GravityFusion: Real-time Dense Mapping without Pose Graph using Deformation and Orientation

Puneet Puri[1], Daoyuan Jia[2], and Michael Kaess[1]

Abstract— **In this paper, we propose a novel approach to integrating inertial sensor data into a pose-graph free dense mapping algorithm that we call GravityFusion. A range of dense mapping algorithms have recently been proposed, though few integrate inertial sensing. We build on ElasticFusion, a particularly elegant approach that fuses color and depth information directly into small surface patches called surfels. Traditional inertial integration happens at the level of camera motion, however, a pose graph is not available here. Instead, we present a novel approach that incorporates the gravity measurements directly into the map: Each surfel is annotated by a gravity measurement, and that measurement is updated with each new observation of the surfel. We use mesh deformation, the same mechanism used for loop closure in ElasticFusion, to enforce a consistent gravity direction among all the surfels. This eliminates drift in two degrees of freedom, avoiding the typical curving of maps that are particularly pronounced in long hallways, as we qualitatively show in the experimental evaluation.**

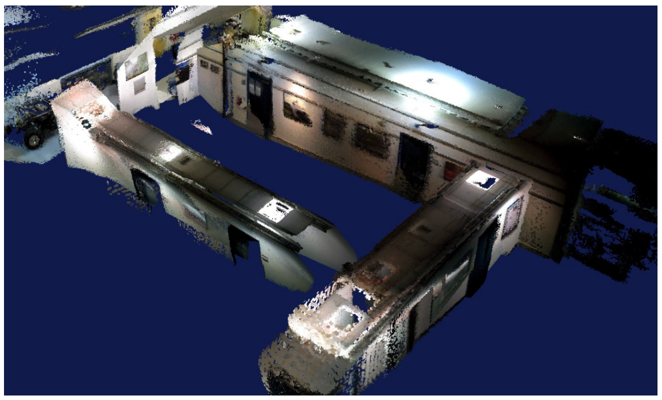
Fig. 1. GravityFusion produces a consistent model through a low-feature corridor circuit, where other methods fail.

I. Introduction

Combining dense 3D mapping with inertial sensing provides significant advantages over pure vision-based mapping. Dense 3D maps are relevant for mobile robot navigation and manipulation, for augmented and virtual reality applications, and for inspection tasks. And while most dense mapping systems rely on only stereo or RGB-D cameras, inertial sensors are becoming ubiquitous because of their low cost and wide range of applications.

In the absence of a global reference such as GPS, mapping algorithms suffer from drift, which can partially be mitigated by loop closures in a simultaneous localization and mapping (SLAM) context. Drift results from the accumulation of noisy sensor data over time, where small errors add up over longer trajectories to yield significant discrepancies between the model and the physical world. A typical solution to reducing drift is loop closure: When we re-observe a previously visited area of the environment, we can create a constraint between the earlier observations and the current sensor data, a loop closure, that allows removal of some of the accumulated drift.

Incorporating loop closures into dense mapping requires the right model representation: Volumetric representations are not suitable, as translation and rotation of parts of the model is computationally very expensive. Surface mesh representation on the other hand can be deformed at relatively low cost, but re-integrating two surface meshes after loop closure is challenging. The surfel representation used in ElasticFusion allows for both, efficient deformation and re-integration upon loop closure. We therefore use ElasticFusion as the basis for our work.

Even with loop closures, some drift remains, and additional measurements, such as from an inertial sensor, are needed to obtain accurate models. When mapping a single floor of a building there is typically insufficient data to constrain drift in some directions, yielding a curved map. The poorly constrained degrees of freedom include pitch, roll and z, while drift in x, y and yaw is bounded by the loop closures. Incorporating additional sensor information can help in limiting that drift. Inertial sensors provide complementary information: by providing an estimate for the gravity direction, they eliminate drift in pitch and roll.

Our contribution is a novel algorithm that integrates inertial measurements directly into the map. Since the ElasticFusion algorithm is pose-free, conventional inertial fusion algorithms cannot be applied. Instead, we fuse inertial measurements inside the map. We then use a mesh deformation technique, used in ElasticFusion for loop closing, to enforce a consistent gravity direction over the entire map, and thereby eliminating much of the drift from the map.

II. Related Work

Various dense mapping techniques have been proposed over the past several years. One approach, BundleFusion by Dai et al. [2], uses bundle adjustment for global frame alignment combined with dense map generation, making it very accurate but computationally expensive. Probabilistic formulations account for estimation uncertainty and include REMODE by Pizzoli et al. [14] as well as planar mapping

[1]P. Puri and M. Kaess are with the Robotics Institute, Carnegie Mellon University, Pittsburgh, PA 15213, USA `puneetp@andrew.cmu.edu, kaess@cmu.edu`

[2]D. Jia is with the Language Technology Institute, Carnegie Mellon University, Pittsburgh, PA 15213, USA `daoyuanj@cs.cmu.edu`

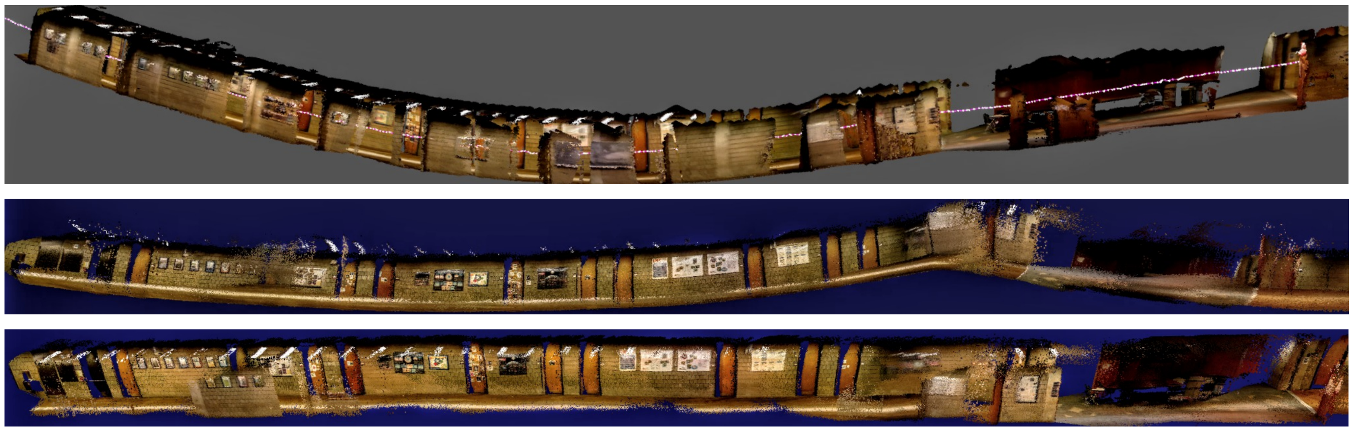

Fig. 2. Dataset of moving the camera through a long corridor. (Top) Kintinuous/RGBD SLAM [18] generates a map that bends. (Middle) ElasticFusion [17] accumulates drift and loses tracking. (Bottom) Our approach, GravityFusion, frequently deforms the model using the measured gravity direction to produce a consistent model.

approaches such as [5] that use infinite planes in a graph optimization to model planar surfaces. Another set of approaches are based on an implicit surface representation in the form of a signed distance function, a volumetric representation that allows fusion of multiple camera frames. The first real-time system is KinectFusion by Newcombe et al. [13], which was later extended in Kintinuous to large scale environments by Whelan et al. [18]. Another common approach called point-based fusion by Keller et al. [7] fuses sensor information into small surface patches with orientation and scale that are called surfels. ElasticFusion by Whelan et al. [17] combines point-based fusion with mesh deformation for large scale loop closure. Because it allows for reintegration of the model after loop closure, we use ElasticFusion as the basis for our inertial mapping system.

Inertial sensor measurements are often integrated into mapping in the context of feature-based methods, some examples follow. Indelman et al. [6] provide a smoothing-based solution to IMU integration with other sensors in a factor graph framework. Forster et al. [3] improve upon the pre-integration method used for integrating the inertial data into the SLAM factor graph. Leutenegger et al. [9] also provide an integrated SLAM solution based on smoothing. Mur-Artal and Tardos [12] integrate the inertial measurements during tracking over a local map.

Inertial sensing integrated with dense mapping is less common to find in the literature. One way is to use a feature-based inertial integration, such as Mur-Artal and Tardos [12], to recover the camera trajectory, and then apply a separate dense mapping system, such as Mur-Artal and Tardos [11]. Concha et al. [1] present a direct visual SLAM method fused with inertial measurements. Usenko et al. [16] recently presented a dense visual inertial odometry method, however, by only using odometry it is not possible to incorporate loop closures for consistent large scale mapping. Ma et al. [10] present a KinectFusion-based inertial fusion, however, without the capability to close loops as a deformation of the dense volumetric representation is computationally expensive. To retain the advantages of fusion while allowing loop closures, we build on point-based fusion, and in particular the ElasticFusion algorithm.

While all of the above methods incorporate inertial sensor data at the level of camera motion, we present a novel approach that directly fuses them with the map. In our GravityFusion algorithm, correction of the map proceeds in much the same way as loop closing in ElasticFusion, using a deformation graph [15] to perform mesh deformation.

III. Approach

GravityFusion corrects accumulated drift in the map using inertial information embedded in surfels. As the camera moves, the system builds a map of surfels similar to ElasticFusion. However, instead of just fusing RGB-D data from multiple frames to create a surfel, GravityFusion also includes orientation information from an inertial sensor at the time of frame capture. This is done by including the measured direction of earth's gravity into the surfel that is being created or updated. Similar to updating position, orientation, and color of a surfel in ElasticFusion, a weighted average is used to fuse gravity vectors of a surfel measured at different times. The map created as a result has measured gravity vectors embedded in every surfel.

While the mapping is being done, the GravityFusion system triggers gravity vector realignment after a certain number of frames to ensure that the map's drift is corrected. This correction is done using a mesh deformation graph, similar to ElasticFusion. The mesh nodes are sampled surfels of the map. These nodes, like surfels, contain gravity information. The nodes of the deformation graph are reoriented to align the gravity vectors to a common direction, while also taking model constraints into account.

Subsequently, all surfels that are not part of the deformation graph are corrected based on neighboring deformation graph nodes. The drift correction of a node is propagated to its neighbouring surfels which reorients and repositions them to ensure correct alignment. The extent of correction is based on vicinity to a correcting graph node. It is possible that multiple nodes affect a surfel and contribute to its reorientation and repositioning. Since these corrections are done for all surfels throughout the map, in the end we get a map which is drift corrected.

Frequent gravity-based map corrections are performed in real-time. While ElasticFusion applies graph deformation only at loop closure, we regularly perform a mesh deformation to incorporate gravity measurements and eliminate drift. To allow for real-time processing, the gravity correction is added to the GPU-based loop closure algorithm in ElasticFusion. Consequently, a good map is always available in our pose-graph-free approach, without the need to maintain past states or deal with issues often related to the marginalization of previous state information.

IV. Map Creation

The mapping process in GravityFusion resembles ElasticFusion's with changes in camera tracking method and map data structure. The map is represented as an unordered surfel list similar to Whelan et al. [17] and Keller et al. [7]. A surfel S consists of gravity vector $\mathbf{g} \in \mathbb{R}^3$, position $\mathbf{v} \in \mathbb{R}^3$, normal $\mathbf{n} \in \mathbb{R}^3$, color $\mathbf{c} \in \mathbb{N}^3$, confidence $\lambda \in \mathbb{R}$, radius $r \in \mathbb{R}$, initialization timestamp t_0 and last updated timestamp t. Sections A and B explain how inertial information is integrated into the tracking and mapping systems.

A. Tracking

Our camera tracking approach closely resembles ElasticFusion's [17] where combined depth tracking and photometric alignment is initialized with photometric alignment only. Our approach differs as it informs this photometric alignment with an inertial measurement from the IMU rotation, thus leading to a more robust camera pose estimation. We only make use of rotation values from the inertial data because incremental angular motion is available with greater accuracy using standard IMUs.

In the RGB-D incoming frame we define the image domain as $\Omega \subset \mathbb{N}^2$ and the depth map D as the depth of pixels $d\colon \Omega \to \mathbb{R}$. The 3D back-projection of a point $\mathbf{u} \in \Omega$ for a given depth map D is given by $\mathbf{p}(\mathbf{u}, D) = \mathbf{K}^{-1} u d(\mathbf{u})$, where $\mathbf{K}$ is the camera intrinsic matrix and u is the homogeneous form of $\mathbf{u}$. The perspective projection of a 3D point $\mathbf{p} = [x, y, z]^\top$ (in camera frame F_c), is represented as $\mathbf{u} = \pi(\mathbf{Kp})$. Here $\pi(\mathbf{p}) = [x/z, y/z]^\top$ represents the de-homogenization operation. The normal map is computed for every depth map using central differences.

The color image C is represented as $c\colon \Omega \to \mathbb{N}^3$. The color-intensity value of a pixel $\mathbf{u} \in \Omega$, given a color image C with color $c(\mathbf{u}) = [c_1, c_2, c_3]^\top$, is defined as $I(u, C) = (c_1 + c_2 + c_3)/3$. The global pose of the camera $\mathbf{P}_t$ (in global frame F_g) is determined by live registration of a depth map and a color frame with the model. The pose estimation iteratively minimizes tracking error $E_{tracking}$ as a weighted sum of geometric error E_{icp} and photometric error E_{rgb}. These errors are calculated between the raw live depth map D_t from the sensor and the active model's depth map using the last frame, $\hat{D}_{t-1}$, in order to find the optimal motion parameter ξ, where $\xi \in \mathbb{R}^6$ and $\exp(\xi) \in SE(3)$. The tracking error is expressed as

$$E_{tracking} = E_{icp} + \omega_{rgb} E_{rgb}, \tag{1}$$

$$E_{icp} = \sum_k ((v^k - exp(\xi)\mathbf{T}.v_t^k).n^k)^2, \tag{2}$$

$$E_{rgb} = \sum_{u \in \Omega} \left(I(u, C_t - I(\pi(K \exp(\xi)\mathbf{T}\mathbf{p}(u, D_t)), C_{t-1}))\right)^2, \tag{3}$$

where ω_{rgb} is the weight of photometric error E_{rgb} over ICP error E_{icp}. Here, v_t^k is the back-projection of the k^{th} vertex in D_t, v^k represents the corresponding vertex in the model, and n^k is the normal of v^k. $\mathbf{T}$ is the current estimation of transformation of camera pose and $exp(\xi)$ is the incremental motion with parameter ξ to be optimized in the current iteration. Similarly the color from the live frame C_t at the current time t and model C_{t-1} is used to find the optimum motion parameter ξ using photometric error—the intensity difference between pixels. For more detailed understanding of the above equations we refer to Whelan et al. [17].

Similar to ElasticFusion, the energy terms from (2) and (3) are jointly optimized to calculate the least squares solution for the optimal ξ to get an accurate camera pose estimate $\mathbf{P}_t = exp(\xi)\mathbf{T}\mathbf{P}_{t-1}$. Unlike ElasticFusion, before the optimization begins, the initial estimate $\mathbf{T}$ is modified to incorporate the measured inertial-orientation $\mathbf{R}_{imu} \in SO(3)$ for a more accurate initialization of $\mathbf{T}$. We call this the $SO(3)$ initialization step.

B. Adding Inertial Information into Surfels

The inertial information is incorporated into the map by embedding gravity direction into every surfel. This is done as the frame is being aligned and fused to the model. To orient the gravity vector into the model frame, a unit vector $[0, 0, 1]^\top$, in model frame, is transformed into the IMU (camera) frame using the inertial readings from the IMU. The inertial data (orientation) is obtained as filtered output of an attitude heading reference system (AHRS) of the IMU, denoted as $\mathbf{R}_{imu}$. We denote this gravity vector in IMU frame as $\mathbf{g}_{imu}^i \in \mathbb{R}^3$ where $i \in \mathbb{R}$ is the surfel's index. We then transform $\mathbf{g}_{imu}^i$ back to the model frame using the tracking estimate of the camera orientation represented as $\mathbf{R}_{tracking} \in SO(3)$. We denote this gravity vector as $\mathbf{g}_{model}^i$. The $\mathbf{g}_{model}^i$ is added into every surfel, providing constraints on two degrees of freedom from the inertial measurements.

$$\mathbf{g}_{imu}^i = (\mathbf{R}_{imu})^{-1} \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^\top \tag{4}$$

$$\mathbf{g}_{model}^i = \mathbf{R}_{tracking} \mathbf{g}_{imu}^i \tag{5}$$

C. Updating Gravity Direction Information

When new frames are fused to the model, surfels are updated for gravity, position, normal and radius according to the confidence of the current and incoming surfels, similar to ElasticFusion. The surfel correspondences are built with map registration of the incoming frame. A confidence value λ is initialized along with the creation of the surfel and accumulates each time we observe the same surfel in the camera frame. The confidence parameter acts as a weight: a measure of the extent to which we should retain the surfel in an update. The update equations for the surfel parameter, given the new incoming surfel's parameters, gravity direction $\mathbf{g}'$, position $\mathbf{v}'$, normal $\mathbf{n}'$ and confidence λ' are:

$$\hat{\mathbf{g}} = \frac{\lambda \cdot \mathbf{g} + \lambda' \cdot \mathbf{g}'}{\lambda + \lambda'} \tag{6}$$

$$\hat{\mathbf{n}} = \frac{\lambda \cdot \mathbf{n} + \lambda' \cdot \mathbf{n}'}{\lambda + \lambda'} \tag{7}$$

$$\hat{\mathbf{v}} = \frac{\lambda \cdot \mathbf{v} + \lambda' \cdot \mathbf{v}'}{\lambda + \lambda'} \tag{8}$$

$$\hat{\lambda} = \lambda + \lambda' \tag{9}$$

The updated gravity direction $\mathbf{g}$ of the surfel is a weighted average of the current estimate with the gravity reading of the incoming surfel $\mathbf{g}'$.

V. Map Correction: Gravity based Deformation Graph

To correct the model for drift and tracking inconsistencies we use a deformation graph. The deformation model used here is general enough to be applied to any map and still provides intuitive manipulation while preserving surface consistency. This formulation allows stretching and realignment of the map while maintaining surface continuity. The nodes of the deformation graph are created by sampling existing surfels from a full model. As a result, the graph created is a sparse model of the original map. The deformation of this graph is a set of affine transformations of graph nodes providing spatial reorganization. These affine transformation influence the neighboring nodes, having an overlapping effect.

Our deformation graph is similar to that in Sumner et al. [15] and Whelan et al. [17] with an essential addition that each node contains inertial information in the form of gravity direction. A single graph node contains a gravity vector $\mathbf{G}_g \in \mathbb{R}^3$, its position $\mathbf{G}_p \in \mathbb{R}^3$ and a timestamp. The graph node also stores an affine transformation as rotation $\mathbf{G}_R \in SO(3)$ and translation $\mathbf{G}_t \in \mathbb{R}^3$ to be applied to itself, while also influencing surfels and graph nodes in its neighborhood. This affine transformation is initialized as $\mathbf{G}_t = \mathbf{0}$ and $\mathbf{G}_R = I$ at the time of graph creation, and again after completion of graph optimization. We consider the total number of graph nodes to be m and each having at most K neighbors. As explained in below subsections, the influence of graph nodes can be broken down into two stages: firstly, how graph nodes affect surfels in their vicinity, as shown in Fig. 3 and secondly, how graph nodes influence each other during graph optimization for deformation as shown in Fig. 4.

A. Influence of Deformation Graph on Surfels

An affine transformation of a graph node is centered around itself. As a graph node rotates ($\mathbf{G}_R$) and translates ($\mathbf{G}_t$), its influence causes the nearby surfels to spatially reorient. The effect of graph node's affine transformation on a nearby surfel at location $\mathbf{v}$ is shown in Fig. 3. This principle

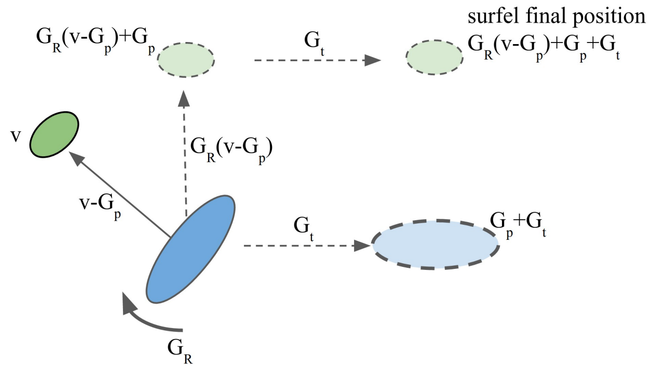

Fig. 3. Influence of affine transformation (G_R,G_t) of a graph node (in green) on a neighboring surfel (in blue) by spatially reorienting the surfel to a new location.

is used throughout the deformation graph to obtain the new location $\hat{\mathbf{v}}$ given by:

$$\hat{\mathbf{v}} = \mathbf{G}_R(\mathbf{v} - \mathbf{G}_p) + \mathbf{G}_p + \mathbf{G}_t \tag{10}$$

The extent of this influence is limited by the distance of surfel i from the graph node j. The influence here is represented as a weight w_j

$$w_j(\mathbf{v}_i) = (1 - ||\mathbf{v}_i - \mathbf{G}_p^j||/d_{max}), \tag{11}$$

where d_{max} is the distance of the surfel to the K_{th} nearest graph node. To smoothly blend the effect of multiple graph nodes on a surfel, we sum over the combined influence. The resultant final position of the surfel is written as

$$\hat{\mathbf{v}}_i = \sum_{j=1}^{m} w_j(\mathbf{v}_i)\mathbf{G}_R^j(\mathbf{v}_i - \mathbf{G}_p^j) + \mathbf{G}_p^j + \mathbf{G}_t^j \tag{12}$$

Any rotation applied to a surfel effect the orientation of its normal $\mathbf{n}_i$. Similar to the position update, the combined influence of graph node's rotation on the surfel's orientation is expressed as

$$\hat{\mathbf{n}}_\mathbf{i} = \sum_{j=1}^{m} w_j(\mathbf{v}_i)(\mathbf{G}_R^j)^{-1}\mathbf{n}_\mathbf{i} \tag{13}$$

As surfel and its normal orientation get updated, a similar update is applied to the gravity direction associated with that surfel. This update ensures that after deformation the gravity direction remains consistent to the surfel's orientation.

$$\hat{\mathbf{g}}_i = \sum_{j=1}^{m} w_j(\mathbf{v}_i)(\mathbf{G}_R^j)^{-1}\mathbf{g}_i \tag{14}$$

We sample the graph nodes densely, ensuring that graph nodes have an accurate representation of the entire map. This dense sampling assures that when deformation is applied, its influence reaches all desired surfels in the map.

B. Optimization of Deformation Graph

We optimize the deformation graph to find the affine transformation of graph nodes, which when applied to our model will correct its drift and make it consistent. This optimization utilizes the direction of gravity vectors in the

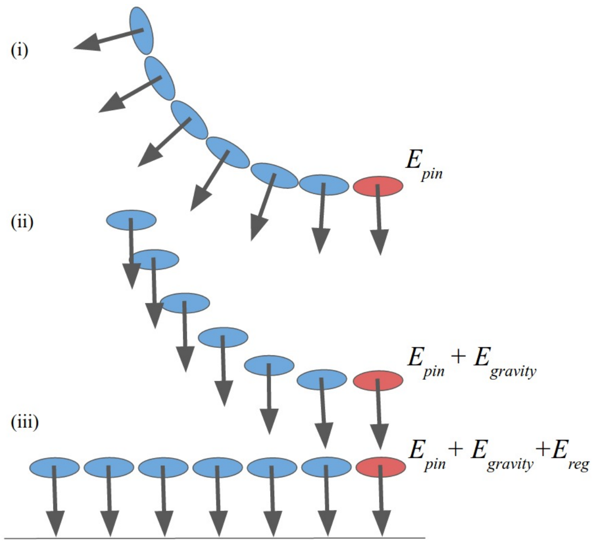

Fig. 4. We highlight here the effect of each energy term on the spatial orientation of the graph nodes in deformation graph. The top image (i) shows constraint energy term of the type pin constraint E_{pin}. The E_{pin} freezes the graph node in its location (in red), acting as a standard reference. The middle figure (ii) shows the effect of the realignment of the gravity vectors caused by the additional $E_{gravity}$ term. The bottom figure (iii) shows the effect of inclusion of regularization E_{reg} term. The E_{reg} causing the graph nodes to remain consistent with each other, enforcing a surface geometry, and keeping the model consistent.

graph nodes to inform the deformation. In an ideal case where there is no drift, all of the gravity vectors on each of the graph nodes will be parallel to each other. However, since the model accumulates drift over time, the nodes' gravity vectors become misaligned as shown in Fig. 4 (i), the optimization explained below penalizes this inconsistent orientation. The graph optimization finds the unknown variables which describe a corrective rotation and translation of every graph node. These rotations and translations, when applied back to graph nodes and in turn onto surfels, addresses the drift by correcting the model.

The optimization of the deformation graph is done using minimization of the combined cost of four components:

1) Constraints: The constraints cost term E_{con}, ensures that these graph nodes are either allowed to move or to remain stationary. A single constraint on a graph node l is a tuple containing its transformed source location as $\phi(\mathbf{G}^l_{p(source)})$ and its destination location as $\mathbf{G}^l_{p(dest)}$. These constraints are of three types: A) Pin constraint: They freeze graph nodes at their position, making them unable to rotate or translate during graph deformation. This constraint type is expressed in (15) as $E_{pin} = E_{con}$. Here the graph node's source and destination location are kept identical. B) Generic constraint: The graph nodes having a generic constraint, spatially re-locate from their source location to their destination location. Generic constraints are used for loop closures as they deform the map from its current location (source) to a similar previously visited location (destination). The system ensures that graph nodes at destination location remain fixed and do not snap towards source location by adding a pin constraint on destination graph node. C) Relative constraint: This type of constraint is similar to a generic constraint with the exception that neither source graph node nor destination graph node are constrained via a pin constraint. The relative constraint is required in to ensure that previous loop closures remain consistent and are not torn off due to newly added generic constraints. Out of the three constraint types, our approach uses pin constraints actively to freeze certain graph nodes, locking their gravity vectors in place to serve as a reference.

$$E_{con} = \sum_l \left\|\phi(\mathbf{G}^l_{p(source)}) - \mathbf{G}^l_{p(dest)}\right\|_2^2 \qquad (15)$$

2) Gravity Alignment: In the process of map creation, the model accumulates drift due to small errors in tracking and the gravity vectors diverge as shown in Fig. 4 (i). These vectors in a drifted model are not entirely parallel. The gravity alignment cost term E_{grav} penalizes this incorrect alignment and minimizes the alignment error by realigning gravity vectors to a standard reference. The standard reference is the direction of gravity vector $\mathbf{G}^k_g$, from a graph node k with a pin constraint, shown in red in Fig. 4. This standard reference graph node remains frozen in its orientation and location and is usually sampled from the recently acquired parts of the model. The graph optimization results in a corrective rotation $\mathbf{G}^j_R$ required for every graph node j in the model, such that gravity vectors of these nodes become parallel as shown in (ii) of Fig. 4.

$$E_{grav} = \sum_{\substack{j=1 \\ j\neq k}}^{m} \left\|(\mathbf{G}^j_R \mathbf{G}^j_g)^\top . \mathbf{G}^k_g - 1\right\|_2^2 \qquad (16)$$

3) Regularization: The regularization term is added to the graph optimization to ensure that the model remains consistent and the surface geometries are enforced. To highlight this, Fig. 4 (ii) shows the effect of inconsistent geometry while not using regularization, and (iii) shows a more consistent geometry with the inclusion of regularization term. The distortion free map deformation is achieved by ensuring overlapping influence of affine transformation from neighbouring graph nodes to be consistent to one another. The regularization term E_{reg} is described in (17), where graph node j influences affine transformation of graph node k, with k one of the neighboring nodes $\mathcal{N}(j)$. The cost E_{reg} is calculated as a sum of squared difference between the position of graph node k predicted by influence from graph node j and the position of graph node k when its affine transformation is applied to itself. The weight α_{jk} is proportional to the degree of which the influence of nodes j and k overlap. For more details, we refer the reader to Sumner et al. [15].

$$E_{reg} = \sum_{j=1}^{m} \sum_{k\in\mathcal{N}(j)} \alpha_{jk} \left\|\mathbf{G}^j_R(\mathbf{G}^k_p - \mathbf{G}^j_p) + \mathbf{G}^j_p + \mathbf{G}^j_t - (\mathbf{G}^k_p + \mathbf{G}^k_t)\right\|_2^2 \qquad (17)$$

4) Rotation: Lastly we want to ensure that the optimized variables are orthonormal to form a valid rotation matrix,

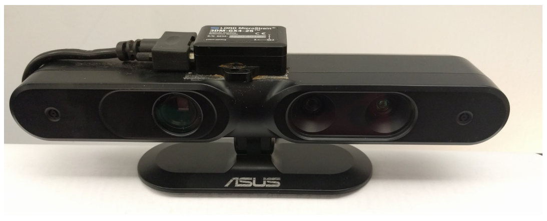
Fig. 5. Our camera setup: ASUS Xtion Pro Live with Microstrain 3DM-GX4-25.

which is enforced by

$$E_{rot} = \sum_{j=1}^{m} \left\|(\mathbf{G}_R^j)^\top(\mathbf{G}_R^j) - I\right\|_2^2 \tag{18}$$

The complete graph deformation is based on optimization of a combined sum of all the components explained above. The weight coefficients control the effect of each energy term:

$$\min_{\substack{\mathbf{G}_R^1..\mathbf{G}_R^m, \\ \mathbf{G}_t^1..\mathbf{G}_t^m}} \omega_{con}E_{con} + \omega_{reg}E_{reg} + \omega_{rot}E_{rot} + \omega_{grav}E_{grav} \tag{19}$$

In our experiments, we set $\omega_{rot} = 1$, $\omega_{reg} = 10$, $\omega_{con} = 100$, and $\omega_{grav} = 100$. We minimize the weighted energy term w.r.t each graph node's rotation $\mathbf{G}_R^j, j \leq m$ and translation $\mathbf{G}_t^j, j \leq m$ using an iterative Gauss-Newton method. The rotation and translation calculated are used in (12), (13), and (14) to align the model for correction.

C. Triggering Graph Optimization

We trigger graph optimization to correct for drift after a fixed interval of frames (C_t=500). This frequent graph optimization ensures that gravity constraints are enforced even if no loop closure occurs. In the event of local and global loop closure, we trigger graph optimization similar to ElasticFusion. For details on local and global loop closure we refer to Whelan et al. [17]

VI. Results

We evaluate the performance of our system qualitatively using data from different real-world environments such as hallways and office alleys. We also present a quantitative surface evaluation on a public dataset in subsection B.

A. Evaluation on Real-world Data

Our hardware setup is a combination of an RGB-D camera (ASUS Xtion Pro Live) and an IMU (Microstrain 3DM-GX4-25) as shown in Fig. 5. We use filtered output of inertial readings from a Microstrain 3DM-GX4-25 sensor.

We focus qualitative testing on long corridors instead of small room environments. In long hallway type environment often dense methods without pose graph fail, as slow drift in map registration causes bending, or lack of features cause loss of tracking due to incorrect photometric alignment. We showcase our system's versatility as it performs well in various environments such as: a) Long corridor with features b) Corridors with fewer features c) Corridors with loops and d) Office desk environment.

1) Experiment for Drift Correction: We capture data through a long corridor which has significant features. Fig. 2 shows the comparison among models generated from various systems. The Kintinuous/RGB-D SLAM [18] in Fig. 2 (top) tracks successfully, however, due to drift accumulation, there is a significant bend in the corridor map. The ElasticFusion in Fig. 2 (center) suffers both from drift accumulation and loss of tracking. Our system in Fig. 2 (bottom) creates a consistent map, generating a straight corridor.

2) Experiment for Tracking Correction: For qualitative evaluation of tracking, we do a 4-way comparison where the IMU based $SO(3)$ initialization of camera tracking is added to ElasticFusion, as described in the tracking section IV.A. In Fig. 6, the first image (from top) shows tracking failure of ElasticFusion due to the incorrect photometric alignment. The second image shows the map created from ElasticFusion (with IMU based tracking initialization). Despite the initialization, the system converges incorrectly and loses tracking. The third image shows the map from GravityFusion (without IMU based tracking initialization). The map deformation is performed using gravity information and the model recovers despite tracking loss, though some misalignment of the corridor remains and is visible as an artifact in the center. The fourth image is of our system GravityFusion (with IMU based tracking initialization). Here our system mitigates tracking loss and creates a consistent model with no visible artifacts.

3) Experiment for Loop Closures: To test loop closure performance, we go through an office environment with alleys connected in a circuit. The top image in Fig. 7 shows a map generated using ElasticFusion. ElasticFusion is unable to track correctly and bends the model creating an inconsistent map. The center image shows a map generated from the Kintinuous/RGB-D SLAM system. Near alley corners, the Kintinuous/RGB-D SLAM system is unable to estimate camera rotation correctly, resulting in an inconsistent model. The bottom image is of our system, as it tracks accurately following through corridors and corners without bending. Though we do not trigger a global loop closure, our system triggers multiple local loop closures, creating a more consistent model.

4) Experiment for Office Environments: In this experiment we test our system ensuring that we retain the ElasticFusion-like ability to do mapping with loopy (and zigzag) camera motion. Here we show a cluttered office desk room which is mapped with similar camera motion. We first do loopy (and zigzag) camera movement to capture the desk and then move the camera around through open doors to arrive back at starting location. Our system handles this camera motion well and generates a consistent map, shown in Fig. 8.

B. Evaluation on Simulated Dataset

Since our system performs frequent graph optimization, which deforms and fixes the model, we here verify that the resulting map does not have any undesired bends or artifacts. We evaluate our system on the ICL-NUIM dataset by Handa

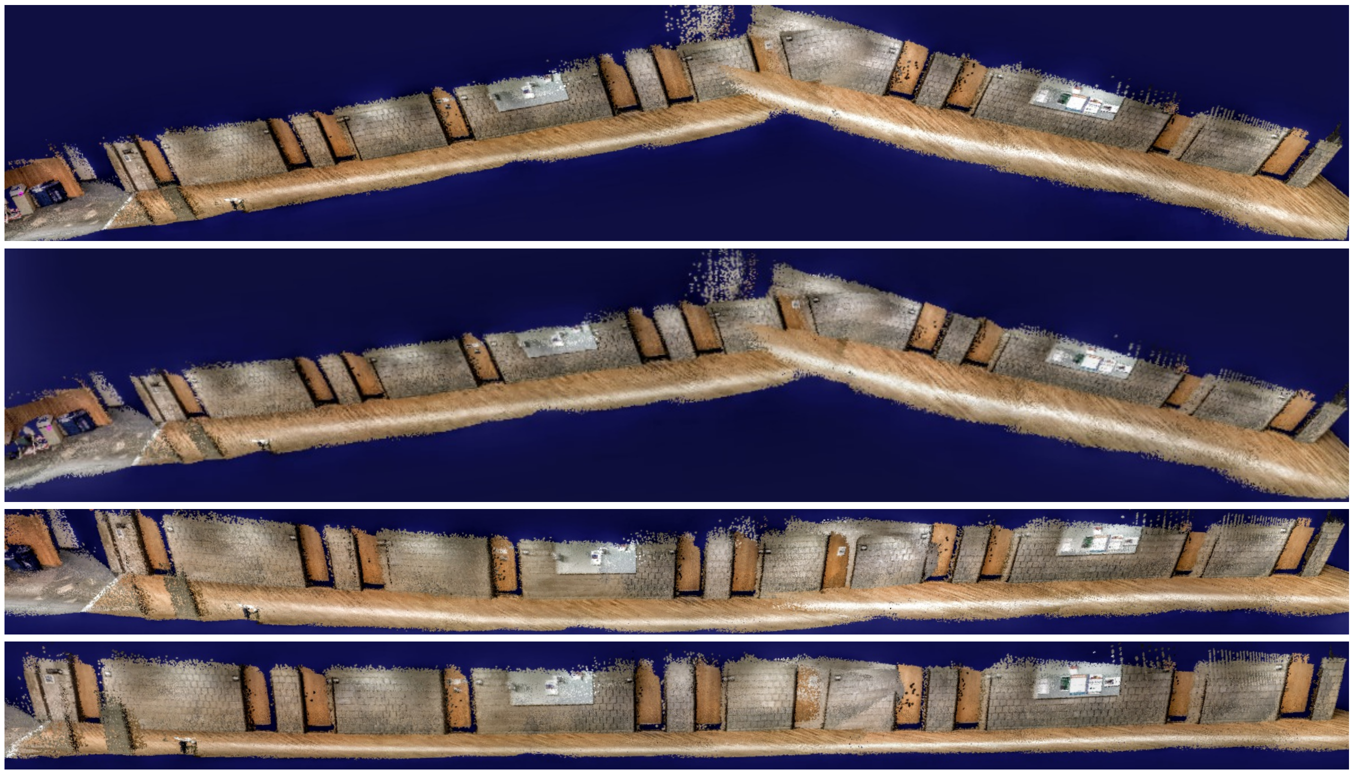
Fig. 6. Comparison of camera tracking loss and recovery on a corridor dataset for the following systems, from the top: 1) ElasticFusion, loses tracking. 2) ElasticFusion with tracking initialized with inertial estimate, still loses tracking. 3) Our approach of GravityFusion without inertial initialization of the camera tracking. It recovers the broken model using deformation, however leaves artifacts in middle of the corridor. 4) Our approach of GravityFusion with inertial initialization of the camera tracking. GravityFusion recovers the model using deformation while maintaining map consistency.

System	lr kt0	lr kt1	lr kt2	lr kt3
DVO SLAM [8]	0.032m	0.061m	0.1119m	0.053m
Kintinuous	0.011m	0.008m	0.009m	0.150m
ElasticFusion	**0.007m**	0.007m	**0.006m**	0.028m
GravityFusion	**0.007m**	**0.006m**	**0.006m**	**0.026m**

TABLE I

ICL-NUIM DATASET PERFORMANCE OF VARIOUS SYSTEMS

et al. [4]. We compute the accuracy of the generated map against the 3D surface from the dataset. Since no IMU value is provided in these datasets, we use the camera trajectory's ground-truth to simulate inertial information by adding Gaussian noise (σ=0.5 degrees). We evaluate on all four datasets, and the results of these surface evaluation errors are listed in Table I. Our system not only matches the model accuracy of the compared systems, but we also exceed their accuracy on two of the datasets: lr kt1 and lr kt3.

VII. CONCLUSIONS

We have presented a novel algorithm for correcting a map with inertial sensor data in the absence of a pose graph. Our GravityFusion algorithm fuses inertial measurements of gravity on a per-surfel basis, and enforces global consistency of the gravity direction through a mesh deformation. Our algorithm eliminates drift in pitch and roll.

Questions for future research include how to make full use of the inertial measurements, i.e. what is the equivalent of tightly coupled integration into the map. At the tracking level inertial information could be used not just for initialization, but also as a constraint in the alignment optimization, which could provide helpful constraints in perceptually ambiguous settings.

ACKNOWLEDGMENT

This work was partially supported by the National Science Foundation, award number 1426703.

We would like to thank Vishal Dugar and Rohan Thakker for implementing parts of the SO(3) initialization.

We would also like to thank Thomas Whelan for patiently answering many of our questions on ElasticFusion.

REFERENCES

[1] A. Concha, G. Loianno, V. Kumar, and J. Civera. "Visual-Inertial Direct SLAM". In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*. 2016.

[2] A. Dai, M. Nießner, M. Zollöfer, S. Izadi, and C. Theobalt. "BundleFusion: Real-time Globally Consistent 3D Reconstruction using On-the-fly Surface Re-integration". In: *ACM Transactions on Graphics 2017 (TOG)* (2017).

[3] C. Forster, L. Carlone, F. Dellaert, and D. Scaramuzza. "On-Manifold Preintegration for Real-Time Visual-Inertial Odometry". In: *IEEE Trans. Robotics* (2016).

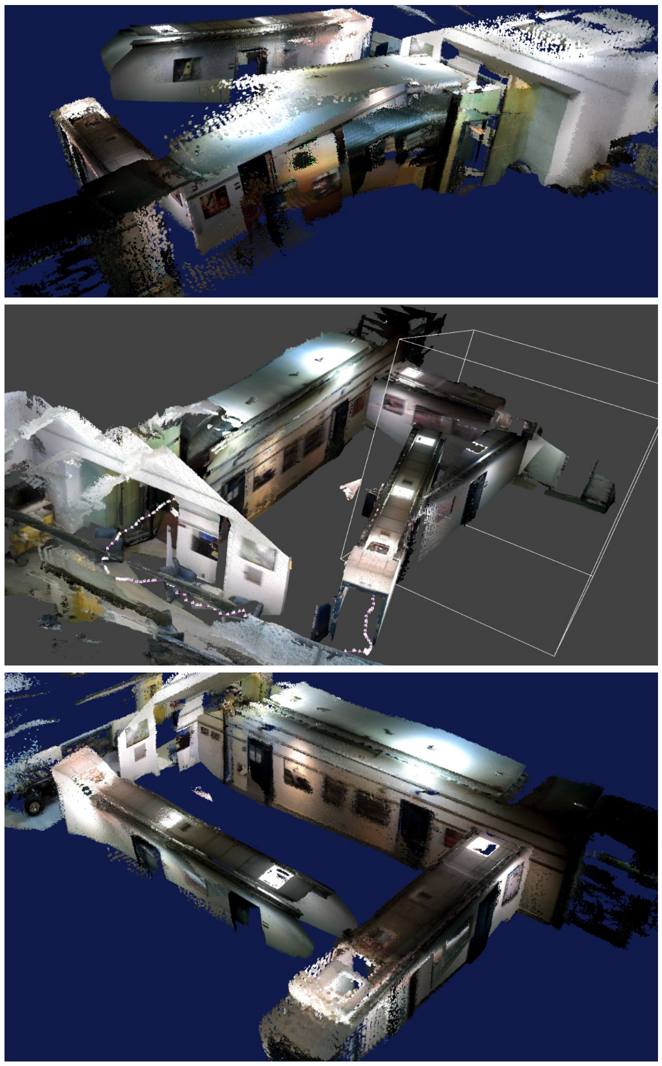

Fig. 7. Map generated from a dataset of office alleys in a circuit by following systems: (Top) ElasticFusion loses tracking. (Middle) Kintinuous incorrectly estimates camera rotation at alley corners, losing tracking. (Bottom) GravityFusion maintains a straight floor profile creating a consistent model.

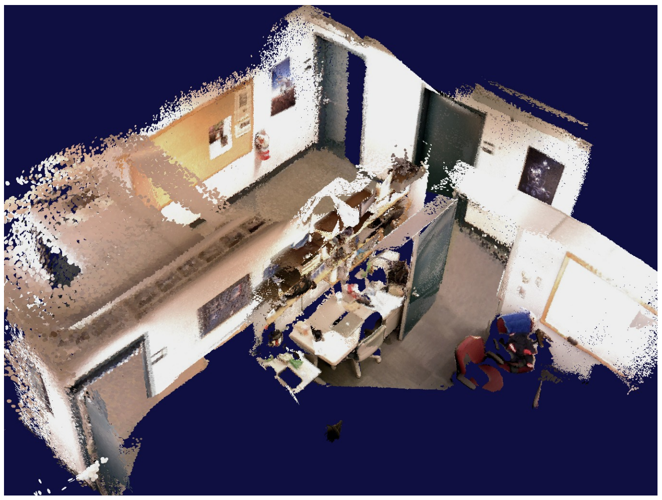

Fig. 8. Map of a cluttered office deskroom using loopy (and zigzag) camera motion.

[4] A. Handa, T. Whelan, J. McDonald, and A. Davison. “A Benchmark for RGB-D Visual Odometry, 3D Reconstruction and SLAM”. In: *IEEE Intl. Conf. on Robotics and Automation, ICRA*. Hong Kong, China, May 2014.

[5] M. Hsiao, E. Westman, G. Zhang, and M. Kaess. “Keyframe-based Dense Planar SLAM”. In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*. To appear. Singapore, May 2017.

[6] V. Indelman, S. Wiliams, M. Kaess, and F. Dellaert. “Information Fusion in Navigation Systems via Factor Graph Based Incremental Smoothing”. In: *Robotics and Autonomous Systems* 61.8 (2013), pp. 721–738.

[7] M. Keller, D. Lefloch, M. Lambers, S. Izadi, T. Weyrich, and A. Kolb. “Real-time 3D Reconstruction in Dynamic Scenes using Point-based Fusion”. In: *Proc. of Joint 3DIM/3DPVT Conference (3DV)*. 2013.

[8] C. Kerl, J. Sturm, and D. Cremers. “Dense Visual SLAM for RGB-D Cameras”. In: *Proc. of the Int. Conf. on Intelligent Robot Systems (IROS)*. 2013.

[9] S. Leutenegger, S. Lynen, M. Bosse, R. Siegwart, and P. Furgale. “Keyframe-based Visual-inertial Odometry using Nonlinear Optimization”. In: *Intl. J. of Robotics Research* 34.3 (2015), pp. 314–334.

[10] L. Ma, J. Falquez, S. McGuire, and G. Sibley. “Large Scale Dense Visual Inertial SLAM”. In: *Field and Service Robotics (FSR)*. 2016, pp. 141–155.

[11] R. Mur-Artal and J. Tardos. “Probabilistic Semi-Dense Mapping from Highly Accurate Feature-Based Monocular SLAM”. In: *Robotics: Science and Systems (RSS)*. July 2015.

[12] R. Mur-Artal and J. Tardos. *Visual-Inertial Monocular SLAM with Map Reuse*. arXiv:1610.05949. 2017.

[13] R. Newcombe, A. Davison, S. Izadi, P. Kohli, O. Hilliges, J. Shotton, D. Molyneaux, S. Hodges, D. Kim, and A. Fitzgibbon. “KinectFusion: Real-Time Dense Surface Mapping and Tracking”. In: *IEEE and ACM Intl. Sym. on Mixed and Augmented Reality (ISMAR)*. Basel, Switzerland, 2011, pp. 127–136.

[14] M. Pizzoli, C. Forster, and D. Scaramuzza. “REMODE: Probabilistic, Monocular Dense Reconstruction in Real Time”. In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*. 2014.

[15] R. Sumner, J. Schmid, and M. Pauly. “Embedded Deformation for Shape Manipulation”. In: *SIGGRAPH*. San Diego, California, 2007.

[16] V. Usenko, J. Engel, J. Stückler, and D. Cremers. “Direct Visual-Inertial Odometry with Stereo Camera”. In: *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*. 2016.

[17] T. Whelan, S. Leutenegger, R. F. Salas-Moreno, B. Glocker, and A. J. Davison. “ElasticFusion: Dense SLAM Without A Pose Graph”. In: *Robotics: Science and Systems (RSS)*. Rome, Italy, July 2015.

[18] T. Whelan, M. Kaess, H. Johannsson, M. Fallon, J. Leonard, and J. McDonald. “Real-time Large Scale Dense RGB-D SLAM with Volumetric Fusion”. In: *Intl. J. of Robotics Research* 34.4-5 (Apr. 2015), pp. 598–626.