Graph SLAM with Signed Distance Function Maps on a Humanoid Robot

René Wagner Udo Frese Berthold Bäuml

Abstract— **For such common tasks as motion planning or object recognition robots need to perceive their environment and create a dense 3D map of it. A recent breakthrough in this area was the KinectFusion algorithm [16], which relies on step by step matching a depth image to the map via ICP to recover the sensor pose and updating the map based on that pose. In so far it ignores techniques developed in the graph-SLAM area such as fusion with odometry, modeling of uncertainty and distributing an observed inconsistency over the map.**

This paper presents a method to integrate a dense geometric truncated signed distance function (TSDF) representation as KinectFusion uses with a sparse parametric representation as common in graph SLAM. The key idea is to have local TSDF sub-maps attached to reference nodes in the SLAM graph and derive graph-SLAM links via ICP by matching a map to a depth image. By moving these reference nodes according to the graph-SLAM estimate, the overall map can be deformed without touching individual sub-maps so that re-building of sub-maps is only needed in case of significant deformation within a sub-map. Also, further information can be added to the graph as common in graph SLAM. Examples are odometry or the fact that the ground is roughly but not exactly planar. Additionally, the paper proposes a modification of the KinectFusion algorithm to improve handling of long range data by taking the range dependent uncertainty into account.

I. Introduction

This work is part of continued effort [21, 22] to equip DLR's humanoid robot Agile Justin [1] with environment modeling capabilities that generate accurate surface models and free-space information (Fig. 1) for motion planning and object recognition. We have previously used a custom re-implementation [21] of the popular KinectFusion algorithm [16] for this purpose. KinectFusion relies on a variant of the well-known iterative closest points (ICP) algorithm [2] to track the sensor pose by minimizing the point-to-plane residuals of the current depth image vs. the one expected according to the current map and the sensor pose estimate. KinectFusion's map is a so called truncated signed distance function (TSDF), an implicit surface defined by a regular 3D voxel grid which, within a truncation region around surfaces, stores the signed distance to the nearest surface. This leads to very smooth, sub-voxel precise surface models well-suited to motion planning [21] and object recognition.

Unfortunately, ICP fails whenever the environment geometry does not constrain the sensor pose in all six degrees of freedom. Additionally minimizing the reprojection residuals in RGB images [23] helps alleviate this but does not fully solve the problem as it relies on texture often not available in indoor environments (e.g. blank walls in long corridors). Taking advantage of the fact that our Kinect is not a free-floating sensor but attached to the robot body we have previously [22] used forward kinematics plus odometry to get the sensor pose. This is very accurate when the robot is stationary and only certain joints are active. However, elasticities in some joints are problematic and odometry errors accumulate.

R. Wagner and B. Bäuml are with DLR Institute of Robotics and Mechatronics, 82234 Wessling, Germany. U. Frese is with Faculty 3 – Mathematics and Computer Science, University of Bremen, 28359 Bremen, Germany. R. Wagner is also with University of Bremen. Contact {rene.wagner,berthold.baeuml}@dlr.de, ufrese@informatik.uni-bremen.de.

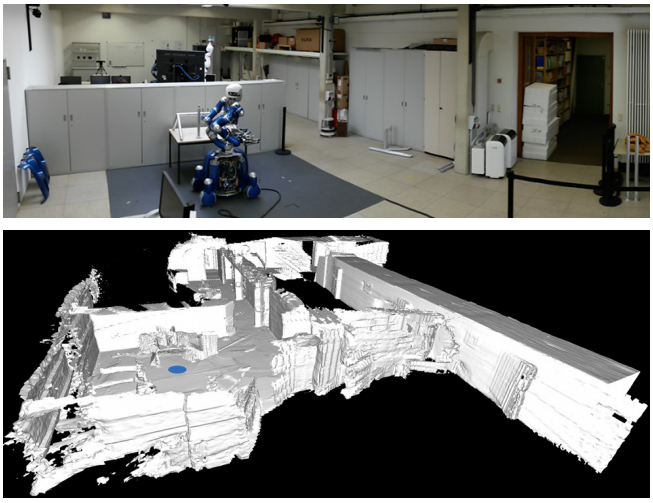

Fig. 1. DLR's humanoid Agile Justin [1] in our mobile manipulation scenario (top) and a marching cubes [15] mesh export of our signed distance function model after graph optimization (bottom, robot pose marked by blue ellipsis). To build the map Justin first observed the tabletop workspace, left the lab through the door at the far end, and returned to the tabletop workspace via two outside corridors and the lab door on the right such that the overall trajectory forms a clockwise loop. The companion video shows the optimization in progress and a fly-through of the generated model.

In contrast to the dense geometric approach of KinectFusion, the graph SLAM (simultaneous localization and mapping) paradigm is purely parametric, i.e. sparse. Variables to be estimated (e.g. robot poses) are nodes in a graph. Measured sensor data and measurement models (predicting measurements based on all dependent variables) form links (edges). This can be transformed into a least squares problem and the maximum likelihood values of all variables obtained through standard numerical optimization methods (Gauss-Newton, Levenberg-Marquardt, etc.). With Agile Justin this has already proved very successful in modeling the robot in a calibration context [20, 3].

Thus, we would like to model the robot including all errors in the forward kinematics chain and odometry using a graph SLAM framework and combine this with KinectFusion-style

dense mapping and ICP. In this setup the different sensors nicely complement each other, e.g. odometry measures the distance traveled quite accurately while the Kinect constrains the orientation of the robot. This paper is the first step towards this goal in so far as it integrates KinectFusion's TSDF maps and ICP with the SLoM [10] graph SLAM framework. The robot model in this paper is the most straight forward one: simple odometry. As soon as this combination works a more sophisticated robot model (e.g. kinematic errors, elasticities, vibration, etc.) can later be implemented purely on the traditional graph SLAM side.

The key idea behind our approach is to split the world into overlapping sub-maps consisting of a small TSDF grid each. The pose of each sub-map's origin is added as an additional variable to the graph SLAM problem. All sub-maps are attached to the robot pose graph via graph SLAM links formed by ICP. As the pose graph is optimized most deformations of the map as a whole only require rigid body transformations of its sub-map parts, i.e. the optimizer can simply change the position and orientation of their origins. Only if the pose graph within a sub-map changes significantly a sub-map will need to be rebuilt. This results in an important property: Sub-maps are generated in real-time and can be used for e.g. planning at all times.

The remainder of this paper is structured as follows. After a discussion of related work (II), we introduce our approach to graph SLAM on signed distance function maps (III), show how this can be used for SLAM on Justin (IV), how sub-maps are managed in our system (V), how an improved sensor model allows us to use KinectFusion on long range Kinect depth data (VI), discuss experimental results (VII), and close with conclusions and future work (VIII).

II. Related Work

The idea to work with a collection of local maps instead of one large map is an old one. To our knowledge the first realization in the SLAM area was the Atlas framework [5]. Graph SLAM is now an established method with several open source software frameworks available, e.g. with a focus on incremental optimization [11], speed [14], or elegant handling of manifold spaces [10]. A separate line of research has lead to the KinectFusion [16] algorithm which was first to combine the previously computationally challenging ICP [2] and dense truncated signed distance function (TSDF) mapping [7] in a real-time GPU implementation. Extensions exist to increase the map coverage by moving volume parts in and out of the GPU on demand [6], using hierarchical data structures based on hash-tables [18], octrees [19] or our own multi-scale TSDF [22]. Unlike our work, these do not address global consistency but they are complementary in that these data structures can be used to represent each individual sub-map at higher resolution. The drift in KinectFusion's ICP can be alleviated by simultaneously minimizing the reprojection residuals in RGB images [23]. To get closer to global consistency, one approach [24] is to perform graph SLAM using a surface mesh extracted from a sliding volume KinectFusion variant as the map and mesh deformation to keep the mesh consistent with the SLAM graph. The original TSDF is discarded, the mesh only represents the mean of the surface distribution thus preventing correct fusion of new depth data into the map. Also, the mesh export loses free vs. occupied space information which is vital for safe motion planning.

Finally, we have been made aware of two very recent approaches after initial submission of our work. One [25] performs KinectFusion on short sub-sequences of temporally adjacent depth images, extracts a mesh fragment from each resulting TSDF and then applies least-squares optimization to minimize errors in overlapping mesh fragments while penalizing non-smooth camera trajectories. Conceptually this is similar to our work. The additional sensor data (odometry) in our approach, however, gives a better idea of the camera trajectory than the smoothness assumption and the reported computing time of 160 times the acquisition time is problematic. The other [9] is very similar to ours in that it also uses a set of small TSDF volumes and graph optimization for global consistency. The TSDF volumes are object-aligned bounding volumes requiring segmentation and volume growing. We believe the parameterization of ICP links is different. Additionally, RGB links in the spirit of [23] are used. They apply some of the sensor model-related changes which we describe in VI. The critical μ-scaling in (24), however, is missing, as is a discussion of the exact graph setup, and timings broken down into ICP vs. graph optimization steps, thus making a computing time comparison with our implementation difficult although it appears that the graph optimization is slightly faster and the rest slower than ours.

III. Graph SLAM with TSDF Maps

To perform graph SLAM with TSDF maps using the SLoM framework we need to find ways to represent TSDF maps in the framework analogously to a SLoM variable and to use the result of KinectFusion's ICP as a SLoM link.

A. Sub-Maps

On the mapping side, rather than using a single large TSDF map, our key idea is to split the world into a set of small sub-maps. The assumption behind this is that the local trajectory of the robot is usually well known. Errors take time (distance travelled) to accumulate. Thus, global consistency can be achieved mainly by rigid body transformation of sub-maps, i.e. without touching the dense data.

To make this possible we define a sub-map to be a TSDF grid plus the 6DOF pose of its origin. This pose is added as a SLoM variable to the graph SLAM problem, so that the optimizer can move maps around if that reduces the overall error in the graph. In our current single-GPU implementation, each sub-map consists of a 128^3 voxels TSDF grid covering a volume of $(7\,\mathrm{m})^3$. We found this rather coarse resolution sufficient due to our modeling changes in VI and since the TSDF achieves sub-voxel precision. However, we have 16 K20 GPUs available (across two machines) and one could

also move sub-maps between GPU and host (CPU) memory on demand to gain higher resolution.

B. KinectFusion ICP Revisited

Let us first have a look at what KinectFusion's ICP [16] does internally. It aims to determine the $\hat{T} \in SE(3)$ that minimizes the point-to-plane residual between two point clouds

$$\sum_i (r_i)^2 := \sum_i \left((\hat{T} v_i - \hat{v}_i)^T \hat{n}_i \right)^2, \tag{1}$$

where $\hat{v}_i$ and $\hat{n}_i$ are the points and normals from the reference point cloud and v_i the corresponding point from the second point cloud (determined via projective data association [16]).

To find this minimum it performs least squares optimization by runnning a fixed number of Gauss-Newton iterations, i.e., starting from an initial guess, e.g., $\hat{T}_0 = I$, in each iteration, it solves the normal system

$$J^T J \delta = J^T r, \tag{2}$$

where r is the residual, i.e. the components of (1), J the Jacobian of r w.r.t. δ, and δ is a 6-vector parameterizing a small relative $SE(3)$ transform. J and r are very large, i.e., $J \in \mathbb{R}^{6\times m}$ and $r \in \mathbb{R}^{m\times 1}$ where m is in the order of the number of depth pixels. Thus, the much more compact $J^T J \in \mathbb{R}^{6\times 6}$ and $J^T r \in \mathbb{R}^{6\times 1}$ are computed in parallel on the GPU. The improved estimate is then

$$\hat{T}_{k+1} = \tilde{T}(\delta_k) \cdot \hat{T}_k, \tag{3}$$

where $\tilde{T}$ turns a parameter vector $\delta = (\beta, \gamma, \alpha, t_x, t_y, t_z)^T$ into a linearized $SE(3)$ transform [16]

$$\tilde{T}(\delta) = \begin{bmatrix} 1 & \alpha & -\gamma & t_x \\ -\alpha & 1 & \beta & t_y \\ \gamma & -\beta & 1 & t_z \end{bmatrix}. \tag{4}$$

At the final iteration, $\hat{T}_n$ is the mean of the Gaussian estimated transformation between the two point clouds and $J^T J$ its information matrix or inverse of the covariance, i.e.

$$J^T J \approx \Omega_{ICP} = \Sigma_{ICP}^{-1}. \tag{5}$$

The mean is a regular $SE(3)$ transform and can be interpreted independently of how it was computed and can thus be directly used in a SLoM link. More care must be taken with Σ_{ICP} as it is defined in terms of the parameterization (4) of δ which we will get back to shortly.

C. KinectFusion ICP as a Link in SLoM

First, let us briefly review the SLoM [10] framework. It solves the least squares problem minimizing the sum of squared residuals

$$\hat{X} = \underset{X}{\operatorname{argmin}} \tfrac{1}{2} \left\| \begin{matrix} f_1(X) \boxminus z_1 \\ \vdots \\ f_m(X) \boxminus z_m \end{matrix} \right\|_\Sigma^2, \tag{6}$$

where X is the stacked vector of all variables to be estimated, the z_i are the measured data, the $f_i(X)$ the measurements predicted by the respective measurement model f_i based on all dependent variables in X, and $\|\cdot\|_\Sigma$ denotes that all measurement residuals have been normalized to be of unit covariance. $\boxminus$ is the difference operator of the $\boxplus$-method [10]. Combined with the matching addition operator $\boxplus$, this enables SLoM's least squares optimizer to manipulate variables from a non-Euclidean manifold $\mathcal{S}$ (such as 3D orientations) as if locally they were plain vectors [10]:

$$\boxplus : \mathcal{S} \times \mathbb{R}^n \to \mathcal{S}, \qquad \boxminus : \mathcal{S} \times \mathcal{S} \to \mathbb{R}^n. \tag{7}$$

This can be thought of as a generalization of the $\exp$ and $\log$ known from Lie group theory to generic manifolds.

SLoM supports the use of different $\boxminus$ operators for different measurements. Thus, we need to find a $\boxminus$ that is compatible with the parameterization of the small transform in KinectFusion's ICP as defined by $\tilde{T}(\delta)$ from (4). Apart from sign and variable ordering, the rotation part is the same as the standard $\boxplus$ on $SO(3)$ [10]. The translation is normally handled independently by SLoM, but we can easily define a new $\boxplus$ that does the same as KinectFusion:

$$\begin{bmatrix} R_1 & t_1 \\ 0 & 1 \end{bmatrix} \boxplus \delta := \begin{bmatrix} R_2 & t_2 \\ 0 & 1 \end{bmatrix}, \tag{8}$$

where

$$R_2 := \exp\left(\widehat{\begin{bmatrix} -\alpha \\ -\beta \\ -\gamma \end{bmatrix}}\right) \cdot R_1 \tag{9}$$

$$t_2 := \exp\left(\widehat{\begin{bmatrix} -\alpha \\ -\beta \\ -\gamma \end{bmatrix}}\right) \cdot t_1 + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}, \tag{10}$$

and $\exp$ denotes the exponential map used in the standard $SO(3)$ $\boxplus$ [10] implemented by means of the Rodriguez formula. The corresponding $\boxminus$ is then simply the inverse:

$$\begin{bmatrix} \beta \\ \gamma \\ \alpha \\ t_x \\ t_y \\ t_z \end{bmatrix} := \begin{bmatrix} R_2 & t_2 \\ 0 & 1 \end{bmatrix} \boxminus \begin{bmatrix} R_1 & t \\ 0 & 1 \end{bmatrix}, \tag{11}$$

where

$$\begin{bmatrix} -\alpha \\ -\beta \\ -\gamma \end{bmatrix} := \log\left(R_2 \cdot R_1^{-1}\right) \tag{12}$$

$$\begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} := t_2 - (R_2 \cdot R_1^{-1}) \cdot t_1 \tag{13}$$

and $\log$ is the inverse of the standard $SO(3)$ $\boxplus$ $\exp$ above.

IV. Defining the SLoM Links for Graph SLAM

Now that we have shown how KinectFusion-style TSDF maps and graph SLAM can be combined in general we will discuss how this can be used for graph SLAM on a wheeled robot such as Agile Justin.

The nodes in the SLoM graph are the variables to be estimated. In our scenario these are the robot poses $\hat{F}_{W\leftarrow R_t}$ and the poses of the sub-map origins $\hat{F}_{W\leftarrow S_k}$ where $\hat{F}_{A\leftarrow B}$ denotes an estimated $SE(3)$ transform from B to A, W is the world, R_t the robot coordinate system at time t, and S_k the k-th sub-map. The measurement models and measured data form the edges or *links* in the graph. For each type of measurement these will be given below.

A. Odometry Links

The odometry measurement model simply determines the relative transform of two poses:

$$\hat{F}_{R_t \leftarrow R_{t+1}} = (\hat{F}_{W \leftarrow R_t})^{-1} \hat{F}_{W \leftarrow R_{t+1}} \tag{14}$$

Comparing this with the transform $F_{R_t \leftarrow R_{t+1}}$ measured by odometry yields the following residual:

$$\hat{F}_{R_t \leftarrow R_{t+1}} \boxminus F_{R_t \leftarrow R_{t+1}} \tag{15}$$

The covariance of the odometry link is derived from the assumption that along the (approximately) straight line segment (for a small distance) traveled in that link noise continuously affects the wheel measurements that made the link. This assumption leads to an analytical formula that correctly models the dependence of the covariance on the link itself and the correlation between position and orientation. We have taken this formula from the function `stdCovariance` in the implementation of [8] and lifted it to 3D by adding noise in pitch and roll direction in a similar way to accomodate for vibrations and steps in the environment.

B. Near Ground Links

Although, locally, we want to permit full 6DOF motion, globally, Justin remains roughly at the same level by virtue of its design as a wheeled robot and its environment. We call this the "near ground measurement" which simply states that up to a standard deviation of $0.05\,\mathrm{m}$ the z-ordinate of all $\hat{F}_{W \leftarrow R_t}$ is zero which corresponds to the height of door steps in our lab. This does not suppress motion in z-direction altogether, but prevents unconstrained drift.

C. Frame-to-Map Links

So far, we have a pure robot pose graph. We will now attach the sub-maps to it via ICP. For these frame-to-map ICP links we first compute a raycast (as KinectFusion does on a single map) on all nearby sub-maps that are intersected by the surface in the current depth image (V-B) and then merge these into a single raycast by taking the minimum depth data point for each pixel. This yields a synthetic depth image as it should have been observed from the sensor pose $(\hat{F}_{W \leftarrow R_t} F_{R_t \leftarrow K_t})$ given the surface model in the sub-maps, where $F_{R_t \leftarrow K_t}$ is the transform from Kinect (K) to robot coordinates at time t according to forward kinematics. Both, the synthetic and the real depth image are converted into point clouds and then we run ICP to estimate the relative transform between the two. Conceptually, however, this is a relation between the depth image and the map. Thus, we transform it to yield the sensor pose relative to each sub-map $F_{S_k \leftarrow K_t}$ and with $F_{R_t \leftarrow K_t}$ obtained from forward kinematics and stored as user data in the SLoM link compute the residual:

$$(\hat{F}_{W \leftarrow S_k})^{-1} (\hat{F}_{W \leftarrow R_t} F_{R_t \leftarrow K_t}) \boxminus F_{S_k \leftarrow K_t} \tag{16}$$

We also transform the covariance as returned from ICP to be in the correct coordinate system via

$$\Sigma'_{ICP} = \begin{bmatrix} Q & 0 \\ 0 & Q \end{bmatrix} \cdot \Sigma_{ICP} \cdot \begin{bmatrix} Q & 0 \\ 0 & Q \end{bmatrix}^T \tag{17}$$

where Q is the rotation in $F_{S_k \leftarrow K_t}$.

D. Frame-to-Frame Links

We also support frame-to-frame ICP links. In this case ICP estimates the relative transform $F_{K_t \leftarrow K_{t+1}}$ between two Kinect poses at two consecutive points in time based on the corresponding depth images. The covariance is taken as-is from the ICP and with $F_{R_t \leftarrow K_t}$, $F_{R_{t+1} \leftarrow K_{t+1}}$ obtained from forward kinematics and stored as user data in the SLoM link the residual is determined as follows:

$$(\hat{F}_{W \leftarrow R_t} F_{R_t \leftarrow K_t})^{-1} (\hat{F}_{W \leftarrow R_{t+1}} F_{R_{t+1} \leftarrow K_{t+1}}) \boxminus F_{K_t \leftarrow K_{t+1}} \tag{18}$$

Note, however, that although also ICP-based this is structurally different from frame-to-map links: While frame-to-map links essentially perform localization against a global frame of reference, frame-to-frame links perform localization against the previous frame only, i.e. localization errors, no matter how small, add up quickly when many frame-to-frame links are used. Thus, the only frame-to-frame link we actually use is to connect the last and first depth frame for loop closure as discussed in the next subsection.

E. Relocalization and Loop Closure

Relocalization refers to determining whether the current view matches a previously visited place. This is useful to reset accumulated pose estimation errors when revisiting known places after long loops. It could be achieved by keypoint-based methods such as [4] but it is not our focus in this paper. Instead, we manually pass in relocalization information as an external signal and compute the spatial relationship via ICP. This relative transformation is added to the graph SLAM problem as a frame-to-frame link. Our main intention here is to show that the rest of our system, particularly the handling of sub-maps, supports this loop closure situation.

F. Loose Sub-Map Attachment Links

Finally, to prevent the sub-map origins from drifting along undefined DOFs (e.g. in a straight corridor) we introduce a loose coupling (standard deviation $1\,\mathrm{m}$, $1\,\mathrm{rad}$) of each sub-map S_k to the robot pose R_{ξ_k} from which it was created. Thus, we compare the predicted sub-map pose relation with the initial relative sub-map pose stored at the time of its creation ξ_k to yield the residual function

$$(\hat{F}_{W \leftarrow R_{\xi_k}})^{-1} \hat{F}_{W \leftarrow S_k} \boxminus F_{R_{\xi_k} \leftarrow S_{k,\xi_k}} \tag{19}$$

V. Sub-Map Management

A. Sub-Map Creation

Our placement policy for sub-maps to be newly created is to place their center at a fixed distance of presently 1.2 times the map radius ($3.5\,\mathrm{m}$) ahead of the robot in the current viewing direction of the Kinect sensor. Maps are newly created if this center is more than the map radius away from any other recently added map. An example of this process is illustrated in Fig. 2. A better policy is certainly possible (less overlap, fewer sub-maps). However, to us it was most important to ensure that no sensor data is "lost" because

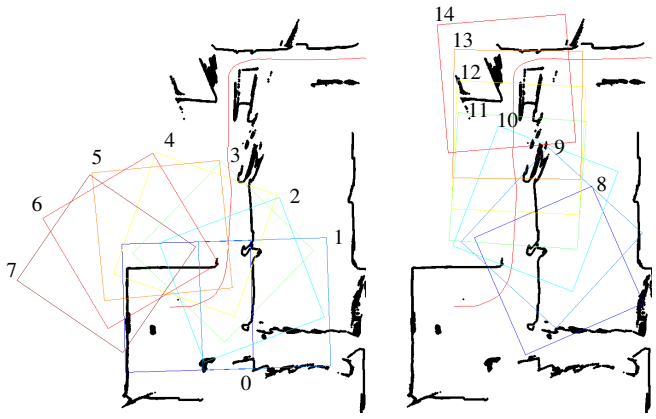

Fig. 2. Sub-map creation policy: New sub-maps are always created ahead of the robot in the viewing direction of the Kinect sensor such that those parts of the world observed by the sensor can be stored in the map. When the head turns (left) this leads to maps being swept around the robot. When the robot moves while the head joints are fixed (right) new maps cover the area ahead of the robot.

it falls into space that is not covered by a sub-map and to ensure overlap such that free-space information in particular is preserved for planning [21].

Note that the division into sub-maps could also be changed after the fact. Whenever the surface within two TSDF sub-maps is well-aligned a combined map can be obtained by simply projecting voxels (signed distance plus weight) from one into the other and performing weighted averaging. In particular, this would be useful upon loop closure (but we have not implemented it yet). Note that perfect alignment is not necessary for this to work: There just needs to be sufficient overlap of the TSDF support regions (i.e. the region where the signed distance is not truncated yet).

B. Sub-Map Updating and Contribution of Sub-Maps to SLoM Links

We still need to answer two related questions: 1. When does a sub-map need to be updated to enter new depth data? 2. When does a sub-map contribute to a SLoM ICP link? Both can be determined by computing the intersection of depth data with the bounding box of a sub-map. We do this on the GPU by checking the following two conditions for each depth pixel. A sub-map update is needed if

$$\lambda_1 < z + \mu(z), \tag{20}$$

where λ_1 is the depth at which a ray through this pixel first intersects the bounding box of the sub-map, z the measured depth and $\mu(z)$ the depth-dependent TSDF truncation distance (VI). Whether a sub-map contributes to a SLoM ICP link ultimately depends on the surface model within the sub-map and will be determined by performing a raycast to generate the point cloud the ICP takes as input. Raycasts, however, are expensive even on the GPU, so we want to do this only if there is a chance that a sub-map can contribute at all. This is the case if for any depth pixel

$$\lambda_1 < z < \lambda_2, \tag{21}$$

where λ_2 is the depth at which a ray through this pixel intersects the back of the bounding box of the sub-map. Executed on the GPU these two checks are rather cheap, taking about 0.15ms per depth image and sub-map.

C. Sub-Map Rebuilding

Deformation of the pose graph can lead to inconsistent sub-maps especially when the robot moves around corners where there is a large odometry error due to friction/wheel slippage and ICP is difficult as the world around the corner has not observed before. Thus, we re-build affected maps by re-running the usual KinectFusion map update procedure on all frames with their new camera pose as determined from the robot pose graph.

To make this possible we keep the raw disparity data as PNG-compressed images ($\sim$ 110kB/frame) in host (CPU) memory which are transferred to the GPU as needed. Keeping the raw data may seem wasteful but host memory is cheap today, GPU memory sizes and transfer bandwidth are steadily increasing. With future GPU/unified memory architectures already announced by AMD/Intel this will be even less problematic as the lines between host and GPU memory will blur further.

Presently, we rebuild all sub-maps. However, we believe this is actually only necessary when the pose graph within a sub-map has been deformed to a certain extent. We have not implemented the book-keeping code required to determine this yet, but it would make the re-building of a sub-map a rare occasion primarily needed when large loops are closed.

Note that alternatingly re-building of sub-maps and graph optimization yields an expectation-maximization (EM) like procedure that converges to the maximum likelihood overall SLAM solution. In our experience, already a single iteration results in very good maps (Fig. 4; Fig. 6, right). The full iterative process with alternating map re-building and re-running the ICP, however, is future work.

VI. An Improved Sensor Model in KinectFusion

The original KinectFusion algorithm [16] neglects an important characteristic of the Kinect sensor: It is essentially a stereo camera and thus measures disparity not depth. This was first described within a calibration context [13], later re-discovered through analysis of the optical geometry [12] or model fitting [17] and most importantly means that the metric depth error is proportional to z^2. Analytically, this is easy to see since

$$z = \frac{b \cdot f}{d} \Rightarrow \frac{\mathrm{d}z}{\mathrm{d}d} = \frac{-b \cdot f}{d^2} = \frac{z^2}{-b \cdot f}. \tag{22}$$

As a by-product of our own Kinect calibration [20] we can confirm that as illustrated in Fig. 3 the disparity error is constant while the depth error grows quadratically with depth where the 2-σ bound can be approximated as

$$2\sigma_z = 0.008\,\mathrm{m}^{-1} \cdot z^2. \tag{23}$$

Our focus here is that this effect has dramatic consequences wherever it is not modeled correctly when using Kinect data at large ranges of up to $10\,$m as it leads to 2-σ depth errors of up to $\pm 0.8\,$m. Thus, we modify several parts of KinectFusion. The combination of these changes allow us

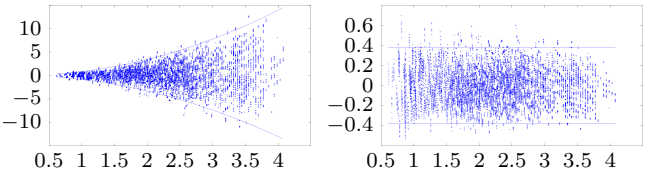

Fig. 3. Kinect depth error [cm] vs. depth [m] (left) and the disparity error [px] vs. depth [m] (right; 2σ bounds as solid lines) as a by-product of Kinect calibration using checkerboard corners with known geometry [20]. The disparity error is constant while the depth error grows quadratically.

to achieve a quality of large-scale maps that to our knowledge has not been described in the literature before.

In the bilateral filter which is used for smoothing of depth images, the depth discontinuity parameter needs to be scaled with σ_z. The same goes for the threshold that determines whether pixels contribute to the averaging filter in the depth image down-sampling. As for the ICP, like any other least squares method it should normalize measurement residuals based on appropriate noise models. SLoM does this based on the Σ in (6). The original KinectFusion ICP [16] treats all point-to-plane residuals equally leading to biased sensor pose estimates. We normalize these by scaling them with $1/\sigma_z$ when the normal system (2) is computed on the GPU. This is not exact as the exact measurement noise obviously depends on the orientation of the surface patch with respect to the sensor (which would be computationally too expensive to consider) but it models the predominant error source.

The most visible effect on the overall mapping performance is achieved in the TSDF mapping code itself. Recall that the TSDF stores in each voxel the signed distance to the nearest surface but only within a support region of $\pm\mu$ around surface. Farther into objects the TSDF is undefined and farther away from objects it is truncated to a maximum value. The weighted depth averaging that is vital to map quality, as over time it eliminates sensor noise, only happens within this support region. We originally assumed μ to be a fixed parameter, upon second reading it turns out that the KinectFusion authors [16] in fact scale μ linearly with distance from the sensor. Given (23) this is obviously not sufficient. Instead we scale μ via

$$\mu(z) := \max(15 \cdot l, 2\sigma_z), \tag{24}$$

where l is the voxel side length and 15 an empirically determined factor [22] that defines the minimum support region size. The second mapping change is in the weights of the weighted depth averaging. The KinectFusion authors propose [16] a weight of $\cos(\theta)/z$ where θ is the angle of the camera ray vs. the surface normal but suggest that in their experience a constant weight of 1 works equally well. We find that this is not true in our setting where surfaces are observed first from very far away (10 m) and later from up close ($<$ 1 m). Here, too, it is crucial to model the sensor noise that increases quadratically with depth. Thus, we use weights of $1/z^2$. We believe the angular factor of $\cos(\theta)$ would help further as Justin often sees some surfaces under a poor viewing angle, but we have not implemented this as, again, it would be computationally too expensive.

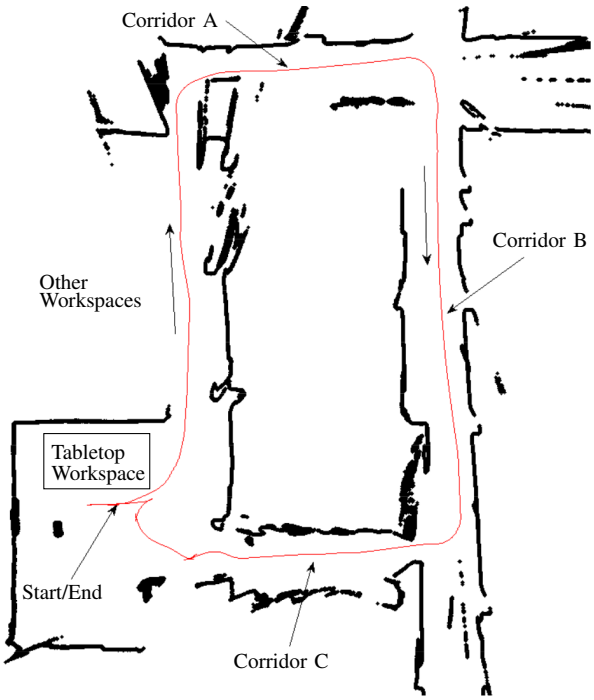

Fig. 4. Overview of our lab environment used for the experiments with several places labeled. The panoramic photo in Fig. 1 was taken from the bottom left corner. The overview map is a 2D slice at $z = 1.5$ m of a 3D map generated in a single large TSDF grid after loop closure and pose graph optimization (i.e. without frame-to-map links).

VII. Experiments

We already know [20] that graph optimization using the $\boxplus$-method works well on problems consisting of 6DOF pose relations. Thus, we believe the key question in evaluating the method in this paper is: What happens to the map under deformation of the SLoM graph – does it remain consistent?

To assess this let us consider the worst case, a loop closure after a long loop. We have acquired several logs of Agile Justin traversing a 50 m loop in the institute building. An overview of this scenario with several labeled places is given in Fig. 4, the final map is depicted in Fig. 1. The trajectory was chosen such that the start and end pose of the robot was identical up to a precision of a few millimeters and a few degrees. As noted above, the loop closure ICP link is added upon a manual signal when the robot reaches this start/end pose after the loop. All processing is done on log data. The initial sub-maps and the full SLoM graph are built in real-time. Graph optimization happens in batch mode once the complete graph is available. In future work, this could easily be changed to run at regular intervals and on a relevant sub-graph only for online operation.

To check for global map consistency we run the optimizer with the frame-to-map ICP links deactivated. What we expect to see is that the sub-maps are moved generally consistently with the pose graph as they are loosely attached to the poses from where they were created via (19). As shown in Fig. 5 this is indeed the case. A closer look, however, reveals that there is some mis-alignment in the form of gaps between sub-maps in corridor B as shown at the bottom of Fig. 5. This is expected as there is no other measurement that would observe this error when the frame-to-map links are

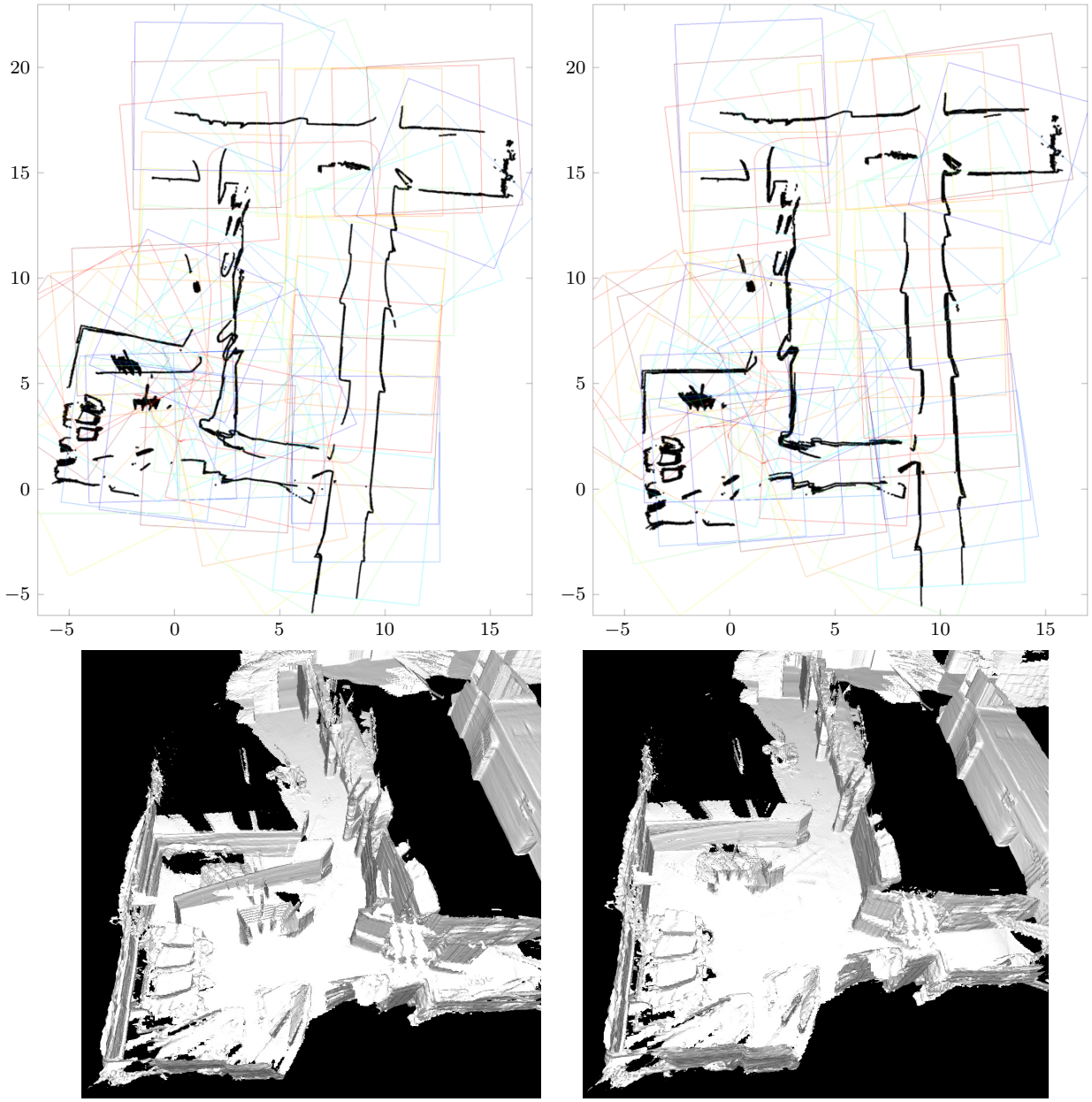

Fig. 5. 2D slice of the map before (top left) and after optimization (top right; axis ticks in 5 m steps). 3D mesh export of the overlapping sub-maps before (bottom left) and after (bottom right) graph optimization. The graph optimization nicely aligns the overlapping sub-maps.

deactivated. If we activate these links as well, as illustrated in Fig. 6, the overall map becomes locally consistent in so far as gaps between sub-maps are now closed and the corridor is "straightened". This straightening effect, however can only work at the resolution chosen for the division into sub-maps as the SLoM graph optimization cannot modify the content of sub-maps. This is handled by re-building the sub-maps based on the optimized sensor poses taken from the SLoM graph and the original Kinect depth data. Within a single map this effect is hard to see in our data sets as Justin's odometry alone is already very precise (less than 1.8 m accumulated position error after the whole loop). Another effect, however, is clearly visible in Fig. 6: Overlapping sub-maps disagree slightly about the location of surfaces before sub-map-rebuilding. After sub-map-rebuilding they are much more consistent. Perfect consistency will only be achieved after iterated graph optimization, ICP, and map-rebuilding, but, as noted in V-C, this iterative process is future work.

All processing was done on a single-core of a Xeon E5-2643 @ 3.30 GHz and an Nvidia K20 GPU with 5 GB GPU memory. Computing times and problem size are as follows. ICP run including preprocessing per frame: 22 ms; map update per frame and 128^3 voxel sub-map: 1 ms; SLoM iteration: 1.5 s; typically 8 iterations until convergence; SLoM graph size: 5313 robot poses, 52 sub-map poses, 5312 odometry links, 5313 near-ground links, 28121 frame-to-map ICP links, and 1 frame-to-frame ICP link.

VIII. Conclusions and Future Work

We have presented a method to integrate dense geometric TSDF maps as used in KinectFusion with a sparse parametric

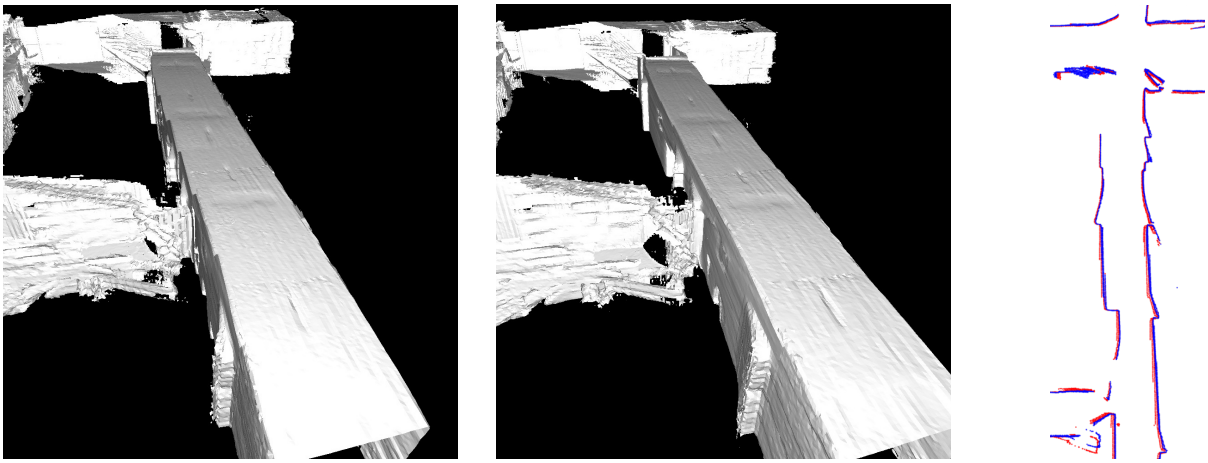

Fig. 6. *Left and Center:* 3D mesh export of sub-maps containing the corridor based on odometry plus loop closure (left) and additionally with frame-to-map ICP links (center), both after graph optimization. While on the left sawtooth-like errors are clearly visible, the frame-to-map ICP links cause the optimizer to better align sub-maps to one another and to generally straighten the corridor. *Right:* 2D slice of corridor B before (red) and after (blue) sub-map re-building (right). Before map-rebuilding overlapping sub-maps disagree slightly about the location of surface so that they appear multiple times across the whole map.

representation common in graph SLAM. The key idea is to have local TSDF sub-maps attached to reference nodes in the SLAM graph and derive graph-SLAM links via ICP. The graph-SLAM optimizer can then move these reference nodes and thus the overall map can be deformed without touching individual sub-maps. Re-building of sub-maps is only needed in case of significant deformation within a sub-map. Additionally, we have modified the KinectFusion algorithm to improve handling of long range data by better respecting the range-dependent uncertainty. We have implemented our method in conjunction with a simple odometry model of a humanoid robot, evaluated it on log data from a loop through corridors of our lab building and confirmed that the map deformation in particular works as intended.

With little implementation effort, the system could be converted from batch optimization to online, incremental operation. The EM-like iterative sub-map re-building (V-C) is not difficult either. In future work, we intend to fully leverage the parametric graph-SLAM capabilities for a more sophisticated robot model (e.g. elasticities in the kinematics chain) and fusion of additional sensor data (e.g. IMUs).

Acknowledgements

This work was partly supported under DFG grant SFB/TR 8 Spatial Cognition. We thank C. Hertzberg for fruitful discussion on SLoM and T. Hammer for the panoramic photo in Fig. 1.

References

[1] B. Bäuml et al. Agile Justin: An upgraded member of DLR's family of lightweight and torque controlled humanoids. In *IEEE Int. Conf. on Robotics and Automation*, 2014.

[2] P. J. Besl and N. McKay. A method for registration of 3-D shapes. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 14(2), 1992.

[3] O. Birbach, B. Bäuml, and U. Frese. Automatic and self-contained calibration of a multi-sensorial humanoid's upper body. In *IEEE Int. Conf. on Robotics and Automation*, 2012.

[4] M. Bosse and R. Zlot. Place recognition using keypoint voting in large 3D lidar datasets. In *IEEE Int. Conf. on Robotics and Automation*, 2013.

[5] M. Bosse, P. Newman, et al. SLAM in large-scale cyclic environments using the Atlas framework. *Int. Journal on Robotics Research*, 23(12), 2004.

[6] J. Chen, D. Bautembach, and S. Izadi. Scalable real-time volumetric surface reconstruction. *ACM Trans. Graph.*, 32(4), 2013.

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

[8] U. Frese and L. Schröder. Closing a million-landmarks loop. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2006.

[9] P. Henry, D. Fox, A. Bhowmik, and R. Mongia. Patch volumes: Segmentation-based consistent mapping with RGB-D cameras. In *Int. Conf. on 3D Vision*, 2013.

[10] C. Hertzberg, R. Wagner, U. Frese, and L. Schröder. Integrating generic sensor fusion algorithms with sound state representations through encapsulation of manifolds. *Information Fusion*, 14(1), 2013.

[11] M. Kaess, A. Ranganathan, and F. Dellaert. iSAM: Incremental smoothing and mapping. *IEEE Trans. on Robotics*, 24(6), 2008.

[12] K. Khoshelham and S. O. Elberink. Accuracy and resolution of Kinect depth data for indoor mapping applications. *Sensors*, 12(2), 2012.

[13] K. Konolige and P. Mihelich. Technical description of Kinect calibration. wiki.ros.org/kinect_calibration/technical, 2010.

[14] R. Kümmerle, G. Grisetti, et al. g2o: A general framework for graph optimization. In *IEEE Int. Conf. on Robotics and Automation*, 2011.

[15] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. *Computer Graphics*, 21(4), 1987.

[16] R. A. Newcombe, S. Izadi, et al. KinectFusion: Real-time dense surface mapping and tracking. In *IEEE Int. Symposium on Mixed and Augmented Reality*, 2011.

[17] C. Nguyen, S. Izadi, and D. Lovell. Modeling Kinect sensor noise for improved 3D reconstruction and tracking. In *3DIMPVT*, 2012.

[18] M. Nießner, M. Zollhöfer, et al. Real-time 3D reconstruction at scale using voxel hashing. *ACM Trans. Graph.*, 32(6), 2013.

[19] F. Steinbruecker, C. Kerl, et al. Large-scale multi-resolution surface reconstruction from RGB-D sequences. In *IEEE Int. Conf. on Computer Vision*, 2013.

[20] R. Wagner, O. Birbach, and U. Frese. Rapid development of manifold-based graph optimization systems for multi-sensor calibration and SLAM. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2011.

[21] R. Wagner, U. Frese, and B. Bäuml. 3D modeling, distance and gradient computation for motion planning: A direct GPGPU approach. In *IEEE Int. Conf. on Robotics and Automation*, 2013.

[22] R. Wagner, U. Frese, and B. Bäuml. Real-time dense multi-scale workspace modeling on a humanoid robot. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2013.

[23] T. Whelan, H. Johannsson, et al. Robust real-time visual odometry for dense RGB-D mapping. In *IEEE Int. Conf. on Robotics and Automation*, 2013.

[24] T. Whelan, M. Kaess, J. Leonard, and J. McDonald. Deformation-based loop closure for large scale dense RGB-D SLAM. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2013.

[25] Q.-Y. Zhou, S. Miller, and V. Koltun. Elastic fragments for dense scene reconstruction. In *Int. Conf. on Computer Vision*, 2013.