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

Puneet Puri

Robotics Institute, School of Computer Science

Carnegie Mellon University

Adviser: Prof. Michael Kaess

June 2017

CMU-RI-TR-17-35

Abstract

In this thesis, 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 sensor 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. We qualitatively show our results in the experimental evaluation using a RGB-D and a stereo camera setup.

Acknowledgements

I would thank my advisor Prof. Michael Kaess for his guidance and support during my Masters journey. His tenacity and patience in our meetings to dig deep into the subject helped me greatly. I would also like to thank my collaborators Daoyuan Jia, Curtis Borium and Prof Nathan Michael and out of group collaborators Vishal Duggar and Rohan Thakker. And finally, I would like to thank the members of my research group with whom discussing and brainstorming in every group meeting help build a better understanding of the SLAM problem.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background

Visual SLAM is the process of creating a map while simultaneously localizing the camera. This process not only creates a representation but also finds the path which camera took as it observes the surroundings to create that map.

Creating an accurate map of real-world is becoming increasingly important as we often want our systems to operate in these surroundings. A good map lends itself to further knowledge extraction like scene understanding, localization, etc. Good quality maps are relevant for mobile robot navigation and manipulation, for augmented and virtual reality applications, and for inspection tasks.

Accurate dense mapping got reignited with the advent of low-cost RGB-D cameras, and now this trend is even driving commodity stereo camera growth. Mapping methods can be broadly classified into sparse and dense. We here look into the dense mapping methods as they offer the possibility of creating a map with details; including complete geometric and color information.

1.2 Problem statement

Most systems rely on only stereo or RGB-D cameras to create dense 3D maps, however inertial sensors are becoming ubiquitous. Combining dense 3D mapping with inertial sensing provides significant advantages over pure vision-based mapping.

In the absence of a global reference such as GPS, mapping algorithms suffer from drift, that 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 is a challenging problem: 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. 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 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 to correct map's drift.

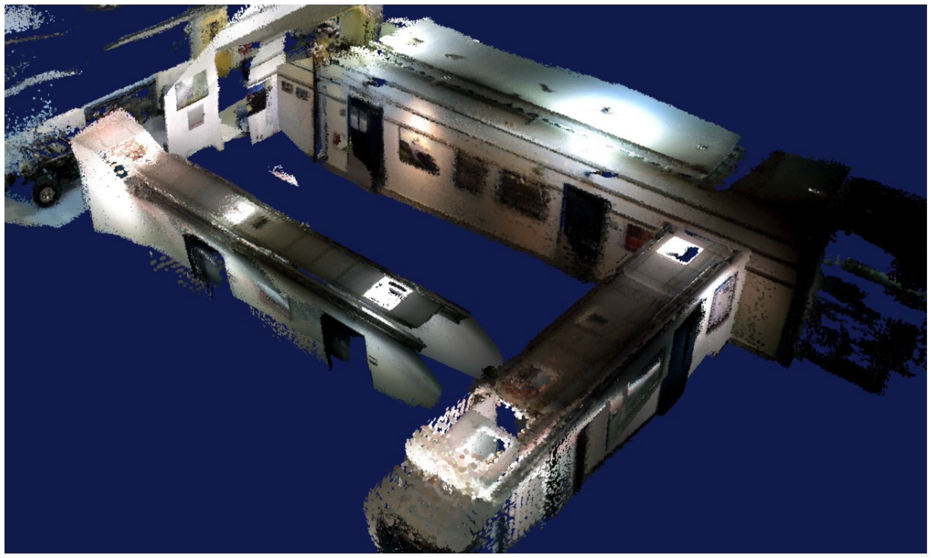


Figure 1.1: Our approach, described in this thesis, produces a consistent model through a low-feature corridor circuit, where other methods fail.

Chapter 2

Related Work

2.1 Brief history of SLAM approaches

Solving SLAM resembles the problem of Bundle Adjustment (in photogrammetry) or Structure from motion (in computer vision) with an important difference; it has an aspect of incremental update during map formation. The earliest approaches such as Smith and Cheeseman [35] solved the SLAM problem using EKF(Extended-Kalman-Filter). The EKF based approaches extended by Davison [6], Castellanos and Tardós [1], and Newman [33] had a limitation that the map creation scaled linearly whereas the co-variance scaled quadratically. Only a few solutions such as Hierarchal SLAM proposed in Estrada et al. [8] bypassed this limitation.

The advent of particle filters, also known as Monte Carlo methods provided a more scalable solution to the EKF. Particle filter approaches such as FastSLAM by Montemerlo et al. [26] overcame the scalability issue by using conditional independence (with Rao-Blackwelization) and a better map management.

The graph-based SLAM methods are where a graph is formed with nodes and edges. The edges are constraints in the form of measurement between variables which are represented as nodes. The graph-based SLAM tackled the issues faced by EKF and particle filters as they could scale well and maintain accuracy. One of the first approach of graph-based SLAM method was Lu and Milios [24] followed by Dellaert [7] Konolige et al. [21] Konolige and Gutmann [20] Kümmerle et al. [22]. The iSAM's factor graph based approach by Kaess et al. [15], Kaess et al. [16] results in a sparse factorization problem which allows incremental matrix optimization, demonstrating that these systems can work at scale.

2.2 Visual SLAM and dense mapping

Visual SLAM uses the camera as a predominant sensor to map surroundings and localize itself. Seminal work on Visual SLAM was done by Davison in his thesis, Davison [5], and as part of Mono-SLAM, Davison [6]. These were capable of capturing a map of sparse landmark features and estimating the pose of the camera in real time. Some of the key contributions were inverse depth parametrization which allowed to handle infinity distances Montiel et al. [27]. Another key milestone in the field of VisualSLAM was PTAM Klein and Murray [19] as it separated the thread for tracking and mapping to run at different rates. The tracking thread was run at camera frame rate, however, the mapping was slowed down to be processed over batches. It also avoided linearization error common with EKF by performing bundle adjustment over a wider array of frames

Davison did some of the earlier work with Dense mapping Newcombe and Davison [30], using the monocular system of Klein and Murray [19], followed by DTAM. DTAM, Newcombe et al. [31], was capable of live dense 3D reconstruction using a monocular camera and used an inverse depth map. The advent of Microsoft Kinect RGB-D camera made the depth sensor easily accessible at a moderate price point, sparking a huge interest in visual SLAM community. Henry et al. [11] and Huang et al. [13] developed one of the earlier systems with Kinect sensor. These approaches used FAST feature correspondence to do visual odometry and made sparse bundle adjustment as a post-processing step to building a dense map. The seminal paper of KinectFusion by Newcombe et al. [32] was capable of real-time 3D reconstruction. It utilized the parallel capability of GPU to run real time and was able to map small spaces such as tables, office cubicles, etc. It used Truncated Sign distance function(TSDF) from Curless and Levoy [3] allowing parallelization and efficient scene representation.

Various dense mapping techniques have been proposed over the past several years . One approach, BundleFusion by Dai et al. [4], uses bundle adjustment for global frame alignment combined with dense map generation, making it very accurate but also computationally expensive. Probabilistic formulations account for estimation uncertainty and include REMODE by Pizzoli et al. [34] as well as planar mapping approaches by Hsiao et al. [12], that use infinite planes in a bundle adjustment setup to model planar surfaces. Another set of approaches is based on an implicit surface representations in the form of a signed distance function, a volumetric representation that allows fusion of multiple camera frames. The KinectFusion by Newcombe et al. [32], was later extended in Kintinuous to large scale environments by Whelan et al. [40]. Another common approach called point-based fusion Keller et al. [17] fuses sensor information into small surface patches with orientation and scale. ElasticFusion by Whelan et al. [39] combines point-based fusion with mesh deformation for large scale 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. [14] provides a smoothing-based solution to IMU integration with other sensors in a factor graph framework. Forster et al. [9] improves upon the pre-integration method used for integrating the inertial data into the SLAM factor graph. Leutenegger et al. [23] also provides an integrated SLAM solution based on smoothing. Mur-Artal and Tardos [29] integrates 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 [29], to recover the camera trajectory, and then apply a separate dense mapping system, such as Mur-Artal and Tardos [28]. Concha et al. [2] presents a direct visual SLAM method fused with inertial measurements. Usenko et al. [38] 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. [25] 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.

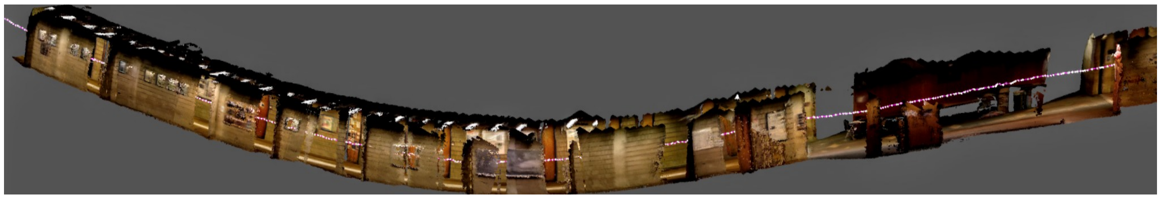
While all of the above methods incorporate inertial sensor data at the level of camera motion, we present a novel approach that directly fuses them with the map. In our GravityFusion algorithm, correction of the map proceeds in much the same way as loop closures in ElasticFusion, using a deformation graph by Sumner et al. [37] to perform mesh deformation.

Chapter 3

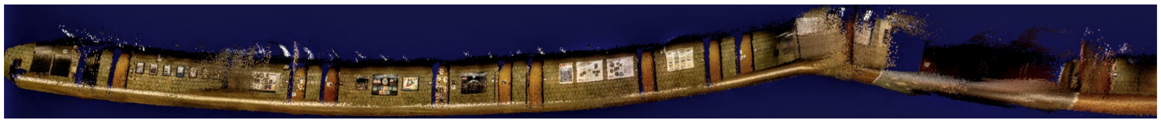
Approach

Our approach, called 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.

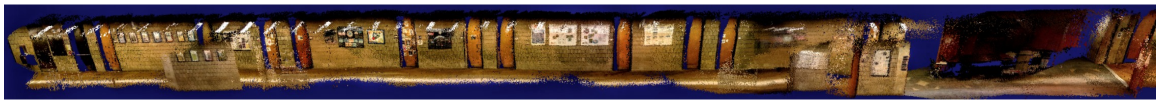
While the mapping is being done, the GravityFusion system triggers gravity vector realignment after a certain number of frames to ensure that the map's drift is corrected. This correction is done using a mesh deformation graph, similar to ElasticFusion. The mesh nodes are sampled surfels of the map. These nodes, like surfels, contain gravity information. The nodes of the deformation graph are reoriented to align the gravity vectors to a common direction. This reorientation of graph nodes results in deformation graph nodes getting realigned. The correction in rotation and position gives the required information to fix the accumulated drift in the map.



(a) Kintinuous/RGBD SLAM [40] generates a map that bends.



(b) Elasticfusion [39] accumulates drift and looses tracking.



(c) Our approach GravityFusion frequently corrects the model using the measured gravity direction to produce a consistent model.

Figure 3.1: Shown is a sequence from walking through a long corridor

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 re-positions 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 a change in its location and orientation. Since these corrections are done for all the sufels 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.

In the following chapters, we describe our approach in detail. In chapter. 4 we discuss the process of map creation and inclusion of inertial information into the map. Following this, in the chapter. 5 we describe the map's drift correction using a deformation graph. We compare our approach to other methods in chapter. 6 and in chapter. 7 we discuss the results obtained using our described method and ElasticFusion system on a stereo camera setup.

Chapter 4

Map Creation

Mapping process in our approach GravityFusion resembles ElasticFusion's with changes in camera tracking method and map datastructure. The map is represented as an unordered surfel list similar to Whelan et al. [39] and Keller et al. [17]. 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. Section 4.1 and subsection 4.1.3, explain the changes in tracking subsystem and how our approach embeds inertial information into the map.

4.1 Camera tracking

Our camera tracking approach closely resembles ElasticFusion's [39] 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.

4.1.1 Geometric and photometric camera pose estimation

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 registering live a depth map and a color frame with the model, where $\mathbf{R}_t \in \mathbb{SO}(3)$ and $\mathbf{t}_t \in \mathbb{R}^3$

$$\mathbf{P}_t = \begin{bmatrix} & \mathbf{R}_t & \mathbf{t}_t & \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.1}$$

Geometric error is calculated between the raw live depth map D_t from the sensor and the active model's depth map using last frame, $\hat{D}_{t-1}$, Here v_t^k is the back projection of the k^{th} vertex in D_t and 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.

$$E_{icp} = \sum_k ((v^k - exp(\xi)\mathbf{T}.v_t^k).n^k)^2, \tag{4.2}$$

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 E_{rgb}; intensity difference between pixels.

$$E_{rgb} = \sum_{u \in \Omega} \left(I(u, C_t - I(\pi(K \exp(\xi) \mathbf{T} \mathbf{p}(u, D_t)), C_{t-1})) \right)^2, \tag{4.3}$$

4.1.2 Joint optimization (without inertial data)

The geometric pose estimation is done by iteratively minimizing tracking error $E_{tracking}$ as a weighted sum of geometric error E_{icp} and photometric error E_{rgb}. These errors are calculated 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,

$$E_{tracking} = E_{icp} + \omega_{rgb} E_{rgb}, \tag{4.4}$$

where ω_{rgb} is the weight of photometric error E_{rgb} over ICP error E_{icp}.The incremental motion parameter, ξ is optimized in current iteration where $\omega \in \mathbb{R}^3$ and $\mathbf{x} \in \mathbb{R}^3$. The cost of system is minimized using non-linear least-squares Gauss Newton . The $\mathbf{J}$ contains the measurement jacobian and $\mathbf{r}$ the residual, where each iteration is solved using Cholesky decomposition similar to ElasticFusion [39]. This results in a optimal ξ that gets a accurate camera pose estimate $\mathbf{P}_t = exp(\xi)\mathbf{T}\mathbf{P}_{t-1}$.

$$argmin \parallel \mathbf{J}\xi + \mathbf{r} \parallel^2 \tag{4.5}$$

$$\xi = \begin{bmatrix} \multicolumn{3}{c}{[w]_\times} & \mathbf{x} \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.6}$$

4.1.3 Bootstrapping with inertial data

ElasticFusion, like most SLAM algorithms, suffers when the quantum of motion between frames is high. Frames with low RGB-D information content are also challenging, since

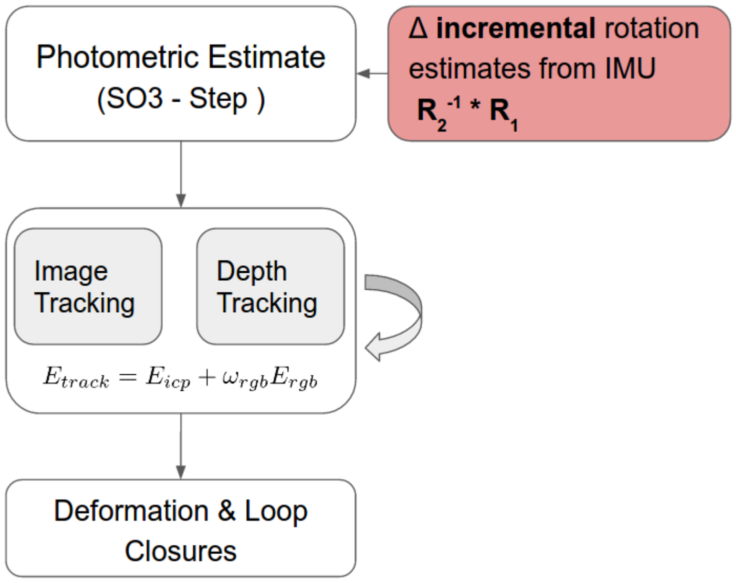

Photometric Estimate
(SO3 - Step)
Δ **incremental** rotation estimates from IMU
$R_2^{-1} * R_1$
Image Tracking
Depth Tracking
$E_{track} = E_{icp} + \omega_{rgb} E_{rgb}$
Deformation & Loop Closures

Figure 4.1: Bootstrapping Optimization with Inertial Data

registration becomes difficult. Specifically, the optimization procedure used to determine the transform matrix between frames relies on an initial point provided by an $SO(3)$ (rotation-only) photometric alignment. If this initial estimate is poor, the subsequent optimization procedure fails to converge to the correct transform.

Using inertial data

The $SO(3)$ alignment step cannot handle large motions or sparse RGB-D frames, and this is precisely the point where we intervene with inertial data. We use inertial measurements to derive the rotation matrix that transforms the current frame to the previous frame, and use this as an initial value for $SO(3)$ alignment. We only make use of rotation values from the inertial data because of two main reasons - firstly, incremental angular motion is available with greater accuracy using standard IMUs, and secondly, angular motion has a much larger impact on the motion between frames than linear motion. Our hypothesis is that reasonably accurate rotation estimate allows the $SO(3)$ alignment procedure to overcome the problem of large inter-frame motion.

Optmizing with inertial data

Similar to ElasticFusion, the energy from both eq. 4.2 and eq. 4.3 are jointly optimized to calculate the least squares solution for optimal ξ to get 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 influenced using incremental rotation data $\mathbf{R}_{imu} \in SO(3)$ from IMU as shown in Fig. 4.1. This results in a more accurate initialization of $\mathbf{T}$. We call this the $SO(3)$ initialization step.

4.2 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 readings(orientation) are obtained from the AHRS of the IMU, denoted as $\mathbf{R}_{imu}$. We denote this gravity vector in IMU frame as $\mathbf{g}^i_{imu} \in \mathbb{R}^3$ where $i \in \mathbb{R}$ is surfel's index. We then transform $\mathbf{g}^i_{imu}$ 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 $\mathbf{g}^i_{model}$. The $\mathbf{g}^i_{model}$ is added into every surfel, providing two degree of freedom inertial information.

$$\mathbf{g}^i_{imu} = (\mathbf{R}_{imu})^{-1} \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^\top \tag{4.7}$$

$$\mathbf{g}^i_{model} = \mathbf{R}_{tracking}\mathbf{g}^i_{imu} \tag{4.8}$$

4.3 Updating gravity direction information

When new frames are fused to the model, surfels are updated for gravity, position, normals and radius according to the confidence of the current and incoming surfels, similar to ElasticFusion. The confidence value is initialized along with the creation of the surfel and accumulates each time we observe the same surfel in the camera frame. The surfel correspondences are built with map registration of the incoming 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 described below. We obtain the updated gravity direction $\mathbf{g}$ of the surfel by a weighted average with the gravity reading of the incoming surfel $\mathbf{g}'$.

$$\mathbf{g} = \frac{\lambda \cdot \mathbf{g} + \lambda' \cdot \mathbf{g}'}{\lambda + \lambda'} \tag{4.9}$$

$$\mathbf{v} = \frac{\lambda \cdot \mathbf{v} + \lambda' \cdot \mathbf{v}'}{\lambda + \lambda'} \tag{4.10}$$

$$\mathbf{n} = \frac{\lambda \cdot \mathbf{n} + \lambda' \cdot \mathbf{n}'}{\lambda + \lambda'} \tag{4.11}$$

$$\lambda = \lambda + \lambda' \tag{4.12}$$

In this chapter we described the process of map creation and embedding inertial information into the map. In next chapter we discuss the process of map correction to address the drift accumulation using a deformation graph.

Chapter 5

Map correction: 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 deformation graph is a mesh made up of nodes connecting its neighbors via edges. The nodes of the deformation graph are created by sampling existing surfels from a full model. As a result, the graph created is a sparser 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. [37] and Whelan et al. [39] 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 and spatially influences 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

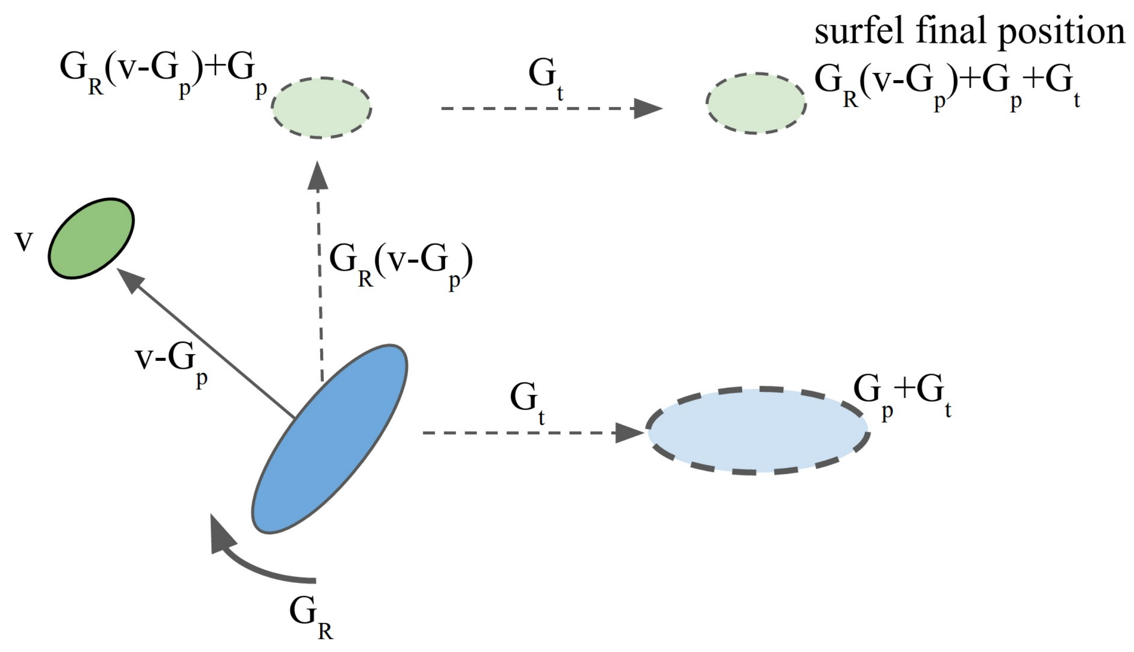


Figure 5.1: Influence of graph nodes on surfels: The surfel (green) is affected by rotation and translation of the graph node (blue).

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. 5.1 and secondly, how graph nodes influence each other during graph optimization for deformation as shown in Fig. 5.2.

5.1 Influence of graph on surfels

An affine transformation of a graph node is centered around itself. As graph node rotates, denoted by $\mathbf{G}_R$, and translates, denoted by $\mathbf{G}_t$, 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. 5.1. This principle is used throughout the deformation graph to obtain the new location $\hat{\mathbf{v}}$ and is given by:

$$\hat{\mathbf{v}} = \mathbf{G}_R(\mathbf{v} - \mathbf{G}_p) + \mathbf{G}_p + \mathbf{G}_t \tag{5.1}$$

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 given by the following equation, where d_{max} is the distance of the surfel to the K_{th} nearest graph node.

$$w_j(\mathbf{v}_i) = (1 - ||\mathbf{v}_i - \mathbf{G}_p^j||/d_{max}) \tag{5.2}$$

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

$$\hat{\mathbf{v}}_i = \sum_{j=1}^{m} w_j(\mathbf{v}_i)\mathbf{G}_R^j(\mathbf{v}_i - \mathbf{G}_p^j) + \mathbf{G}_p^j + \mathbf{G}_t^j \tag{5.3}$$

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

$$\hat{\mathbf{n}}_\mathbf{i} = \sum_{j=1}^{m} w_j(\mathbf{v}_i)(\mathbf{G}_R^j)^{-1}\mathbf{n}_\mathbf{i} \tag{5.4}$$

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.

$$\hat{\mathbf{g}}_i = \sum_{j=1}^{m} w_j(\mathbf{v}_i)(\mathbf{G}_R^j)^{-1}\mathbf{g}_i \tag{5.5}$$

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.

5.2 Deformation graph optimization

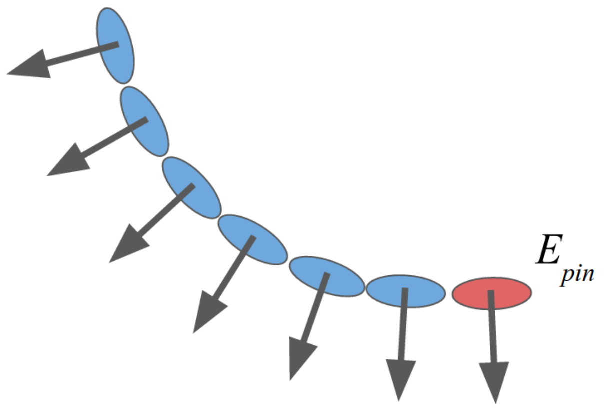
We optimize the deformation graph to find the affine transformation of graph nodes, which when applied to our model will correct its drift and make it consistent. This optimization utilizes the direction of gravity vectors in the graph nodes to inform the deformation. In an ideal case where there is 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, the nodes' gravity vectors become misaligned as shown in Fig. 5.2a, the optimization explained below penalizes this incorrect 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 fixing the model. The optimization of the deformation graph is done using minimization of the combined cost of four components:

5.2.1 Constraints

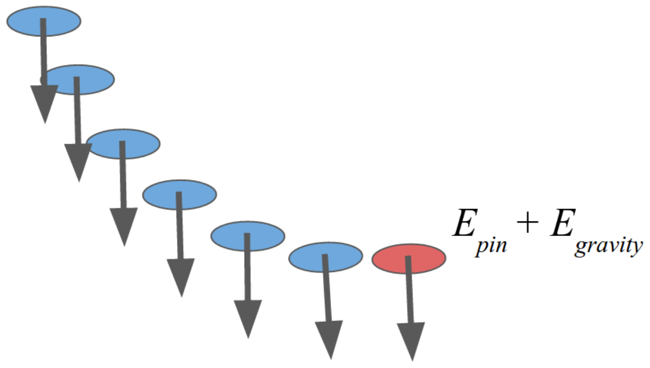
The constraints cost term E_{con}, ensures that these graph nodes are either allowed to move or to remain stationary. A single constraint on a graph node l is a tuple containing its source location, as $\mathbf{G}^l_{p(source)}$, and its destination location, as $\mathbf{G}^l_{p(dest)}$. These constraints 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 eq. 5.6 as $E_{pin} = E_{con}$. Here the graph node's source and destination location are kept identical.

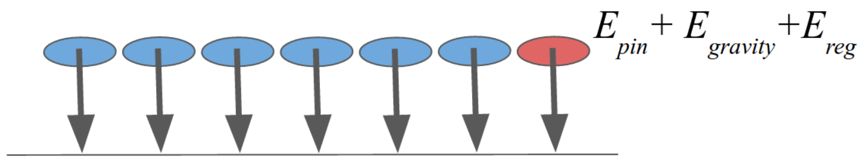
B) Generic constraint: The graph nodes having a generic constraint, spatially re-locate from their source location to their destination location. Generic constraints are used for loop closures as they deform the map from its current location (source) to a similar previously visited location (destination). The system ensures that graph nodes at destination location remain fixed and do not snap towards source location by adding a pin constraint



(a) This 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



(b) Effect of the realignment of the gravity vectors caused by the additional $E_{gravity}$ term, making them parallel.



(c) 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, keeping the model consistent and preventing it from degeneration

Figure 5.2: Affect of different energy terms on the spatial orientation of the graph nodes in the deformation graph.

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 as they freeze the graph nodes, locking their gravity vectors in place and making them as a reference.

$$E_{con} = \sum_{l} \left\| \mathbf{G}^{l}_{p(source)} - \mathbf{G}^{l}_{p(dest)} \right\|_2^2 \tag{5.6}$$

5.2.2 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. 5.2a. 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. 5.2. 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 aligned as shown in Fig. 5.2b.

$$E_{grav} = \sum_{\substack{j=1 \\ j \neq k}}^{m} \left\| (\mathbf{G}^{j}_{R}\mathbf{G}^{j}_{g})^{\top} . \mathbf{G}^{k}_{g} - 1 \right\|_2^2 \tag{5.7}$$

5.2.3 Regularization

The regularization term is added to the graph optimization to ensure that model remains consistent and the surface geometries are enforced. To highlight this, Fig. 5.2b shows the effect of inconsistent geometry without regularization term E_{reg}, and Fig. 5.2c shows a more consistent geometry with the inclusion of the regularization term. The distortion free map deformation is achieved by ensuring overlapping influence of affine transformation from neighboring graph nodes to be consistent to one another. The regularization term E_{reg} is described in eq. 5.8, where graph node j influences affine transformation of graph node k as k is one of its neighboring node given by $\mathcal{N}(j)$. 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. [37].

$$E_{reg} = \sum_{j=1}^{m} \sum_{k \in \mathcal{N}(j)} \alpha_{jk} \left\| \mathbf{G}_R^j (\mathbf{G}_p^k - \mathbf{G}_p^j) + \mathbf{G}_p^j + \mathbf{G}_t^j - (\mathbf{G}_p^k + \mathbf{G}_t^k) \right\|_2^2 \tag{5.8}$$

5.2.4 Rotation

Lastly we want to ensure that the variables optimized are orthonormal to form a valid rotation matrix, which is enforced by

$$E_{rot} = \sum_{j=1}^{m} \left\| (\mathbf{G}_R^j)^\top (\mathbf{G}_R^j) - I \right\|_2^2 \tag{5.9}$$

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:

$$\min_{\substack{\mathbf{G}_R^1..\mathbf{G}_R^m,\\ \mathbf{G}_t^1..\mathbf{G}_t^m}} \omega_{con} E_{con} + \omega_{reg} E_{reg} + \omega_{rot} E_{rot} + \omega_{grav} E_{grav} \tag{5.10}$$

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 iterative Gauss-Newton method. The rotation and translation calculated are used in above eqs. 5.3, 5.4, and 5.5 to align the model for correction.

5.3 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. [39].

In this chapter we described the process of map correction by addressing the drift accumulation using a deformation graph informed by gravity direction. In the next chapter we compare our approach to the other state of the art dense systems.

Chapter 6

Results: RGB-D camera

We discuss the results obtained using our system GravityFusion with the RGB-D camera and an inertial sensor. In the next chapter, ch.7, we discuss the stereo camera based reconstruction using both ElasticFusion and our system GravityFusion.

6.1 Dense reconstruction using RGB-D

We evaluate the performance of our system against other approaches, qualitatively using data from different real-world environments such as hallways and office alleys. We also do the quantitative surface evaluation on public datasets as described in following section 6.2.

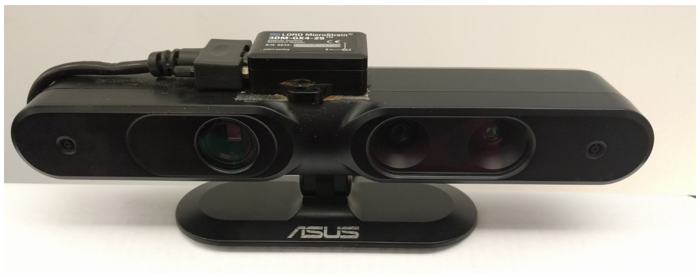
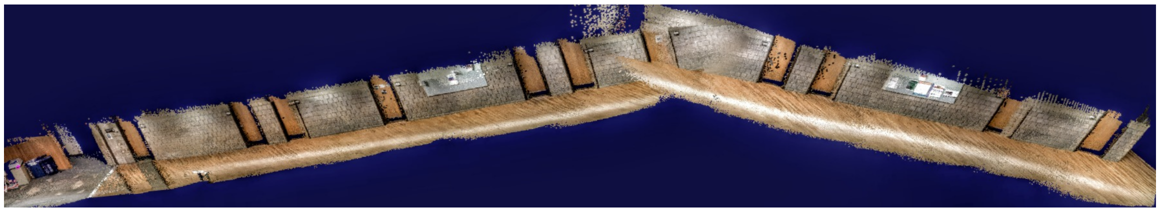
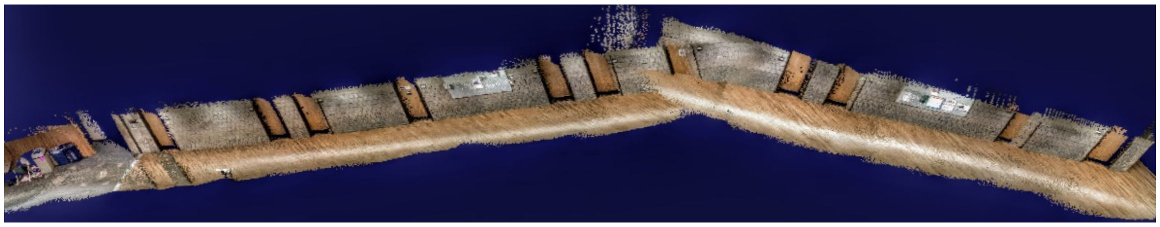
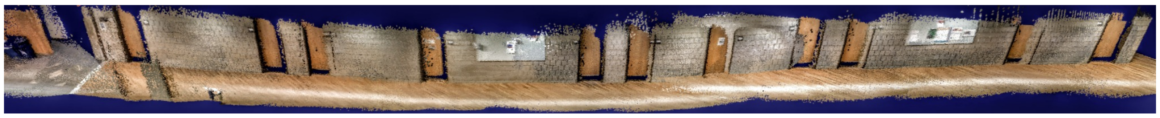


Figure 6.1: Our camera setup: ASUS Xtion Pro Live with Microstrain 3DM-GX4-25


(a) Broken map generated by ElasticFusion[39] due to loss of tracking


(b) ElasticFusion with camera tracking initialized with IMU, yet still, it loses tracking


(c) 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.

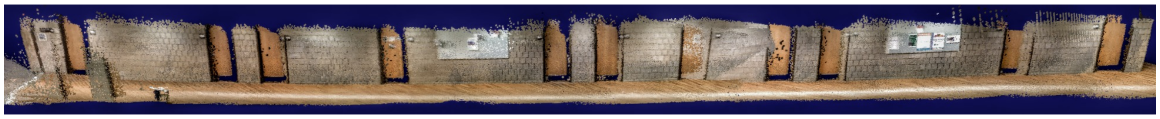

(d) Our approach of GravityFusion with inertial initialization of the camera tracking. GravityFusion recovers the model using deformation while maintaining map consistency.

Figure 6.2: Comparison of camera tracking loss and recovery on a corridor dataset

6.1.1 Hardware

Our hardware setup is a combination of a RGB-D camera (ASUS Xtion Pro Live) and a IMU (Microstrain 3DM-GX4-25) as shown in Fig. 6.1. We use filtered output of inertial readings from Microstrain 3DM-GX4-25 sensor.

6.1.2 Experiment setup

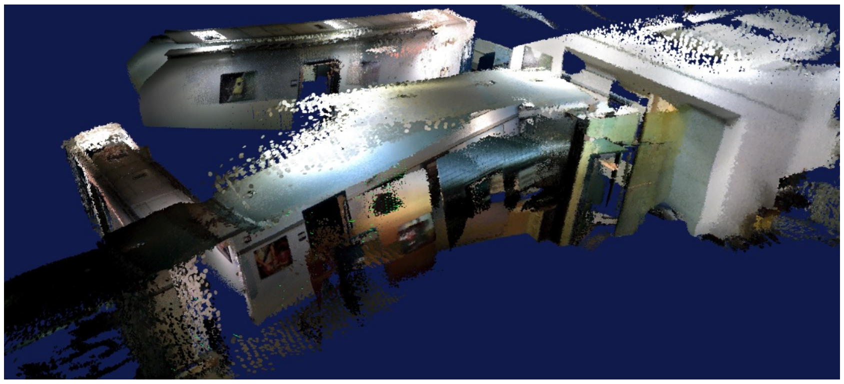
We focus qualitative testing on long corridors instead of small room environments. In long hallway type environment, often dense methods without pose graph fail as slow drift in map registration causes bending, or lack of features cause loss of tracking due to incorrect photometric alignment. We showcase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

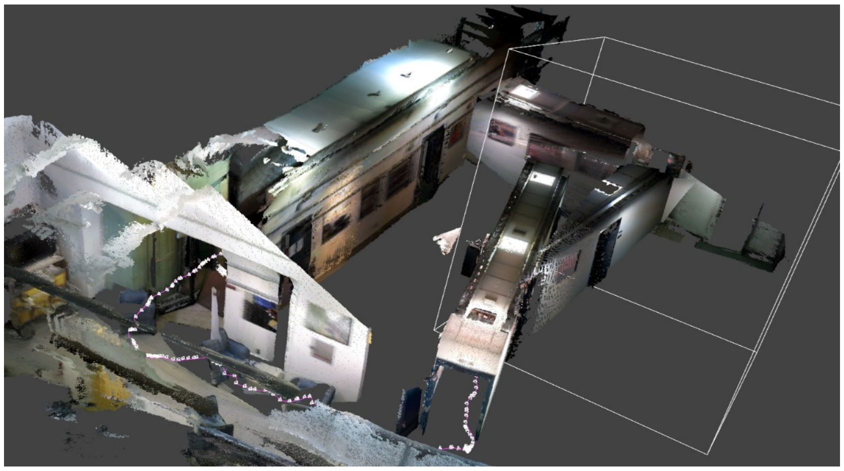
Experiment for drift correction

We capture data through a long corridor which has significant features. The Fig. 3.1 shows the comparison among models generated from various systems. The Kintinuous/RGB-D SLAM[40] in Fig. 3.1a, tracks successfully however due to drift accumulation there is a significant bent in the corridor map. The ElasticFusion in Fig. 3.1b suffers both from drift accumulation and loss of tracking. Our system in Fig. 3.1c maintains a consistent map generating a straight corridor.

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, similar to as described in the tracking 4.1. The Fig.6.2a shows tracking failure of ElasticFusion due to the incorrect photometric alignment. The second image Fig.6.2b shows the map created from ElasticFusion (with IMU based tracking initialization), despite the initialization, the system converges incorrectly and loses tracking. The third image Fig.6.2c 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 Fig.6.2d, 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.


(a) ElasticFusion loses tracking.


(b) Kintinuous incorrectly estimates camera rotation at alley corners, losing tracking.

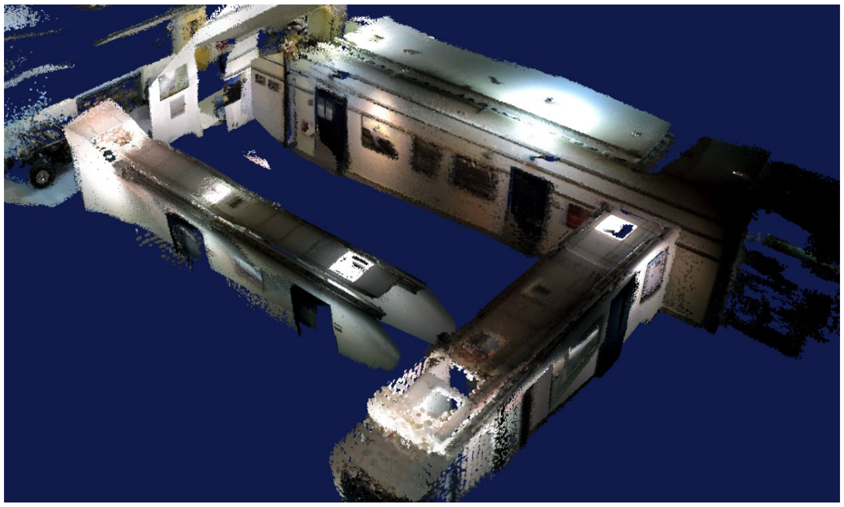

(c) GravityFusion maintains a straight floor profile creating a consistent model.

Figure 6.3: Map generated from a dataset of office alleys in a circuit

Experiment for loop closures

To test loop closure performance, we go through an office environment with alleys connected in a circuit. The Fig.6.3a shows a map generated using ElasticFusion. The ElasticFusion is unable to track correctly and bends the model creating an inconsistent map. The center image Fig.6.3b, shows a map generated from Kintinious/RGB-D SLAM system. The Kintinious/RGB-D SLAM system near alley corners is unable to estimate camera rotation correctly, resulting in an inconsistent model. The bottom image Fig.6.3c, 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 and does creates a more consistent model.

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 zig-zag) camera motion. Here we show a cluttered office desk room which is mapped with similar camera motion. We first do loopy (and zig-zag) camera movement to capture the desk then take the camera around the room through the open doors to arrive back to the point we started. Our system handles this camera motion well and generates a consistent map as shown in Fig. 6.4.

6.2 Evaluation on simulated dataset

Since our system performs frequent graph optimization, which deforms and fixes the model, we here verify that map generated as a result does not have any undesired bends or artifacts. We evaluate our system on ICL-NUIM dataset of Handa et al. [10]. 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 value from ground-truth to simulate the inertial information. We evaluate on all four datasets, and the results of these surface

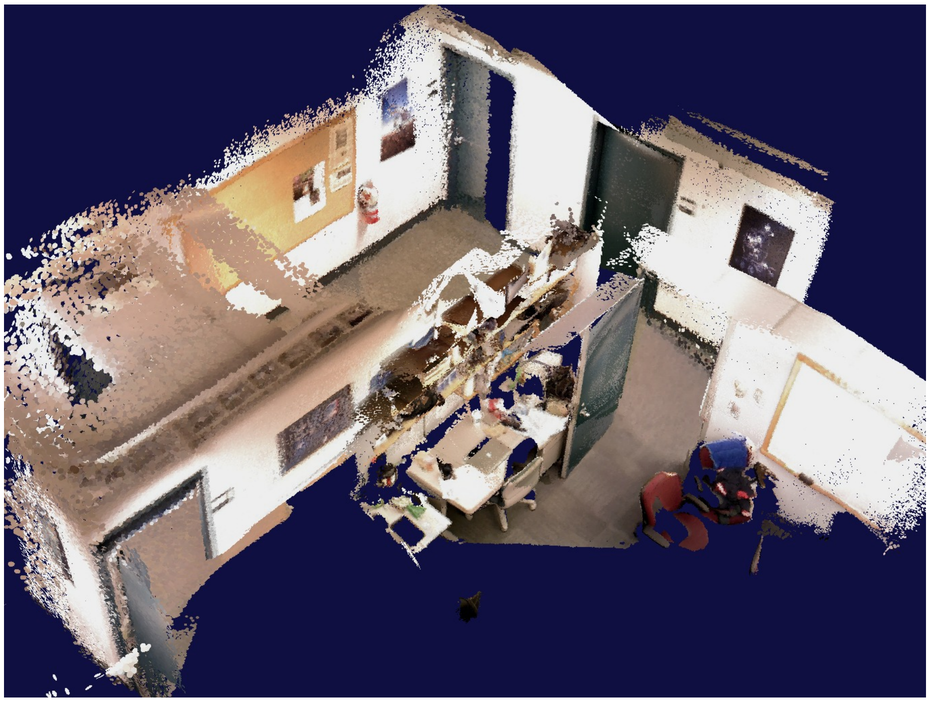


Figure 6.4: Mapping of a cluttered office deskroom using loopy(and zig-zag) camera motion.

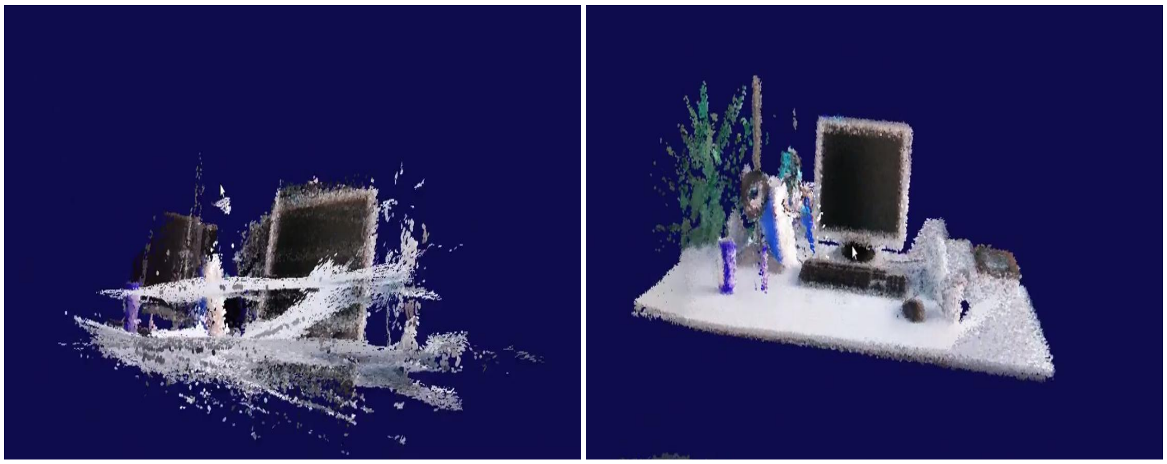
System	lr kt0	lr kt1	lr kt2	lr kt3
DVO SLAM [18]	0.032m	0.061m	0.1119m	0.053m
Kintinuous	0.011m	0.008m	0.009m	0.150m
Elasticfusion	**0.007m**	0.007m	**0.005m**	0.028m
Gravity Fusion	0.007m	**0.005m**	**0.005m**	**0.025m**

Table 6.1: ICL-NUIM dataset performance of various systems

evaluation errors are listed under Table 6.1. Our system not only matches the model accuracy of compared system, but we also exceed their accuracy on two of the datasets: lr kt1 and lr kt3.

6.3 Evaluation of inertial camera tracking

Unlike the previous section, here we wanted to validate our approach of initializing the camera tracking described in the section. 4.1, ensuring it leads to consistent results. In this section, we discuss the results from inertial data being used to initialize the photometric alignment required in camera tracking optimization. We make use of the well-known TUM SLAM dataset Sturm et al. [36] to benchmark our modifications. The dataset provides both RGB-D frames and groundtruth data for the camera's trajectory in a variety of scenarios. We make use of incremental inter-frame motions computed from these ground-truth trajectories as a simulation for an actual IMU. We modified the data-set by only retaining every 40th frame - this increased the average rotation rate from 5.7 deg/sec to a massive 225 deg/sec, a 40x increase. A qualitative comparison of the reconstructed map with this dataset is shown in figure 6.5, where IMU based initializing of camera tracking results in a more consistent model. We also compare the angular errors of IMU based initialization of camera tracking against camera tracking without IMU initialization. We notice that norm of error represented in axis-angle notation, is lesser when using inertial data, this is shown in figure 6.6.



(a) Without inertial initialization (b) With inertial initialization

Figure 6.5: Inertial initialization of camera tracking estimate

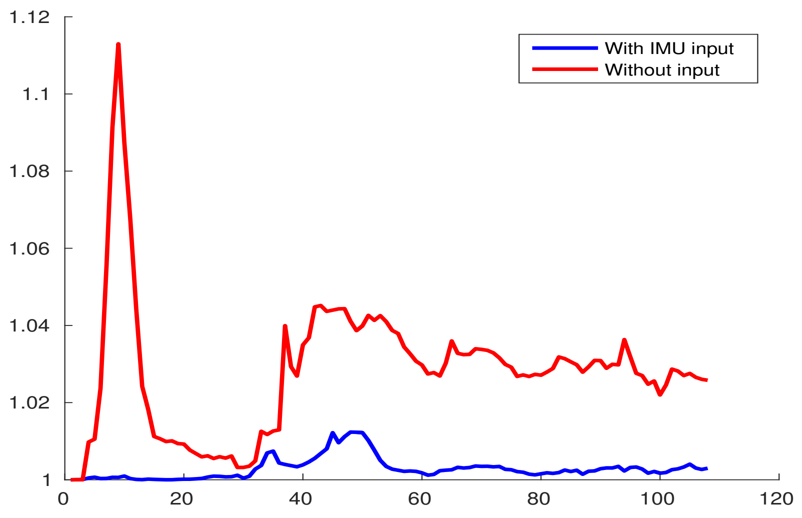


Figure 6.6: Norm of error (in axis-angle notation) w.r.t ground truth

Video links:

Without IMU initialization:`https://www.youtube.com/watch?v=uUvrVxtKLM0`

With IMU initialization:`https://www.youtube.com/watch?v=9DkLUx3MDh4`

In this chapter, we showcased our system compared to the other state of the art dense systems using public datasets and our custom datasets. We also validated our camera tracking initialization using a modified public data set. In the next chapter, we showcase results obtained using a stereo camera using the ElasticFusion and our approach.

Chapter 7

Results: Stereo camera

Here we discuss the qualitative results obtained with dense reconstruction using a stereo camera setup. The ElasticFusion system is designed to be used with the RGB-D camera. This limits the systems use to primarily indoor environments as the infrared projector in RGB-D camera is not intended to work in outdoor lighting. We modify the ElasticFusion toolchain and discuss some of the dense mapping results we achieved with our stereo camera setup. We also show how some of the mapping failures of ElasticFusion are mitigated when doing the same scanning using our system GravityFusion. Our broader problem definition is to obtain high-quality scans of a surface from a close-up distance (<2meters).

7.1 Camera setup

We experimented with multiple cameras listed here to evaluate the reconstruction quality. Dense reconstruction requires stereo camera system with high quality lenses and preferably global shutter functionality, this aids in getting a more accurate disparity match. We below discuss details of the cameras systems and some of there results.

7.1.1 BumbleBee2 camera

The BumbleBee2 stereo camera is setup with a FireWire 1394 card. The system is designed to use a ROS node which controls camera parameters. The camera was set to publish images of 648 x 488 at 30fps (48fps max). Since the camera had high-quality lenses and utilized a global shutter, we were able to generate consistent reconstruction. The map produced were of high-quality, however, as the camera was tethered via FireWire port to a desktop we were restricted to the office environment. We did not use BumbleBee2 in our outdoor field testing.

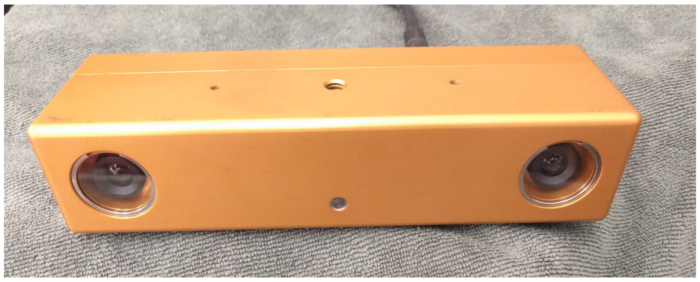


Figure 7.1: Bumbleebe2 stereo camera

7.1.2 Stereo sensor pack V1: Head mounted

This system of the stereo-camera pair was housed in a custom aluminum CNC'ed block along with an inertial sensor as shown in Fig.7.2. The system composed of two global shutter cameras of 752x480 with 70-degree FOV and a VN-100 IMU, which were connected to an onboard computer. This was intended to be a low-power device; hence an odroid-board was used. Due to sensor bandwidth and computation limitations, the frame rate of these cameras was set to 9fps. This system was designed and assembled by Curt Boirum (of FRC-CMU).

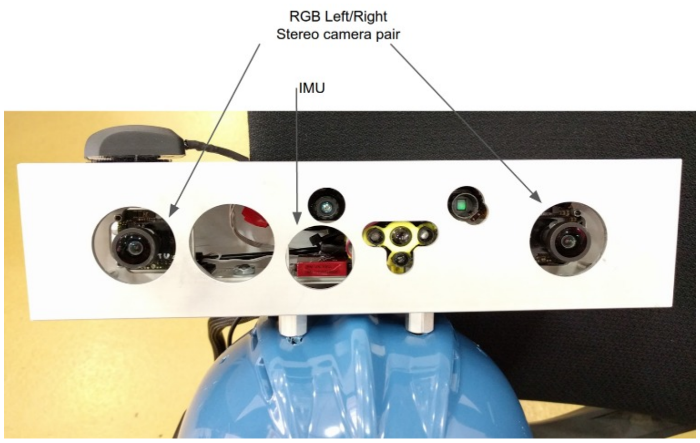


Figure 7.2: Head mounted V1-sensor-pack with stereo camera and IMU

7.1.3 Stereo sensor pack V2: Body mounted

This V2 stereo-camera sensor pack, designed by Curt Boirum, used a body mounting setup as shown in Fig. 7.3. This design allowed for a better weight distribution and hence permitted a stable scanning motion pattern. The system composed of two global shutter cameras of 752x480 with 70-degree FOV and a VN-100 IMU, similar to V1. The system had a significantly more potent on board system using a i7-5600U based single board computer to process frames faster. We achieved 50fps with 752x480 and 20fps with 1280x960 cameras(experimental). Here we discuss results derived using 752x480 resolution frames.

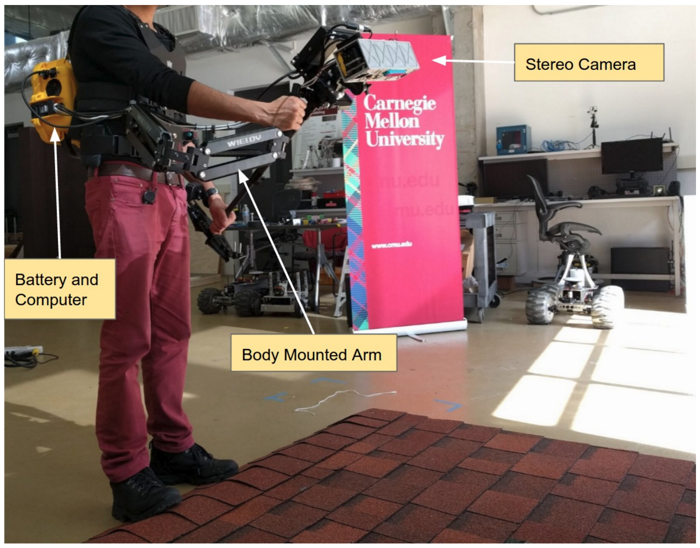


Figure 7.3: Body mounted V2-sensor-pack with stereo camera and IMU

7.1.4 Calibration

We perform the calibration using the ROS stereo camera calibration utility. We ensure calibration is done at a focal length and within the range at which we intend to operate. This results in an accurate calibration and a more true stereo setup. These stereo calibration values for left and right camera : camera intrinsic K, projection matrix P, distortion model parameters D and Rectification matrix R are used in below described stereo processing software stack.

7.2 Stereo software stack

Here we discuss our reconstruction software stack using images from our custom stereo camera. The system makes use of Robot Operating System(ROS) infrastructure to do the pre-processing and disparity generation. The packaging for reconstruction is done using customized software. The complete stack is shown in Fig. 7.4

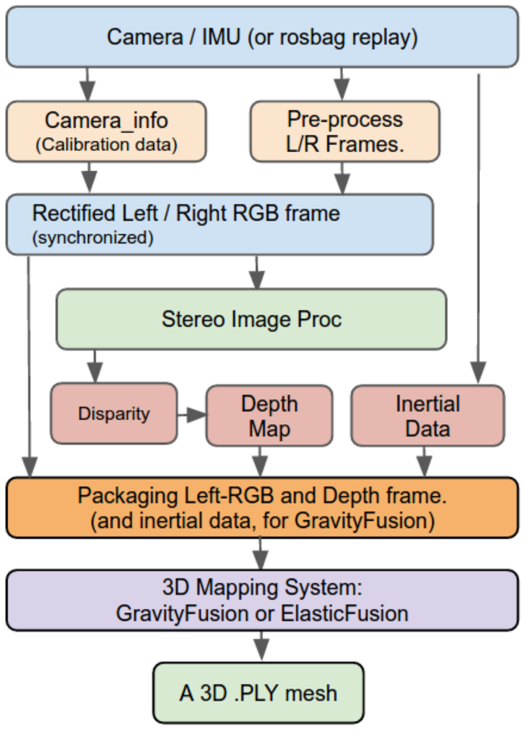

Camera / IMU (or rosbag replay)
Camera_info (Calibration data)
Pre-process L/R Frames.
Rectified Left / Right RGB frame (synchronized)
Stereo Image Proc
Disparity
Depth Map
Inertial Data
Packaging Left-RGB and Depth frame. (and inertial data, for GravityFusion)
3D Mapping System: GravityFusion or ElasticFusion
A 3D .PLY mesh

Figure 7.4: Software stack for 3D Mapping using stereo frames and IMU

7.2.1 Image pre-processing

As part of the image pre-processing we take an input image stream from the stereo RGB camera pair and crop it to 640x480. We also increase the brightness and contrast of the image using $g(i,j) = \alpha.f(i,j) + \beta$ where $f(i,j)$ is the source image, α is the gain (contrast) parameter and β is the bias parameter to add brightness. We perform the image rectification using the ROS-stereo-image-proc's rectification module. Here we publish the camera intrinsic K, Projection matrix P, Distortion model parameters and Rectification matrix R to correct the image as shown in Fig. 7.5

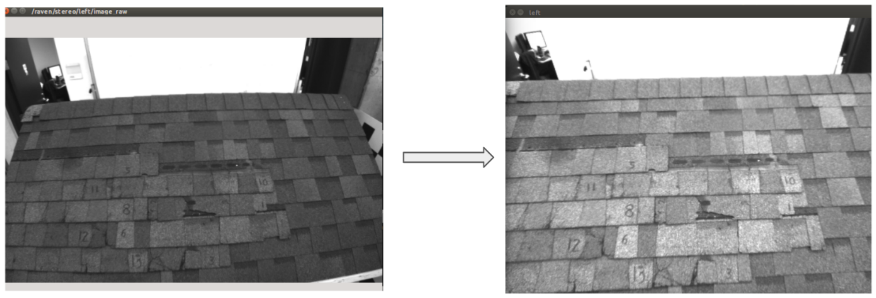


Figure 7.5: Pre-processing and rectification of image frame

7.2.2 Disparity generation

We use the ROS module stereo-image-proc to generate disparity. We utilize the stereo block matching algorithm and the semi-global block matching algorithm for disparity generation. Inorder to get high-quality 3-D reconstruction, a dense disparity frame is required. We get dense disparity generation by setting the hyperparameters such as 'maximum search range' on the epipolar line to be 200(pixels) and the 'minimum disparity' to be beyond 30(pixels). This provides a cut-off threshold for objects further away and avoids any incorrect correlation at the time of generating disparity .

7.2.3 Depth computation

We use the ROS stereo-image-proc module to generate the point cloud frames. These frames are processed to extract the z value from points to generate a depth map. Here we

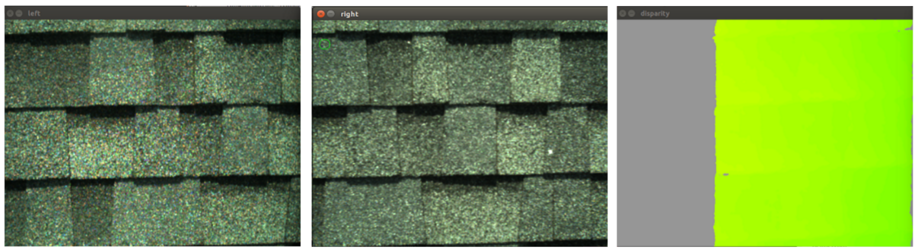


Figure 7.6: Stereo image dense disparity generation

list out the method to calculate depth(in mm) $Z \in \mathbb{R}$ given $d \in \mathbb{R}$ as disparity(pixels), given the focal length(in pixels) as $f \in \mathbb{R}$, and stereo camera baseline(in mm) as $B \in \mathbb{R}$. We express depth as $Z = f * B/d$.

7.2.4 Packaging depth and RGB frames

To do dense reconstruction, we feed the image and depth frames into GravityFusion and ElasticFusion via a custom format. known as '.klg'. This format requires the RGB and depth frames to be compressed. These compressed frames of RGB and depth are stored along with a respective timestamp. To time synchronize the RGB/Depth frames with the corresponding IMU readings, we use the IMU timestamp during file creation.

7.3 Reconstruction

We showcase qualitative results from ElasticFusion and GravityFusion. V2-sensor-pack stereo camera described above in section 7.1.3 was used along with the stack described in section 7.2 to create the following models. Here our intent is not to compare instead to highlight the quality of stereo camera based reconstruction with both ElasticFusion and our system GravityFusion.

7.3.1 Scanning with stereo camera

We collect our dataset by moving the sensor pack in lawn mower pattern as shown in figure 7.7. We maintain a 25-30% overlap during the scan strides. This type of scan pattern allows for the system to have frequent local loop closure at the overlapping strides hence keeping the model consistent.

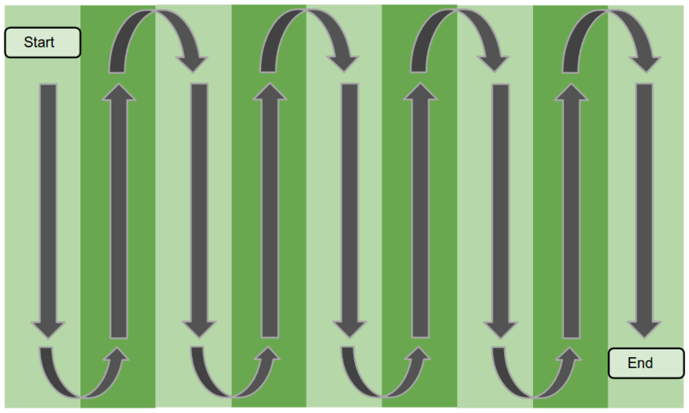


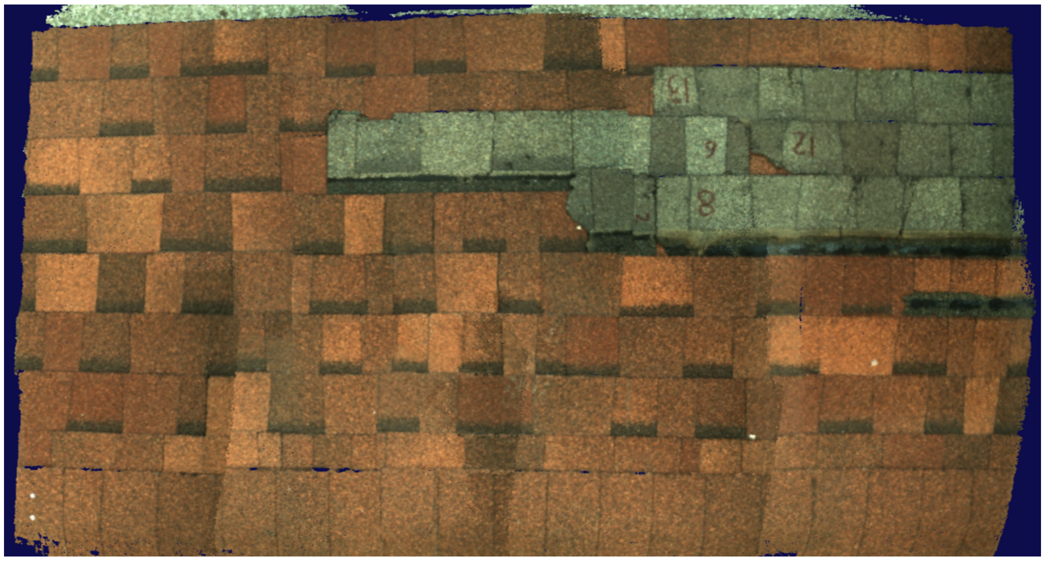
Figure 7.7: Camera pattern for scanning the surface

7.3.2 Stereo reconstruction with ElasticFusion

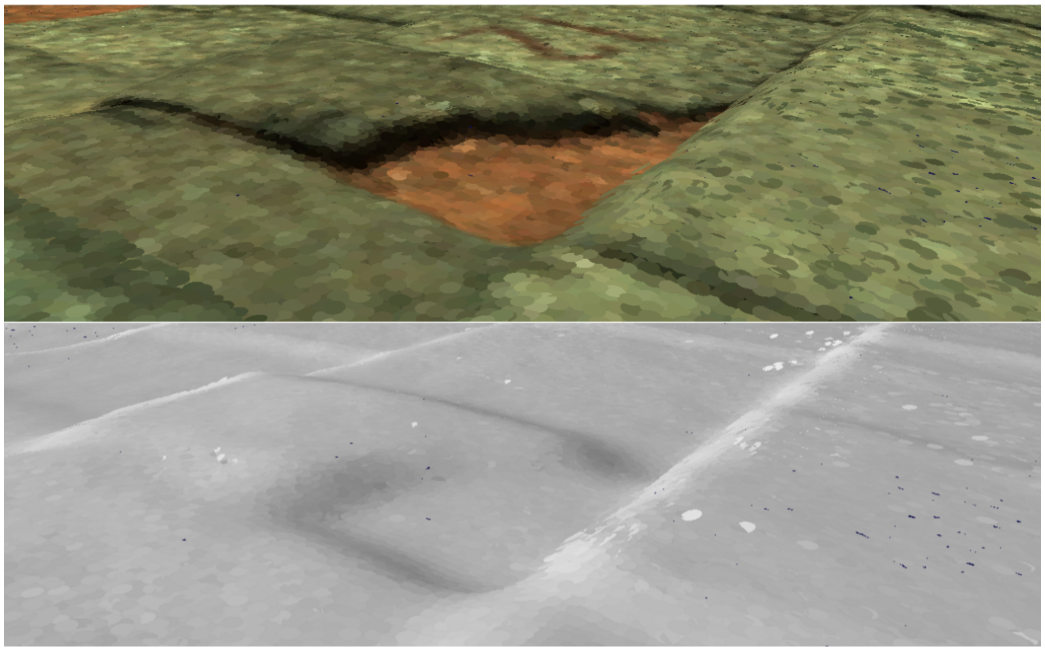
We use the ElasticFusion to generate the following dense 3D reconstructions. The 3D models generated are shown in figure 7.8 and figure 7.9. The details of scan quality are highlighted in the sub-figure 7.8b and sub-figure.7.9c. Both these results are captured in full daylight showcasing a high-quality reconstruction without the need for an RGB-D camera.

7.3.3 Stereo GravityFusion system

The dense reconstruction generated with ElasticFusion either due to drift or due to tracking loss sometime results in an inconsistent model. Here we showcase a dataset where Elastic-Fusion fails to create a correct model due to tracking loss shown in figure 7.10. However using our system GravityFusion, we are able to recover tracking loss and create a more consistent model.

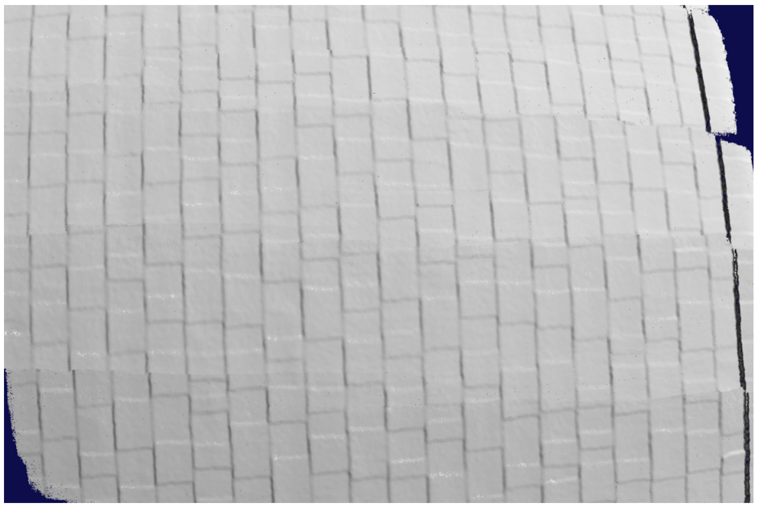


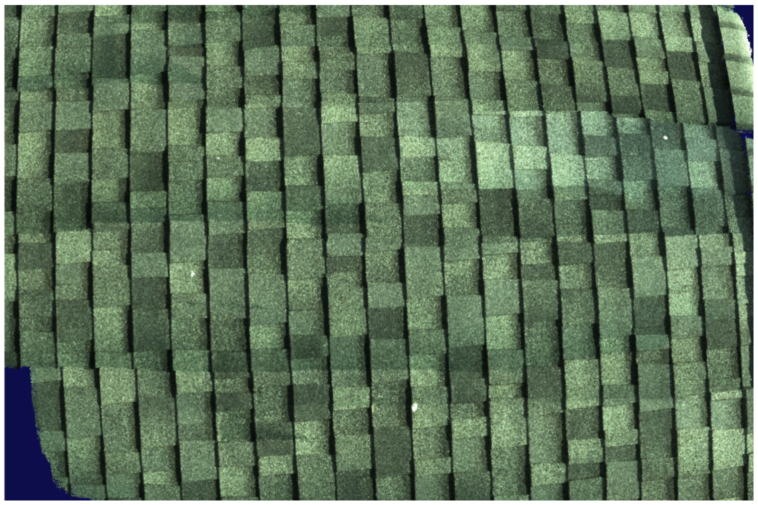
(a) 3D model using lawnmower pattern

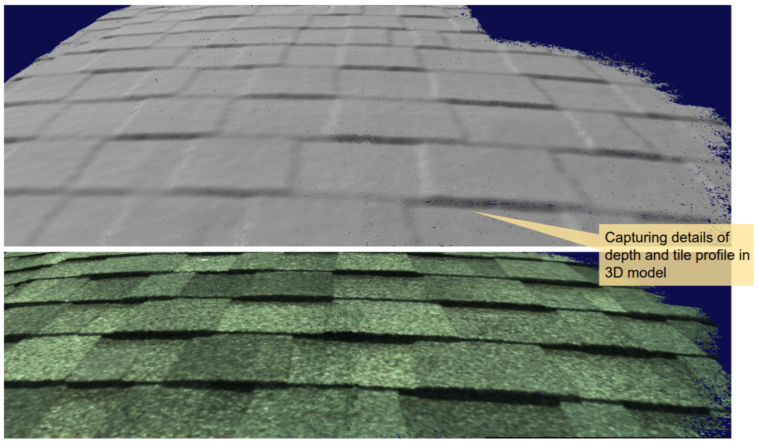


(b) Detailed model : highlights deformities in the surface

Figure 7.8: 3D model of the inhouse demo roof

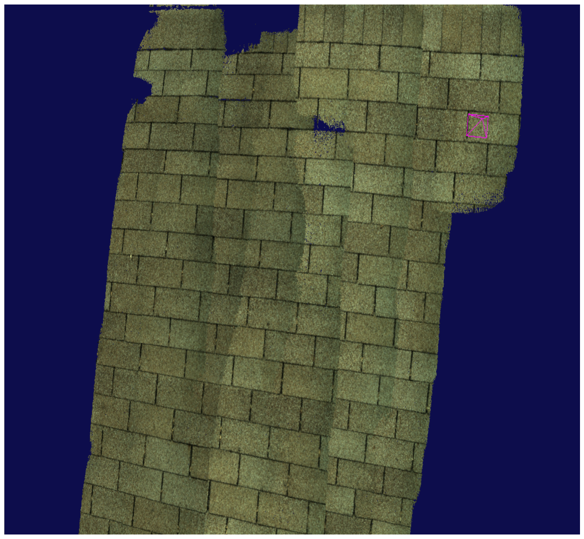

(a) Full depth cloud model


(b) Color 3D point cloud model



(c) Close up of the scan: Individual tile depth visible

Figure 7.9: Scan of tiles using V2-sensor-pack Stereo camera


(a) Color model

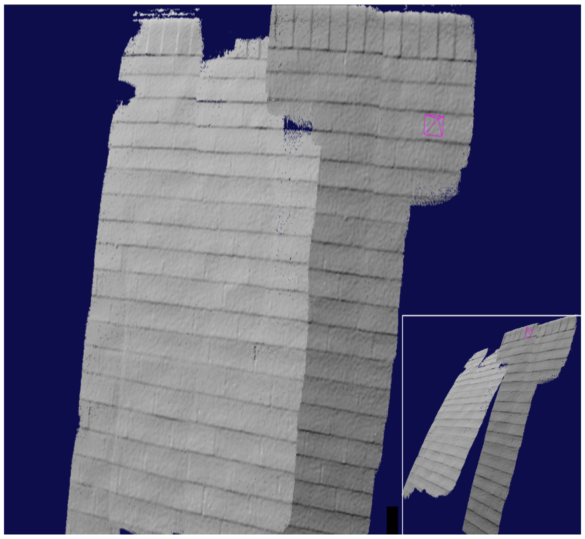
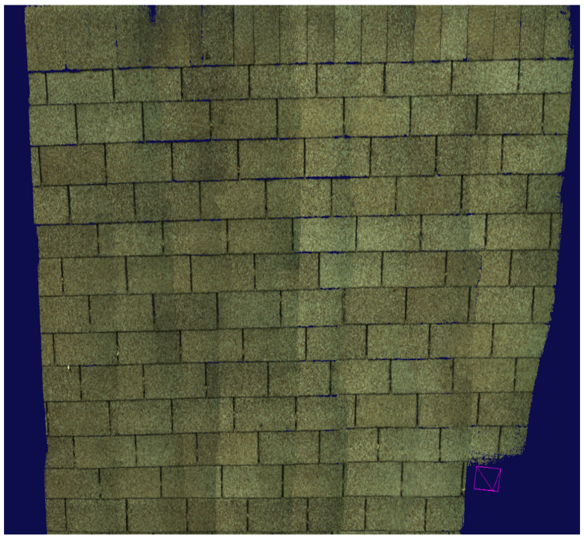

(b) Inconsistent model, side profile in small image

Figure 7.10: Model broken due to tracking loss, scan patterns are not aligned


(a) Color model

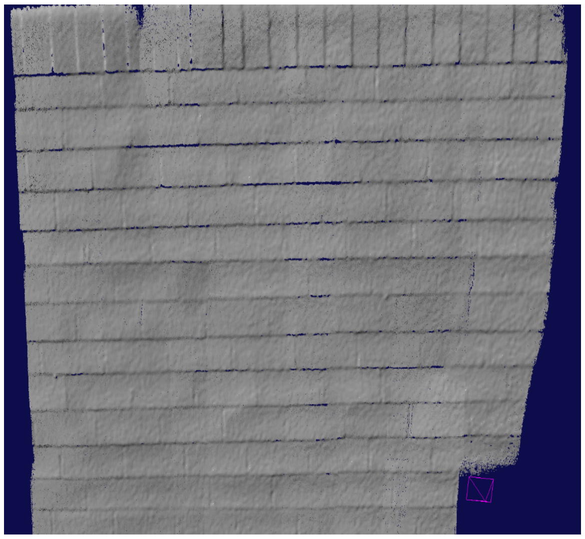

(b) Consistent model

Figure 7.11: Gravity-Fusion using inertial information aligns scan patterns making model consistent

Chapter 8

Conclusion and future work

We have presented a novel algorithm for correcting a map with inertial sensor data in the absence of a pose graph. Our GravityFusion algorithm works in real time and is capable of fusing inertial measurements of gravity on a per-surfel basis. Our detailed approach enforces global consistency of the gravity direction in the model through a mesh deformation. The deformation eliminates drift in pitch and roll in the model making it consistent. The evaluation on custom and public datasets validates our approach.

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. This could provide helpful constraints in perceptually challenging and ambiguous settings. Another direction is towards formalizing the deformation graph as a factor graph in the current setup utilizing gravity information. This formulation will extend the geometric consistency of the current approach. Alongside since our approach uses a surfel based representation, the current method of mapping will be able to overlap surfaces without the issues faced in other pose graph based approaches such as Kintinuous, hence resulting in a more robust and accurate map over a much larger scale.

Bibliography

[1] J. Castellanos and J. Tardós. *Mobile Robot Localization and Map Building: A Multisensor Fusion Approach*. Boston, MA: Kluwer Academic Publishers, 2000.

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

[3] B. Curless and M. Levoy. “A volumetric method for building complex models from range images”. In: *SIGGRAPH*. 1996, pp. 303–312.

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

[5] A. Davison. “Mobile Robot Navigation using Active Vision”. PhD thesis. Keble College, Oxford, 1998.

[6] A. Davison. “Real-Time Simultaneous Localisation and Mapping with a Single Camera”. In: *Intl. Conf. on Computer Vision (ICCV)*. Nice, France, 2003, pp. 1403–1410.

[7] F. Dellaert. “Square Root SAM: Simultaneous Location and Mapping via Square Root Information Smoothing”. In: *Robotics: Science and Systems (RSS)*. 2005.

[8] C. Estrada, J. Neira, and J. Tardós. “Hierarchical SLAM: Real-Time Accurate Mapping of Large Environments”. In: *IEEE Trans. Robotics* 21.4 (2005), pp. 588–596.

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

[10] 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, 2014.

[11] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox. “RGB-D mapping: Using Kinect-style depth cameras for dense 3D modeling of indoor environments”. In: *The International Journal of Robotics Research* 31.5 (2012), pp. 647–663.

[12] 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.

[13] A. S. Huang, A. Bachrach, P. Henry, M. Krainin, D. Fox, and N. Roy. “Visual odometry and mapping for autonomous flight using an RGB-D camera”. In: *In Proc. of the Intl. Sym. of Robot. Research*. 2011.

[14] 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.

[15] M. Kaess, A. Ranganathan, and F. Dellaert. “iSAM: Fast Incremental Smoothing and Mapping with Efficient Data Association”. In: *IEEE Intl. Conf. on Robotics and Automation (ICRA)*. Rome, Italy, 2007, pp. 1670–1677.

[16] M. Kaess, A. Ranganathan, and F. Dellaert. “iSAM: Incremental Smoothing and Mapping”. In: *IEEE Trans. Robotics* 24.6 (2008), pp. 1365–1378.

[17] 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.

[18] 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.

[19] G. Klein and D. Murray. “Parallel Tracking and Mapping for Small AR Workspaces”. In: *IEEE and ACM Intl. Sym. on Mixed and Augmented Reality (ISMAR)*. Nara, Japan, 2007, pp. 225–234.

[20] K. Konolige and J.-S. Gutmann. “Incremental Mapping of Large Cyclic Environments”. In: *International Symposium on Computational Intelligence in Robotics and Automation (CIRA’99)*. 1999.

[21] K. Konolige, J. Bowman, J. Chen, P. Mihelich, M. Calonder, V. Lepetit, and P. Fua. “View-Based Maps”. In: *Robotics: Science and Systems (RSS)*. 2009.

[22] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. “g2o: A General Framework for Graph Optimization”. In: *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*. Shanghai, China, 2011.

[23] 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.

[24] F. Lu and E. Milios. “Globally consistent range scan alignment for environment mapping”. In: *Autonomous Robots* (1997), pp. 333–349.

[25] 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.

[26] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. “FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem”. In: *Proc. 19^{th} AAAI National Conference on AI*. Edmonton, Alberta, Canada, 2002.

[27] J. Montiel, J. Civera, and A. Davison. “Unified Inverse Depth Parametrization for Monocular SLAM”. In: *Robotics: Science and Systems (RSS)*. Philadelphia, PA, 2006.

[28] 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.

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

[30] R. Newcombe and A. Davison. “Live dense reconstruction with a single moving camera”. In: *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*. IEEE. 2010, pp. 1498–1505.

[31] R. Newcombe, S. Lovegrove, and A. Davison. “DTAM: Dense Tracking and Mapping in Real-Time”. In: *Intl. Conf. on Computer Vision (ICCV)*. Barcelona, Spain, 2011, pp. 2320–2327.

[32] 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.

[33] P. Newman. “On the Structure and Solution of the Simultaneous Localisation and Map Building Problem”. PhD thesis. The University of Sydney, 1999.

[34] 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.

[35] R. Smith and P. Cheeseman. “On the representation and estimation of spatial uncertainty”. In: *Intl. J. of Robotics Research* 5.4 (1987), pp. 56–68.

[36] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. “A Benchmark for the Evaluation of RGB-D SLAM Systems”. In: *Proc. of the International Conference on Intelligent Robot Systems (IROS)*. 2012.

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

[38] 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.

[39] 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.

[40] 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.