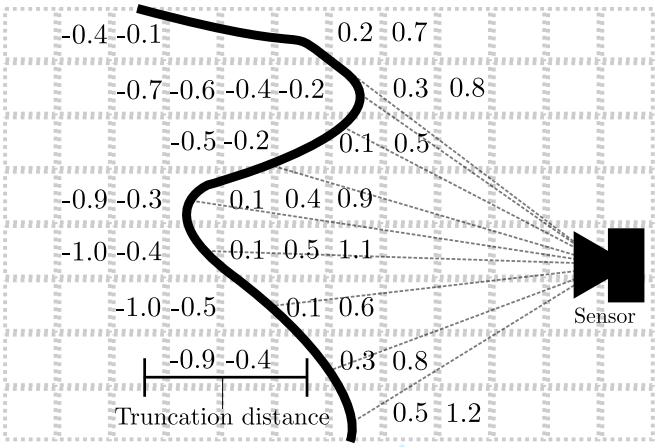


Figure 1: Two dimensional example of the structure of the truncated signed distance function representation of an implicit surface. Shown are example signed distance values stored at voxels within the truncation distance of the observed surface, with rays cast from the observing sensor.

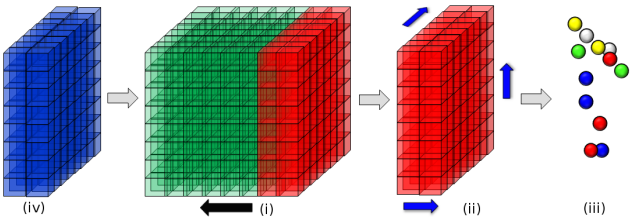


Figure 2: Visualisation of the volume shifting process for spatially extended mapping; (i) The camera motion exceeds the movement threshold m_s (direction of camera motion shown by the black arrow); (ii) Volume slice leaving the volume (red) is raycast along all three axes to extract surface points and reset to free space; (iii) The raycast surface is extracted as a point cloud and fed into the Greedy Projection Triangulation (GPT) algorithm of Marton et al. (2009); (iv) New region of space (blue) enters the volume and is integrated using new modulo addressing of the volume.

DOF) pose which is rigidly optimised upon loop closure detection. This approach can be seen as similar to the SLAM++ approach of Salas-Moreno et al. (2013) whereby the patch volumes are analogous to objects. While achieving global consistency between each volume, there is no clear solution presented for correcting the surface within any one given volume or stitching surfaces which are split between volumes, leaving local surfaces disconnected.

As discussed there exists a large number of systems utilising RGB-D data for SLAM and related problems. However, most are either unable to operate in real-time, provide an up-to-date optimised representation of the map at runtime or any time it is requested or efficiently incorporate large non-rigid updates to the map. Non-rigid surface correction is of great interest specifically in the realm of volumetric fusion as typically reconstructions are locally extremely accurate but drift slowly over large scales over time, where a smooth continuous deformation of the surface is most suitable for correction. In the following sections we will fully describe our approach to RGB-D SLAM with volumetric fusion which is capable of functioning in real-time over large scale trajectories, while efficiently applying non-rigid updates to the dense map upon loop closure to ensure global consistency.

2 Extended Scale Volumetric Fusion

In this section we will provide some background on the usage of volumetric fusion for dense RGB-D-based tracking and mapping and describe our extension to the premier system using this approach to allow spatially extended mapping.

2.1 Background

Real-time volumetric fusion with RGB-D cameras was brought to the forefront by Newcombe et al. (2011) with the KinectFusion system. A significant component of the system is the cyclical pipeline used for camera tracking and scene mapping, whereby full depth maps are fused into a volumetric data structure (TSDF), which is then raycast to produce a predicted surface that the subsequently captured depth map is matched against using ICP. The truncated signed distance function (TSDF) is a volumetric data structure that encodes implicit surfaces by storing the signed distance to the closest surface at each voxel up to a given truncation distance from the actual surface position. Points at which the sign of the distance value changes are known as zero crossings, which represent the actual position of the surface, shown in Figure 1. Each voxel also stores a weight for the distance measurement at that point, effectively providing a moving average of the surface position. In the case of KinectFusion, the TSDF is stored as a three dimensional voxel grid in GPU memory where dense depth map integration is accomplished by sweeping through the volume and updating distance measurements accordingly, while surface raycasting is carried out by simply projecting rays from the current camera pose and returning the depth and surface normals at the first zero crossings encountered. Surface normals are easily computed by taking the finite difference around a given position within the TSDF, as exploited by Bylow et al. (2013) and Canelhas et al. (2013). The entire process is very amenable to parallelisation and greatly benefits in execution time from being implemented on a GPU (Newcombe et al. (2011)).

2.2 Volume Representation

Defining the voxel space domain as $\Psi \subset \mathbb{N}^3$ the TSDF volume S at some location $\mathbf{s} \in \Psi$ has the mapping $S(\mathbf{s}) : \Psi \rightarrow \mathbb{R} \times \mathbb{N} \times \mathbb{N}^3$. Within GPU memory the TSDF is represented as a 3D array of voxels. Each voxel contains a signed distance value ($S(\mathbf{s})^T$, truncated float16), an unsigned weight value ($S(\mathbf{s})^W$, unsigned int8) and a byte for each color component R, G and B ($S(\mathbf{s})^R, S(\mathbf{s})^G, S(\mathbf{s})^B$) for a total of 6 bytes per voxel. The integration of new surface measurements is carried out in a similar fashion to Newcombe et al. (2011), when integrating a new signed distance function measurement $S(\mathbf{s})_i^T$ during the fusion of a new depth map, each voxel $\mathbf{s} \in \Psi$ at time i is

For Peer Review

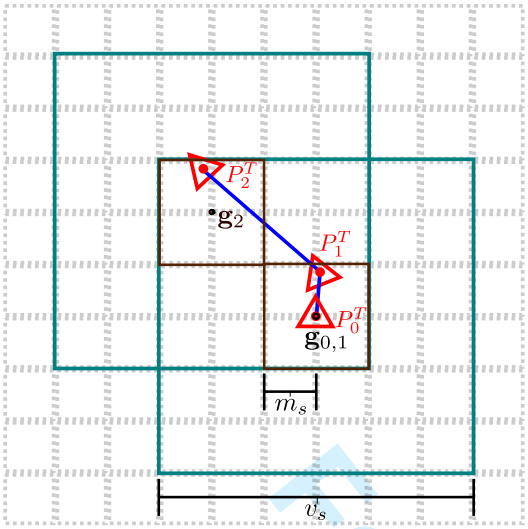


Figure 3: Visualisation of the interaction between the movement threshold m_s and the shifting process. Between frames 0 and 1 the camera does not cross the movement boundary (dark brown) and no shift occurs. At frame 2, the pose crosses the boundary and causes a volume shift, recentering the volume (teal) around P_2^T and updating $\mathbf{g}_2$. The underlying voxel grid quantisation is shown in light dashed lines.

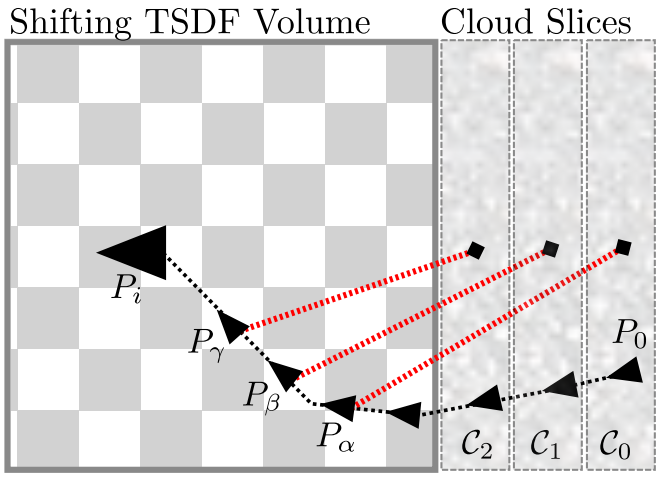


Figure 4: Two dimensional visualisation of the association between extracted cloud slices, the camera poses and the TSDF volume. Note that the camera poses here are in global coordinates rather than internal TSDF coordinates. A red dashed line links camera poses with extracted slices of the TSDF volume (P_γ, P_β and P_α with C_2, C_1 and C_0 respectively). The large triangles represent camera poses that caused volume shifts while the small black squares represent those that didn't.

updated with:

$$S(\mathbf{s})_i^{T'} = \frac{S(\mathbf{s})_{i-1}^W S(\mathbf{s})_{i-1}^T + S(\mathbf{s})_i^W S(\mathbf{s})_i^T}{S(\mathbf{s})_{i-1}^W + S(\mathbf{s})_i^W} \quad (1)$$

$$S(\mathbf{s})_i^{W'} = \min(S(\mathbf{s})_{i-1}^W + S(\mathbf{s})_i^W, max_weight) \quad (2)$$

As is the case with previous approaches, we take $S(\mathbf{s})_i^W = 1$ to provide a simple moving average. Using only a cubic volume, we parameterise the TSDF by the side length in voxels v_s and the dimension in metres v_d. Both of these parameters control the resolution of the reconstruction along with the size of the immediate "active area" of reconstruction. In all experiments in this paper we set $v_s = 512$ for total GPU memory usage of 768MB. The 6-DOF camera pose within the TSDF at time i is denoted as P_i^T, composed of a rotation $P_{i_\mathbf{R}}^T \in \mathbb{SO}_3$ and a translation $P_{i_\mathbf{t}}^T \in \mathbb{R}^3$. The origin of the TSDF coordinate system is positioned at the center of the volume with basis vectors aligned with the axes of the TSDF. Initially $P_{0_\mathbf{R}}^T = \mathbf{I}$ and $P_{0_\mathbf{t}}^T = (0,0,0)^\top$. The position of the TSDF volume in voxel units in the global frame is initialised to be $\mathbf{g}_0 = (0,0,0)^\top$.

2.3 Volume Shifting

Unlike Newcombe et al. (2011) camera pose estimation and surface reconstruction is not restricted to only the region around which the TSDF was initialised. By employing modulo arithmetic in how the TSDF volume is addressed in GPU memory we can treat the structure like a cyclical buffer which virtually translates as the camera moves through an environment. Figure 2 provides a visual example and description of the shifting process. It is parameterised by an integer movement threshold m_s, defining the cubic movement boundary (in voxels) around $\mathbf{g}_i$ which upon crossing, causes a volume shift, shown in Figure 3. Discussion on the choice of value for m_s is provided in Section 5.3. Each dimension is treated independently during a shift. When a shift is triggered, the TSDF is virtually translated about the camera pose (in voxel units) to bring the camera's position to within one voxel of $\mathbf{g}_{i+1}$. The new pose of the camera P_{i+1}^T has no change in rotation, while the shift corrected camera position $P_{i+1_\mathbf{t}}^{T'}$ is calculated from $P_{i+1_\mathbf{t}}^T$ by first computing the number of voxel units crossed:

$$\mathbf{u} = \left\lfloor \frac{v_s P_{i+1_\mathbf{t}}^T}{v_d} \right\rfloor \quad (3)$$

And then shifting the pose while updating the global position of the TSDF:

$$P_{i+1_\mathbf{t}}^{T'} = P_{i+1_\mathbf{t}}^T - \frac{v_d \mathbf{u}}{v_s} \quad (4)$$

$$\mathbf{g}_{i+1} = \mathbf{g}_i + \mathbf{u} \quad (5)$$

2.3.1 *Implementation*

There are two parts of volumetric fusion as described by Newcombe et al. (2011) that require indexed access to the TSDF volume; 1) Volume Integration and 2) Volume Raycasting. Referring again to Figure 2, the new surface measurements shown in blue can be integrated into the memory previously used for the old surface contained within the red region of the TSDF by ensuring all element look ups in the 3D block of GPU memory reflect the virtual voxel translation computed in Equation 5. Assuming row major memory ordering, an element in the unshifted cubic 3D voxel grid can be found at the 1D memory location a given by:

For Peer Review

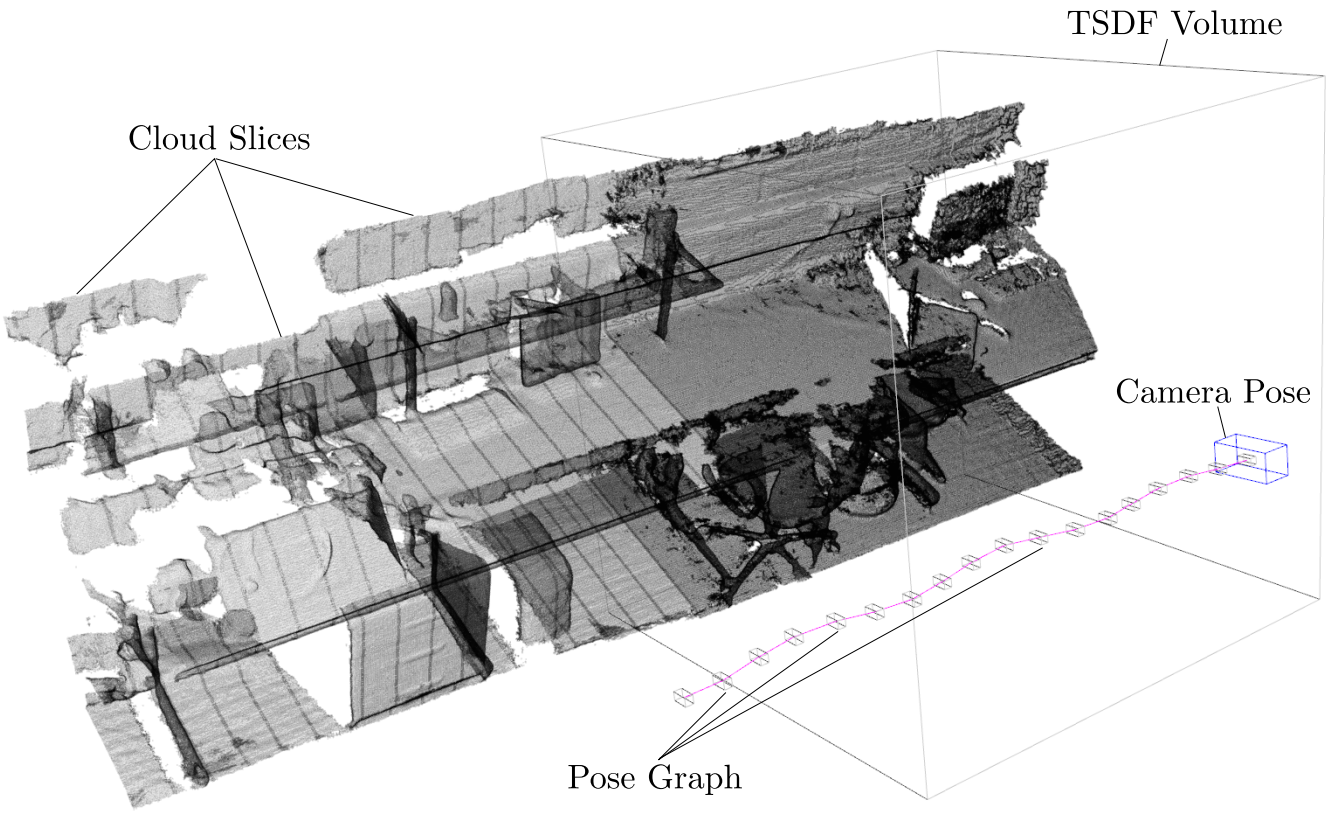


Figure 5: Visualisation of a shifted TSDF volume with extracted cloud slices and pose graph highlighted, using dynamic cube positioning discussed in Section 2.4. The pose graph is drawn in pink, while small cuboids are drawn for camera poses that have cloud slices associated with them.

$$a = (x + yv_s + zv_s^2) \tag{6}$$

The volume's translation can be reflected in how the TSDF is addressed for integration and raycasting by substituting the indices in Equation 6 with values that are offset by the current global position of the TSDF and bound within the dimensions of the voxel grid using the modulus operator:

$$x' = (x + \mathbf{g}_{ix}) \mod v_s \tag{7}$$
$$y' = (y + \mathbf{g}_{iy}) \mod v_s \tag{8}$$
$$z' = (z + \mathbf{g}_{iz}) \mod v_s \tag{9}$$
$$a = (x' + y'v_s + z'v_s^2) \tag{10}$$

2.3.2 *Surface Extraction*

In order to recover the surface from the TSDF that moves out of the region of space encompassed by the volume the **u** value computed in Equation 3 is used with $\mathbf{g}_i$ to index a three dimensional slice of the volume to extract surface points from. These points are extracted by three orthogonal raycasts aligned with the axes of the TSDF through the slice, extracting zero crossings as individual surface vertices. We filter out noisy measurements at this point by only extracting points that have a minimum voxel weight. The same 3D slice of the volume is then reset to free space to allow integration of new surface measurements. The extracted vertices are transferred to main system memory where further processing takes place. The orthogonal raycast can result in duplicate vertices if the TSDF is obliquely aligned to the surface being reconstructed. A voxel grid filter is used to remove these points by overlaying a voxel grid (with the same voxel size as the TSDF) on the extracted point cloud and returning a new point cloud with a point for each voxel that represents the centroid of all points that fell inside that voxel. Each set of vertices extracted from the TSDF in this fashion is known as a "cloud slice". From here, we rebuild the surface by incrementally triangulating successive cloud slices using an incremental mesh growing variant of the GPT algorithm to ensure surface connectivity between slices (Marton et al. (2009)). We associate with each cloud slice the pose of the camera at the time of the slice's extraction. This is visualised in Figure 4. At this point we introduce camera poses in the global coordinate frame outside of the TSDF volume. The global pose of a camera from the TSDF at time i is given as:

$$P_i = P_i^T + \frac{v_d \mathbf{g}_i}{v_s} \tag{11}$$

We construct a pose graph incrementally using each global camera pose P_i, that is, a camera pose for every frame where some poses are attached to cloud slices. The full shifting and surface extraction process is shown in Figure 5, where only the poses with associated cloud slices are drawn.

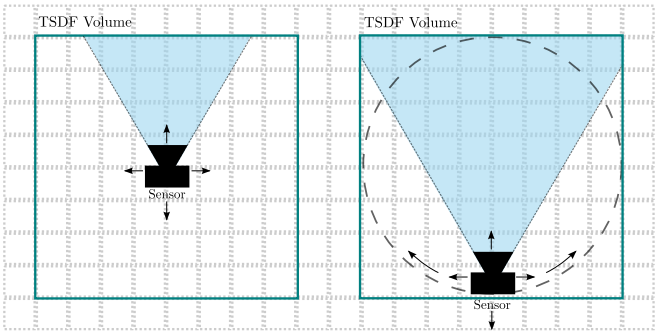


Figure 6: Visualisation of frustum-volume overlap for regular and dynamic cube positioning, from left to right; (i) By keeping the camera centered in the TSDF, there is poor overlap between the camera's field of view and the volume; (ii) By using a circular (or spherical) parameterisation of the volume's position relative to the camera, greater overlap with and usage of the TSDF can be achieved.

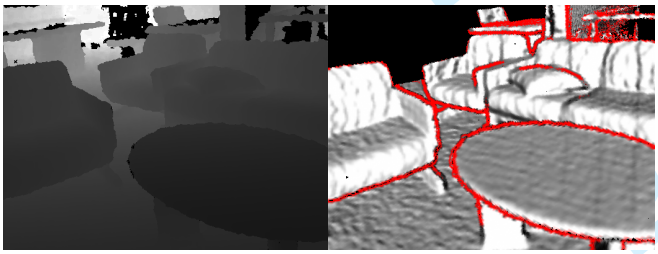


Figure 7: From left to right; (i) Input depth map registered to RGB channel; (ii) Color measurements from pixels highlighted in red are rejected for being on depth discontinuities. Lighter surfaces are weighted higher in color integration due to being well aligned with the camera sensor.

2.4 Dynamic Cube Positioning

As mentioned in Section 2.2, we position the camera in the center of the TSDF volume and roughly maintain this position inside the TSDF at all times. This parameterisation of the camera position relative to the volume is wasteful as most of the volume is unused (i.e. behind the camera) and there is little overlap between the camera frustum and the volume, shown in Figure 6. By dynamically changing the position of the volume relative to the camera depending on the camera's orientation we can achieve greater frustum-volume overlap and make better use of the entire TSDF volume. In a typical SLAM setting a circular parameterisation is sufficient.

Defining $P^T_{i_\beta}$ to be the rotation around the y-axis of the camera pose at time i, we can compute the new position of the center of the TSDF volume relative to the camera as:

$$\mathbf{r}^T = (\frac{v_d}{2} \cdot \cos(P^T_{i_\beta} + \frac{\pi}{2}), 0, \frac{v_d}{2} \cdot \sin(P^T_{i_\beta} - \frac{\pi}{2}))^\top \quad (12)$$

This dynamic parameterisation enables more intelligent use of the volume and maintains a larger active reconstruction area on front of the camera at all times.

2.5 Color Estimation

As well as estimating the surface itself in the reconstruction process, we also estimate the color of the surface. Color is integrated into the TSDF in a similar manner to depth measurements including value truncation and averaging. The only distinction is that the predicted surface color values obtained from the volume raycast are not used in camera pose estimation. The motivation for this decision is discussed into Section 3.2. Color fusion has similar advantages to depth map fusion in that sensor noise and other optical phenomena are averaged out from the final reconstruction over time.

2.5.1 *Artifact Reduction*

Algorithm 1: Color Integration

Input: $I_{i_{\mathbf{rgb}}}$ Current RGB image
$I_{i_{\mathbf{d}}}$ Current depth map
$I_{i_{\mathbf{n}}}$ Current normal map
$S(\mathbf{s})_i$ Current TSDF volume
$\mathbf{s} \in \Psi$ Current voxel
$\mathbf{p} \in \Omega$ Current pixel

do
- $c \leftarrow 0$
- **for** each $\mathbf{p}_k$ in 7×7 area around $\mathbf{p}$ **do**
  - **if** $\left|I_{i_{\mathbf{d}}}(\mathbf{p}_k) - I_{i_{\mathbf{d}}}(\mathbf{p})\right| >$ depth threshold **or** $I_{i_{\mathbf{d}}}(\mathbf{p}_k) = 0$ **then**
    - $c \leftarrow c + 1$
- **if** $c <$ count threshold **then**
  - $w_c = \min(1.0, I_{i_{\mathbf{n}}}(\mathbf{p})_z / max_weight)$
  - $S(\mathbf{s})^{R'}_i = (S(\mathbf{s})^W_{i-1} S(\mathbf{s})^R_{i-1} + w_c I_{i_{\mathbf{rgb}}}(\mathbf{p})^R)/(S(\mathbf{s})^W_{i-1} + w_c)$
  - $S(\mathbf{s})^{G'}_i = (S(\mathbf{s})^W_{i-1} S(\mathbf{s})^G_{i-1} + w_c I_{i_{\mathbf{rgb}}}(\mathbf{p})^G)/(S(\mathbf{s})^W_{i-1} + w_c)$
  - $S(\mathbf{s})^{B'}_i = (S(\mathbf{s})^W_{i-1} S(\mathbf{s})^B_{i-1} + w_c I_{i_{\mathbf{rgb}}}(\mathbf{p})^B)/(S(\mathbf{s})^W_{i-1} + w_c)$

end

The estimated surface color is sometimes inaccurate around the edges of closed objects in a scene due to poor calibration between the RGB and depth cameras or light diffraction around objects. We have observed that there typically exists stark discontinuities in the depth channel around such edges which can in turn cause the background to blend with the foreground surface or vice-versa. To address this issue we opt to reject the integration of color measurements close to or on strong boundaries in the depth image. A color measurement is deemed to be on a boundary if some of its neighbours are more than a given distance away from it in depth. We consider a pixel neighbourhood window of 7×7 pixels around each RGB value to be integrated. Figure 7 shows a source depth image and rejected measurements on the TSDF surface model. In addition to this it is ideal to weight color measurements on surfaces well aligned with the sensor higher than those at extreme angles. We weight each color measurement update by the normal angle with respect to the sensor, visualised in Figure 7.

Defining the image space domain as $\Omega \subset \mathbb{N}^2$, an RGB-D frame I_i is composed of an RGB image $I_{i_{\mathbf{rgb}}} : \Omega \rightarrow \mathbb{N}^3$, a depth image $I_{i_{\mathbf{d}}} : \Omega \rightarrow \mathbb{R}$ and a timestamp i. We also define a normal map computed for $I_{i_{\mathbf{d}}}$ as $I_{i_{\mathbf{n}}} : \Omega \rightarrow \mathbb{R}^3$. We list the

For Peer Review

(i) (ii)

Figure 8: From left to right; (i) Light diffraction behind a foreground surface has caused incorrect color integration (ii) Incorporating a discontinuity check with surface angle weighting greatly reduces the visual artifacts captured.

algorithm for color integration in Algorithm 1. An example reconstruction is shown in Figure 8 comparing surface coloring with and without the described measures.

3 Camera Pose Estimation

A number of volumetric fusion systems use only depth information for camera pose estimation (Newcombe et al. (2011), Chen et al. (2013), Bylow et al. (2013), Keller et al. (2013), Roth and Vona (2012), Zeng et al. (2012), Canelhas et al. (2013)). A reliance on geometric information alone for camera pose estimation has a number of well understood problems, such as the inability to function in corridor-like environments and other scenes with few 3D features. To avoid these problems like Henry et al. (2013) we combine dense geometric camera pose constraints with dense photometric constraints to achieve a more robust pose estimate in more challenging scenes. We base our approach on the dense photometric image warping method of Steinbruecker et al. (2011), performing dense RGB-D alignment every frame in real-time. In tune with other components of the pipeline we utilise a GPU implementation of the algorithm. Following we describe the geometric and photometric components of the camera pose estimation pipeline and our method for combining them to form a single joint pose constraint.

3.1 Geometric Camera Pose Estimation

Many of the previous works on volumetric fusion estimate the pose of the camera each frame relative to the TSDF by aligning the current depth map with the TSDF, either by raycasting the volume to retrieve a vertex and normal map of the predicted surface (as done originally by Newcombe et al. (2011)) and performing iterative closest point (ICP) or by directly minimising the distance to the surface in the TSDF (Bylow et al. (2013), Canelhas et al. (2013)). We perform the former in order to avoid expensive global memory accesses in the TSDF volume in GPU memory.

We aim to find the motion parameters ξ that minimize the cost over the point-to-plane error between vertices in the current depth frame and the predicted raycast surface:

$$E_{icp} = \sum_k \left\| \left(\mathbf{v}^k - \exp(\hat{\xi}) \mathbf{T} \mathbf{v}_n^k \right) \cdot \mathbf{n}^k \right\|^2, \tag{13}$$

where $\mathbf{v}_n^k$ is the k-th vertex in frame n, $\mathbf{v}^k, \mathbf{n}^k$ are the corresponding vertex and normal in the model, and $\mathbf{T}$ is the current estimate of the transformation from the current frame to the model frame. For simplicity of notation we omit conversions between 3-vectors (as needed for dot and cross products) and their corresponding homogeneous 4-vectors (as needed for multiplications with $\mathbf{T}$). We utilise projective data association as originally proposed by Newcombe et al. (2011) for fast point correspondence between the vertex maps by projecting the vertices from the predicted surface $\mathbf{v}$ onto the depth map vertices $\mathbf{v}_n$. Linearizing the transformation around the identity we get:

$$E_{icp} \approx \sum_k \left\| \left(\mathbf{v}^k - (\mathbf{I} + \hat{\xi}) \mathbf{T} \mathbf{v}_n^k \right) \cdot \mathbf{n}^k \right\|^2 \tag{14}$$

$$= \sum_k \left\| \left(\mathbf{v}^k - \mathbf{T} \mathbf{v}_n^k \right) \cdot \mathbf{n}^k - \hat{\xi} \mathbf{T} \mathbf{v}_n^k \cdot \mathbf{n}^k \right\|^2 \tag{15}$$

$$= \sum_k \left\| \begin{bmatrix} -\mathbf{T}\mathbf{v}_n^k \times \mathbf{n}^k \\ -\mathbf{n}^k \end{bmatrix}^\top \xi + (\mathbf{v}^k - \mathbf{v}_n^k) \cdot \mathbf{n}^k \right\|^2 \tag{16}$$

$$= \left\| \mathbf{J}_{icp} \xi + \mathbf{r}_{icp} \right\|^2 \tag{17}$$

Blocks of the measurement Jacobian and residual can be populated in tandem and solved with a highly parallel tree reduction on the GPU to produce a 6×6 system of normal equations which are then transferred to the CPU and solved with Cholesky decomposition to yield $\hat{\xi}$. As in previous work we compute the alignment iteratively with a three level coarse-to-fine depth map pyramid scheme.

3.2 Photometric Camera Pose Estimation

As mentioned previously we choose to match between consecutive RGB-D frames with the photometric component instead of matching to the predict surface reconstruction. Depending on the configuration of the TSDF there maybe be poor overlap between the camera frustum and the volume, which limits the amount of photometric information which can be used, where distant photometric features are desirable to constrain camera rotation. As well as this, the resolution of the TSDF in terms of voxels may produce a raycast image with a much lower resolution than the image produced by the RGB sensor. By default the Microsoft Kinect and Asus Xtion Pro Live, two of the most popular RGB-D sensors, have automatic exposure and white balance enabled, which can cause unusual coloring of the surface reconstruction over time, again hindering model-based photometric tracking.

For Peer Review

Given two consecutive RGB-D frames $[I_{i_{\mathbf{rgb}}}, I_{i_{\mathbf{d}}}]$ and $[I_{i+1_{\mathbf{rgb}}}, I_{i+1_{\mathbf{d}}}]$ we compute a rigid camera transformation between the two that maximises photoconsistency. Similar to the geometric pose estimation method we solve for this transformation iteratively with a three level image pyramid.

3.2.1 Preprocessing

For both pairs we perform preprocessing on the RGB image and depth map. For each depth map we convert raw sensor values to a metric depth map $M : \Omega \to \mathbb{R}$ and we compute an intensity image $I = (I^R_{\mathbf{rgb}} * 0.299 + I^G_{\mathbf{rgb}} * 0.587 + I^B_{\mathbf{rgb}} * 0.114)$ with $I : \Omega \to \mathbb{N}$. Following this a three level intensity and depth pyramid is constructed using a 5×5 Gaussian kernel for downsampling. We compute the partial derivatives $\frac{\partial I_{n+1}}{\partial x}$ and $\frac{\partial I_{n+1}}{\partial y}$ using a 3×3 Sobel operator coupled with a 3×3 Gaussian blur with $\sigma = 0.8$. Each of these steps is carried out on the GPU acting in parallel with one GPU thread per pixel.

Algorithm 2: Interest Point Accumulation

Input: $\frac{\partial I_{n+1}}{\partial x}$ and $\frac{\partial I_{n+1}}{\partial y}$ intensity image derivatives
s minimum gradient scale for pyramid level
Output: $\mathcal{L}$ list of interest points
$k_{\mathcal{L}}$ global point count
Data: α thread block x-dimension
β thread block y-dimension
γ pixels per thread
ι shared memory local list
κ shared memory local index
blockIdx CUDA block index
threadIdx CUDA thread index

```
in parallel do
    i ← β * blockIdx.y + threadIdx.y
    j ← α * γ * blockIdx.x + γ * threadIdx.x
    if threadIdx.x = 0 and threadIdx.y = 0 then
        κ ← 0
    syncthreads()
    for l ← 0 to γ do
        p ← (i, j + l)
        g^2 = ∂I_{n+1}/∂x(p)^2 + ∂I_{n+1}/∂y(p)^2
        if g^2 ≥ s then
            idx ← atomicInc(κ)
            ι_idx ← p
    syncthreads()
    b ← α * γ * threadIdx.y + γ * threadIdx.x
    for l ← 0 to γ do
        a ← b + l
        if a < κ then
            idx ← atomicInc(k_L)
            L_idx ← ι_a
end
```

3.2.2 Precomputation

As with the ICP method described in Section 3.1, we use project data association between frames to population point correspondences. For the sake of speed we only include point correspondences with a minimum gradient in the intensity image, with the motivation that other low gradient points will not have a significant effect on the final transformation. We implement this optimisation by using a list of interest points. Compiling this list of points as a parallel operation is done using a basic parallel reduction exploiting shared memory in each CUDA thread block as inspired by a similar operation by van den Braak et al. (2011). Algorithm 2 lists the operation as it would operate for each level of the pyramid.

In the computation of the Jacobian matrix the projection of each point in M_n is required. For each pyramid level the 3D projection $V_n(\mathbf{p})$ of each point $\mathbf{p}$ in the depth map is computed prior to beginning iteration with $V : \Omega \to \mathbb{R}^3$. Only projecting certain points based on a condition results in performance hindering branching and a reduction in pipelining. Empirically it was found to be faster to simply project the entire depth map rather than only project points required in correspondences. Given the intrinsic camera calibration matrix $\mathbf{K}$ of the camera we can obtain the principal points c_x and c_y and the focal lengths f_x and f_y. The 3D reconstruction of each point $\mathbf{p}$ is computed in parallel with one thread per point as $V_n(\mathbf{p}) = (\frac{(\mathbf{p}_x - c_x)M_n(\mathbf{p})}{f_x}, \frac{(\mathbf{p}_y - c_y)M_n(\mathbf{p})}{f_y}, M_n(\mathbf{p}))^\top$

3.2.3 Iterative Transformation Estimation

Our iterative estimation process takes two main steps; (i) populating a list of valid correspondences from the precomputed list of interest points and (ii) solving the linear system for an incremental transformation and concatenating these transformations. The first step involves a reduction similar to the one in Algorithm 2, but rather than reducing from a 2D array to a 1D array it reduces from a 1D array to another 1D array; a distinction which results in a notable difference in implementation. On the first iteration for frame n we set the estimated camera transformation matrix $\mathbf{T}_n$ to the identity, where

$$\mathbf{T}_n = \begin{bmatrix} & \mathbf{R}_n & & \mathbf{t}_n \\ 0 & 0 & 0 & 1 \end{bmatrix} \in \mathbb{SE}_3 \tag{18}$$

with a rotation $\mathbf{R}_n \in \mathbb{SO}_3$ and translation $\mathbf{t}_n \in \mathbb{R}^3$. Before each iteration we compute the projection of $\mathbf{T}_n$ into the image before uploading to the GPU as

$$\mathbf{R}^I_n = \mathbf{K}\mathbf{R}_n\mathbf{K}^{-1}, \qquad \mathbf{t}^I_n = \mathbf{K}\mathbf{t}_n. \tag{19}$$

Algorithm 3 lists the process of populating a list of point correspondences from the list of interest points which can then be used to construct the Jacobian. With a list of valid correspondences we need only solve a least-squares equation

$$\arg\min_{\xi} \left\| \mathbf{J}_{rgbd}\xi + \mathbf{r}_{rgbd} \right\|^2 \tag{20}$$

to compute an improved camera transformation estimate

$$\mathbf{T}'_n = \exp(\hat{\xi})\mathbf{T}_n \tag{21}$$

$$\hat{\xi} = \begin{bmatrix} & [\boldsymbol{\omega}]_\times & & \mathbf{x} \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{22}$$

with $\xi = [\boldsymbol{\omega}^\top \mathbf{x}^\top]^\top$, $\boldsymbol{\omega} \in \mathbb{R}^3$ and $\mathbf{x} \in \mathbb{R}^3$. We first normalise the intensity difference sum σ computed in Algorithm 3 to enable a weighted optimisation $\sigma' = \sqrt{\sigma / k_C}$. Computation of

For Peer Review

the σ value in parallel is in fact an optimisation exploiting the atomic arithmetic functions available in the CUDA API. From here $\mathbf{J}_{rgbd}$ and $\mathbf{r}_{rgbd}$ can be populated according to the original algorithm documented by Steinbruecker et al. (2011), including usage of σ' for weighting. Equation 20 is then solved using a tree reduction on the GPU followed by Cholesky factorisation of the linear system on the CPU.

Algorithm 3: Correspondence Accumulation

Input: $\mathcal{L}$ list of interest points
d_δ maximum change in point depth
$[I_n, M_n]$ previous intensity depth pair
$[I_{n+1}, M_{n+1}]$ current intensity depth pair
$\mathbf{R}_n^I$ camera rotation in image
$\mathbf{t}_n^I$ camera translation in image
Output: C correspondence list of the form $(\mathbf{p}, \mathbf{p}', \Delta)$
k_C global point count
σ global intensity difference sum
Data: α thread block x-dimension
γ pixels per thread
ι shared memory local list
κ shared memory local index
blockIdx CUDA block index
threadIdx CUDA thread index

```
in parallel do
    i ← α * γ * blockIdx.x + γ * threadIdx.x
    if threadIdx.x = 0 then
        κ ← 0
    syncthreads()
    for l ← 0 to γ do
        p ← L_{i+l}
        z ← M_{n+1}(p)
        if isValid(z) then
            (x', y', z')^⊤ ← z(R_n^I(p, 1)^⊤) + t_n^I
            p' ← (x'/z', y'/z')^⊤
            if isInImage(p') then
                d ← M_n(p')
                if isValid(d) and |z' − d| ≤ d_δ then
                    idx ← atomicInc(κ)
                    ι_idx ← (p, p', I_{n+1}(p) − I_n(p'))
    syncthreads()
    b ← γ * threadIdx.x
    for l ← 0 to γ do
        a ← b + l
        if a < κ then
            atomicAdd(σ, ι_a^2)
            idx ← atomicInc(k_C)
            C_idx ← ι_a
end
```

3.3 Combined Camera Pose Estimate

We combine the cost functions of both the geometric and photometric estimates in a weighted sum. The sum of the RGB-D and ICP cost is defined as

$$E = E_{icp} + w_{rgbd} E_{rgbd} \tag{23}$$

where w_{rgbd} is the weight and was set empirically to 0.1 to reflect the difference in metrics used for ICP and RGB-D costs. For each step we minimize the linear least-squares problem by solving the normal equations

$$\begin{bmatrix} \mathbf{J}_{icp} \\ v\mathbf{J}_{rgbd} \end{bmatrix}^\top \begin{bmatrix} \mathbf{J}_{icp} \\ v\mathbf{J}_{rgbd} \end{bmatrix} \xi = \begin{bmatrix} \mathbf{J}_{icp} \\ v\mathbf{J}_{rgbd} \end{bmatrix}^\top \begin{bmatrix} \mathbf{r}_{icp} \\ \mathbf{r}_{rgbd} \end{bmatrix} \tag{24}$$

$$(\mathbf{J}_{icp}^\top \mathbf{J}_{icp} + w_{rgbd} \mathbf{J}_{rgbd}^\top \mathbf{J}_{rgbd}) \xi = \mathbf{J}_{icp}^\top \mathbf{r}_{icp} + v\mathbf{J}_{rgbd}^\top \mathbf{r}_{rgbd} \tag{25}$$

where $v = \sqrt{w_{rgbd}}$. The products $\mathbf{J}^\top\mathbf{J}$ and $\mathbf{J}^\top\mathbf{r}$ are computed on the GPU using a tree reduction. The normal equations are then solved on the CPU using Cholesky factorisation. The final estimate returns a locally optimal (in the least-squares sense) camera pose which jointly minimizes the photometric error between the current RGB-D frame and the last and the geometric error between the current depth map and the TSDF surface reconstruction. This combined method provides a very accurate and stable trajectory estimate as well as surface reconstruction, which we expand upon in Section 5.

4 Loop Closure

At this point we have a method for creating large scale dense 3D mesh-based maps in real-time, however like all egomotion estimation systems drift will accumulate over space and time, warranting a need to correct the map to achieve global consistency when possible. We now frame the system as a more traditional SLAM setup with a frontend (for camera tracking and surface extraction) and a backend (for pose graph optimisation and map optimisation). A detailed system architecture diagram is shown in Figure 9.

The frontend is made up of the extended scale volumetric fusion method described in Section 2 coupled with the combined geometric and photometric camera pose estimation method described in Section 3. There is one other component of the frontend yet to be described which is a visual place recognition module that relies on the DBoW place recognition system (Galvez-Lopez and Tardos (2011)) that we describe in Section 4.2.

The backend provides a means of performing deformation-based dense map correction making use of incremental pose graph optimisation coupled with a non-rigid map optimisation. We use iSAM (Kaess et al. (2008)) to optimise the camera pose graph according to loop closure constraints provided by our place recognition module. The optimised trajectory is then used in conjunction with matched visual features to constrain a non-rigid space deformation of the map. We adapt the embedded deformation technique of Sumner et al. (2007) to apply it to large scale dense maps captured using a pose graph frontend and utilise efficient incremental methods to prepare the map for deformation.

Following we provide a detailed description of each component involved in the global consistency pipeline including pose graph representation, place recognition and loop closure, deformation graph construction and map optimisation.

For Peer Review

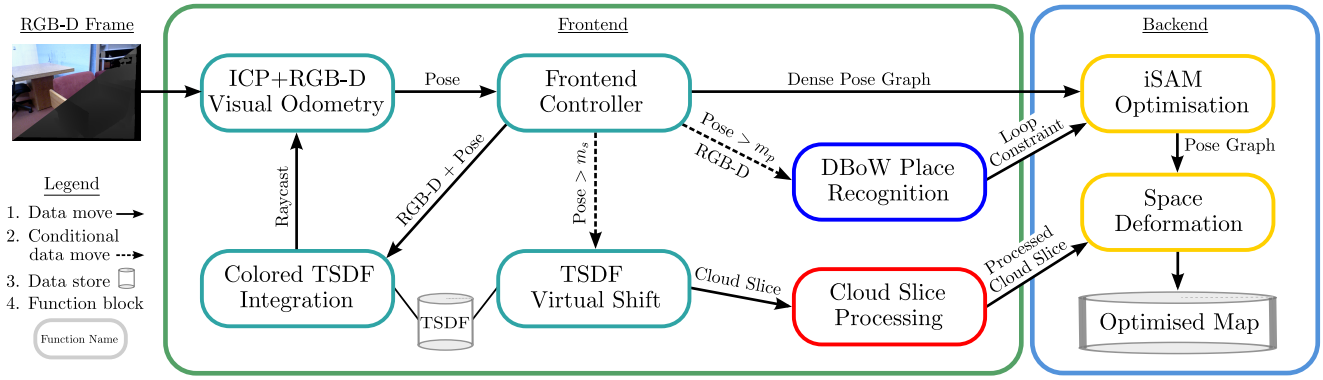


Figure 9: System architecture diagram. Differently colored function blocks are executing asynchronously in separate CPU threads. The m_s quantity denotes the volume shifting threshold and m_p denotes the place recognition movement threshold.

4.1 Pose Graph

All camera poses added to the pose graph are given in global coordinates, as described in Section 2.3.2. A camera pose P_i is estimated for every processed frame. We evaluate the trade offs of using every pose versus a subset of poses in Section 5. As discussed in Section 2.3.2 some camera poses also have an associated cloud slice as shown in Figure 10 where the relationship between pose P_γ and cloud slice C_j is shown. This provides a useful association between camera poses and the extracted surface, capturing both temporal and spatial proximity. We define C_{j_p} to be the pose associated with cloud slice C_j. In order to model the uncertainty of inter-pose constraints derived from dense visual odometry we can approximate the constraint uncertainty with the Hessian as $\Sigma = (\mathbf{J}^\top \mathbf{J})^{-1}$, where $\mathbf{J}$ is the combined measurement Jacobian computed in Equation 25.

4.2 Place Recognition

We use Speeded Up Robust Feature (SURF) descriptors with the bag-of-words-based DBoW loop detector for place recognition (Galvez-Lopez and Tardos (2011)). Adding every RGB-D frame to the place recognition system is non-optimal, therefore we utilise a movement metric sensitive to both rotation and translation which indicates when to add a new frame to the place recognition system. Defining $\mathbf{r}(\mathbf{R}) : \mathbb{SO}(3) \rightarrow \mathbb{R}^3$ to provide the rotation vector form of some rotation matrix $\mathbf{R}$, we compute a movement distance between two poses a and b that compounds both translation and rotation into a single quantity as:

$$m_{ab} = \left\| \mathbf{r}(P_{a_\mathbf{R}}^{-1} P_{b_\mathbf{R}}) \right\|_2 + \left\| P_{a_\mathbf{t}} - P_{b_\mathbf{t}} \right\|_2 \quad (26)$$

For each frame we evaluate the movement distance between the current frame pose and the pose of the last frame added to the place recognition system according to Equation 26. If this metric is above some threshold m_p, a new frame is added. Empirically we found $m_p = 0.3$ provides good performance.

Upon receiving a new RGB-D frame $[I_{i_{\mathbf{rgb}}}, I_{i_\mathbf{d}}]$ the place recognition module first computes a set of SURF keypoints and associated descriptors $U_i \in \Omega \times \mathbb{R}^{64}$ for that frame. These features are cached in memory for future queries. The depth image $I_{i_\mathbf{d}}$ is also cached, however to ensure low memory usage it is compressed on-the-fly using lossless compression (Deutsch and Gailly (1996)). Following this, the existing bag-of-words descriptor database is queried. If a match is found the SURF keypoints and descriptors U_m and depth data $I_{m_\mathbf{d}}$ (on-the-fly decompressed) for the matched image are retrieved for constraint computation. A number of validation steps are performed to minimise the chance of false positives. Overall we choose very high threshold parameters to prevent any false place recognitions in our experiments. They are as follows:

4.2.1 SURF Correspondence Threshold

Given U_i and U_m we find correspondences by a k-nearest neighbour search in the SURF descriptor space. We use the Fast Library for Approximate Nearest Neighbors (FLANN) to perform this search and populate a set of valid correspondences $G \in \Omega \times \Omega$, thresholding matches using an L_2-norm between descriptors in $\mathbb{R}^{64}$. We discard the loop closure candidate if $|G|$ is less than some threshold; a value of 35 has been found to provide adequate performance in our experiments.

4.2.2 RANSAC Transformation Estimation

Given G and $I_{m_\mathbf{d}}$, we first attempt to approximate a 6-DOF relative transformation between the camera poses of frames i and m using a RANSAC-based 3-point algorithm (Fischler and Bolles (1981)). Given a calibrated camera intrinsics matrix $\mathbf{K}$, depth image $I_{m_\mathbf{d}}$ and keypoint location $\mathbf{p} \in \Omega$, we can compute the 3D back-projection $\mathbf{p}_w = I_{m_\mathbf{d}}(\mathbf{p})\mathbf{K}^{-1}(\mathbf{p}|1)^\top$, where $\mathbf{p}_w \in \mathbb{R}^3$. Each matching keypoint in G is back-projected from image m to a 3D point, transformed according to the current RANSAC model and reprojected into the image plane of frame i (using standard perspective projection onto an image plane) where the reprojection error quantified by the L_2-norm in $\mathbb{R}^2$ is used for outlier detection. Empirically we chose a

For Peer Review

maximum reprojection error of 2.0 for inliers. If the percentage of inliers for the RANSAC estimation is below 25% the loop closure is discarded. Otherwise, we refine the estimated transformation by minimising all inlier feature reprojection errors in a Levenberg-Marquardt optimisation.

4.2.3 *Point Cloud ICP*

At this point only candidate loop closures with strong geometrically consistent visual feature correspondences remain. As a final step we perform a non-linear ICP step between $I_{i_\mathbf{d}}$ and $I_{m_\mathbf{d}}$. Firstly we back-project each point in both depth images to produce two point clouds. In order to speed up the computation, we carry out a uniform downsampling of each point cloud in $\mathbb{R}^3$ using a voxel grid filter. Finally, using the RANSAC approximate transformation estimate as an initial guess, we iteratively minimise nearest neighbour correspondence distances between the two point clouds using a Levenberg-Marquardt optimisation. We accept the final refined transformation if the mean L_2^2-norm of all correspondence errors is below a threshold. Typically we found a threshold of 0.01 to provide good results.

Once a loop closure candidate has passed all of the described tests, the relative transformation constraint between the two camera poses is added to the pose graph maintained by the iSAM module. Section 4.4 describes how this constraint is used to update the map.

4.3 Space Deformation

Our approach to non-rigid space deformation of the map is based on the embedded deformation approach of Sumner et al. (2007). Their system allows deformation of open triangular meshes and point clouds; no connectivity information is required as is the case with many deformation algorithms (Karan (2000); Jacobson and Sorkine (2011)). Exploiting this characteristic, Chen et al. (2012) applied embedded deformation to automatic skeletonised rigging and real-time animation of arbitrary objects in their KinÊtre system. Next we describe our adaptation of embedded deformation to apply to large scale dense maps with a focus on automatic incremental deformation graph construction.

4.3.1 *Deformation Graph*

Sumner et al. (2007) propose the use of a deformation graph to facilitate space deformation of a set of vertices. A deformation graph is composed of nodes and edges spread across the surface to be deformed. Each node N_l has an associated position $N_{l_\mathbf{g}} \in \mathbb{R}^3$ and set of neighbouring nodes $\mathcal{N}(N_l)$. The neighbours of each node are what make up the edges of the graph. Each node also stores an affine transformation in the form of a 3×3 matrix $N_{l_\mathbf{R}}$ and a 3×1 vector $N_{l_\mathbf{t}}$, initialised by default to the identity and $(0,0,0)^\top$ respectively. The effect of this affine transformation on any vertex which that node influences is centered at the node's position $N_{l_\mathbf{g}}$.

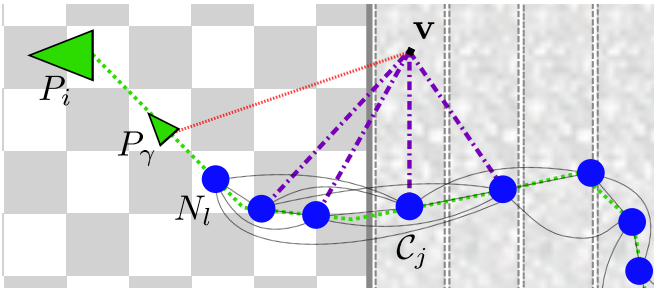


Figure 10: Two-dimensional example showing the current position of the TSDF shifting volume as a checkerboard pattern and the previously extracted cloud slices as textured columns. Also shown is the pose graph as small green points as well as a pose P_γ which caused a volume shift. The association between P_γ and the extracted cloud slice is shown with a dotted red line. A $k = 4$ connected sequential deformation graph is also shown, demonstrating the back-traversal vertex association algorithm on a random vertex $\mathbf{v}$.

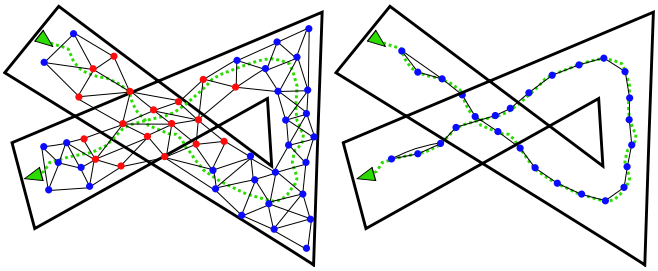


Figure 11: Two-dimensional example of deformation graph construction. On the left a spatially-constrained graph is constructed over a pre-loop closure map suffering from significant drift. The nodes highlighted in red are connected to nodes which belong in potentially completely unrelated areas of the map. On the right our incremental sampling and connectivity strategy is shown (two-nearest neighbours for simplicity) which samples and connects nodes along the pose graph, preventing unrelated areas of the map being connected by the deformation graph.

4.3.2 *Incremental Graph Construction*

The original approach to embedded deformation relies on a uniform sampling of the vertices in $\mathbb{R}^3$ to construct the deformation graph. Chen et al. (2012) substitute this with a method that uses a 5D orientation-aware sampling strategy based on the Mahalanobis distance between surface points in order to prevent links in the graph between physically unrelated areas of the model. Neither strategy is appropriate in a dense mapping context as drift in odometry estimation before loop detection may cause unrelated areas of the map to completely overlap in space. This issue also arises in determining connectivity of the graph. Applying sampling and connectivity strategies that are only spatially aware can result in links between completely unrelated areas of the map, as shown in Figure 11. The effects of applying a nearest neighbour strategy are visualised in Figure 12. For this reason we derive a sampling and connectivity strategy that exploits the camera pose graph for deformation graph construction and connection. The method is computationally efficient and incremental, enabling real-time execution. Our sampling strategy is

For Peer Review

listed in Algorithm 4.

Algorithm 4: Incremental Deformation Node Sampling

Input: P camera pose graph
i pose id of last added node
d_p pose sampling rate
Output: N set of deformation graph nodes
do
  $l \leftarrow |N|$
  if $l = 0$ **then**
    $N_{l_\mathbf{g}} \leftarrow P_{0_\mathbf{t}}$
    $l \leftarrow l + 1$
    $i \leftarrow 0$
  $P_{last} \leftarrow P_i$
  for i **to** $|P|$ **do**
    if $\left\| P_{i_\mathbf{t}} - P_{last_\mathbf{t}} \right\|_2 > d_p$ **then**
      $N_{l_\mathbf{g}} \leftarrow P_{i_\mathbf{t}}$
      $l \leftarrow l + 1$
      $P_{last} \leftarrow P_i$
end

We connect deformation graph nodes returned by our sampling strategy in a sequential manner, following the temporal order of the pose graph itself. That is to say our set of graph nodes N is ordered. We sequentially connect nodes up to a value k. We use $k = 4$ in all of our experiments. For example, a node l will be connected to nodes $(l \pm 1, l \pm 2)$. We show $k = 2$ connectivity in Figure 11. Note the connectivity of end nodes which maintains k-connectivity.

4.3.3 *Incremental Vertex Weighting*

Each vertex $\mathbf{v}$ has a set of influencing nodes in the deformation graph $\mathcal{N}(\mathbf{v})$. The deformed position of a vertex is given by Sumner et al. (2007):

$$\hat{\mathbf{v}} = \sum_{k \in \mathcal{N}(\mathbf{v})} w_k(\mathbf{v}) \left[N_{k_\mathbf{R}}(\mathbf{v} - N_{k_\mathbf{g}}) + N_{k_\mathbf{g}} + N_{k_\mathbf{t}} \right] \tag{27}$$

where $w_k(\mathbf{v})$ is defined as (all k summing to 1):

$$w_k(\mathbf{v}) = (1 - \left\| \mathbf{v} - N_{k_\mathbf{g}} \right\|_2 / d_{max})^2 \tag{28}$$

Here d_{max} is the Euclidean distance to the $k+1$-nearest node of $\mathbf{v}$. In previous work based on this technique the sets $\mathcal{N}(\mathbf{v})$ for each vertex are computed in batch using a k-nearest neighbour technique. Again, being based on spatial constraints alone this method fails in the example shown in Figure 11. To overcome this issue we derive an algorithm that assigns nearest neighbour nodes to each vertex using a greedy back-traversal of the sampled pose graph nodes.

Referring back to Figure 10 and Section 2.3.2, we recall that each pose that causes a volume shift has an associated set of vertices contained within a cloud slice. We can exploit the inverse mapping of this association to map each vertex onto a single pose in the pose graph. However, the associated pose is at least a distance of $\frac{v_d}{2}$ away from the vertex, which is not ideal for the deformation. In order to pick sampled pose graph nodes for each vertex that are spatially and temporally

Algorithm 5: Back-Traversal Vertex Association

Input: C cloud slices
N set of deformation graph nodes
b_p number of poses to traverse back
Output: $\mathcal{N}(\mathbf{v})$ for each $\mathbf{v}$
do
  foreach C_j **do**
    foreach $\mathbf{v} \in C_j$ **do**
      $l \leftarrow$ binary_search_closest(C_{j_P}, N)
      $N' \leftarrow \emptyset$
      $n \leftarrow 0$
      for $i \leftarrow 0$ **to** b_p **do**
        $N'_n \leftarrow N_l$
        $n \leftarrow n + 1$
        $l \leftarrow l - 1$
      sort_by_distance$(N', \mathbf{v})$
      $\mathcal{N}(\mathbf{v}) \leftarrow N'_{1 \rightarrow k}$
end

optimal, we use the closest sampled pose to the associated cloud slice pose as a starting point to traverse back through the sampled pose graph nodes to populate a set of candidate nodes. From these candidates the k-nearest neighbours of the vertex are chosen. We list the algorithm for this procedure in Algorithm 5 and provide a visual example in Figure 10.

The per-vertex node weights can be computed within the back-traversal algorithm, which itself can be carried out incrementally online while the frontend volume shifting component provides new cloud slices. The ability to avoid computationally expensive batch steps for deformation graph construction and per-vertex weighting by using incremental methods is the key to allowing low latency online map optimisation at any time.

4.4 Optimisation

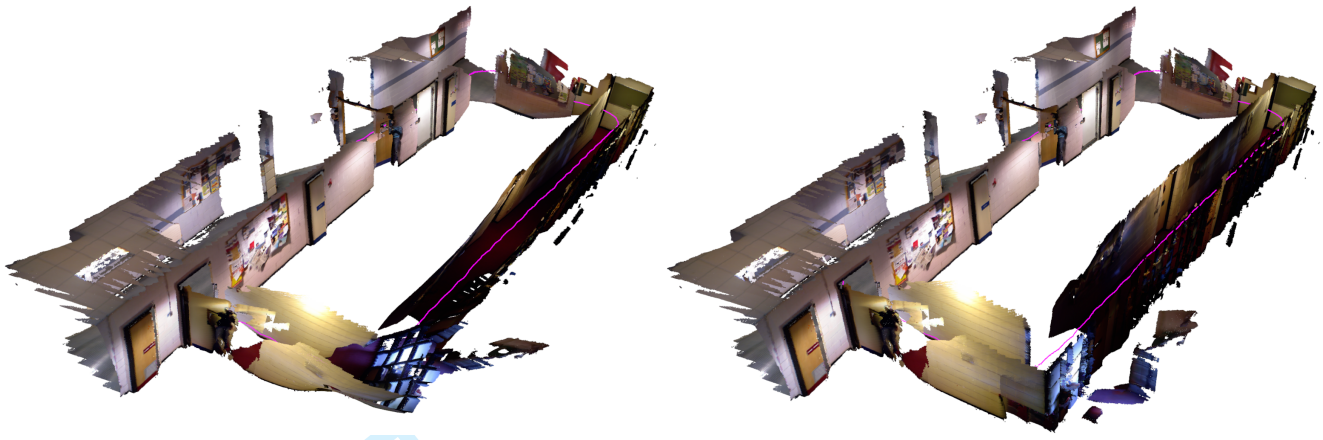
On acceptance of a loop closure constraint as described in Section 4.2 we perform two optimisation steps, firstly on the pose graph and secondly on the dense vertex map. The pose graph optimisation provides the measurement constraints for the dense map deformation optimisation in place of user specified constraints that were necessary in the original embedded deformation approach. Pose graph optimisation is carried out using the iSAM framework (Kaess et al. (2008)). We benefit from the incremental sparse linear algebra representation used internally in iSAM, such that execution time is reasonable in terms of online operation.

4.4.1 *Map Deformation*

Sumner et al. (2007) define three cost functions over the deformation graph and user constraints to optimise the set of affine transformations over all graph nodes N. The first maximises rigidity in the deformation:

$$E_{rot} = \sum_l \left\| N_{l_\mathbf{R}}^\top N_{l_\mathbf{R}} - \mathbf{I} \right\|_F^2 \tag{29}$$

For Peer Review



(i) (ii)

Figure 12: From left to right; (i) Highly distorted map produced when a naïve nearest neighbour sampling and connectivity strategy is used; (ii) Non-distorted map loop closure using our proposed sampling and connectivity strategy.

Where Equation 29 is the alternative Frobenius-norm form provided by Chen et al. (2012). The second is a regularisation term that ensures a smooth deformation across the graph:

$$E_{reg} = \sum_{l} \sum_{n \in N(N_l)} \left\| N_{l_\mathbf{R}}(N_{n_\mathbf{g}} - N_{l_\mathbf{g}}) + N_{l_\mathbf{g}} + N_{l_\mathbf{t}} - (N_{n_\mathbf{g}} + N_{n_\mathbf{t}}) \right\|_2^2 \quad (30)$$

The third is a constraint term that minimises the error on a set of user specified vertex position constraints Q, where a given constraint $Q_p \in \mathbb{R}^3$ and $\phi(\mathbf{v})$ is the result of applying Equation 27 to $\mathbf{v}$:

$$E_{con} = \sum_{p} \left\| \phi(\mathbf{v}) - Q_p \right\|_2^2 \quad (31)$$

We link the optimised pose graph to the map deformation through the E_{con} cost function. With P being the pose graph before loop constraint integration we set P' to be the optimised pose graph returned from iSAM. We then add each of the camera pose translations to the deformation cost as if they were user specified vertex constraints, redefining Equation 31 as:

$$E_{con_P} = \sum_{i} \left\| \phi(P_{i_\mathbf{t}}) - P'_{i_\mathbf{t}} \right\|_2^2 \quad (32)$$

A uniform constraint distribution across the surface obtained from this parameterisation aids in constraining both surface translation and orientation. However at some points the surface orientation may not be well constrained. In order to overcome this issue we add additional vertex constraints between the unoptimised and optimised 3D back-projections of each of the matched inlier SURF keypoints detected in Section 4.2, where P_i is the camera pose of the matched loop closure frame:

$$E_{surf} = \sum_{q} \left\| \phi((P_{i_\mathbf{R}} G_q) + P_{i_\mathbf{t}}) - ((P'_{i_\mathbf{R}} G_q) + P'_{i_\mathbf{t}}) \right\|_2^2 \quad (33)$$

The final total cost function is defined as:

$$w_{rot}E_{rot} + w_{reg}E_{reg} + w_{con_P}E_{con_P} + w_{surf}E_{surf} \quad (34)$$

With $w_{rot} = 1$, $w_{reg} = 10$, $w_{con_P} = 100$ and $w_{surf} = 100$, we minimise this cost function using the iterative Gauss-Newton algorithm choosing weighting values in line with those used in Sumner et al. (2007). As highlighted in previous work, the Jacobian matrix in this problem is sparse, enabling the use of sparse linear algebra libraries for efficient optimisation. We use the CHOLMOD library to perform sparse Cholesky factorisation and efficiently solve the system (Davis and Hager (1999)). We then apply the optimised deformation graph N to all vertices over all cloud slices C in parallel across multiple CPU threads. As discussed in Section 2.3.2 we compute an incremental mesh surface representation of the cloud slices as they are produced by the frontend. The incremental mesh can be deformed by applying the deformation graph to its vertices. In our experience an incremental mesh typically contains more minuscule holes than a batch mesh, which in path planning is functionally almost identical but less visually appealing. In all results we show the batch mesh computed over the set of optimised vertices.

5 Evaluation

We evaluate our system both quantitatively and qualitatively in terms of trajectory estimation, surface reconstruction and computational performance. We processed a combined total of over 70,000 unique RGB-D frames in our evaluation.

5.1 Trajectory Estimation

To evaluate the accuracy of our camera trajectory estimation we present results on the widely used RGB-D benchmark of Sturm et al. (2012). This benchmark provides synchronised

For Peer Review

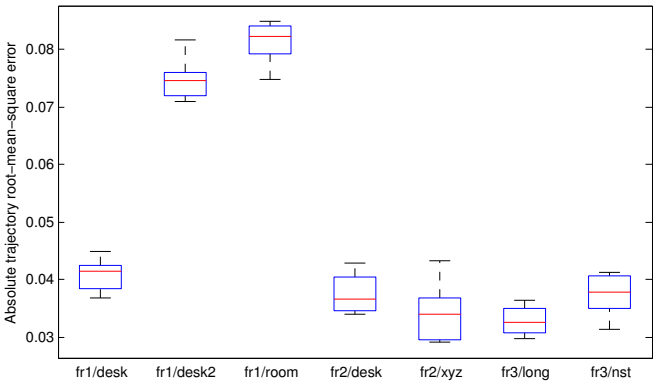


Figure 13: Boxplot of the ATE RMSE in metres per sequence evaluated. In each box the red central line is the median, the box edges the 25th and 75th percentiles and the whiskers extend to the minimum and maximum estimates. Each dataset was ran ten times to account for the randomness induced by the place recognition system in Section 4.2.

Dataset	RMSE	Median	Max	$\bar{\omega}$
fr1/desk	0.0407	0.0352	0.0905	23.33
fr1/desk2	0.0747	0.0639	0.2309	29.31
fr1/room	0.0813	0.0739	0.2511	29.88
fr2/desk	0.0376	0.0315	0.0879	6.34
fr2/xyz	0.0341	0.0234	0.0979	1.72
fr3/long	0.0329	0.0297	0.0698	10.19
fr3/nst	0.0372	0.0335	0.0735	7.43

Table 1: Statistics on ATE on evaluated datasets. Trajectory values are in metres as the mean over ten runs of each dataset. The mean angular velocity is given as $\bar{\omega}$ in degrees per second, retrieved from the dataset specifications.

ground truth poses for an RGB-D sensor moved through an environment, captured with a highly precise motion capture system. We evaluated multiple runs over seven datasets with quantitative results shown in Table 1 and a boxplot shown in Figure 13. We use the absolute trajectory (ATE) root-mean-square error metric (RMSE) to evaluate our system, which measures the root-mean-square of the Euclidean distances between all estimated camera poses and the ground truth poses associated by timestamp (Sturm et al. (2012)).

Consistent performance is achieved on all sequences evaluated, with a notably higher error on the fr1/desk2 and fr1/room datasets. This can be explained by the high average angular velocity on these sequences which causes motion blur, increases the effect of rolling shutter and violates the assumption of projective data association. Provided there is a low standard deviation in frame rate and good overlap between successive frames a strong trajectory estimate is achievable. Figure 14 shows two dimensional plots of the differences between the estimated trajectories and the ground truth trajectories. In all real world datasets evaluated in this paper the auto exposure and auto white balance features of the RGB-D camera were enabled.

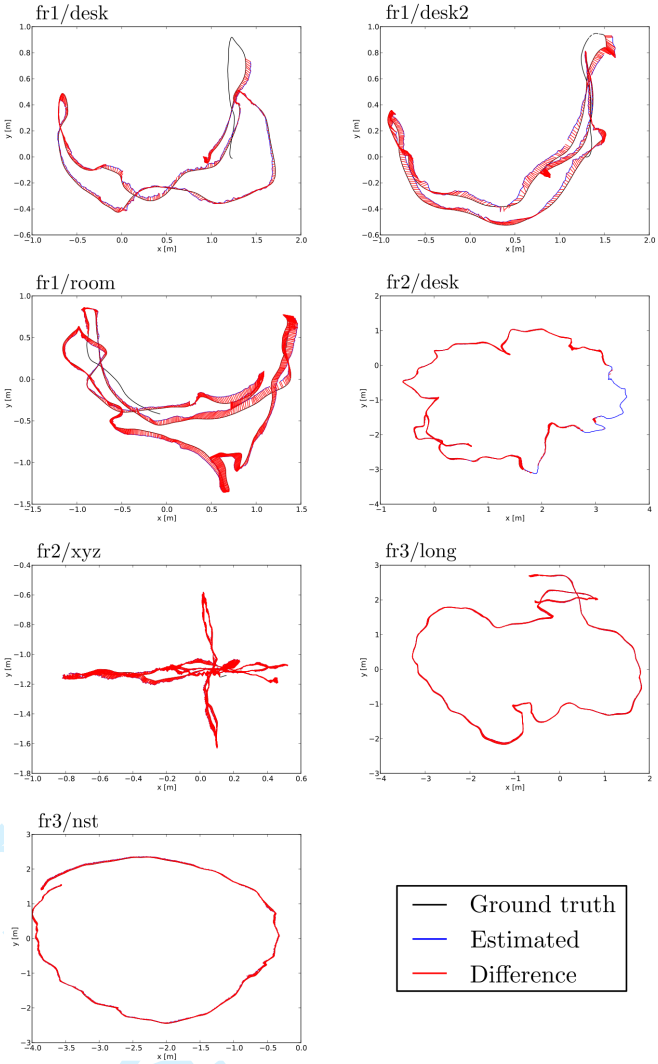


Figure 14: Two dimensional plot of estimated trajectories versus ground truth trajectories on evaluated sequences.

5.2 Surface Reconstruction

We present a number of quantitative and qualitative results on evaluating the surface reconstructions produced by our system. In our experience a high score on a camera trajectory benchmark does not imply a high quality surface reconstruction due to the frame-to-model tracking component of the system. In previous work we found that although other methods for camera pose estimation may score better on benchmarks, the resulting reconstructions are not as accurate if frame-to-model tracking is not being utilised (Whelan et al. (2013a)). We evaluate six different datasets captured in a handheld fashion across a wide range of environments, demonstrating the viability of our system for use over large scale trajectories both indoors and outdoors (within sensing limitations) and across multiple floors.

For Peer Review

5.2.1 Comparison to 2-pass Optimisation

In order to evaluate the accuracy of the deformation process we compare the resulting maps produced when a 2-pass approach is taken versus a single pass approach with a deformation for map correction. The 2-pass approach involves the following steps;

1. Build a pose graph with a camera pose for every frame.
2. Detect visual loop closures using the method described in Section 4.2.
3. At the end of the dataset, optimise the camera pose graph taking loop closure constraints into account.
4. Rerun the dataset using the optimised pose graph in place of the visual odometry frontend.

From here we can compare the two maps to determine a measure of similarity. This presents an interesting question as although the pose graphs for both the 2-pass and deformation-based maps are identical, the maps themselves may differ slightly due to the fact the 2-pass approach gives up frame-to-model registration on the second pass where the frustum-volume intersection may also slightly change. This means there will not be any reliable 1-to-1 point correspondences between the maps. For this reason we measure the map similarity by the residual error of a round of dense ICP between the maps. Given that both maps lie in the global coordinate frame we can iteratively minimise nearest neighbour point-wise correspondences between the two maps using standard point-to-plane ICP. This allows us to account for a small rigid transformation error between the two maps. We measure the remaining root-mean-square residual error between point correspondences as the residual similarity error between the two maps. Table 2 lists statistics on the six evaluated datasets including the 2-pass residual registration error as well as the same error computed on maps deformed with a subsampled pose graph, which we discuss in Section 5.3. It is clear that the deformation approach brings the map into strong alignment with the 2-pass output, with only a few millimetres in difference. This can be seen in Figure 15. Extension 1 shows the map correcting deformation occurring for the Indoors and Two floors datasets, as well as flythroughs of the final meshes.

5.2.2 Surface Ground Truth

We also evaluate the surface reconstruction quality quantitatively using synthetic data produced in an identical manner to the datasets created by Handa et al. (2012). Each dataset contains 30Hz RGB-D frames from a camera placed in a synthetic office environment. The camera trajectories were generated from real world data which was previously ran through our visual odometry frontend. Given that the datasets were produced using a procedural raytracing process (using POVRay), there is no actual surface to compare against. However, each RGB-D frame does have ground truth depth information which we compare against. For each frame in a dataset

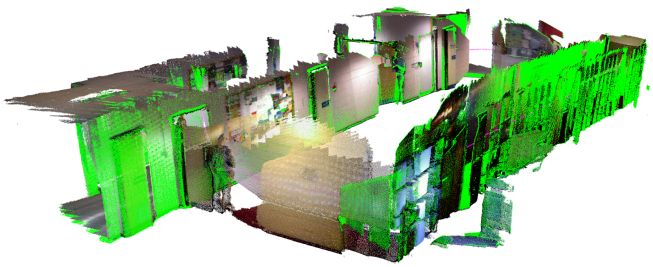


Figure 15: Deformed map (shown in color) aligned with the 2-pass produced map (shown in green). The two maps are barely distinguishable, evident by how interwoven the vertices are.

Figure 16: Mesh reconstruction of the first synthetic dataset. Note that the rough triangulation of parts of the chairs is due to a poor viewing angle throughout the sequence.

we compute a histogram of the per depth pixel L_1-norm error between the ground truth depth map and the predicted surface depth map raycast from the TSDF, normalising by the number of valid pixels before aligning all histograms into a two dimensional area plot. We evaluated two synthetic datasets of the same scene with different camera motions. The temporal error histograms are shown in Figure 17 while frames from each dataset are shown in Figure 18. Overall the synthetic surfaces are reconstructed very well, however occasional raycasting artifacts (particularly around the edges of objects and on nearby surfaces) can hinder the reconstruction quality score, as in the first dataset. Observing the final reconstruction in Figure 16 it is clear that the slight dip in accuracy did not effect the reconstruction quality by any significant amount. Typically around 95% of the estimated depth of the surface is within 5mm of ground truth. In future work we aim to explore testing against synthetic surfaces themselves rather than depth maps alone, avoiding the need to rely on a robust raycaster for meaningful evaluation.

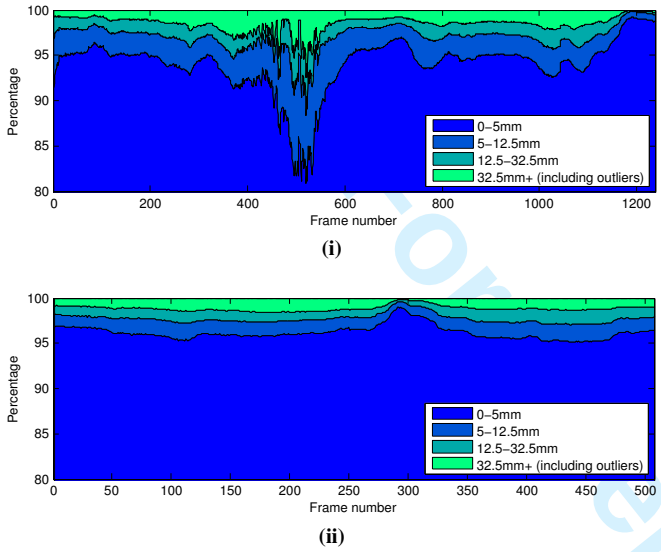
5.3 Computational Performance

We evaluate the computational performance of both the frontend and backend of the system. The evaluation platform was a standard desktop PC running Ubuntu 12.04 with an Intel Core i7-3960X CPU at 3.30GHz, 16GB of RAM and an

For Peer Review

Dataset	Length(m)	v_d(m)	d_p(m)	Vertices	Volume(m^3)	2-pass(mm)	2-pass fast(mm)	Figure
Coffee	30.18	4.5	0.4	909,422	7,993	1.2	1.8	19
Indoors	49.57	7	0.4	1,603,116	21,918	2.7	4.9	20
Garden	71.49	6	0.8	2,418,331	28,340	2.1	2.5	21
Outdoors	152.05	6	0.8	2,961,966	34,711	2.3	8.8	22
Two floors	173.88	6	0.8	4,016,273	47,066	2.1	8.0	23
In/outdoors	317.95	6	0.8	5,985,669	70,145	2.8	7.5	24

Table 2: Statistics on six handheld datasets captured over a wide variety of environments.



(i)

(ii)

Figure 17: Temporal histograms of predicted depth versus ground truth depth on synthetic datasets. A frame from the dip in accuracy around the center of the first dataset is shown in Figure 18 (i) while a frame from the peak in accuracy in the center of the second dataset is shown in Figure 18 (ii).

m_s	Avg	Min	Max	StdDev
1	34.15	25.93	41.58	3.30
2	32.21	25.63	39.29	3.14
4	31.08	25.38	39.02	2.77
8	30.57	25.42	37.44	2.48
16	**29.94**	**24.97**	**37.25**	**2.26**
32	30.26	25.33	40.30	2.39
64	30.49	25.06	43.95	2.73

Table 3: Computational performance of the volumetric fusion thread on the fr1/desk dataset. The shifting threshold m_s is given in voxels while the frame processing timings are given in milliseconds. Highlighted is the optimal choice based on execution time.

nVidia GeForce 680GTX GPU with 2GB of memory.

5.3.1 *Frontend Performance*

To evaluate the performance of the frontend (including volume integration, camera pose estimation, volume raycasting and volume shifting, essentially all teal colored function blocks in Figure 9) we provide frame processing timing results on the fr1/desk sequence comparing different choices of the m_s parameter discussed in Section 2.3. This parameter effects the frequency and size of each volume shift, which in turn effects frontend performance. Results are shown in Table 3. A shifting threshold of 16 voxels was found to be optimal, providing the best computational performance with an average frame rate comfortably below the frame rate of the sensor (30Hz) and with minimal spikes in execution time.

5.3.2 *Backend Performance*

We quantify the computational performance of the backend in the context of an online real-time SLAM system by measuring the latency of the system. That is, how long is takes for 1) a loop closure to be recognised when one is encountered and 2) map correction to be completed. Table 4 shows execution time and latency statistics on our test platform. We also experimented with subsampling the pose graph used in the iSAM-based pose graph optimisation by the same sampling metric used in Algorithm 4. This effects the number of poses used in the final pose graph optimisation and the number of points available to constrain the map deformation in Equation 32. Our results (shown in Table 5) show that using a subsampled pose graph (akin to using only keyframes) instead of an every frame pose graph reduces execution time (and therefore latency) by almost an order of magnitude, while only mildly effecting map quality (quantified as "2-pass fast" in Table 2). As expected the appearance-based frontend scales very well over hundreds of metres while the backend is capable of correcting millions of vertices for global consistency in only 1-3 seconds. Extension 2 shows the entirety of the In/outdoors dataset running in real-time including the two online loop closures. Note that in this video the vertex count is higher due to the weight-based filtering mentioned in Section 2.3.2 being disabled, resulting in more extracted vertices from the TSDF slices.

6 Conclusion

In this paper we have presented a real-time dense SLAM system which makes use of a dense every-frame volumetric fusion frontend for camera pose estimation and surface reconstruction in combination with a non-rigid map deformation backend to correct the mapped dense surface upon loop closure. We have provided an extensive evaluation, both quantitatively and qualitatively on common benchmarks and our own datasets demonstrating the system's ability to produce

Quantities	Datasets						
	Coffee	Indoors	Garden	Outdoors	Two floors	In/outdoors(1)	In/outdoors(2)
DBoW images	280	301	658	1171	1584	1662	2706
Poses	1544	2993	8634	5240	12952	17306	25586
Nodes	58	55	72	178	191	211	364
Vertices	932,056	1,352,919	2,256,475	2,805,083	3,896,281	3,560,994	5,867,125
Process	Timings (ms)						
Frontend	465	602	622	587	657	521	543
iSAM	257	510	1412	1299	3326	4386	6545
Deformation	266	390	1112	928	2197	3040	4473
Total latency	988	1502	3146	2814	6180	7947	11561

Table 4: Computational performance statistics on six datasets using an every frame pose graph. Quantities shown are at the moment of loop closure. The In/outdoors dataset contains two looping points which are both listed.

Quantities	Datasets						
	Coffee	Indoors	Garden	Outdoors	Two floors	In/outdoors(1)	In/outdoors(2)
DBoW images	277	305	672	1173	1593	1713	2782
Poses	283	307	674	1186	1594	1716	2783
Nodes	52	49	68	167	181	196	339
Vertices	943,721	1,371,560	2,246,028	2,841,135	3,904,113	3,569,842	5,850,152
Process	Timings (ms)						
Frontend	488	589	651	597	467	540	793
iSAM	46	67	110	288	378	271	1140
Deformation	110	105	170	377	381	148	842
Total latency	644	761	931	1262	1226	959	2775

Table 5: Computational performance statistics on six datasets using a subsampled pose graph. Quantities shown are at the moment of loop closure. The In/outdoors dataset contains two looping points which are both listed.

large scale dense globally consistent maps in real-time.

6.1 Failure Cases & Future Work

Our current implementation does not support the reintegration of areas of the map which are revisited into the volumetric fusion frontend. This results in aliasing in areas that receive multiple passes. However representing the surface as a set of cloud slices maintains spatiotemporal information about the map which can be used for change detection, scene differencing or even the merging of cloud slices from multiple passes. Additionally, reliance on projective data association for camera pose estimation limits the kinds of motion which our visual odometry frontend can handle. However this restriction works in our favour as with increased camera motion comes increased motion blur and rolling shutter effects, which are computationally expensive to compensate for in real-time to preserve reconstruction quality. Our future work includes overcoming these issues as well as looking at improved methods for estimating camera pose uncertainty and scalability over hundreds of metres.

7 Acknowledgements

Research presented in this paper was funded by a Strategic Research Cluster grant (07/SRC/I1168) by Science Foundation Ireland under the Irish National Development Plan, the Embark Initiative of the Irish Research Council and by ONR grants N00014-10-1-0936, N00014-11-1-0688, N00014-12-1-0093, and N00014-12-10020. The authors would like to thank Guillaume Gales, Richard H. Middleton and Ross Finman for their input and discussion and also Ankur Handa for his synthetic surface ground truth datasets.

References

Audras, C., Comport, A. I., Meilland, M., and Rives, P. (2011). Real-time dense RGB-D localisation and mapping. In *Australian Conf. on Robotics and Automation*, Monash University, Australia.

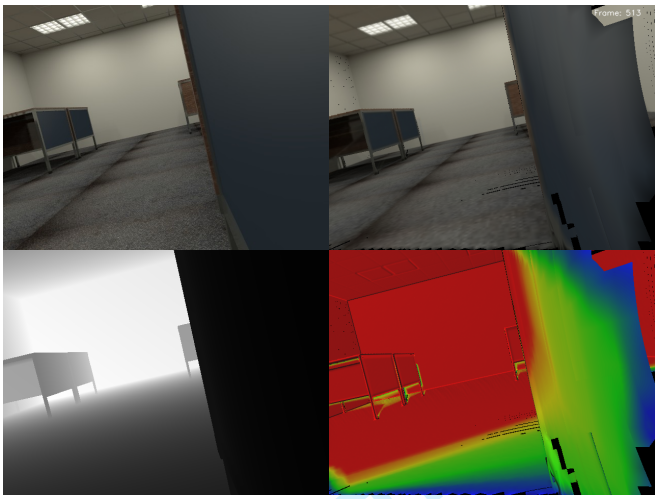
Bylow, E., Sturm, J., Kerl, C., Kahl, F., and Cremers, D. (2013). Real-time camera tracking and 3d reconstruction using signed distance functions. In *Robotics: Science and Systems Conference (RSS)*.

Canelhas, D. R., Stoyanov, T., and Lilienthal, A. J. (2013). SDF Tracker: A parallel algorithm for on-line pose estimation and scene reconstruction from depth images. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems, IROS*, Tokyo, Japan. (to appear).

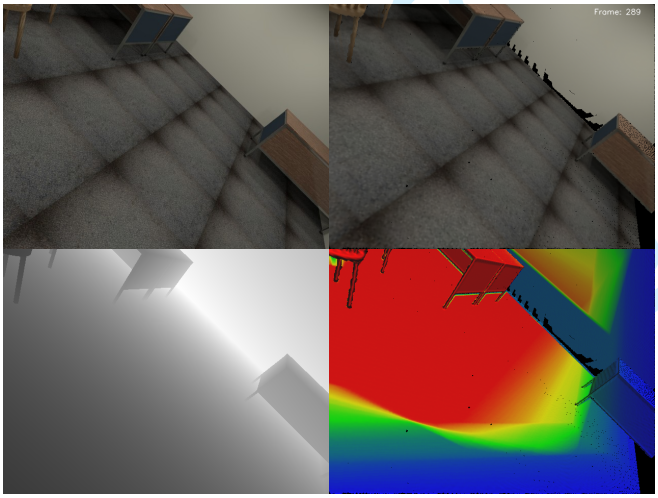
Chen, J., Bautembach, D., and Izadi, S. (2013). Scalable real-time volumetric surface reconstruction. In *SIGGRAPH 2013*, Anaheim, CA, USA. ACM.

Chen, J., Izadi, S., and Fitzgibbon, A. (2012). KinÊtre: animating the world with the human body. In *Proceedings of the 25th annual ACM symposium on User interface software and technology*, UIST '12, pages 435–444, New York, NY, USA. ACM.

For Peer Review



(i)



(ii)

Figure 18: One frame from each surface ground truth evaluation dataset. Each shows in clockwise order the ground truth RGB, predicted RGB, ground truth depth map and predicted surface phong shaded colored by voxel weight. From top to bottom; (i) Here raycasting artifacts are visible in the predicted surface in the bottom right causing a high error in the evaluation; (ii) Overall the surface is being well estimated and there are no raycasting artifacts.

Davis, T. A. and Hager, W. W. (1999). Modifying a sparse Cholesky factorization. *SIAM J. Matrix Anal. Appl.*, 20(3):606–627.

Deutsch, P. and Gailly, J.-L. (1996). Zlib compressed data format specification version 3.3.

Endres, F., Hess, J., Engelhard, N., Sturm, J., Cremers, D., and Burgard, W. (2012). An evaluation of the RGB-D SLAM system. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, St. Paul, MA, USA.

Fischler, M. A. and Bolles, R. C. (1981). Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395.

Galvez-Lopez, D. and Tardos, J. D. (2011). Real-time loop detection with bags of binary words. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ Int. Conf. on*, pages 51 –58.

Handa, A., Newcombe, R., Angeli, A., and Davison, A. (2012). Real-time camera tracking: When is high frame-rate best? In *ECCV 2012*, volume 7578 of *Lecture Notes in Computer Science*, pages 222–235.

Henry, P., Fox, D., Bhowmik, A., and Mongia, R. (2013). Patch Volumes: Segmentation-based Consistent Mapping with RGB-D Cameras. In *Third Joint 3DIM/3DPVT Conference (3DV)*.

Henry, P., Krainin, M., Herbst, E., Ren, X., and Fox, D. (2012). RGB-D mapping: Using Kinect-style depth cameras for dense 3D modeling of indoor environments. *The Int. Journal of Robotics Research*.

Hornung, A., Wurm, K. M., Bennewitz, M., Stachniss, C., and Burgard, W. (2013). OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*. Software available at `http://octomap.github.com`.

Hu, G., Huang, S., Zhao, L., Alempijevic, A., and Dissanayake, G. (2012). A robust RGB-D SLAM algorithm. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 1714 –1719.

Huang, A. S., Bachrach, A., Henry, P., Krainin, M., Maturana, D., Fox, D., and Roy, N. (2011). Visual odometry and mapping for autonomous flight using an RGB-D camera. In *Int. Symposium on Robotics Research (ISRR)*, Flagstaff, Arizona, USA.

Jacobson, A. and Sorkine, O. (2011). Stretchable and twistable bones for skeletal shape deformation. *ACM Transactions on Graphics (proceedings of ACM SIGGRAPH ASIA)*, 30(6):165:1–165:8.

Kaess, M., Ranganathan, A., and Dellaert, F. (2008). iSAM: Incremental smoothing and mapping. *IEEE Transactions on Robotics (TRO)*, 24(6):1365–1378.

Karan, K. S. (2000). Skinning characters using surface-oriented free-form deformations. In *In Graphics Interface 2000*, pages 35–42.

Karpathy, A., Miller, S., and Fei-Fei, L. (2013). Object discovery in 3d scenes via shape analysis. In *International Conference on Robotics and Automation (ICRA)*.

Keller, M., Lefloch, D., Lambers, M., Izadi, S., Weyrich, T., and Kolb, A. (2013). Real-time 3d reconstruction in dynamic scenes using point-based fusion. In *Third Joint 3DIM/3DPVT Conference (3DV)*.

Kerl, C., Sturm, J., and Cremers, D. (2013). Robust odometry estimation for RGB-D cameras. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*.

Lee, D., Kim, H., and Myung, H. (2012). GPU-based real-time RGB-D 3D SLAM. In *Ubiquitous Robots and Ambient Intelligence (URAI), 2012 9th International Conference on*, pages 46 –48.

Marton, Z. C., Rusu, R. B., and Beetz, M. (2009). On fast surface reconstruction methods for large and noisy datasets. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, Kobe, Japan.

Newcombe, R. A., Izadi, S., Hilliges, O., Molyneaux, D., Kim, D., Davison, A. J., Kohli, P., Shotton, J., Hodges, S., and Fitzgibbon, A. (2011). KinectFusion: Real-time Dense Surface Mapping and Tracking. In *Proc. of the 2011 10th IEEE Int. Symposium on Mixed and Augmented Reality*, ISMAR '11, pages 127–136,

For Peer Review

Washington, DC, USA.

Pirker, K., Rüther, M., Schweighofer, G., and Bischof, H. (2011). GPSlam: Marrying sparse geometric and dense probabilistic visual mapping. In *Proc. of the British Machine Vision Conf.*, pages 115.1–115.12.

Roth, H. and Vona, M. (2012). Moving volume KinectFusion. In *British Machine Vision Conf. (BMVC)*, Surrey, UK.

Salas-Moreno, R. F., Newcombe, R. A., Strasdat, H., Kelly, P. H. J., and Davison, A. J. (2013). SLAM++: Simultaneous localisation and mapping at the level of objects. In *Proc. Computer Vision and Pattern Recognition (CVPR)*.

Steinbruecker, F., Sturm, J., and Cremers, D. (2011). Real-Time Visual Odometry from Dense RGB-D Images. In *Workshop on Live Dense Reconstruction with Moving Cameras at the Int. Conf. on Computer Vision (ICCV)*.

Stückler, J. and Behnke, S. (2013). Multi-resolution surfel maps for efficient dense 3d modeling and tracking. In *Journal of Visual Communication and Image Representation*.

Sturm, J., Engelhard, N., Endres, F., Burgard, W., and Cremers, D. (2012). A benchmark for the evaluation of RGB-D SLAM systems. In *Proc. of the Int. Conf. on Intelligent Robot Systems (IROS)*.

Sumner, R. W., Schmid, J., and Pauly, M. (2007). Embedded deformation for shape manipulation. In *ACM SIGGRAPH 2007 papers*, SIGGRAPH '07, New York, NY, USA. ACM.

van den Braak, G.-J., Nugteren, C., Mesman, B., and Corporaal, H. (2011). Fast hough transform on GPUs: Exploration of algorithm trade-offs. In *Advances Concepts for Intelligent Vision Systems*, volume 6915 of *Lecture Notes in Computer Science*, pages 611–622.

Wagner, R., Frese, U., and Bäuml, B. (2013). 3D Modeling, Distance and Gradient Computation for Motion Planning: A Direct GPGPU Approach. In *International Conference on Robotics and Automation (ICRA)*.

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

Whelan, T., Kaess, M., Fallon, M., Johannsson, H., Leonard, J., and McDonald, J. (2012). Kintinuous: Spatially extended KinectFusion. In *RSS Workshop on RGB-D: Advanced Reasoning with Depth Cameras*, Sydney, Australia.

Whelan, T., Kaess, M., Leonard, J., and McDonald, J. (2013b). Deformation-based loop closure for large scale dense RGB-D SLAM. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems, IROS*, Tokyo, Japan. (to appear).

Zeng, M., Zhao, F., Zheng, J., and Liu, X. (2012). A Memory-Efficient KinectFusion Using Octree. In *Computational Visual Media*, volume 7633 of *Lecture Notes in Computer Science*, pages 234–241. Springer.

A Index to Multimedia Extensions

The multimedia extensions to this article are at: http://www.ijrr.org.

Extension	Type	Description
1	Video	Indoors and Two floors dataset deformation visualisations.
2	Video	In/outdoors dataset full real-time reconstruction.

For Peer Review

Figure 19: Dataset of a small coffee room. Inset shows everyday objects such as bins and fridges are captured in high detail and how the deformation approach works well in smaller environments.

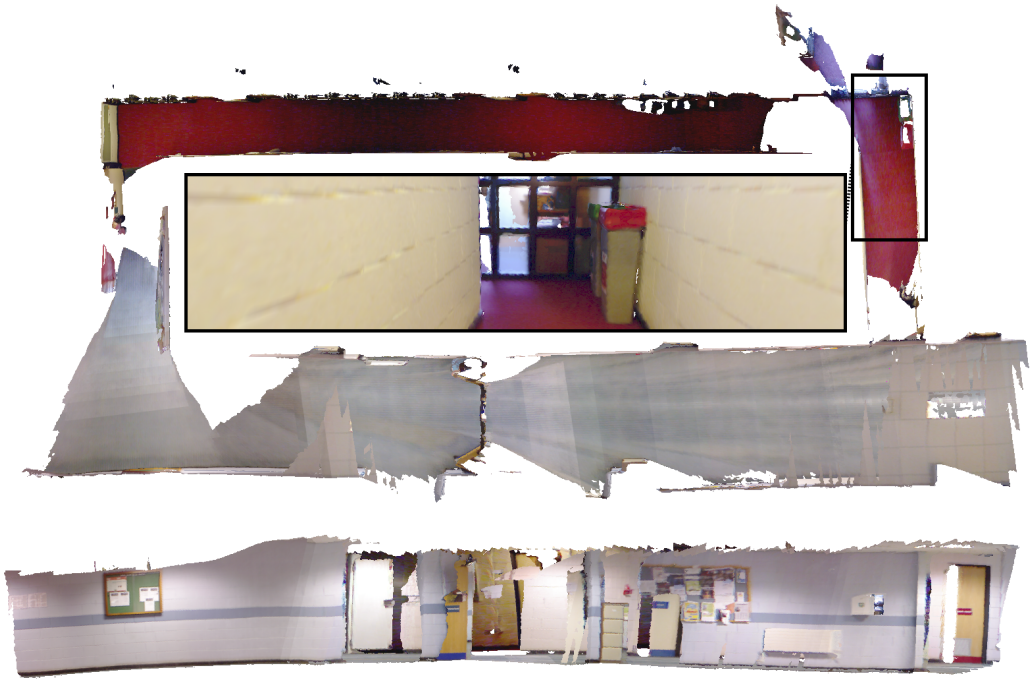


Figure 20: Corridor loop closure dataset. The inset shows map consistency at the point of loop closure. Extension 1 shows the actual map correcting deformation occurring.

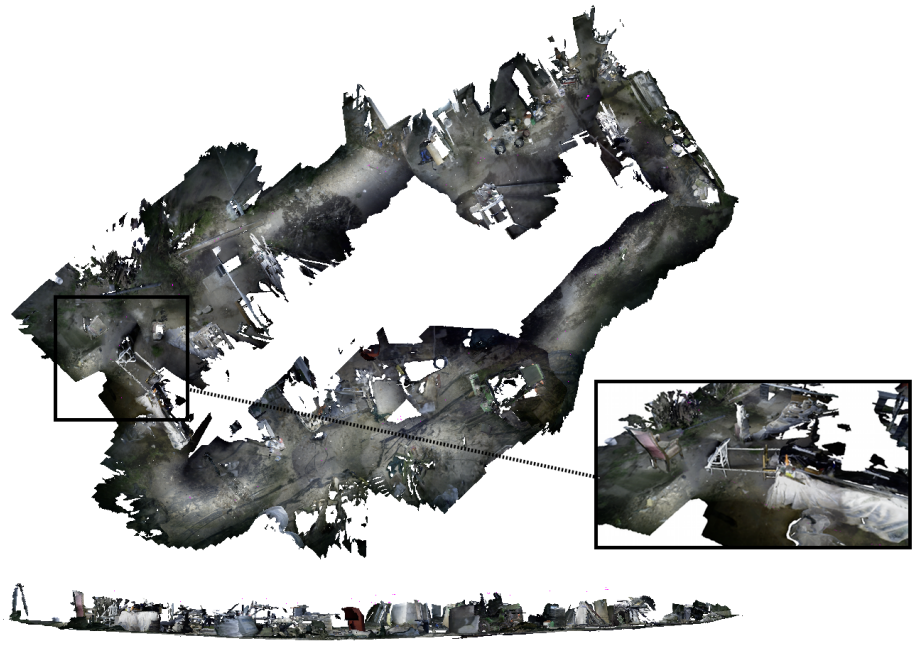


Figure 21: Large cluttered outdoor dataset. Inset shows chairs and metal bars are reconstructed well.

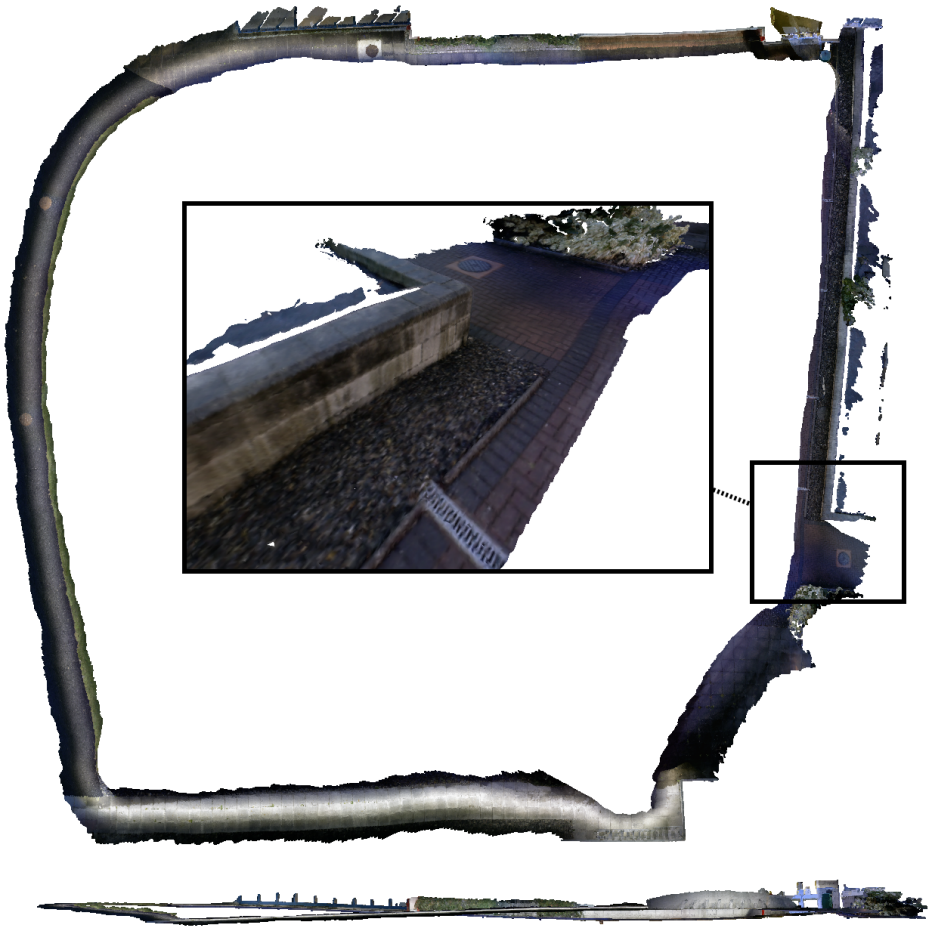


Figure 22: Large outdoor dataset. Inset shows brickwork is clearly visible.

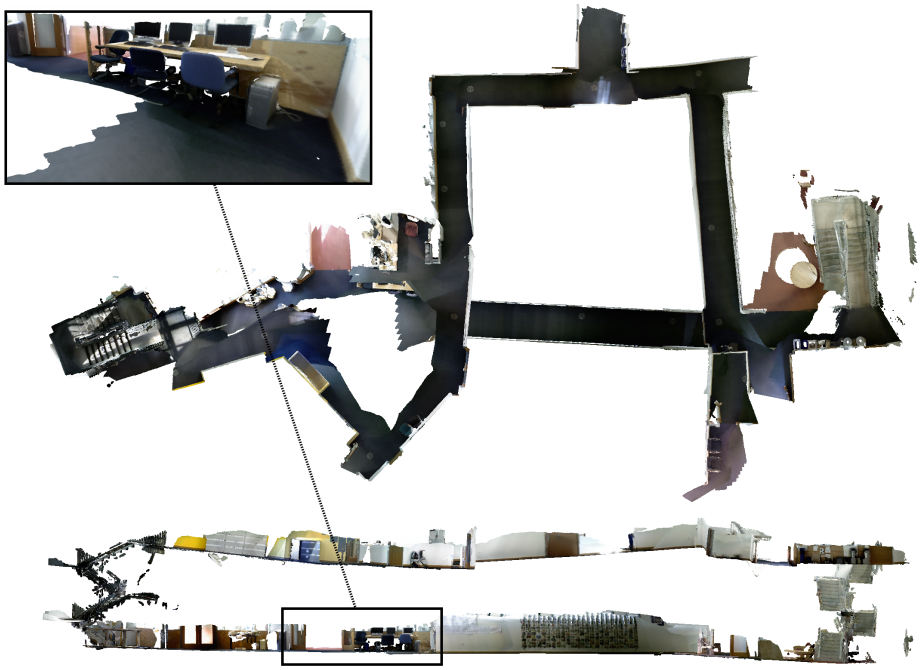


Figure 23: Dataset composed of two floors. Inset shows everyday objects such as chairs and computers are captured in high detail. Extension 1 shows the actual map correcting deformation occurring.

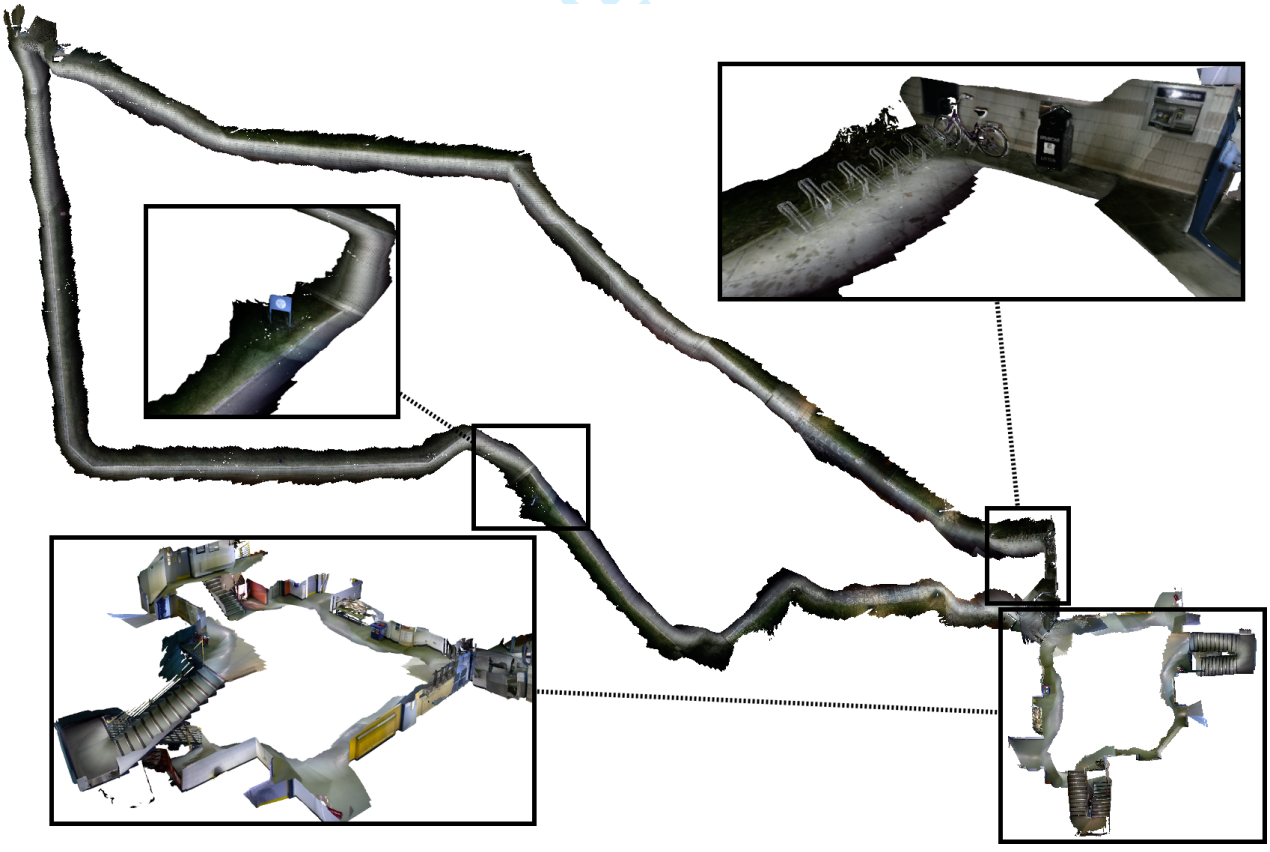


Figure 24: Large indoor and outdoor dataset made up of over five million vertices. Insets show the high fidelity of small scale features in the map. Extension 2 shows this entire dataset running from start to finish in real-time, including online loop closure.