Large Scale 3D Mapping of Indoor Environments Using a Handheld RGBD Camera

Brian Peasley
Clemson University, bpeasle@clemson.edu

Large Scale 3D Mapping of Indoor Environments Using a Handheld RGBD Camera

---

A Dissertation
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy
Computer Engineering

---

by
Brian K. Peasley
December 2013

Accepted by:
Dr. Ian D. Walker, Committee Chair
Dr. Adam W. Hoover
Dr. Timothy C. Burg
Dr. Christopher J. Post

Abstract

The goal of this research is to investigate the problem of reconstructing a 3D representation of an environment, of arbitrary size, using a handheld color and depth (RGBD) sensor. The focus of this dissertation is to examine four of the underlying subproblems to this system: camera tracking, loop closure, data storage, and integration.

First, a system for 3D reconstruction of large indoor planar environments with data captured from an RGBD sensor mounted on a mobile robotic platform is presented. An algorithm for constructing nearly drift-free 3D occupancy grids of large indoor environments in an online manner is also presented. This approach combines data from an odometry sensor with output from a visual registration algorithm, and it enforces a Manhattan world constraint by utilizing factor graphs to produce an accurate online estimate of the trajectory of the mobile robotic platform. Through several experiments in environments with varying sizes and construction it is shown that this method reduces rotational and translational drift significantly without performing any loop closing techniques. In addition the advantages and limitations of an octree data structure representation of a 3D environment is examined.

Second, the problem of sensor tracking, specifically the use of the KinectFusion algorithm to align two subsequent point clouds generated by an RGBD sensor, is studied. A method to overcome a significant limitation of the Iterative Closest

Point (ICP) algorithm used in KinectFusion is proposed, namely, its sole reliance upon geometric information. The proposed method uses both geometric and color information in a direct manner that uses all the data in order to accurately estimate camera pose. Data association is performed by computing a warp between the two color images associated with two RGBD point clouds using the Lucas-Kanade algorithm. A subsequent step then estimates the transformation between the point clouds using either a point-to-point or point-to-plane error metric. Scenarios in which each of these metrics fails are described, and a normal covariance test for automatically selecting between them is proposed. Together, Lucas-Kanade data association (LKDA) along with covariance testing enables robust camera tracking through areas of low geometrical features, while at the same time retaining accuracy in environments in which the existing ICP technique succeeds. Experimental results on several publicly available datasets demonstrate the improved performance both qualitatively and quantitatively.

Third, the choice of state space in the context of performing loop closure is revisited. Although a relative state space has been discounted by previous authors, it is shown that such a state space is actually extremely powerful, able to achieve recognizable results after just one iteration. The power behind the technique is that changing the orientation of one node is able to affect other nodes. At the same time, the approach — which is referred to as Pose Optimization using a Relative State Space (POReSS) — is fast because, like the more popular incremental state space, the Jacobian never needs to be explicitly computed. Furthermore, it is shown that while POReSS is able to quickly compute a solution near the global optimum, it is not precise enough to perform the fine adjustments necessary to achieve acceptable results. As a result, a method to augment POReSS with a fast variant of Gauss-Seidel — which is referred to as Graph-Seidel — on a global state space to allow the solution to settle closer to the global minimum is proposed. Through a set of experiments, it is shown that this combination of POReSS and Graph-Seidel is not only faster but achieves a lower residual than other non-linear algebra techniques. Moreover, unlike the linear algebra-based techniques, it is shown that this approach scales to very large graphs. In addition to revisiting the idea of using a relative state space, the benefits of only optimizing the rotational components of a trajectory in order to perform loop closing is examined (rPOReSS). Finally, an incremental implementation of the rotational optimization is proposed (irPOReSS).

Acknowledgments

I would most like to acknowledge and express my appreciation for the immense support and guidance contributed by Dr. Stan T. Birchfield , my advisor, colleague, and friend, who advised me through many situations, and provided support that made my graduate program easier and more enjoyable than it could have been.

I would like to thank all the students and faculty, past and present, of Clemson University who directly and indirectly provided helpful discussion, and assistance.

Finally, I gratefully acknowledge financial support from the National Science Foundation under grant IIS-1017007.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Overview

In this chapter the concept of 3D reconstruction will be introduced as well as a brief discussion on why it is important. This chapter starts by giving a brief overview of some of the applications of 3D reconstruction to motivate the work in this dissertation. Secondly 3D reconstruction will be reviewed as well as its relationship to previous work in Simultaneous Localization and Mapping (SLAM) and Structure From Motion (SfM). Finally this chapter will discuss in detail the focus of this dissertation.

1.2 Motivation

Three dimensional (3D) reconstruction of an environment is an important problem that has received much attention in recent years. The advent of ubiquitous range sensing, as well as significant advances in algorithms, has greatly improved fidelity and speed with which such reconstructions can be made, making them more relevant than ever. 3D reconstruction of environments has been used in many applications. These applications range from the field of robotics to biomedical imaging. The remainder of this section will present a brief survey of these applications in order to demonstrate the need for the study and improvement upon current 3D reconstruction techniques.

Biomedical Imaging The use of 3D reconstruction in biomedical imaging can allow for early detection of disease and abnormalities as well as aid in the study of diseases. 3D reconstruction has been applied for modeling of neuronal tissue samples in order to study the relationships between morphology and disease [10]. Recent work with ultrasound has led to development of 3D ultrasound devices which provide width, height, and depth. Combining these 3D ultrasound images can yield reconstructions of a fetus. These reconstructions aid in the detection of diseases like prenatal-onset skeletal dysplasia [47] and Down syndrome [4]. In addition to the detection and study of diseases, 3D reconstruction has been used to learn the 3D shapes of organs which aids in positioning and use of instruments during surgery [57].

Computer Graphics One of the more widely used applications of 3D reconstructions in computer graphics is augmented reality, superimposing 3D graphics onto a video or image. By constructing a 3D representation of an environment, CGI models can then be overlaid onto a video or image of that environment in a realistic way. In recent work by [60] models of historical buildings are merged with videos of the original locations of the buildings in order to provide a virtual heritage. In addition to augmented reality, computer graphics has applied 3D reconstruction techniques to create photo realistic 3D morphable models from a single 2D image of a person that allows for quick, easy, and cheap creation of models of actors to be used in movies and video games [7].

Robotics The robotics field, where the research in this dissertation is focused, has made extensive use of 3D reconstructions. Its primary use has been for autonomous navigation of environments [44, 56, 28] that exist outside the typical 2D planar environments that most robots are designed to operate in. In addition to navigation of 3D environments, 3D reconstruction has been applied to the problem of object grasping and manipulation [84, 72, 85].

1.3 Reconstruction Overview

3D reconstruction has been studied by both the computer vision and robotics community. 3D reconstruction in the robotics community is a subset of the widely studied area of Simultaneous Localization and Mapping (SLAM). In the computer vision community 3D reconstruction is referred to as Structure From Motion (SfM).

SLAM, pioneered by Smith et al. [75, 76], is a probabilistic framework to build a map of an unknown environment or to update a map of a known environment. SLAM has received considerable attention in the robotics community which has led to many landmark papers [54, 61] that allow for real-time implementations of SLAM. The sensors used to construct/update these maps is arbitrary and work in SLAM has used a wide variety of sensors that include, but is not limited to, sonars, laser scanners, mono cameras, stereo cameras, inertial measurement units (IMU), and time of flight sensors.

SfM, although similar to SLAM, differs in its specific use of mono, stereo, or multi-camera rigs, to reconstruct an environment. The specific use of cameras as a sensor makes SfM to a subset of Visual-SLAM (V-SLAM). Like SLAM, SfM has received considerable attention in the computer vision community with major contributions to the field like [13] that allows for the recovery of a camera trajectory in real-time and [59] that provides a system for real-time dense 3D reconstructions for a single camera. Additionally, SLAM does not work specifically in 3D whereas SfM is specific to reconstructing a 3D environment from 2D images.

Most research in 3D reconstruction has focused on sensor tracking, cameras for SfM and predominately lasers and sonars for SLAM. Recent advents in range sensing technologies have provided sensors that allow for the acquisition of high resolution colored point clouds in real-time using structured light. However a full 3D reconstruction system encompasses more than sensor tracking. Reconstruction, typically performed in 2D or 3D, is an iterative process of localizing the pose of a sensor given a map and previous poses (sensor tracking), then building onto that map with the readings from the given sensor. In this dissertation the 3D reconstruction process is divided into five distinct subproblems: data storage, camera tracking, integration, loop closure detection, loop closing. These five subproblems will be explained in detail in the following subsections. A pipeline of the reconstruction process can be seen in Figure 1.1. It is not claimed that this is the only valid pipeline for 3D reconstruction but rather one interpretation of the current literature. Additionally, since work in this dissertation explicitly uses a camera as a sensor, sensor tracking will be referred to as camera tracking.

1.3.1 Camera Tracking

Camera tracking is the problem of determining a camera's pose at time t using previous knowledge such as its previous locations, any generated maps, or any a priori assumptions about the motion (i.e, motion models) of the camera. This process is

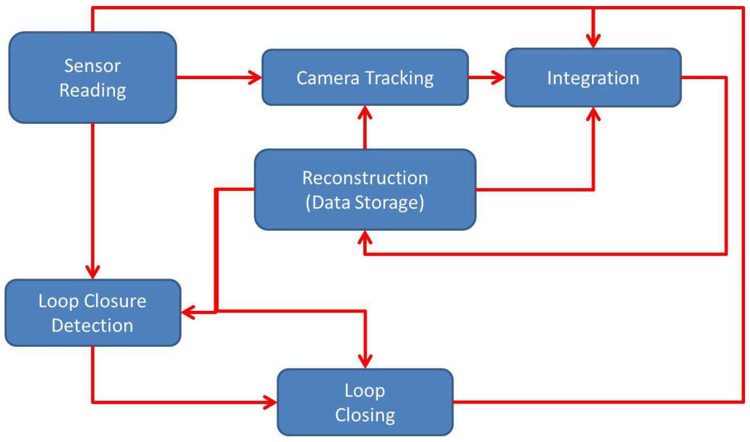


Figure 1.1: A pipeline of the reconstruction approach with loop closure for reconstruction of an environment. Readings from a camera, along with the current reconstruction are used for localization. Once localized the camera readings are integrated into the reconstruction. In addition, the camera reading is checked against the reconstruction to detect loop closures. If a loop closure is detected then the reconstruction is optimized and the optimized reconstruction is then integrated into the old reconstruction.

typically performed incrementally and uses the camera's pose at $t-1$ as an initial estimate for the pose at time t and the estimate is refined to minimize some error function.

Localization, typically seen in SLAM , also estimates the pose of sensor (camera in this dissertation). It differs from camera tracking in that localization typically uses the current sensor reading and compares it against a generated map to estimate the pose. So, localization can be seen as a global estimate of a sensor pose while camera tracking can be viewed as estimating the relative pose with respect to some starting coordinate frame.

1.3.2 Integration

Integration is the process of fusing new data into a global reconstruction. Given an estimate of a camera's pose, any data read seen by the camera can be integrated into a global reconstruction of the environment. The representation of the environment will dictate how new data is fused into the global reconstruction. In the context of this dissertation all reconstructions are represented using a Truncated Signed Distance Function (TSDF) or a 3D occupancy grid which are explained in more detail in Section 1.7.

1.3.3 Loop Closure Detection

Loop closures can be defined as a point of a reconstructed environment in which there are multiple sensor readings, typically separated by a large temporal spacing. More simply, the sensor has returned to a previously visited area. Depending on the environment finding loop closures may or may not be necessary. For environments in which the sensor visits the same area multiple times (interior of a building) the detection of loop closures is very prevalent, while very large environments (desert) where there is little, if any, overlap, loop closing may not be prevalent. Correctly asserting these loop closures is important because any errors in the detection of loop closures will be manifested during loop closing, which could greatly affect the final reconstruction. Loop closure is not focused on in this dissertation, instead all loop closures are assumed to be known.

1.3.4 Loop Closing

Loop closing is the process of taking all the detected loop closures, along with local estimates, to minimize an error function. Typically this error function describes the estimated trajectory of a sensor and minimizing this error finds the most optimal trajectory given all the available measurements. Loop closing can be done both incrementally (optimize as loop closures are detected) or batch (optimize after all loop closures have been detected). Loop closing by optimizing a graph based representation of the estimated sensor trajectory, which is commonly referred to as PoseSLAM since it only operates on measurements between sensor poses, is studied in this dissertation.

1.3.5 Data Storage

As a sensor is being tracked and its readings are being integrated into a reconstruction the reconstruction must be stored, preferably in memory for fast access. Typically in 2D this reconstruction is stored as an occupancy map since a two dimensional occupancy map does not require much memory and can be used to store very large environments at high resolution. However in 3D memory becomes a commodity and more efficient representations must be used to store the dense 3D reconstructions of environments.

1.4 Focus of Dissertation

In the previous sections the problem of reconstruction was outlined. First a simple 3D reconstruction system is proposed to motivate the work in this dissertation. This simple 3D reconstruction system that makes a simplifying assumption about the environment being reconstructed, namely the "Manhattan World" assumption which obviates the need for loop closure detection. Along with this assumption this system will begin an initial examination into multi-resolution volumetric occupancy grids in order to store the reconstructed environment in a memory efficient manner.

In this dissertation four of the five subproblems to the reconstruction process with no assumptions about the type environment made are examined, namely: camera tracking, data storage, integration, loop closing. In the following sections these four subproblems will be discussed in more detail as well as the approach taken in this dissertation to solve these problems. In addition the use of a color and depth camera (RGBD) is for 3D reconstruction in this dissertation. An RGBD camera provides a color and depth image that are aligned allowing for the creation of colored point clouds of the cameras field of view in real-time.

1.5 Manhattan Construction

In order to initiate a study of the 3D reconstruction process the type of environments to be reconstructed are limited to "Manhattan World" environments, all walls are parallel or perpendicular and the camera only exists inside a 2D plane. In addition the use of an octree, a multi-resolution representation of the environment, will be used to construct the environment allowing for reconstruction of large scale. As a robot drives around a previously unexplored indoor environment, the data acquired by the sensor is used not only to populate the map but also to compute the transformation of the robot between consecutive frames. These transformations, along with readings acquired by an odometry sensor, are fed to a pose-based SLAM algorithm to estimate the robot's pose on-line. Feature correspondence along with the Manhattan world assumption combine in a powerful way to significantly reduce translational drift and to essentially remove rotational drift. Results on several large environments validate the method's ability to build online octree-based maps without the need for correction, even in the presence of loop closure.

1.6 Camera Tracking

In general, the problem of camera tracking is the process of determining a camera's pose at time t using prior knowledge of an environment and the camera's trajectory. In the context of this dissertation the environment used for camera tracking is represented as a truncated signed distance function (TSDF) of the surfaces. An SDF is volumetric representation of an environment where each voxel contains the distance to the nearest surface, a truncated SDF (TSDF) stores the same values with the exception that no absolute value can be larger than some threshold. Additionally this dissertation focuses on tracking of an RGBD camera.

Given a current camera reading the camera can be aligned to a previous image reading or it can be aligned to a model (some representation of the environment) to estimate the current pose. In this dissertation current camera readings are aligned using a hybrid approach, using both the model and the previous images. If aligning to a model then camera tracking can be broken into 2 distinct sub problems: measurement synthesis, and pose estimation. Measurement synthesis is the process of converting between a sensor reading and the representation of the environment. For example, if the environment is represented as a color mesh and the sensor was an RGB camera then measurement synthesis would project the color mesh onto the camera to create a color image.

Of the many camera tracking techniques that have been developed, the landmark KinectFusion method [58, 37] has established itself as perhaps the most accurate real-time camera tracking system using an inexpensive sensor. Recent work by others [87] has extended this algorithm to operate over large-scale environments. However, both the standard and the extended version use geometric information alone to align the camera with the model, thus requiring the environment to contain sufficient geometric features in order for the iterative closest point (ICP) algorithm to accurately estimate the camera pose. This deficiency was noted in [87], where preliminary experiments were conducted to explore the advantage of replacing the ICP algorithm with the output of a feature-based visual odometry system called FOVIS [34] to provide a more stable estimate of camera pose. The frame-to-frame approach of FOVIS, however, loses one of the key advantages of KinectFusion, namely the reduced drift that result from matching the current image to the model.

Rather than replacing ICP with a technique that uses sparse feature points, a method to modify KinectFusion in a way that uses both color and depth from the RGBD sensor is proposed. This direct approach uses all the data, preserves the advantages of image-to-model matching, and obviates the need for the feature point extraction and feature correspondence steps inherit in feature-based visual odometry systems. The key to the approach is to replace the projective data association (PDA) point matching algorithm [5] used by KinectFusion with a data association technique driven by Lucas-Kanade [51, 82, 3]. This matching algorithm, referred to as Lucas-Kanade data association (LKDA) [65]. An automatic method for selecting the appropriate error metric based on the geometry of the scene is also proposed. Experimental results on standard datasets demonstrate that these two innovations enable camera tracking to succeed in areas of low geometry, without sacrificing either computational efficiency or accurate camera tracking in highly geometric environments.

1.7 Data Storage and Integration

Given that the overall goal of this dissertation is to present a system that can create a dense 3D reconstruction of an environment a memory efficient representation of this 3D environment must be used. Moreover, an efficient representation of the surface of a 3D environment is desired. In addition to memory efficiency the desired representation should have the following attributes: quickly indexed, continuous, and easily maintained/updated. The representation must be memory efficient due to the need to be able to handle large environments. It must be quickly indexed so that measurement synthesis (Section 1.6) of the 3D map can be done in real-time; if the 3D representation is not quickly indexed then measurement synthesis process will suffer in computational performance. The representation must be continuous as to not yield any discontinuities in a generated map; any discontinuities in the map will yield discontinuities in synthesized data, causing camera tracking to suffer in accuracy. Finally the representation must also be quickly and easily updated so that integration of new sensor readings is not a detriment to the computational performance of the system.

Of the many available 3D representations, multi-resolution volumetric occupancy grids are a promising approach for robotics. Other representations include point clouds and meshes. Point clouds, while the easiest representation of a 3D environment, only provides one of the desired attributes, easily updated. The memory cost for point clouds is $O(n)$, where n is the number of frames, while at first thought seems reasonable, however when considering large scenes the amount of memory needed to represent the scene becomes unmanageable. Point clouds are not easily indexed in the sense that to find all points around a current estimate of a camera's pose each point must be examined. Additionally, point clouds are not continuous which can cause camera tracking to become less accurate. The use of meshes overcomes the continuity and the indexing problem of point clouds. Meshes are also more memory efficient than point clouds having memory cost of $O(n)$ where n is the size of environment. However, meshes are not easily maintained and are expensive to compute.

Multi-resolution volumetric occupancy grids overcome the problems of point clouds and meshes by fusing the high-volume stream of measurements into a finite grid, both storage and computational requirements remain bounded and manageable, especially if multi-resolution storage schemes are used. A recent implementation of these ideas is OctoMap [88], which uses an octree-based data structure to accumulate data probabilistically while at the same time compressing the required storage down to a mere couple of bits per child node.

However, the main strength of an occupancy-based map, namely its ability to provide a compact representation of the scene, is perhaps also its greatest weakness. Promising to obviate the need to store raw range data for long periods of time, the representation is unable to correct large mistakes because the data are discarded as soon as they are assimilated into the map. This drawback is particularly apparent in the case of loop closure, where a single frame of data can necessitate large adjustments in the map representation. Because of this limitation, current implementations (such as [88]) assume that the robot's pose throughout a sequence is known at map construction time.

In addition to the rigidity of the volumetric occupancy grids, even when stored in an octree, another limitation is their discontinuity. Each voxel only represents the probability of that space being occupied, effectively leading to a discrete representation of the environment. Since the camera tracking algorithm proposed in this dissertation operates on a TSDF representation of the environment (Section 1.6), this work is motivated to store the TSDF in an octree. A TSDF, while a voxel representation, can be trilinearly interpolated to provide a continuous representation of the environment. More simply, given any 3D point the distance to the nearest surface can be computed at that point, whereas a occupancy grid would give the probability that the voxel in which that 3D point resides is occupied. In addition to overcoming the discontinuity of the occupancy grid, an octree based representation of a TSDF allows for a large-scale implementation of the proposed camera tracking algoritm allowing for large environments to be reconstructed.

Integration is tightly coupled with data storage and the sensor that is used for reconstruction. Throughout this dissertation an RGBD sensor is the only sensor used for reconstruction. However, two types of environment representations will be discussed: the TSDF and 3D occupancy grid. Since two different representations to build an environment are used integratation of new depth readings from an RGBD camera into these representations as well as highlight differences between the representations is discussed. In addition integration of depth readings into a TSDF when it is stored as an octree is also examined, which allows for large scale reconstructions when using a TSDF representation of the environment.

1.8 Loop Closing

Given that tracking of an RGBD camera over an extended sequence is successful, within some tolerance, over time the estimated pose of the RGBD camera will drift from its true pose. This drift can be corrected when the RGBD camera revisits an area of the environment which has already been mapped. This detection and recognition of previously mapped areas allows us to optimize the estimated trajectory of the RGBD sensor so that it is globally consistent. Detection or recognition of loop closures, which are non-trivial problems, are not examined in this dissertation. In this dissertation all loop closures are assumed to be known and trajectories are optimized using PoseSLAM.

PoseSLAM is the problem of simultaneous localization and mapping (SLAM) using only constraints between sensor poses, in which only the sensor poses (as opposed to landmark positions) are estimated. To a large extent, once the sensor poses have been determined, a map of the environment can be created by overlaying the sensor data obtained at the poses. Assuming a graph-based approach, the primary problem in PoseSLAM is to optimize the graph in the presence of loop closure.

The nodes of a graph represent the state space of a trajectory where each node is the state of a sensor at some point in the trajectory. The simplest state space is the global state space where each node represents the global pose of a sensor at some point in the trajectory. The edges in the graph represent measurements taken from the sensor. The choice of state space can have an enormous impact on the ability of an algorithm to optimize a trajectory. In their influential work on PoseSLAM, Olson et al. [61] proposed the use of an incremental state space (ISS). The advantage of this choice is that it leads to a very simple linear optimization problem. Curiously, in the same paper they briefly mention that one could form the problem using a relative state space (RSS) but discarded its use, saying that the resulting system is highly nonlinear and non-sparse, yielding a computationally expensive algorithm. The first two reasons are no doubt true, and as a result (to our knowledge) no one has attempted to use an RSS for loop closure.

The claim that using an RSS is computationally expensive is revisited. In fact, this dissertation arrives at a surprising result, namely that the opposite conclusion is true. By formulating the loop closure problem using an RSS, it is shown that the same variation of stochastic gradient descent — which we call non-stochastic gradient descent — is able to converge very quickly, typically in just one iteration. Like the ISS, the RSS leads to a formulation that is very straightforward, leading to an implementation that requires less than 100 lines of C++ code. This algorithm is called POReSS (Pose Optimization by a Relative State Space) in this dissertation.

While POReSS is able to achieve recognizable results (meaning that the basic shape of the map is present) in just one iteration, the coarse movements of the algorithm prevent it from ever reaching the global minimum. Therefore, the use of POReSS as a starting point for another algorithm, a fast variant of Gauss-Seidel, referred to as Graph-Seidel is proposed. This approach using a global state space (GSS) is much more able to make fine adjustments to the poses, thus enabling it to settle into a good solution. While Graph-Seidel requires many iterations, each iteration is extremely fast. It is shown that the combination of POReSS and Graph-Seidel is able to achieve competitive results compared with state-of-the-art. Moreover, the approach scales well, able to operate on graphs with tens of millions of nodes.

In addition to the POReSS algorithm two variants of POReSS are also introduced. The first variation, called rPOReSS, only optimizes the rotational component and using that output as a starting point for Graph-Seidel. This variation examines the claim made later in this dissertation, Section 3, that rotational drift is the main source of error. The second variation is irPOReSS which is an incremental implementation of the rPOReSS algorithm. It is designed to run online without being effected greatly by the number of edges or nodes in the graph.

Chapter 2

Previous work

2.1 Data Storage

Occupancy grids have been a popular representation for robot mapping since the pioneering work of Moravec and Elfes [55]. However, as pointed out by a number of researchers [35, 81], the grid-based approach does not facilitate loop closing because it is unable to handle pose uncertainty. The most common approach to simultaneous localization and mapping (SLAM) is to store a separate map with each particle, so that when information is obtained that renders previous calculations invalid, the data stored in the particle set can be used to correct the mistake [30]. However, this requires either all the data to be stored, or for multiple maps to be retained, both of which negate one of the main strengths of the grid-based representation. One solution would be to quickly rasterize the map into an occupancy grid whenever requested, as in [78], but this solution also requires the raw data to be stored.

Several researchers have extended the idea of grid-based representations to height maps that include the distance above the ground for each grid cell. Such an approach is explored by Marks et al. [52], in which the robot is run in an environment with high visibility, so that the large overlap in field of view between various viewpoints minimizes the effects of loop closure. Another approach is that of Pfaff et al. [67], in which a graph-based algorithm operating on all the data is used for loop closure, though an occupancy grid is used for the final representation. A similar approach for a flat ground is adopted by [29], which also builds on the idea of Lu and Milios [50] that requires all data to be retained.

In the computer vision literature, several methods have been developed in recent years to use the Manhattan world assumption for reconstruction. Furukawa et al. [23] describe an algorithm that employs a multiview stereo approach for estimating the 3D coordinates of a sparse set of feature points. From these points, dominant plane directions are extracted, from which plane hypotheses are generated. Markov random fields are then used to compute per-view depth maps, even for relatively textureless scenes. In followup work [24], an automated system for 3D reconstruction of architectural scenes is described using a combination of Manhattan world multiview stereo, structure-from-motion, and graph cuts for axis-aligned depth map integration. Additional research endeavors [63] [64] demonstrate the ability to perform online SLAM using planes extracted from point clouds.

The approach of Flint et al. [21] uses visual SLAM to obtain key frames in a video sequence, along with the pose of the sensor for each key frame. Using these poses, along with line segments detected in the key frames, an EM algorithm is used to estimate the rotation of the SLAM coordinate frame with the axis-aligned coordinate frame. This rotation yields the vanishing points in the images, which imposes a powerful constraint for detecting even faint axis-aligned edges, from which the wall, ceiling, and floor planes can be reconstructed.

2.2 Lucas-Kanade Data Association

One approach to using visual information to improve upon geometric mapping is that of Henry et al. [31], in which an initial estimate of the 3D camera transformation is found by applying RANSAC to SIFT feature matches with depth values. These visual feature associations are then combined with dense point associations in an ICP framework to minimize both geometric point-to-plane error and visual point-to-point error. The final transformations are used to produce a pose graph, which is then optimized to yield a globally consistent map as a surfel representation. Other researchers have followed a similar approach. Endres et al. [20] compare SURF, SIFT, and ORB features on public datasets using a system that also yields globally consistent transformations from post-processing optimization of pose-graph maps.

Another body of work improves upon ICP by incorporating color information. Druon et al. [16] segment point clouds based on the hue component of the HSV color space, then perform ICP while requiring matching points to belong to the same color class. Douadi et al. [15] incorporate both geometric and color information into the distance metric used by ICP. Huhle et al. [36] register scans of 3D point data by extending the standard metric with color information using Gaussian mixture models in a color space. Joung et al. [38] extract feature points from images using SIFT to find a set of correspondences between two laser scans, which provide an initial alignment for the standard ICP algorithm.

In work that is perhaps most similar to work in this dissertation, Tykkälä et al. [83] formulate the 3D registration of point clouds as a direct image-based minimization task, adding depth to visual odometry. In a manner similar to projective data association (PDA), their approach stores point cloud data as images to avoid expensive nearest-neighbor searches in 3D. In followup work the same researchers [2] reduce the computational load by adopting the direct approach of minimizing the sum-of-squared distances of image intensities under a projective model without using depth. The resulting system is able to produce 2D reconstructions of environments. Whelan et al. [86] augment KinectFusion's ICP algorithm with the RGBD alignment approach of Steinbrücker et al. [77]. An additional step is then provided to switch between this combined method and FOVIS. Kern et al. [43] present a real-time method using a single-core CPU that estimates the 3D rotation and translation by minimizing the photo-consistency between the two RGB images, utilizing depth to compute the 3D coordinates of the points being warped. Hoover et al. create space envelopes that model the scene using planar surfaces from range images and find corresponding planes between range images to estimate the cameras motion [32].

2.3 PoseSLAM

Work in this dissertation falls within the framework of graph-based SLAM, which was pioneered by Lu and Milios [50], who performed scan matching to determine the relative motion between two laser scans, and applied an iterative linearization method for graph optimization. Duckett et al. [17] proposed optimizing the map via relaxation, but this early work assumed knowledge of global orientation, which makes the problem linear. Frese et al. [22] propose multi-level relaxation (MLR), a variant of Gauss-Seidel, to find the non-linear maximum likelihood solution. Howard et al. [33] show how the general relaxation framework of Lu and Milios can be applied to a broad range of problems, including not only SLAM but also multi-robot SLAM and sensor network calibration.

Olson et al. [61] correctly noted the tendency of Gauss-Seidel to get trapped in local minima. Their approach contains two contributions: an alternative state space representation (incremental state space) so that a single iteration updates many poses, and a variant of stochastic gradient descent that is robust to local minima and converges quickly (compared with Gauss-Seidel). An extension of this work to incremental optimization of pose graphs was presented in [62]. Grisetti et al. [27] propose TORO (Tree-based netwORk Optimizer) that also extends the work of Olson et al. to use a tree-based parameterization for describing the configuration of nodes in the graph, as well as slerp functions for handling 3D rotations [25].

Other researchers have investigated the problem of nonlinear least squares minimization. R. Kummerle et al. [48] propose g2o, a flexible open-source framework for 2D or 3D SLAM and bundle adjustment, using sparse linear algebra techniques. Square Root SAM (simultaneous localization and mapping) [14] formulates the problem as a factor graph, also relying upon sparse linear algebra. In followup work, the approach was extended to provide incremental updates [41]. Both Konolige [45] and Montemerlo and Thrun [53] use the preconditioned conjugate gradient (PCG), while later work by Konolige et al. [46] exploits the sparse structure of the linear system. Ranganathan et al. [68] show that loopy belief propagation (LBP) is equivalent to Gauss-Seidel relaxation but also recovers the marginal covariances. In computer vision similar approaches are used to solve the related problem of bundle adjustment [1, 9].

Chapter 3

Manhattan Construction

3.1 Overview

This chapter will examine the use of a multi-resolution volumetric occupancy grid to store a reconstructed 3D environment of single level building interiors. More specifically this chapter will use an octree representation of a binary 3D occupancy grid. It will be shown that an octree representation of a 3D environment greatly reduces the amount of memory needed to store large scale reconstructions allowing for the storage of environments of sizes at least up to $50x50$ meters. One disadvantage to an occupancy grid representation of an environment, 2D or 3D, is its rigidity. An occupancy grid is unable to correct large mistakes because the data is discarded as soon as they it is assimilated into the occupancy grid. This chapter will not fully address this issue but rather focus on specific 3D environments, namely single level building interiors, where the “Manhattan” assumption can be combined with visual and wheel odometry to greatly reduce rotational and translational drift, obviating the need for any loop closing.

3.2 Octree Scene Representation

An occupancy grid [19, 80] is an efficient way to integrate sensor readings, while an octree efficiently represents a 3D occupancy grid. An octree is a hierarchical data structure, where each node represents a cubic volume of space (voxel), and each node is either a leaf node or has eight children representing eight equally-sized cubic subsets of the parent's cubic volume. Octrees are flexible representations, able to capture arbitrarily shaped environments at any desired level of resolution, the resolution being determined by the minimum voxel size. Research has shown [8] that octrees are able to efficently represent scenes, requiring approximately 2.6 bits to store each cubic volume. An illustrative example of the octree data structure can be seen in Figure 3.1.

One of the more compelling implementations of octrees is the OctoMap, recently introduced by Wurm *et al.*[88]. By explicitly representing three types of voxels (occupied, unoccupied, and unknown), the data structure is able to differentiate between areas of the environment that have been determined by the sensor to be free of obstacles and areas for which no information has yet been obtained. Each node is represented by two bits capturing one of four states, that is, whether the voxel is a leaf node, and therefore one of the three types just mentioned, or whether it is a parent node. Utilizing a clamping update policy, nodes that are saturated to either a minimum or maximum value indicate with a high degree of certainty whether they are occupied, leading to a binarized maximum likelihood decision. In combination, these implementation details yield a compact representation that, when binarized, can represent sub-meter resolution of areas more than 10,000 square meters in size with considerably less than one megabyte of storage.

However, this tremendous gain in efficiency comes at a price. The reason

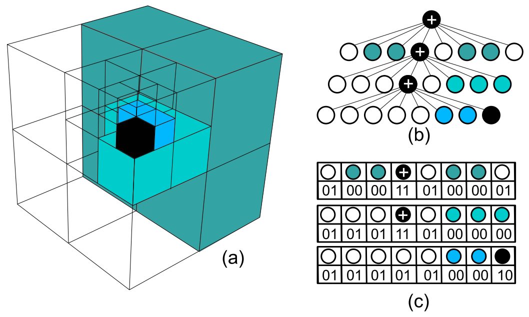


Figure 3.1: An octree representation (b) and corresponding compact bit encoding (c) of a simple 3D model (a). White indicates areas of the map that are unoccupied, gray indicates areas that are unknown, black indicates occupied areas, and black with a white cross indicates nodes that have children. At each level the 3D model is scanned in clockwise order around the top half, then the bottom half.

that occupancy grid maps are able to save so much space is that they discretize the sensor readings prior to storage. This discretization discards information and is a reasonable approach only when the pose of the sensor is known. Therefore, occupancy grid-based approaches typically perform in a batch fashion, first estimating the robot pose throughout the entire data collection process, then compressing the data in the occupancy grid structure. Such an approach does not naturally extend to online operation because drift in the pose estimation causes increased errors in the map over time. A constraint is emposed on the environment to reduce online pose estimation errors.

3.3 Factor Graphs for PoseSLAM

The underlying inference technique used in this chapter employs the Smoothing and Mapping (SAM) technique for representation and incremental solving of the

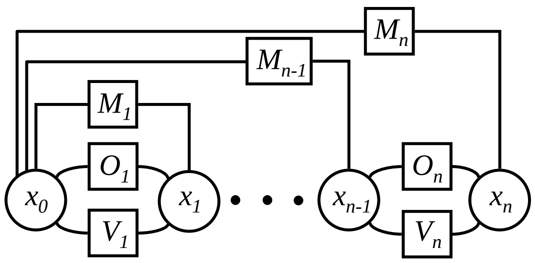


Figure 3.2: Factor graph used to estimate trajectory of robot. Three types of factors are used: relative pose by robot odometry (O_i), relative pose by visual registration (V_i), and Manhattan world constraint (M_i). The ith pose is given by x_i, $i = 0, \dots, n$.

SLAM problem as inference over an undirected graphical model; a detailed explanation of the approach can be found in [14]. In a pose-only formulation of SLAM, the trajectory $X \triangleq \{x_i\}$ is solved, given the measurements $Z \triangleq \{z_k\}$, which is represent in an undirected, bipartite *factor graph.*

The measurements are typically connected to a small number of variables, such as binary pose constraints calculated by visual registration or odometry. As an inference problem, a MAP estimate over all measurements is computed

$$\begin{aligned} X^* \triangleq \underset{X}{\operatorname{argmax}} \; P(X|Z) &= \underset{X}{\operatorname{argmax}} \; P(X,Z) \\ &= \underset{X}{\operatorname{argmin}} \; -\log P(X,Z), \end{aligned} \tag{3.1}$$

using Bayes' rule to cast inference as a nonlinear least-squares optimization problem in which the negative log likelihood is minimized:

$$X^* = \underset{X}{\operatorname{argmin}} \; \frac{1}{2} \left\| h(X) - Z \right\|_{\Sigma}^{2}, \tag{3.2}$$

where $h(X)$ is a generative measurement model that predicts all sensor measurements given the poses.

The sparsity of the relationships between variables motivates the use of a graphical formulation, in which the optimization problem is factored into separate *factors* $f_k(X_k, z_k)$, where each factor is a loss function $f_k(X_k, z_k) = \frac{1}{2} \left\| h(X)_k - z_k \right\|^2_{\Sigma_k}$ operating on the subset X_k of X associated with z_k. In this framework, $h_k(X_k)$ is the generative measurement model for the given sensing modality, with a local measurement covariance Σ_k. The full loss function can be defined as $L(X) = \sum^k f_k(X, z_k)$. As these factors are independent different types of constraints can easily be added to the graph.

Direct nonlinear optimization algorithms, such as Levenberg-Marquardt, can solve this problem in batch through recursive linearization of the full system around the current estimate X, successively computing updates δ until convergence:

$$\begin{aligned} \delta^* &= \underset{\delta}{\operatorname{argmin}} \ \frac{1}{2} \left\| h(X) + H(X)\delta - Z \right\|^2_{\Sigma} && (3.3) \\ &= \underset{\delta}{\operatorname{argmin}} \ \frac{1}{2} \left\| A\delta - b \right\|^2_{\Sigma}, && (3.4) \end{aligned}$$

where $H(X)$ is the Jacobian of $h(X)$ at X.

The full linearized system of (3.3) is reduced to a large block-wise sparse least-squares problem (3.4) to solve for δ^*. To avoid repeatedly solving a large system online, we again exploit sparsity and represent the solution process with a Bayes tree [39], which performs incremental multi-frontal Cholesky factorization to update the current estimate as new pose constraints are added. For more details on the iSAM (incremental SAM) algorithm, see [41].

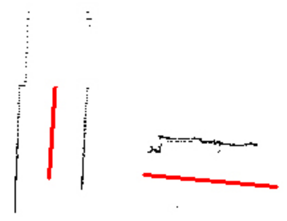
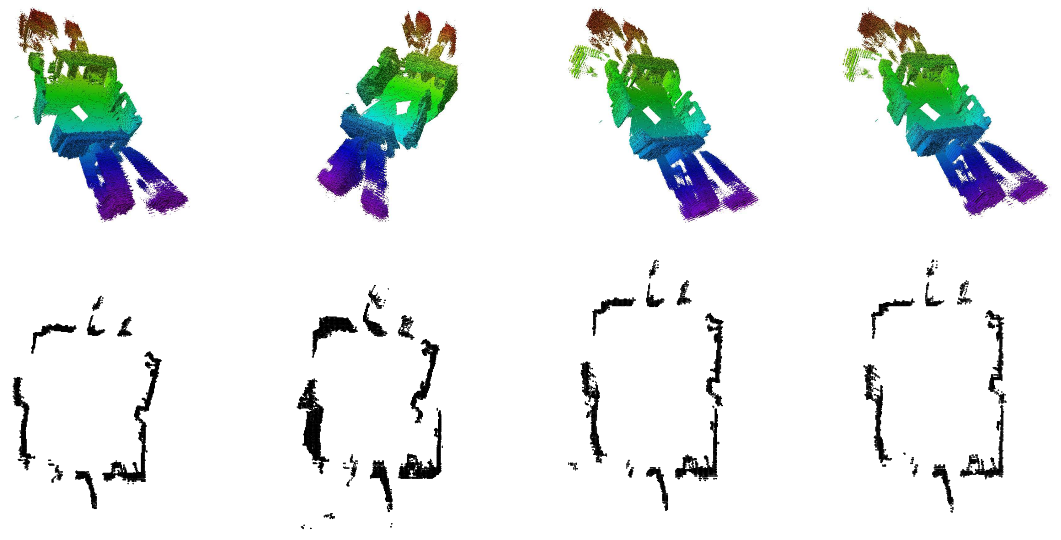

(a) (b)

Figure 3.3: Output from the RANSAC line fitting algorithm. The black dots indicate the points along a scan line from the point cloud. The red line is in the dominant direction of the points in the scan line. (a) The robot in the middle of a corridor, (b) The robot in an area where only one wall is visible.



(a)	(b)	(c)	(d)
Robot Odom	Visual Reg + Robot Odom	Man Constraint + Robot Odom	Man Constraint +Visual Reg + Robot Odom

Figure 3.4: Top: A 3D octree-based map of a laboratory environment of size 10.6 by 20.6 meters, constructed using 590 scans from the RGBD sensor and modeled with 30 mm resolution. Bottom: A 2D plan view of the map obtained by taking a horizontal slice through the 3D map. From left to right: results from various versions of the algorithm, demonstrating the ability of visual registration to reduce translational drift, and the Manhattan constraint to remove rotational drift. The final map required just 1.9 MB of disk space.

3.4 Manhattan Constraint

While the visual registration between consecutive frames helps significantly to reduce drift in the pose estimation of the robot, errors nevertheless persist. For large environments, even small rotational errors cause large errors over time, because positional errors are on the order of $\ell \sin \Delta\theta$, where ℓ is the length traveled, and $\Delta\theta$ is the rotational error. For example, even a rotational error of just 1 degree will produce positional errors of nearly two meters when traversing a length of 100 meters.

To overcome this rotational drift error, the use of a Manhattan world assumption is proposed. According to this assumption, every pair of surfaces of interest are either parallel or perpendicular to one another. One key advantage of the Manhattan world assumption is that its enforcement does not require precise correspondence to be established between pairs of frames. Rather, in the context of a 3D sensor, all that is required is that planes be clustered appropriately into one of three mutually orthogonal bins. Because the relative rotation between consecutive frames is on the order of a few degrees at most, and because the bins are 90 degrees apart, essentially *zero rotational drift for indefinite periods of time* can be achieved in environments in which the assumption holds, with only mild assumptions on the ability of the algorithm to associate planes correctly. This removal of the most dangerous of the two types of drift enables the compression abilities of the occupancy grid-based approach to be fully utilized without significant fear of regretting the loss of data that would otherwise have been imperative for proper handling of loop closure.

Unlike the other factors which are added to join consecutive robot poses in the graph, the Manhattan constraint always connects the current pose to the initial pose, where the world coordinate frame is defined. This is illustrated in Figure 3.2.

In order to apply the Manhattan constraint on the geometry of the scene it is necessary to isolate features in the environment that will allow the rotation parameters for this constraint to be found. The most obvious features in indoor environments that are either parallel or orthogonal are walls. As mentioned earlier, in order to calculate the rotation appropriate for geometric alignment, explicit correspondence of items in the point cloud is not necessary. Rather, only the normal of a single wall in the current frame is needed, along with the assignment to the same plane sensed in the previous frame. To determine such a plane, RANSAC is applied to the depth data in a horizontal scan of the RGBD sensor to find the dominant line. Output from this approach can be seen in Figure 3.3.

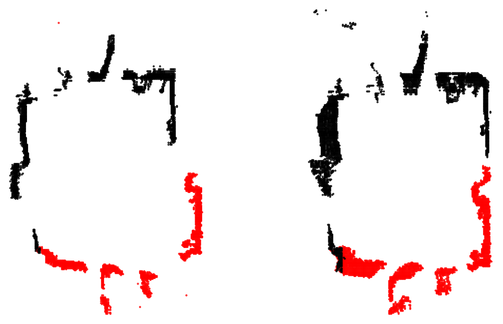
Once the relative rotation between walls of consecutive frames has been established, the orientation of the current frame and the global coordinate frame is automatically achieved, since the orientation of the previous frame is already known. This zero-drift principle of the Manhattan World constraint is similar to the driftless approach of matching the current sensor reading to the model rather than to a previous sensor reading, employed in KinectFusion [37].

3.5 Experimental Results

To evaluate the proposed approach maps are constructed from data recorded in three different indoor environments. The first environment was a small laboratory where a mobile robot drove around the perimeter of the room. The purpose of this experiment was to test the ability of the proposed method to handle rotational and translational drift without explicitly handling any loop closures. The second environment was a building on our campus consisting of a long main corridor and two side corridors, with no opportunity for loop closure. This experiment tested the performance of the rotational constraint imposed by the Manhattan world assumption over a long distance. The third environment was another large building on our campus containing many opportunities for loop closure, thus allowing the error in the results to be measured with and without such techniques. The hardware platform consisted of an ActivMedia Pioneer P3AT mobile robot with a forward-facing Kinect RGBD sensor.

Results from the first experiment in the laboratory can be seen in Figure 3.4. Four different reconstructed maps demonstrate the influence of the various terms in the factor graph. Figure 3.4(a) shows the constructed map using only odometry data. As expected, both rotational and translational drift are present, causing noticeable errors in the map. Figure 3.4(b) shows the map constructed using cues from both the robot odometry and the visual registration. Although visual registration could be used to reduce both translational and rotational drift, it is employed for the former in order to better show the power of the Manhattan assumption. As a result of this limitation, the addition of the visual registration causes the right wall to move to the left. At first inspection it may not be obvious that the map constructed in Figure 3.4(a) is worse than that of Figure 3.4(b). However, if the coincidental combination of translational and rotational drift is separated, then the errors due to odometry alone are more readily apparent, see Figure 3.5. While the Manhattan world constraint is sufficient for removing the rotational drift from the map, it does not address the problem of translational drift. The effects of the latter can be seen by the slight misalignment of the two pieces of the wall on the right side of the map (just above the concavity) in Figure 3.4(c). This gap is removed in Figure 3.4(d) by the addition of visual registration.

Figure 3.6 shows the resulting maps of the second environment. Due to the size of the building (the length of the main corridor is approximately 55 meters), there is much room for the robot odometry and visual registration to drift. This drift



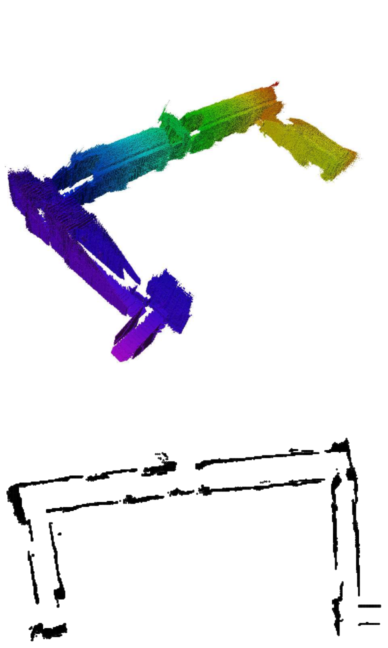
(a) Robot Odometry

(b) Visual Registration + Robot Odometry

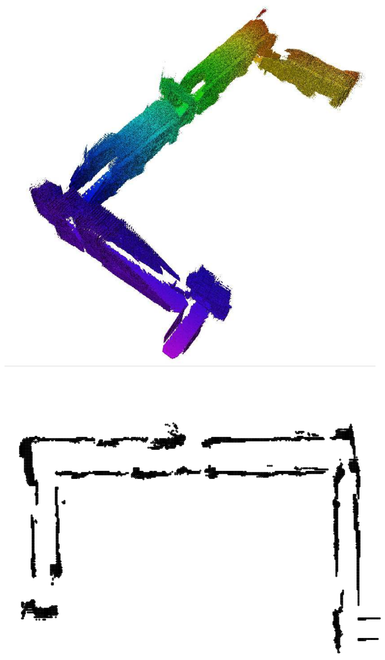
Figure 3.5: A comparison of the system with and without visual registration, showing the rather larger translational error reduction. The points in red were rotated about the bottom left corner of their respective maps to isolate the translational drift from the rotational drift.

is shown in Figure 3.6(a), where significant rotational drift causes noticeable errors in the map. By adding in the Manhattan world constraint to the factor graph all rotational drift is removed from the map, see Figure 3.6(b), even though there is no opportunity to perform loop closure.

In the third experiment the robot was driven around the floor of a large building containing several intersecting hallways. This experiment shows not only the ability of the Manhattan world constraint to remove rotational drift over an extended period of time, but also the ability of the visual registration to reduce the translational drift to a surprisingly low level, without any loop closure. Figure 3.7(a) shows the map with robot odometry and visual registration, which exhibits noticeable distortions over the length of the path. Of course, existing techniques can handle such environments, but only by requiring that raw data are kept until such a time as loop closure is performed. In contrast, the proposed approach, shown in Figure 3.7(b), is



(a)
Robot Odometry
+ Visual Registration



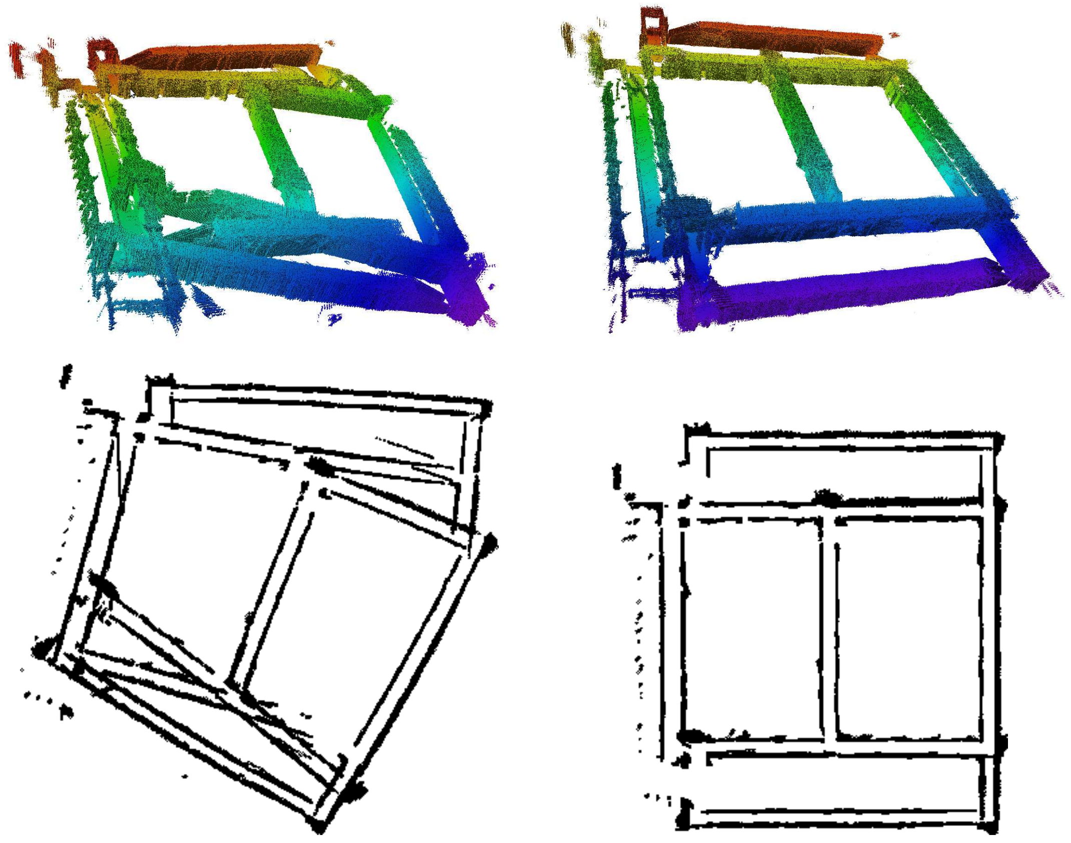
(b)
Manhattan Constraint
+ Visual Registration
+ Robot Odometry

Figure 3.6: A large building with no loops. (Size was 23.9 by 47.8 meters, modeled with 30 mm resolution using 3,300 scans). The addition of the Manhattan constraint enforces perpendicularity of the walls. The map was saved to disk using only 2.1 MB.

able to significantly reduce rotational and translational drift over an extended period of time, thus enabling data to be discarded as they are assimilated into the map.

The particular path driven by the robot is illustrated in Figure 3.8. Starting at location 1, the robot drove (from a bird's eye point of view) up and to the right, then down to 2. Turning left (the robot's right), it traveled through 3, then down to 4, then over to 5, after which it encountered 3 and 2 again before heading down to 6 and then completing the bottom loop to end at 6. Opportunities for loop closure therefore occurred at locations 1, 3, 2, 4, 5, and 6, in that order. Table 3.1 shows the errors occurring at the six different potential loop closure locations for the particular path driven. These errors were obtained by manually viewing the video and selecting, for each intersection, two key frames in which the robot was approximately in the middle of the intersection; the distance between the two estimated robot locations yielded the error. Due to imprecision in this measurement technique, these numbers should be used as relative rather than absolute assessments of error. Nevertheless, the Manhattan assumption reduces the error by about an order of magnitude.

The amount of memory saved in using the octree-based representation rather than retaining all the raw data is approximately three orders of magnitude, as shown in Table 3.2 and Figure 3.9.



(a)
Robot Odometry
+ Visual Registration

(b)
Manhattan Constraint
+ Visual Registration
+ Robot Odometry

Figure 3.7: An environment with several intersecting corridors. The building is 52.6 by 53.2 meters and modeled with 30 mm resolution using 7,789 RGBD scans. The building was traversed multiple times in order to map the environment in its entirety. The benefit of the Manhattan assumption is evident. The map required just 5.6 MB of disk space.

Intersection	RO	VR + RO	VR + RO + Manhattan
1	12.0 m	1.0 m	0.1 m
2	31.5 m	8.3 m	0.5 m
3	16.1 m	3.7 m	1.0 m
4	25.0 m	6.7 m	0.8 m
5	12.7 m	10.7 m	0.5 m
6	15.5 m	4.5 m	0.5 m

Table 3.1: Error for six intersections from the environment shown in Figure 3.7. The columns show the results using various combinations of robot odometry (RO), visual registration (VR), and the Manhattan constraint. The path of the robot and the intersection points can be seen in Figure 3.8.

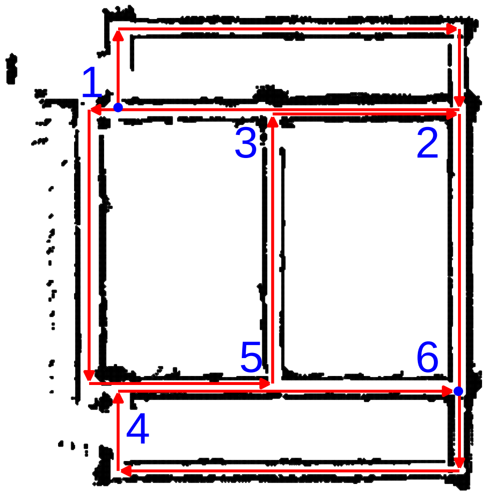


Figure 3.8: Path taken by the robot in the generation of map in Figure 3.7. The robot moved in the direction of the arrows, encountering the intersection points in the following order: $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 3 \rightarrow 2 \rightarrow 6 \rightarrow 4 \rightarrow 5 \rightarrow 6$. Therefore, the potential loop closures would have been, in order, $(1, 3, 2, 4, 5, 6)$.

	in memory	on disk
Point Cloud	4,350,000 kB	590,000 kB
Octree	941 kB	363 kB
Compression	4622:1	1625:1

Table 3.2: Amount of space Required to store the entire map in Figure 3.6 in both memory and on disk.

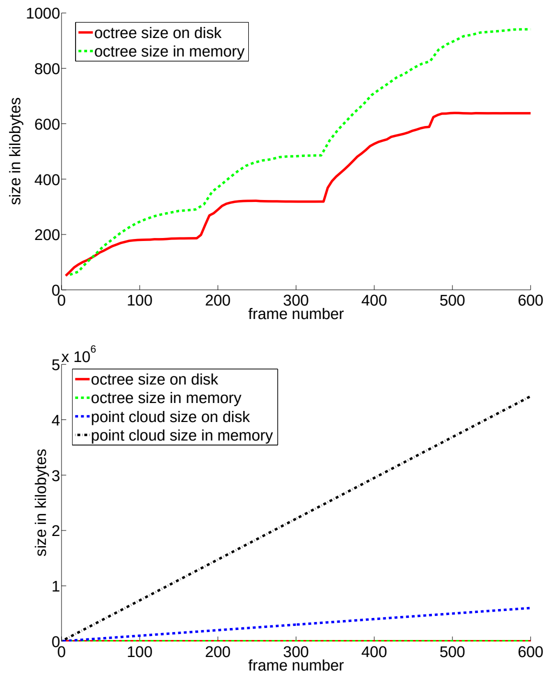


Figure 3.9: The amount of space required by the map on disk and in memory by point cloud and octree representations, as a function of frame number. The plots show the sizes for the map constructed in Figure 3.4. The top plot is a zoomed-in view of the bottom plot (notice the red and green lines overlaid on the x axis in the bottom). The octree reduces storage requirements by more than three orders of magnitude.

Chapter 4

Camera Tracking

4.1 Overview

A key challenge to any mapping system is to maintain camera tracking and estimate the 3D Euclidean pose of the camera (or sensor). Work in the landmark KinectFusion papers [58, 37] solves this problem through a particular variant of Iterative Closest Point (ICP), which is a family of algorithms to incrementally align two point clouds. The variant of ICP employed by KinectFusion uses Projective Data Associate (PDA) for correspondence. This limits the KinectFusion algorithm to use only geometric information which subsequently limits the type of environments which it can maintain camera tracking. This chapter proposes to overcome this reliance on geometric information by replacing PDA with a correspondence scheme driven by Lucas-Kanade. This data association technique will be refered to as Lucas-Kanade Data Association (LKDA). PDA and LKDA are illustrated in Figure 4.1. In addition to replacing the data correspondence step this chapter also modifies two additional steps of the ICP variant used by KinectFusion. As explained in [70], each iteration of ICP can be broken into six steps, and different choices within these steps lead to different variations of the algorithm. These six steps, along with a description of their implementation in both KinectFusion and the proposed approach, is as follows:

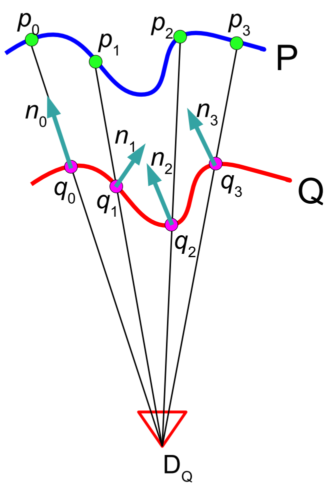
1. *Point selection.* First it must be determined which points from the two clouds to use. Both KinectFusion and the proposed approach use all points.

2. *Point matching (Data Correspondence).* The next step is to establish data association, or correspondence, between the points in the two clouds. This is achieved in KinectFusion by Projective Data Association (PDA) [5], a camera-centric approach to data association that is especially suitable for point clouds obtained from a depth sensor. By assuming that the change between the two point clouds to be aligned is small, which is warranted due to the real-time nature of the system, PDA achieves data association in an efficient manner. In contrast, Lucas-Kanade Data Association (LKDA) is introduced to overcome the inability of PDA to handle scenes with little geometric texture.

3. *Weighting.* As with point selection, both KinectFusion and the proposed approach weight all point correspondences equally.

4. *Rejection.* Outliers must be removed from the correspondence to avoid corrupting the computed transformation. In KinectFusion, a corresponding pair of points is rejected if either the distance between them or the difference between their normal vectors is too large. The proposed approach adds a third condition to test whether their colors are significantly different.

5. *Error metric.* KinectFusion minimizes the point-to-plane error, which has been shown [69] to converge more quickly than a point-to-point error metric, when coupled with PDA. The proposed approach automatically selects either of these



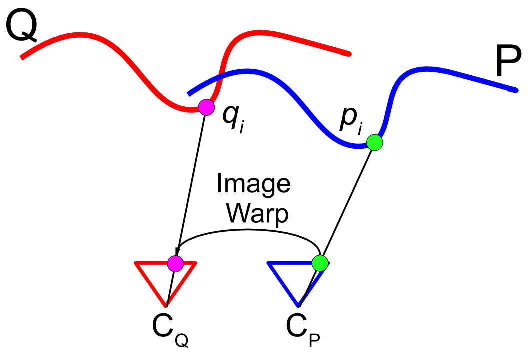
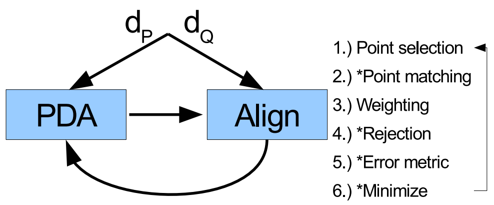


Figure 4.1: Left: PDA establishes correspondence between two point clouds by projecting one onto the depth camera, D_Q, of the other. In addition to point correspondences ($p_i \to q_i$) the normals n_i, which are used in the point-to-plane error metric are depicted. Right: LKDA establishes correspondence by projecting a point p_i onto RGB camera C_P. The projected point is then warped onto the RGB camera C_Q using the estimated warp from Lucas-Kanade, and is finally mapped back into a 3D coordinate, q_i.

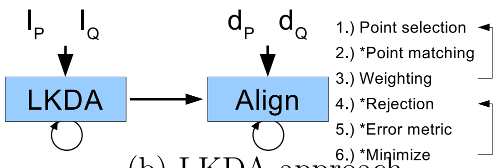
two metrics depending upon the covariances of the normals, in order to handle scenes with low or high geometric texture.

6. *Minimize.* The point-to-point error metric is minimized in closed form, while the point-to-plane error metric requires an iterative method.

An illustration comparing the two approaches is given in Figure 4.2. Note that, while the proposed approach follows the same six steps of ICP, it is not necessarily an ICP algorithm since the correspondences are not recomputed each time the parameters are estimated. For the remainder of this chapter the proposed approach will be refered to by its data correspondence technique, LKDA.



(a) ICP implementation in KinectFusion



(b) LKDA approach

Figure 4.2: Top: Given the current and reference depth images, KinectFusion iterates through the six ICP steps. Steps 1-3 perform data association (PDA), while steps 4-6 perform alignment. Bottom: In contrast, LKDA iteratively finds the correspondences using the RGB images (LKDA, steps 1-3), then iteratively estimates the alignment parameters using the depth images (steps 4-6). The steps marked with an asterisk (*) are different in the two approaches.

4.2 Mapping to Model

In addition to a real-time camera tracking system, one of the key novelties of the KinectFusion algorithm is its introduction of camera tracking with respect to a model as opposed to a previous frame. When mapping to a model new sensor readings are aligned to the current reconstruction of the model and not to the previos sensor readings allowing for a reduction in drift. Drift is not reduced when the sensor readings do not have large overlaps (i.e. going down a hallway). In order to align the current sensor reading to the model both need to be represented in the same way, or have a way of converting between the sensor and model representations. In this dissertation this process is called measurement synthesis.

In the context of KinectFusion a synthetic depth image is generated from a truncated signed distance function (TSDF) which the current depth image is aligned too using ICP. In the context of the LKDA approach proposed in this dissertation a synthetic depth image is generated in the same way, however correspondence between the current depth image and the synthetic depth image is determined by the previous current and previous color images. LKDA drives the geometric alignment of the current depth image to a model with the use of temporally subsequent color images.

4.2.1 TSDF

TSDFs, also referred to as implicit functions or level sets, have been used heavily in the computer graphics community, especially for volumetric rendering (i.e. smoke, clouds, fire). The use of TSDFs to create complex models was was pioneered by [12] but has received little attention in the computer vision and robotics community up until recent years. Typical representations of 3D models in robotics is the combination of several point clouds which only provided a rudimentrary scene reconstruction. Extending upon point clouds triangular meshes can be created from a point cloud to generate a more robust reconstruction. However, maintaining a triangular mesh is both a complicated and computationally expensive operation. TSDFs provide a robust and continuous representation of the environment that is easily maintained.

In the context of 3D reconstructions a TSDF is a function that represents the surface of a scene by providing the distance to the nearest surface. The surfaces can be recovered from a TSDF by finding zero crossings in a TSDF. Addtionally, while a TSDF is typically used in 2D or 3D they can be used in any dimensionality. A graphical depiction of TSDF can be seen in Figure 4.3.

To create a synthetic depth image of a TSDF a ray is cast from the location of the RGBD camera through a pixel in the depth image into the TSDF until a surface is hit (i.e. zero crossing). When a surface is hit the orthognal distance to the surface

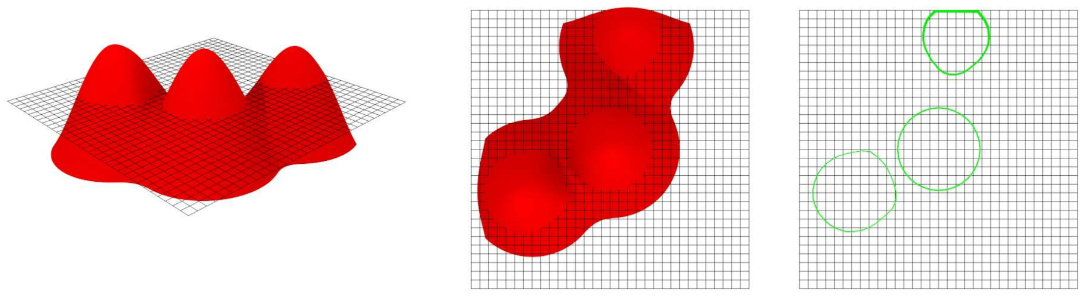


Figure 4.3: A graphical depiction of a 2D TSDF. The far left and middle images are a view of the same TSDF from different angles. The far right image shows the zero crossings of the TSDF which could easily be used to represent the contours of a segment in an image or similar applications.

from the camera is stored at the pixel in the depth image that the ray passes through. An illustration of this process can be seen in Figure 4.4. .

4.3 Lucas-Kanade Data Association

Let $P = \{p_i\}_{i=1}^N$ and $Q = \{q_i\}_{i=1}^N$, where $p_i, q_i \in \mathbb{R}^3$, be two point clouds in 3D Euclidean space. The goal of data association is to compute a function $L : \mathbb{Z}_{1:N} \rightarrow \mathbb{Z}_{1:N}$ that maps indices in one point cloud to those of the other, so that $L(i) = j$ indicates a corresponding pair of points $p_i \leftrightarrow q_j$, where $\mathbb{Z}_{1:N} = \{1, \ldots, N\}$ is the set of the first N natural numbers.

As shown in Figure 4.1, PDA establishes correspondences by projecting the points from both point clouds onto the same image plane — points that project onto the same pixel are established as corresponding points. This approach relies solely upon geometric information, i.e., the spatial coordinates of the points in the two clouds. As a result, environments that do not provide high geometric texture (e.g., a plane such as a wall) can cause KinectFusion to fail to track the camera. Another failure mode occurs when sufficient depth readings occur outside the truncated signed

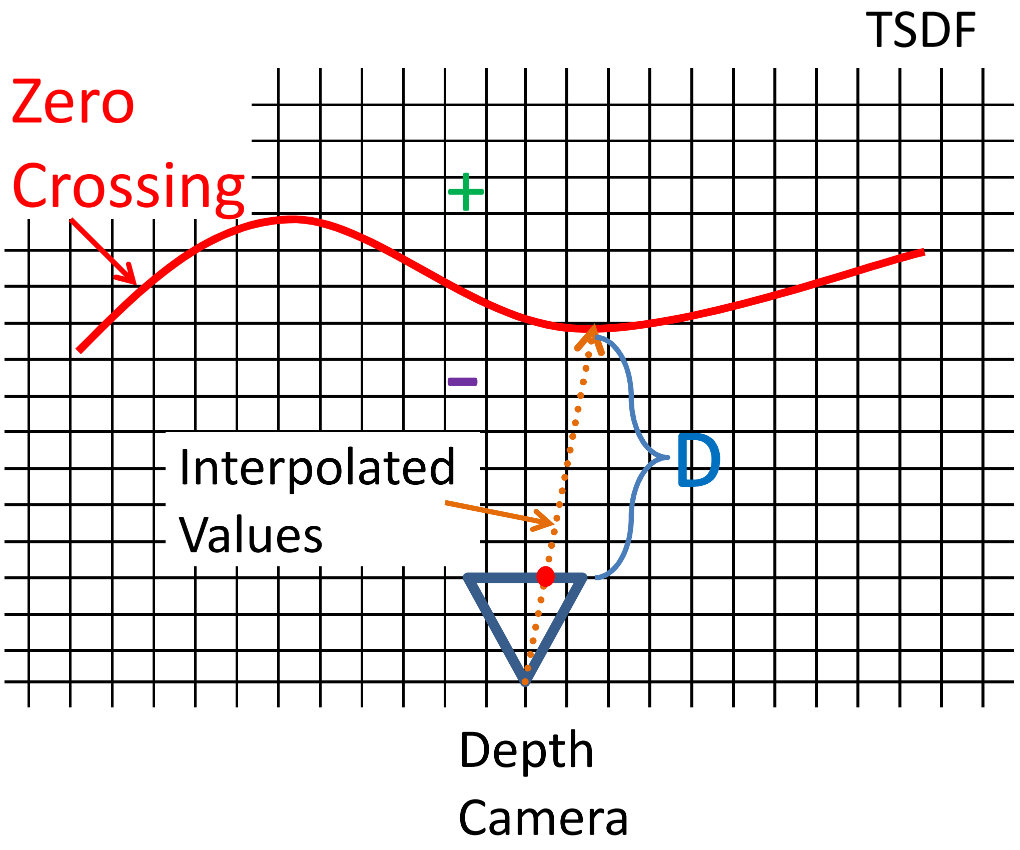


Figure 4.4: Illustration depicting the measurement synthesis process for a TSDF and a depth camera. A ray, orange, is cast from the center of projection of the depth camera into the TSDF. Points along this ray are sampled and the TSDF is trilinearly interpolated at these points to obtain the signed distance to the nearest surface. Once the sign of the TSDF has changed along the ray (i.e. zero crossing/ray intersection with surface), the orthogonal distance, D, is stored in the pixel the cast ray passes through (red circle) in the depth image.

distance function (TSDF) maintained by KinectFusion; such readings cannot be used for camera tracking. The RGB color information, on the other hand, is not affected by the scene geometry or the size of the TSDF, and therefore can improve the alignment of point clouds in such environments. While existing systems such as [34] and [31] use visual cues alongside the depth information for camera tracking, these approaches rely upon extracted sparse feature points to estimate the relative transformation of a camera. In contrast, this section describes an approach for incorporating *all* the RGB information in a natural way to make the data association step more robust.

4.3.1 Lucas-Kanade

The Lucas-Kanade algorithm is a differential method for computing the optical flow of an image. The goal of Lucas-Kanade is to find the parameters ζ that minimize the sum of squared distance error

$$E_{LK} = \sum_x \sum_y \left(I(W^{-1}(x, y; \zeta)) - J(x, y) \right)^2, \tag{4.1}$$

where I and J are two consecutive image frames, the double summation is over all the pixels, and $W(x, y; \zeta)$ is a parametric warp function that brings the two images into alignment. This equation is minimized by linearizing about the current estimate and repeatedly solving a linear system:

$$\Delta\zeta = H^{-1} \sum_x \sum_y \left(\nabla J \frac{\partial W}{\partial \zeta} \right)^T \delta_{IJ}, \tag{4.2}$$

where

$$\begin{aligned} H &= \sum_x \sum_y \left(\nabla J \frac{\partial W}{\partial \zeta} \right)^T \nabla J \frac{\partial W}{\partial T} && (4.3) \\ \delta_{IJ} &= I(W^{-1}(x, y; \zeta)) - J(x, y), && (4.4) \end{aligned}$$

and where ∇J is the 1×2 vector containing the x and y gradients of image J. $\Delta\zeta$ is computed incrementally until the algorithm converges, or a maximum number of iterations has been reached. For efficiency, the inverse compositional algorithm [3] is used.

The above formulation is valid for any warp function. Since a global, featureless mapping technique that warps an entire image into another is desired, an affine warp to more accurately model the relationship between the two images is used:

$$\tilde{W}(x, y; \zeta) = \underbrace{\begin{bmatrix} R_{xx} + 1 & R_{xy} & a_x \\ R_{yx} & R_{yy} + 1 & a_y \\ 0 & 0 & 1 \end{bmatrix}}_{\text{affine warp}} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}, \tag{4.5}$$

which in inhomogeneous coordinates is

$$W(x, y; \zeta) = \begin{bmatrix} x(R_{xx} + 1) + yR_{xy} + a_x \\ xR_{yx} + y(R_{yy} + 1) + a_y \end{bmatrix}. \tag{4.6}$$

Differentiating leads to

$$\frac{\partial W}{\partial \zeta} = \begin{bmatrix} x & y & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x & y & 1 \end{bmatrix}. \tag{4.7}$$

A projective warp was also tried, but it incured considerable computational expense with no appreciable difference in the results.

4.3.2 Point Matching

Projective data association (PDA) [49] establishes correspondence by finding the closest point from the other cloud as projected onto the image plane. In a similar manner, Lucas-Kanade data association (LKDA) also finds the closest point on the image plane, but only after first transforming the projected coordinates according to the warp function found by Lucas-Kanade:

$$L(i) = \arg\min_j \|\varphi(C_Q\tilde{q}_j) - \varphi(\tilde{W}(C_Q\tilde{p}_i;\zeta))\|, \tag{4.8}$$

where φ dehomogenizes the coordinates.

Unlike traditional ICP, the LKDA point matching step is performed only once per pair of image frames. The correspondences found in this step are then used in each iteration of the alignment process. Correspondences are found by computing the parameters, ζ, that will warp image I to J. Once these parameters are found, a correspondence map C_{map} is generated so that finding a corresponding point is simply a lookup. Once a correspondence map is generated the 3D Euclidean transformation can be iteratively estimated, T, between the two point clouds.

4.3.3 Rejection

This process is presented in Algorithm 1. Point correspondences are rejected in a similar fashion as described in [58], where τ_{dist} and τ_{ang} are the distance (0.1m) and angular thresholds (20°), respectively. In addition the constraint of requiring the

Algorithm 1 LKDA Point Cloud Alignment

```
// Generate correspondence map ($C_{map}$)
$\zeta \leftarrow$ perform Lucas-Kanade
for each pixel $(x, y)$ in image domain do
    $C_{map}(x, y) \leftarrow W(x, y; \zeta)$
end for
// Compute vertex and normal maps ($V_P$, $V_Q$, and $N$)
for each pixel $(x, y)$ in the depth maps $d_P$ and $d_Q$ do
    $V_P(x, y) \leftarrow Proj^{-1}(x, y, d_P(x, y))$
    $V_Q(x, y) \leftarrow Proj^{-1}(x, y, d_Q(x, y))$
    $N(x, y) \leftarrow$ normal vector of $Q(x, y)$
end for
// Compute alignment
$T \leftarrow$ identity $4 \times 4$ Euclidean transformation
while not aligned do
    for each pixel $(x, y)$ in depth map $d_P$ do
        $p \leftarrow TV_P(x, y)$
        $q \leftarrow V_Q(C_{map}(x, y))$
        $n \leftarrow N(C_{map}(x, y))$
        if $||p - q|| > \tau_{dist}$ or $||n \times p||/||p|| > \tau_{ang}$ or
            $||I(x, y) - J(W(x, y; \zeta))|| > \rho$
         then
            reject point correspondence
        end if
    end for
    $T^{(k)} \leftarrow$ solve linear system
    Update $T \leftarrow T^{(k)} \cdots T^{(2)}T^{(1)}$
end while
```

corresponding points to have a similar color is added:

$$||I(x, y) - J(W(x, y; \zeta))|| > \rho, \tag{4.9}$$

where ρ is a predefined color threshold, defined to be 25. The rejected points in each iteration of the alignment process are not necessarily the same, so that a different set of points contribute to the error metric in each iteration.

4.4 Automatic Selection of the Error Metric

Once correspondence has been found between the two image frames, the two point clouds can be aligned. Two common choices of the metric used for alignment are point-to-point and point-to-plane [11]. If there is sufficient geometric texture in the environment, then the latter converges faster than the former [70], which explains its adoption by KinectFusion. On the other hand, if all point correspondences are on a flat wall then the point-to-plane error metric will not result in correct alignment, because there will be no mechanism to induce a lateral shift between the point clouds in the direction perpendicular to the normals (see Figure 4.5).

To overcome this limitation a linear combination of the 3D Euclidean transformations computed using the point-to-point (T_{pp}) and the point-to-plane ($T_{p\pi}$) error metric is employed:

$$T = (1 - \lambda)T_{pp} + \lambda T_{p\pi}, \tag{4.10}$$

where $0 \leq \lambda \leq 1$ is a weighting factor. The transformation T_{pp} is computed in closed-form using orthogonal Procrustes analysis, while $T_{p\pi}$ is computed iteratively as described in [49].

The value of λ is determined automatically at run time by solving for the condition number, κ, of the covariance matrix of the geometric normals seen by the current frame. In areas of low geometry the value of κ^{-1} will be low, and conversely in areas of high geometry it will be high. λ is modeled as a sigmoid

$$\lambda = \left(1 + e^{-\alpha(\gamma - \hat{\gamma})}\right)^{-1} \tag{4.11}$$

where α is a large constant, found experimentally to be 300, $\gamma \equiv \kappa^{-1}$, and $\hat{\gamma}$ indicates where the two terms are weighted equally. To find $\hat{\gamma}$, for each frame over several

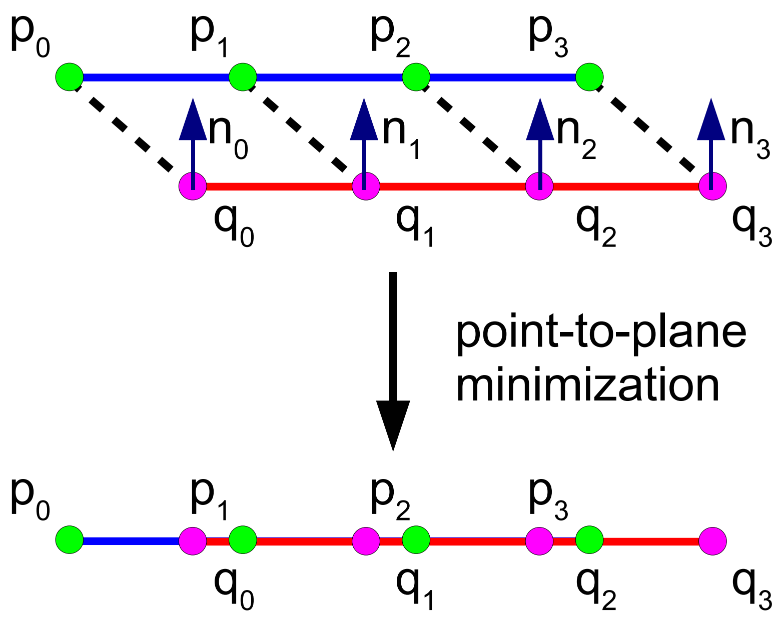


Figure 4.5: Illustration showing how the point-to-plane error metric would result in incorrectly aligned point clouds despite having perfect correspondence and no noise in the system.

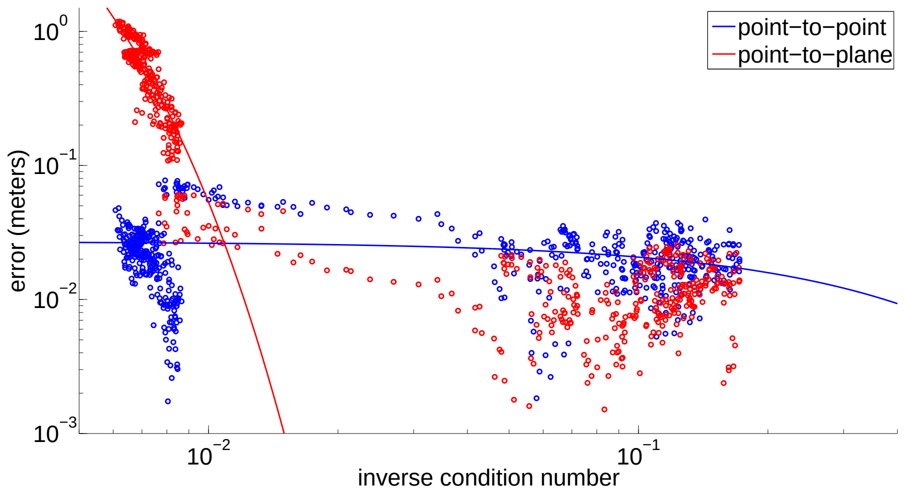


Figure 4.6: Scatter plot of γ, the inverse of the condition number of the covariance matrix of the normals, versus the error of the estimated translation (averaged over all frames of several sequences). The red markers are the errors computed when using the point-to-plane error metric to estimate odometry, while the blue markers are from the point-to-point error metric. Robust exponential curves were fit to both data sets and the intersection of these two curves yield $\hat{\gamma} \approx 0.01$.

sequences $T_{p\pi}$, T_{pp}, and κ^{-1}, were computed and compared the former two with the ground truth. The resulting scatter plot can been seen in Figure 4.6, along with robust exponential curves fit to both sets of data; the intersection of the two curves yielded $\hat{\gamma} = 0.01$. In addition, a plot of the cumulative camera tracking errors from two different scenes can be seen in Figure 4.7, which again illustrates that areas of low geometry exhibit significantly less tracking error when employing the point-to-point error metric, while areas of high geometry tend to do slightly better with the point-to-plane error metric.

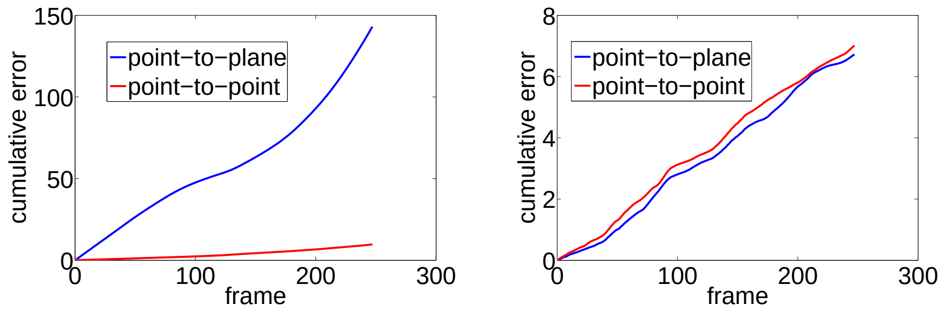


Figure 4.7: **Left**: Cumulative camera tracking error when reconstructing a flat environment (low geometry). **Right**: Cumulative camera tracking error when reconstructing a rich environment (high geometry).

4.5 Experimental Results

To evaluate the effectiveness of the proposed approach, along with the ability of the error metric selection to handle a variety of scenes, a set of experiments comparing the results of LKDA with the standard KinectFusion algorithm along with results from [86] and [42] were conducted. We modified the open-source version of KinectFusion known as *KinFu*, which is found in the open-source PCL library [71], with our own C++ and GPGPU code. The experiments compare the unmodified version of the code with four different modified versions in which LKDA uses either color or grayscale images, and the warp allowed is either translation or affine. (As mentioned earlier, no significant difference between affine and projective were found, though the latter requires significantly more computation.)

To allow quantitative comparison, data obtained from the RGB-D SLAM Dataset and Benchmark [79],[1] was used, which provides synchronized depth and color data from the Kinect or Xtion sensor as well as ground truth sensor trajectory from a high-accuracy motion capture system in several different environments. For these ex-

[1]http://vision.in.tum.de/data/datasets/rgbd-dataset

periments, 5 different datasets exhibiting a variety of small environments were tested: fr1/xyz, fr1/rpy, fr1/floor, fr1/desk2, and fr3/nostructure_texture_near_withloop (shortened as fr3/ntnw), where "fr" stands for "Freiburg". These sequences can roughly be categorized as one of the following:

- *high-geometry.* The environment contains a large amount of geometric variation throughout, enabling geometry-based sensor tracking. (fr1/xyz, fr1/floor, fr1/desk2)
- *planar.* The environment is largely planar (that is, without much geometric variation), making geometry-based sensor tracking difficult. (fr3/ntnw)
- *out-of-bounds.* For at least some frames of the sequence, the sensor is placed so that a majority of the depth values are outside the TSDF for at least some frames. (fr1/rpy)

The results of the 5 algorithms on these 5 datasets are shown in Table 4.1 which shows the position drift in meters per second and orientation drift in radians per second. For the high-geometry sequences, all the methods succeed, and there is little difference in accuracy in either position or orientation. For the planar or out-of-bounds sequences, however, the standard KinectFusion algorithm fails to maintain sensor tracking over the life of the sequence. While the translation-only version of LKDA works well even when many of the sensor readings are out-of-bounds of the TSDF, it is not sufficient to handle planar scenes. In contrast, the affine version of LKDA succeeds in both types of environments. There is little difference between grayscale and color versions, thus indicating that chrominance information is not important for this task.

A plot of the odometry from each algorithm for fr1/rpy is shown in Figure 4.9, as well as the reconstruction of the environment in Figure 4.11. The odometry plots

	fr1/xyz	fr1/floor	fr1/desk2	fr3/ntnw	fr1/rpy
Kinect Fusion	0.026 (1.76)	.078 (3.30)	0.096 (4.11)	0.791 * (26.40) *	1.864 * (103.53) *
LKDA Gray Translation	0.026 (1.76)	0.089 (3.60)	0.096 (4.03)	0.534 * (30.56) *	0.061 (3.28)
LKDA Color Translation	0.026 (1.76)	0.089 (3.60)	0.096 (4.03)	0.535 * (30.54) *	0.060 (3.35)
LKDA Gray Affine	0.026 (1.76)	0.077 (3.23)	0.095 (4.06)	0.033 (1.78)	0.057 (3.27)
LKDA Color Affine	0.026 (1.76)	0.077 (3.23)	0.095 (4.06)	0.034 (1.80)	0.057 (3.26)

Table 4.1: RMSE camera tracking error. Each cell shows the RMSE translational drift in meters per second (top), and absolute RMSE rotational drift in degrees per second (bottom, in parentheses). Failed tracking results are indicated by an asterisk (*).

were generated by the absolute trajectory error tool for evaluating error of estimated trajectories by comparing to ground truth [79].[2] These plots show that KinectFusion with PDA is unable to maintain tracking in environments in which the depth data exceeds the bounds of the TSDF, while all variations of the LKDA algorithm are able to maintain camera tracking.

Finally in planar scenes, as expected, KinectFusion with PDA is unable to maintain tracking due to the lack of geometric features to align. However, when using LKDA both the color and gray versions of the affine LKDA alignment process are able to maintain camera tracking. A plot of the odometry from each algorithm for this environment can be seen in Figure 4.8 as well as the reconstruction of the environment in Figure 4.10. This experiment not only shows a restriction to PDA but it also shows the need to estimate at least an affine warp with Lucas-Kanade. It also shows that the addition of color information does not have an appreciable difference to only using gray scale information. The reconstruction from PDA is the last frame

[2]http://vision.in.tum.de/data/datasets/rgbd-dataset/tools#evaluation

of the test sequence frame since PDA estimated that the camera remained relatively stationary while the actual camera motion can be clearly seen in reconstruction with LKDA.

In addition to the above experiments, we also directly compared our method with the approach of Whelan et al. [86] and the Dense Visual Odometry (DVO) algorithm of Kerl et al. [42]. Results of these comparisons are displayed in Tables 4.2 and 4.3, respectively. While the algorithm of [86] performs better on a frame-to-frame basis, our approach yields a better overall estimated trajectory, leading to lower absolute trajectory errors. Compared with [42], our approach is competitive, yielding the lowest error on several sequences.

Finally, LKDA was integrated into a 2D SLAM system that had previously only used the depth values to obtain a robot trajectory and map. Output from the wheel odometry of the robot was used as a starting point for the LKDA algorithm. This starting point was achieved by taking the previous colored point cloud and warping it by a Euclidean transformation calculated from the wheel odometry. This point cloud was backprojected onto the current depth and color image planes to form a depth and color image that reflected the motion of the robot estimated by the wheel odometry. This new depth and color image were aligned to the current depth and color image using LKDA. The warp computed when using LKDA was composed with the warp from the wheel odometry to obtain the final transformation. Results from this integration can be seen in Figure 4.12.

Runtimes for the different variations of LKDA can be seen in Figure 4.13. On our test platform the cloud alignment algorithm implemented in KinectFusion runs at 20Hz while others, such as [86] , have test platforms in which KinectFusions runs at 100Hz. So, in addition to the absolute runtime per frame we give the Lucas-Kande to KinectFusion cloud alignment runtime ratio, which is platform independent, for

	LKDA			Whelan		
	ATE		RPE	ATE		RPE
	med	max	RMSE	med	max	RMSE
fr1/desk	**0.021**	**0.049**	**0.040**	0.068	0.231	**0.040**
fr2/desk	**0.026**	**0.050**	0.041	0.118	0.346	**0.021**
fr1/room	**0.041**	**0.079**	**0.075**	0.152	0.419	0.076
fr2/lnl	**0.219**	**0.555**	0.553	0.309	1.032	**0.165**

Table 4.2: Comparison with Whelan et al. [86], showing median and maximum values of the Absolute Translation Error (m), and RMSE translational drift (m/s).

	LKDA		DVO	
	ATE (m)	RPE (m/s)	ATE (m)	RPE (m/s)
fr1/desk	0.022	0.040	**0.021**	**0.030**
fr1/desk2	**0.032**	0.095	0.046	**0.055**
fr1/room	**0.045**	0.075	0.053	**0.048**
fr1/360	**0.075**	**0.114**	0.083	0.119
fr1/teddy	0.057	0.100	**0.034**	**0.067**
fr1/floor	**0.048**	**0.077**	−−	0.090
fr1/xyz	0.017	0.026	**0.011**	**0.024**
fr1/rpy	0.039	0.050	**0.020**	**0.043**
fr1/plant	0.054	0.083	**0.028**	**0.036**

Table 4.3: Comparison with DVO [43], showing the RMSE Absolute Trajectory Error (m) and the RMSE translational drift (m/s).

	pt-pt	pt-pl
High Geometry		
fr1/rpy	1.097	0.039
fr1/floor	0.058	0.048
fr1/teddy	0.178	0.057
fr1/xyz	0.024	0.017
fr1/plant	0.234	0.054
Low Geometry		
fr3/ntnw	0.020	0.479
fr3/ntf	0.033	0.452

Table 4.4: Comparison of the absolute trajectory error (meters) of the point-to-point error metric (left) against the point-to-plane error (right). Scenes of low geometry the point-to-point error metric performed better while in scenes with high geometry the point-to-plane error metric performed better.

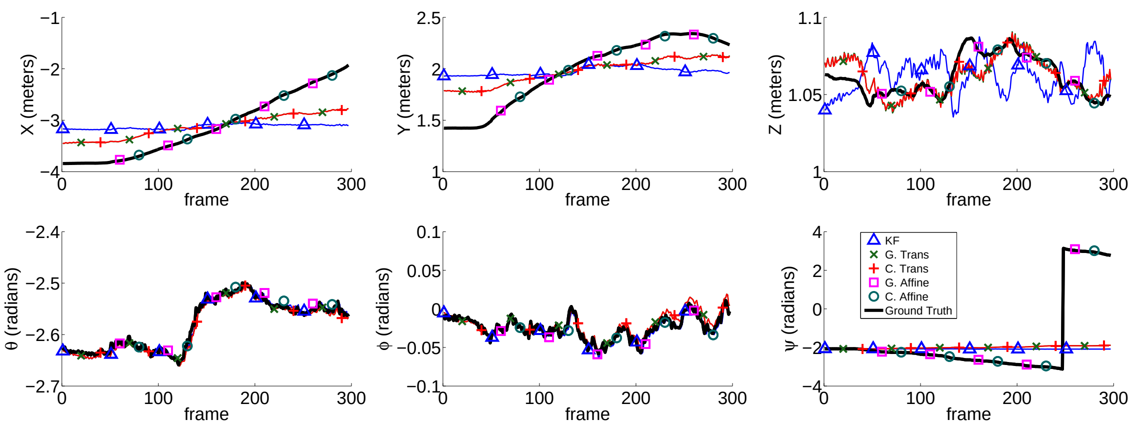


Figure 4.8: Camera pose estimation in a planar environment (fr3/ntnw). The plots show the following: ground truth (solid black line), KinectFusion with PDA (blue triangles), grayscale translation LKDA (green x's), color translation LKDA (red crosses), grayscale affine LKDA (magenta squares), and color affine LKDA (teal circles). The Gray Affine and the Color Affine are able to maintain camera tracking while the translation variations of LKDA, along with KinectFusion, are unable to handle a scene with a strictly planar environment.

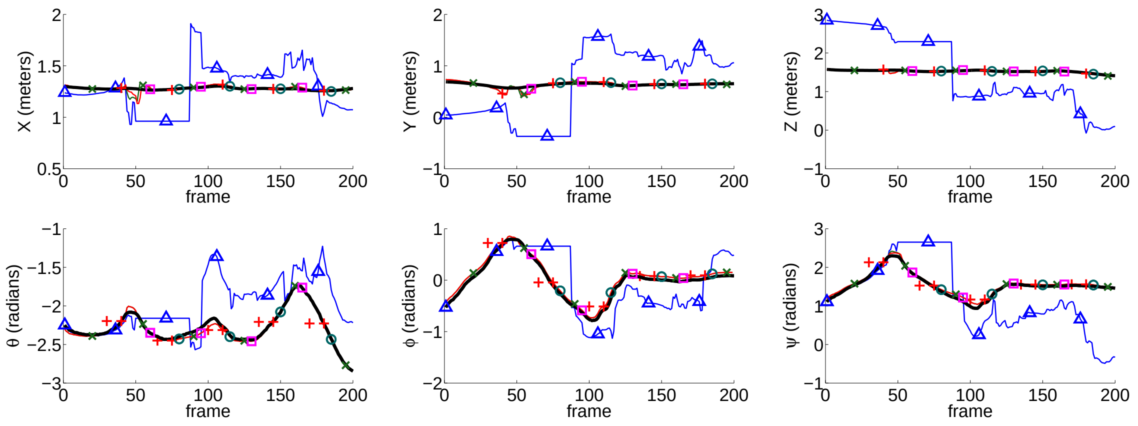
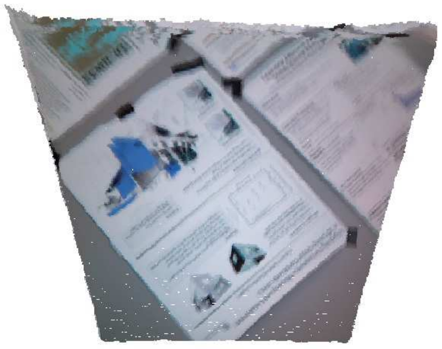
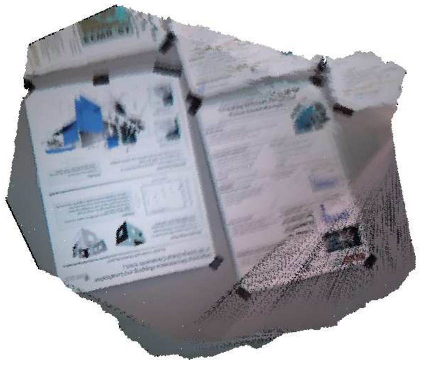


Figure 4.9: Camera odometry in an out-of-bounds environment (fr1/rpy). The legend is the same as the previous figure. When using LKDA for correspondence, regardless of variation, tracking can be maintained when the majority of depth readings are beyond the bounds of the TSDF. KinectFusion is unable to handle the lack of geometrical information which causes it to lose camera tracking.

a fair comparison against others runtimes. While the cloud alignment algorithm in KinectFusion uses PDA for correspondence, and LKDA does not, the computational cost for PDA is neglible due to the parallel implementation on the GPU.

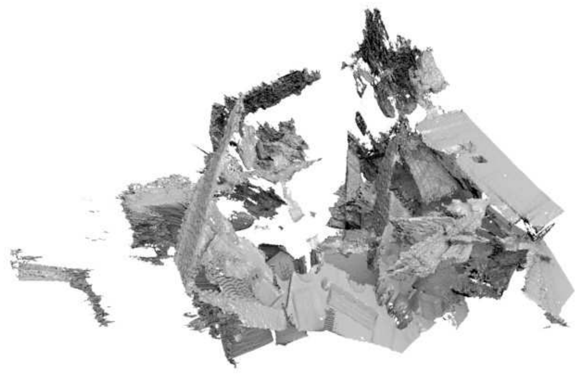


KinectFusion with PDA

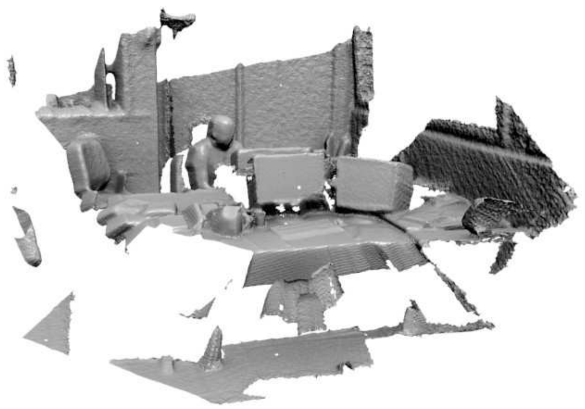


KinectFusion with LKDA

Figure 4.10: A planar environment (flat wall) with almost no geometric information (fr3/ntnw) reconstructed using standard KinectFusion (top) and the grayscale affine LKDA approach (bottom). Because of the lack of geometry, the former was unable to estimate the camera's pose and therefore computed a relatively motionless pose, resulting in a reconstruction that was simply the last frame of the sequence. In the LKDA approach, the camera's motion is accurately computed.

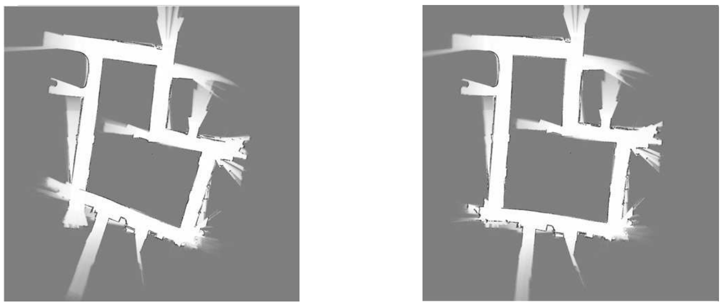


KinectFusion with PDA



KinectFusion with LKDA

Figure 4.11: An environment in which the camera was oriented in a way such that a majority of the depth values went beyond the boundaries of the TSDF volume (fr1/rpy). The inability of standard KinectFusion (top) to handle depth values outside the TSDF volume caused a severe loss in camera tracking, thus resulting in a poor reconstruction. The grayscale affine LKDA approach (bottom) resulted in a much more accurate reconstruction.



Position Error = 0.048 m
Rotation Error = 0.054 rads

Position Error = 0.038 m
Rotation Error = 0.048 rads

Figure 4.12: Results of 2D SLAM system without (left) and with (right) LKDA used for camera tracking. The use of LKDA reduces the amount of drift incured in the system.

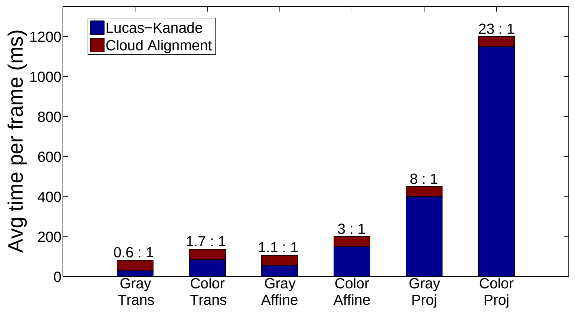


Figure 4.13: Runtime, in ms, of each LKDA variant. Timings are split into average time to run Lucas-Kanade (blue) and the average time to run align the clouds after correspondences have been determined (red) which is .05 ms. The Lucas-Kanade time shows the time used to calculate correspondence while the Alignment time is the time needed to bring point clouds into alignment given a correspondence map. The addition of these two times is the total average time, per frame, to track the camera. The Lucas-Kanade to cloud alignment ratio is given to allow for fair comparisons with different algorithms run on different platforms.

Chapter 5

Data Storage and Integration

5.1 Overview

Previous chapters in this dissertation have discussed how to maintain camera tracking and how to efficiently store large scale environments. Once the current pose of the camera is deterimed, its data is used to update the reconstructed environment. This chapter will discuss how to fuse the current depth readings, obtained from the RGBD camera, into the reconstructed environment given the pose estimated by the camera tracking algorithm.

Discussion of the use of two different volumetric representations of an environment has been made in this dissertation, the truncated signed distance function (TSDF) (Chapter 4) and a 3D occupancy grid stored as an octree (Chapter 3). The TSDF is chosen to be the final representation of a reconstruction environment for two reasons: it retains more detail and the camera tracking algorithm employed in this dissertation uses a TSDF, a more detailed comparisons between the TSDF and 3D occupancy grids is presented in the following section. However, there has been no discussion in this dissertation on how to efficiently save TSDFs in order to allow for the reconstruction of large scale environments, so this chapter will also discuss how to represent TSDFs using an octree.

5.2 TSDF vs 3D Occupancy Grid

While both the 3D occupancy grid and the TSDF are voxel representations of an environment the resulting representation of any model stamped into these volumes is substaintially different, the ouccpancy grid saves space while the TSDF reduces the loss of detail. An illustration of these differences can be seen in Figure 5.1 and a reconstruction of real data comparing the two data structures can be seen in Figure fig:chair. In chapter 3 of this dissertation, the space saving attributes of an octree in order to hold large environments were studied. The octrees in chapter 3 held binary 3D occupancy grids where each cell was labeled free or occupied. While this method of storing the environment was memory efficient and easily maintained it lacks a continuous representation of the environments surfaces. Given the discrete nature of the occupancy grid any environment stamped into that grid would inevitably be discretized to the same resolution as the grid, losing fidelity that may be desired later on.

Alternatively, in chapter 4 TSDFs were employed to represent the reconstructed environment. The TSDF is a continuous representation of the surfaces of an environment which allows for the retention of finer detail. This retention of detail allowed for accurate camera tracking and highly detailed reconstructions. However, while maintaining accurate 3D reconstructions, the size of the environment was limited to the bounds of the TSDF, which, due to memory constraints, has to remain relatively small. The TSDF representation is only continuous in areas which have had sensor readings. If a voxel in the TSDF has had no sensor reading then it will be undefined in that area and subsequently is not continuous, depending on the resolution of the sensor and the resolution of the TSDF this can lead to small holes in a scene reconstruction.

Additionally a 3D occupancy grid is more sensitive to noisy measurements. Once a cell has been marked as occupied it will stay occupied unless the map is marginalized in order to remove noisy readings. While there are techniques to do this, such as Markov Chain Monte Carlo (MCMC), it would take considerable amounts of time to do this on large scale or dense 3D occupancy grid. After marginalization the occupancy grid will contain values that represent the probability that a cell is occupied, causing it to lose its binary state, and ultimately leading to larger memory costs. Local TSDFs created by a camera can be fused together, as described in Section 5.3, using a running weighted average which effectively marginializes out the majority of noisy readings. The sensitivity of 3D occupancy grids to noisy readings can be seen in Figure 5.2.

5.3 Fusing TSDFs

5.3.1 TSDF From Depth Image

Data obtained from an RGBD sensor is a color image and a depth image. To recover the surfaces of the environment the information from the depth image needs to be integrated into the current TSDF that represents the environment. However, data provided by a depth image cannot be directly fused into the current reconstruction, it must first be transformed into a local TSDF of the environment as seen by the camera. This local TSDF can then combined with the global TSDF to provide an update to the reconstruction. To create a local TSDF each pixel in a depth image,

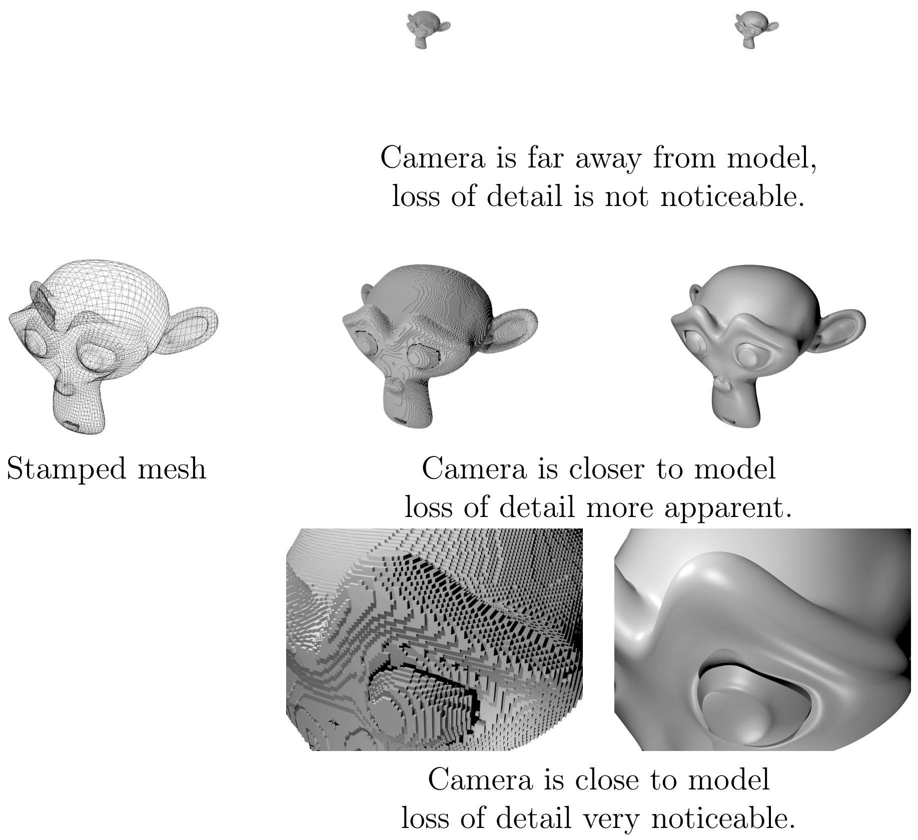

Camera is far away from model,
loss of detail is not noticeable.
Stamped mesh
Camera is closer to model
loss of detail more apparent.
Camera is close to model
loss of detail very noticeable.

Figure 5.1: A 3d mesh monkey (left) stamped into a 512^3 binary 3D occupancy grid (middle) and TSDF (right). The amount of memory used to save the binary 3D occupancy grid is 16.7 MB while the TSDF used 536.87 MB. While the occupancy grid used much less memory to store the model it loses detail that the TSDF is able to retain. The loss of detail becomes more apparent the closer to the model the camera gets.

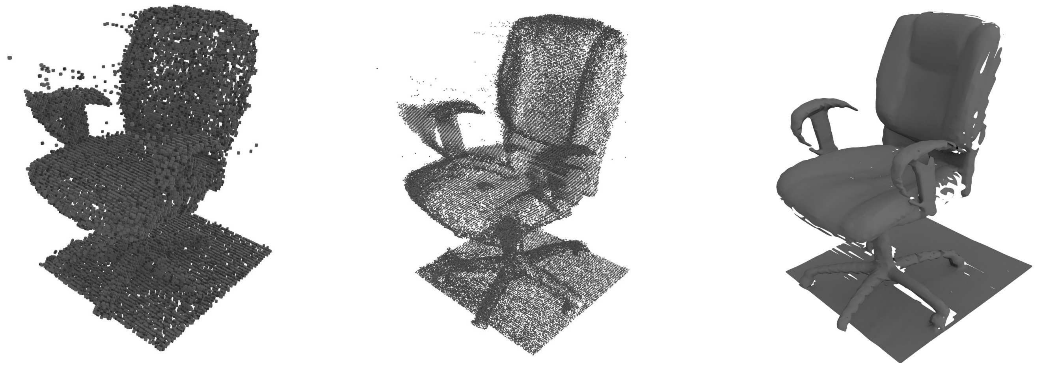


Figure 5.2: A reconstruction of a typical office chair. The chair was reconstructed using a 3D occupancy grid (left and middle) and a TSDF (right). The left occupancy grid and the TSDF reconstructions had a resolution of 1cm while the middle occupancy grid had a resolution of 5 mm. It is clear that the TSDF provides a more detailed reconstruction and does not contain as much noise as the 3D occupancy grid. When the left occupancy grid and the TSDF representations were stored in an octree the 3D occupancy grid consumed 1.5 MB of memory and the TSDF consumed 3.6 MB of memory. The middle occupancy grid consumed 2.96 MB. Without the use of an octree the left 3D occupancy grid would have required 12.5 MB of memory to store and the TSDF would have required 402.65 MB of memory to store when using volumes that were 512 x 384 x 512 voxels in size. The octree provided 8.3 : 1 compression ratio for the 3D occupancy grid and a 111.85 : 1 compression ratio for the TSDF.

D, determines the value of all the voxels that lie on the ray cast through that pixel from the camera. Each pixel in a depth image gives the orthogonal distance from the camera to the nearest surface, an illustration of this can be seen in Figure 5.3. Each voxel in a TSDF represents the signed distance to the nearest surface. The value for each voxel, V, in the TSDF can be obtained as follows

$$\begin{aligned}
P &= \omega(V) && (5.1)\\
\omega(V) &= \begin{bmatrix} (V_{\text{row}} - \text{size}/2) * \text{resolution} \\ (V_{\text{col}} - \text{size}/2) * \text{resolution} \\ (V_{\text{slice}} - \text{size}/2) * \text{resolution} \end{bmatrix} && (5.2)\\
d &= \begin{bmatrix} \frac{V_x f_x}{V_z} + c_x \\ \frac{V_y f_y}{V_z} + c_y \end{bmatrix} && (5.3)\\
\text{TSDF(V)} &= min(\lambda^{-1}||P||_2 - D(d_x, d_y), \mu) && (5.4)\\
\lambda &= ||K^{-1}[d_x, d_y, 1]^T||_2 && (5.5)
\end{aligned}$$

where K is a constant single camera calibration matrix, μ is the maximum distance from a surface to be considered, and λ^{-1} converts the ray distance to a point P to a depth. $\omega(V)$ takes a voxel in the TSDF which is indexed by its row, col, and slice and computes the 3D point of that voxel based on the resolution of the TSDF.

5.3.2 Upating Global TSDF

Once a local TSDF is computed from the current depth reading the global TSDF can be updated with this local TSDF given an estimate of the camera pose. The local TSDF, F_{loc}, is transformed by the current estimate of the pose, T, to be in

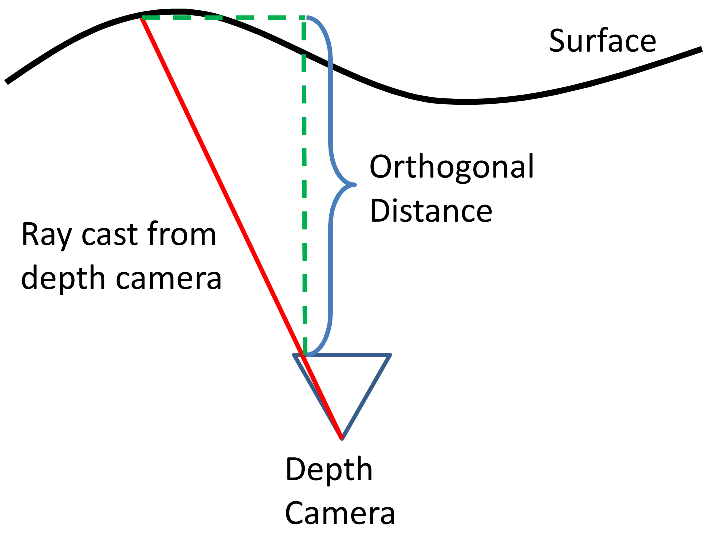


Figure 5.3: Illustration depicting the values stored at each in pixel in a depth image.

the same coordinate frame as the global TSDF, F_{glob}. A voxel, V, in the global TSDF is updated by taking the weighted average of all voxels in F_{loc} that, when warped, are contained within a cubic region of space for which V is a corner of. Algorithm 2 provides psuedocode for this update process.

5.4 Storing TSDF in an Octree

The previous section of this chapter discusses the details of updating a TSDF volume with the current readings from a depth image. However, since the TSDF allocates memory for every voxel in a bounded space, the size of this space is limited by memory. This restriction was also seen in 3D occupancy grids and was overcome by storing them in an Octree. This section examines a similar approach for a TSDF.

In a 3D occupancy grid all occupied space was stored in an octree. In a TSDF there is no concept of free or occupied space. However, it is easy to detect which voxels in the TSDF are close to a surface, which is really what a 3D reconstruction

Algorithm 2 Update Global TSDF

```
for $v \epsilon F_{loc}$ do
    //Compute 3D point in $F_{loc}$ coordinate frame and warp to global coordinate frame
    $P = T\omega(v)$

    //Trilinearly Interpolate, $\eta$, point $P$ in $F_{glob}$
    [indices, weights] = $\eta(F_{glob}, P)$
    for i = 0:7 do
        $F_{glob}(\text{indices[i]})_{\text{update}} = F_{glob}(\text{indices[i]})_{\text{update}} + \text{weights[i]} * v$
        $F_{glob}(\text{indices[i]})_{\text{count}} = F_{glob}(\text{indices[i]})_{\text{count}} + 1$
    end for
end for
for $v \epsilon F_{glob}$ do
    $v = (v_{\text{weight}} * v + v_{\text{update}}/v_{\text{count}})/(v_{\text{weight}} + 1)$
    $v_{\text{weight}} = \arg\max(\text{max weight}, v_{\text{weight}} + 1)$
end for
```

is interested in representing. By storing all voxels that contain absolute distance less than μ, the value at which distances become truncated, a TSDF can be efficiently saved in memory, allowing for large scale reconstructions that retain a high level of detail. A pipeline of this approach can be seen in Figure 5.4.

By storing the TSDF in an octree large scale environments can be reconstucted with high fidelity. Figure 5.2 demonstrates the effectives of an octree version of the TSDF to save on space while not sacrificing detail.

5.5 Experimental Results

To compare the resulting reconstructions made when using a TSDF and a 3D occupancy grid several environments. These reconstructions were made using ground truth odometry data to ensure no errors from camera tracking or loop closing would affect the results. Two of the environments, Figures 5.5 and fig:teddy are reconstructions of typical office environments. Figure 5.7 provides a reconstruction in

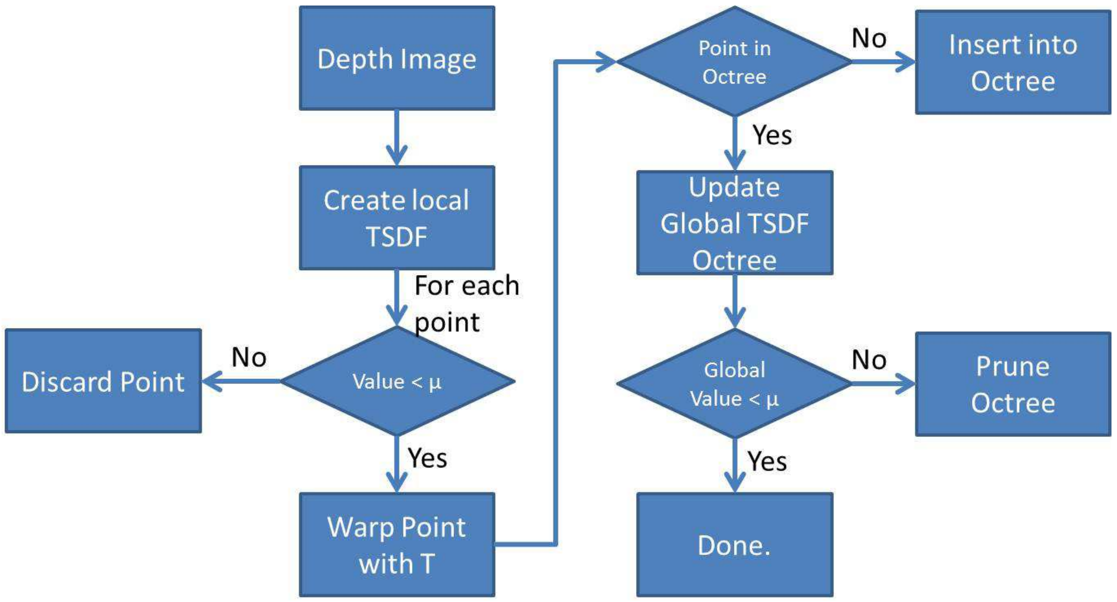


Figure 5.4: Pipeline of maintaining a TSDF representation of an environment in an octree.

a large scale environment. It can be seen in all of these reconstructions that the TSDF is able to create reconstructions with little noise, which leads to a smaller memory footprint, than the 3D occupancy grid. Additionally, unlike the 3D occupancy grid, the TSDF is able to retain a high level of detail.

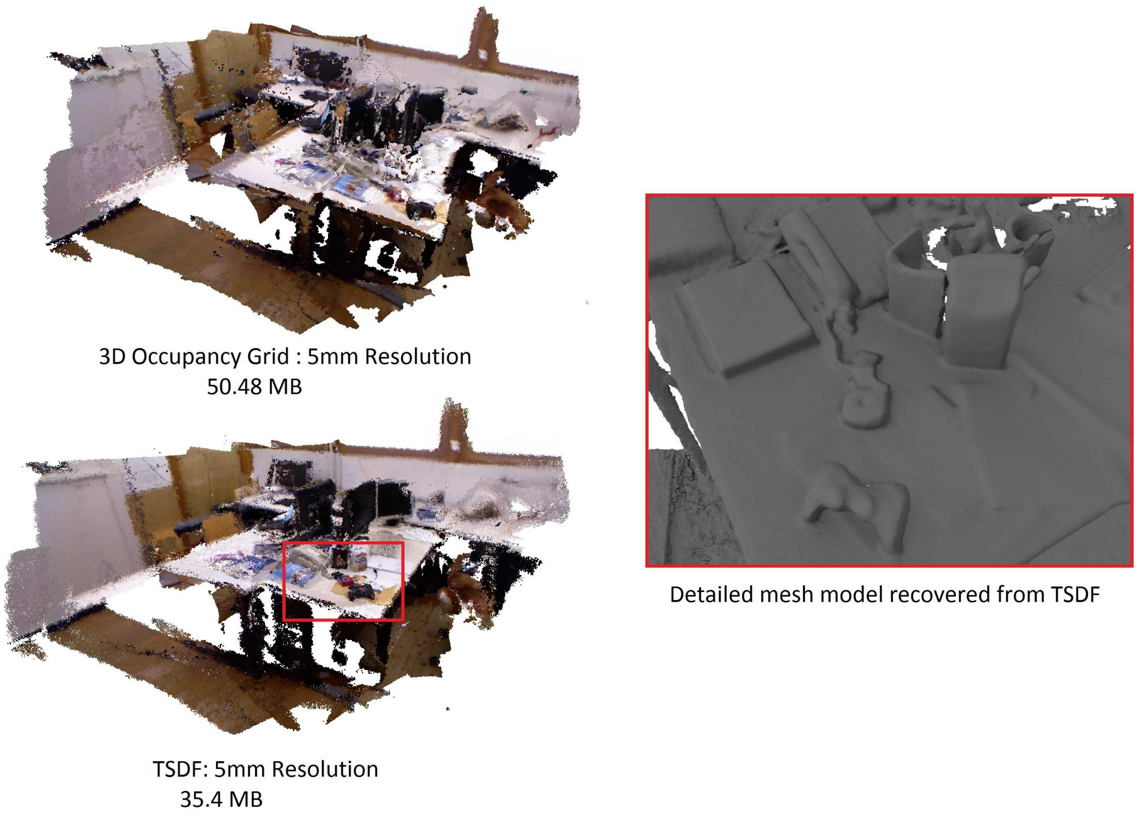

3D Occupancy Grid : 5mm Resolution
50.48 MB
TSDF: 5mm Resolution
35.4 MB
Detailed mesh model recovered from TSDF

Figure 5.5: Reconstruction of the fr1_desk1 dataset obtained from [79]. Reconstructions were made using ground truth data. The size of the environment is 4.34m x 4.37m x 1.93m (36.6m^3) and was stored at a 5mm resolution. The top left reconstruction was stored using a 3D occupancy grid and the bottom left reconstruction was stored using a TSDF. Both representations were saved in an octree. Without an octree the TSDF reconstruction would have required 1.1713 GB to store and the 3D occupancy grid would have required 292.8 MB. Due to the amount of noise seen in the 3D occupancy grid reconstruction the memory usage of the octree surpasses the memory required to store the TSDF. The TSDF reconstruction has less noise, less memory required to store the same space, and can be interpolated to generate high resolution mesh models of the environment (right).

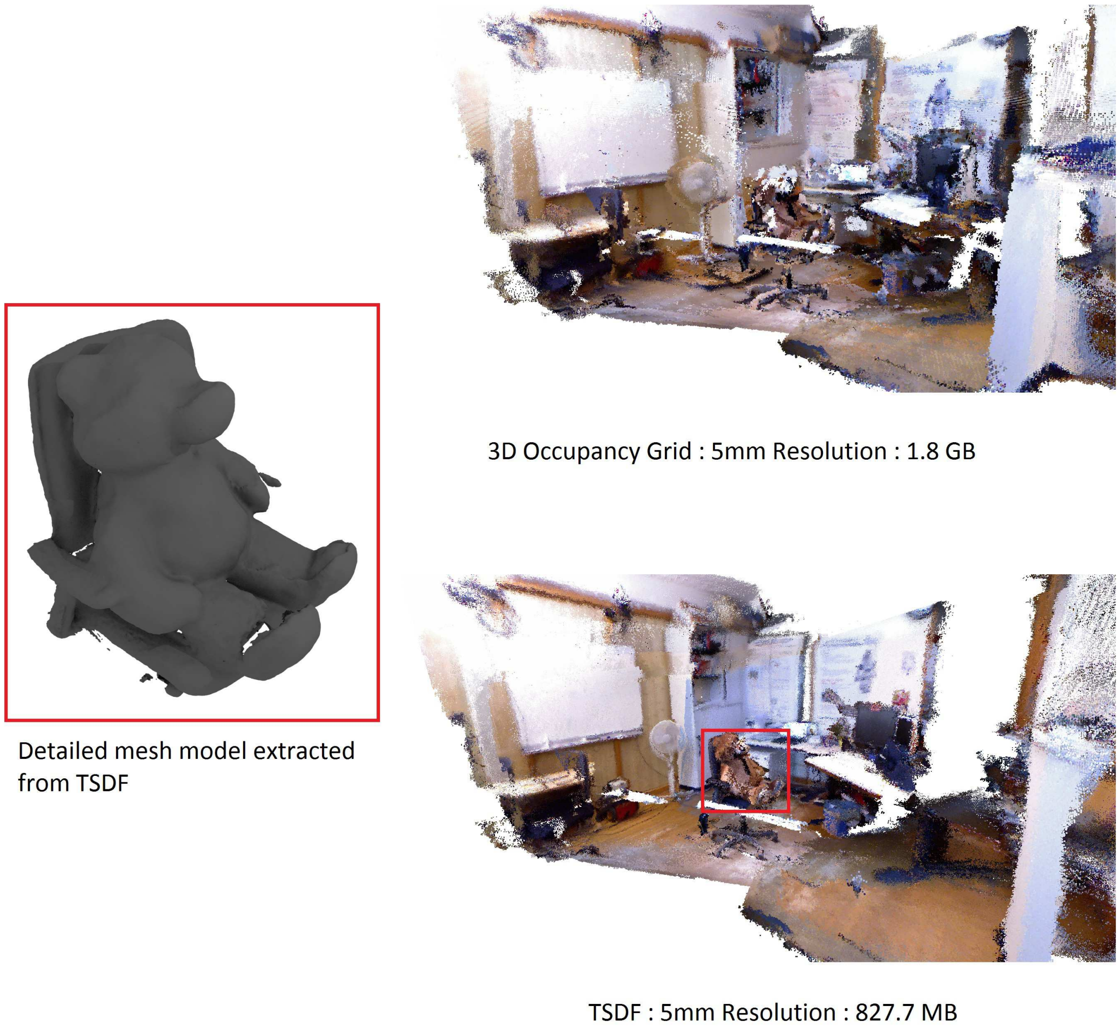
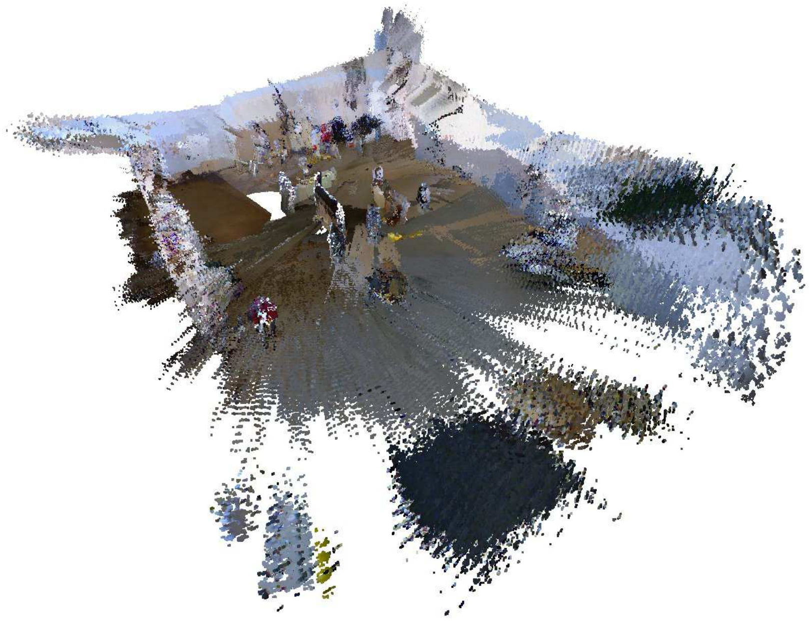
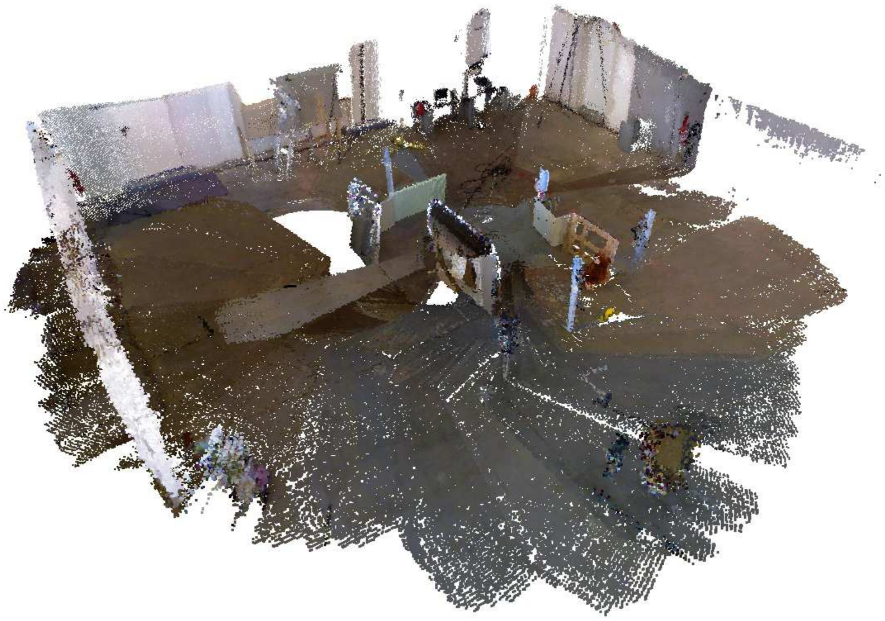

3D Occupancy Grid : 5mm Resolution : 1.8 GB
Detailed mesh model extracted from TSDF
TSDF : 5mm Resolution : 827.7 MB

Figure 5.6: Reconstruction of the fr1_teddy dataset obtained from [79]. Reconstructions were made using ground truth data. The size of the environment is 11.38m x 9.75m x 3.04m (337.3m^3) and was stored at a 5mm resolution. The top right reconstruction was stored using a 3D occupancy grid and the bottom right reconstruction was stored using a TSDF. Both representations were saved in an octree. Without an octree the TSDF reconstruction would have required 10.79 GB to store and the 3D occupancy grid would have required 2.70 GB. Due to the amount of noise seen in the 3D occupancy grid reconstruction the memory usage of the octree surpasses the memory required to store the TSDF. The TSDF reconstruction has less noise, less memory required to store the same space, and can be interpolated to generate high resolution mesh models of the environment (left).



3D Occupancy Grid, 50mm resolution, Stored in an octree with 497 MB of memory.



TSDF, 50mm resolution, Stored in an octree with 66 MB of memory.

Figure 5.7: The size of the environment is 13.84m x 13.22m x 2.62m (479.3m^3) and was stored at a 50mm resolution.

Chapter 6

Loop Closing

6.1 Overview

Previous chapters of this dissertation have discussed how to maintain camera tracking over large environments as well as how to efficiently represent a large scale 3D reconstruction in memory. Previous chapters have even discussed how to remove drift using a "Manhattan" world assumption when reconstruction building interiors. However, this assumption can not be used in all environments. This motivates an investigation of methods for optimizing a trajectory of a sensor so that when a loop closure is detected the trajectory can be optimized to remove any drift. In this chapter the problem of PoseSLAM is considered in order to optimize an estimated trajectory. PoseSLAM is the problem of simultaneous localization and mapping (SLAM) using only constraints between robot poses, in which only the robot poses (as opposed to landmark positions) are estimated. To a large extent, once the robot poses have been determined, a map of the environment can be created by overlaying the sensor data obtained at the poses. Assuming a graph-based approach, the primary problem in PoseSLAM is to optimize the graph in the presence of loop closure.

Given that the trajectory of a sensor is represented in a graph framework each node in the graph represents a state of the sensor and each edge represents a relative measurements between poses. In the context of 3D reconstruction, this measurement would be obtained from camera tracking and any detected loop closures. The state of each node can represent many things. If the states of the node represent a global pose of a sensor then the state space is said to be a Global State Space (GSS). In their influential work on PoseSLAM, Olson et al. [61] proposed the use of an incremental state space (ISS) where each node represented the change in state and each variable in the state is independent on the others. In this chapter the use of a Relative State Space (RSS) is proposed where each node also represents the change in state however the position variables are dependent upon the orientation variables. This approach is referred to as Pose Optimization using a Relative State Space (POReSS). Figure 6.1 illustrates the subtle yet important difference between an ISS and RSS.

This chapter also presents a two phase approach to graph optimization. The first phase uses a RSS and a stochastic gradient descent minimization technique that allows for fast minimizations even in the presence of poor initial conditions. The first phase, however, can take a long time to converge. The second phase, which uses a GSS and a graph based implentation of Gauss-Seidel, which is referred to as Graph-Seidel in this dissertation, allows for fast convergence when provided a good starting point. This two phase system allows for quick optimization of graphs even in the presence of poor initial conditions.

After showing results for the proposed two phase approach, the benefits of only optimizing the rotational component using a RSS during the first phase of an optimization are examined. This is referred to as rotational POReSS (rPOReSS). In addition an incremental implentation of rPOReSS called irPOReSS is proposed.

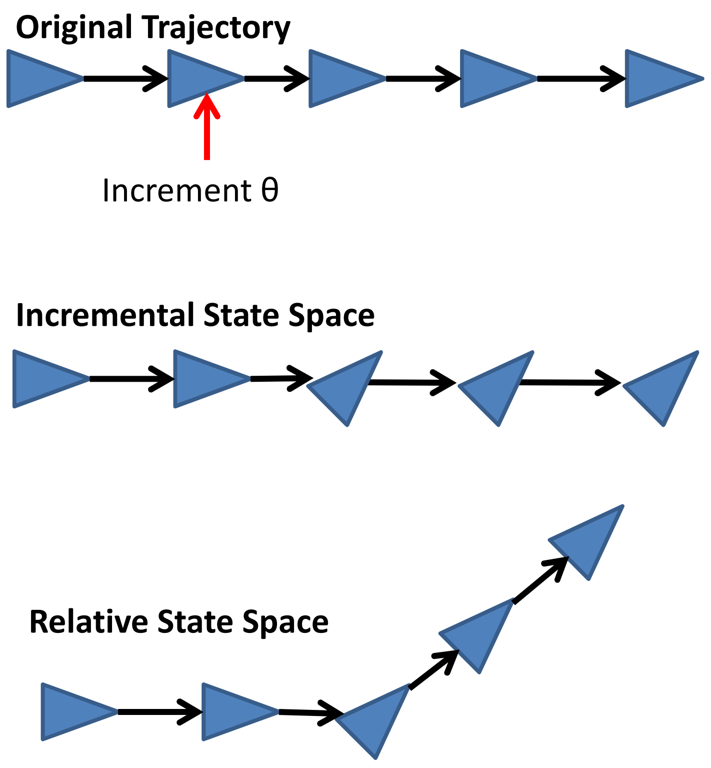


Figure 6.1: Illustration showing the difference between an incremental state space (ISS) and a relative state space (RSS). The top figure shows some trajectory with one of the nodes changing its orientation. In an ISS only the orientation of all subsequent nodes is changed while in a RSS both the orientations and positions of subsequent nodes are effected.

6.2 Graph-based SLAM

The graph-based approach to SLAM attempts to find the maximum likelihood configuration given a set of measurements. This section briefly reviews this approach, loosely adopting the notation of [27]. The graph is given by $\mathcal{G} = (\mathcal{P}, \mathcal{E})$ consisting of a set of vertices $\mathcal{P} = \{\mathbf{p}_i\}_{i=0}^{n}$ representing robot poses, and a set of edges $\mathcal{E}$ between pairs of robot poses. Assuming a ground-based robot rolling on a horizontal floor plane, the ith pose is given by $\mathbf{p}_i = \begin{bmatrix} x_i & y_i & \theta_i \end{bmatrix}^T$, where $\begin{bmatrix} x_i & y_i \end{bmatrix}^T \in \mathbb{R}^2$, and $\theta_i \in SO(2)$. Typically the poses are traversed in a sequential manner, so for convenience this sequence is stacked into the vector $\mathbf{p} = \begin{bmatrix} \mathbf{p}_1^T & \cdots & \mathbf{p}_n^T \end{bmatrix}^T$. These poses are in a global coordinate system fixed by convention at $\mathbf{p}_0 \equiv \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T$.

Let $\mathbf{x} = \begin{bmatrix} \mathbf{x}_1^T & \cdots \mathbf{x}_n^T \end{bmatrix}^T$ be a state vector that is uniquely related to the sequence of poses through a bijective function g such that $\mathbf{x} = g(\mathbf{p})$ and $\mathbf{p} = g^{-1}(\mathbf{x})$. In the simplest case $\mathbf{x}_i \equiv \mathbf{p}_i$, so that the states are equivalent to the global poses, but this is not required; as we shall see, the choice of state space can have significant impact upon the results.

Each edge $(a, b) \in \mathcal{E}$ captures a constraint δ_{ab} between poses $\mathbf{p}_a$ and $\mathbf{p}_b$ obtained by sensor measurements or by some other means. For ease of presentation at most one edge between any two given poses is assumed, but the extension to a multigraph is straightforward. The uncertainty of the measurement is given by the *information matrix* Ω_{ab}, which is the inverse of the covariance matrix. If $\mathbf{f}_{ab}(\mathbf{x})$ is let to be the zero-noise observation between poses $\mathbf{p}_a$ and $\mathbf{p}_b$ given the current configuration $\mathbf{x}$, then the discrepancy between the predicted observation and the actual observation is the *residual*:

$$\mathbf{r}_{ab}(\mathbf{x}) \equiv \delta_{ab} - \mathbf{f}_{ab}(\mathbf{x}). \tag{6.1}$$

Assuming a Gaussian observation model, the negative log-likelihood of the observation is given by the squared Mahalanobis distance

$$\epsilon_{ab}(\mathbf{x}) \propto \mathbf{r}_{ab}^T(\mathbf{x})\Omega_{ab}\mathbf{r}_{ab}(\mathbf{x}). \tag{6.2}$$

The goal of graph-based SLAM is to find the configuration $\mathbf{x}$ that minimizes the energy $\epsilon(\mathbf{x}) \equiv \sum_{(a,b)\in\mathcal{E}} \epsilon_{ab}$. This energy is a non-linear expression due to the orientation parameters, thus requiring an iterative approach. Let $\tilde{\mathbf{x}}$ be the current estimate for the state. The linearized energy about this current estimate is given by

$$\tilde{\epsilon}(\mathbf{x}) \equiv \sum_{(a,b)\in\mathcal{E}} \tilde{\mathbf{r}}_{ab}^T(\mathbf{x})\Omega_{ab}\tilde{\mathbf{r}}_{ab}(\mathbf{x}), \tag{6.3}$$

where the linearized residual function is given by the first-order Taylor expansion:

$$\tilde{\mathbf{r}}_{ab}(\mathbf{x}) \equiv \mathbf{r}_{ab}(\tilde{\mathbf{x}}) - J_{ab}(\tilde{\mathbf{x}})\underbrace{(\mathbf{x} - \tilde{\mathbf{x}})}_{\Delta\mathbf{x}}, \tag{6.4}$$

where $J_{ab}(\tilde{\mathbf{x}})$ is the Jacobian of the *error* $\mathbf{e}_{ab}(\mathbf{x}) \equiv -\mathbf{r}_{ab}(\mathbf{x})$ evaluated at the current state.

Expanding the linear system, rearranging terms, differentiating $\partial\tilde{\epsilon}(\mathbf{x})/\partial\Delta\mathbf{x}$, and setting to zero yields

$$\sum_{(a,b)\in\mathcal{E}} \Omega_{ab}\left(\mathbf{r}_{ab}(\tilde{\mathbf{x}}) - J_{ab}(\tilde{\mathbf{x}})\Delta\mathbf{x}\right) = 0. \tag{6.5}$$

Defining K as the matrix obtained by concatenating the Ω_{ab} horizontally, $\mathbf{r}(\tilde{\mathbf{x}})$ as the vector obtained by stacking $\mathbf{r}_{ab}(\tilde{\mathbf{x}})$ vertically, and $J(\tilde{\mathbf{x}})$ as the matrix obtained by stacking $J_{ab}(\tilde{\mathbf{x}})$ vertically, standard least squares system is obtained

$$K\mathbf{r}(\tilde{\mathbf{x}}) = KJ(\tilde{\mathbf{x}})\Delta\mathbf{x}. \tag{6.6}$$

Multiplying both sides by $(KJ)^T$ yields the so-called *normal equations*:

$$J^T(\tilde{\mathbf{x}})\Omega J(\tilde{\mathbf{x}})\Delta\mathbf{x} = J^T(\tilde{\mathbf{x}})\Omega\mathbf{r}(\tilde{\mathbf{x}}), \tag{6.7}$$

where $\Omega = K^T K$.

For reference let us pause to consider the dimensions of these matrices. Defining m to be the number of elements in the state vector, then $\mathbf{x}_i$ is an $m \times 1$ vector; typically for 2D pose optimization $m = 3$ due to the translation and orientation parameters. The vectors $\mathbf{x}$, $\tilde{\mathbf{x}}$, and $\Delta\mathbf{x}$ are all $mn \times 1$. Let m' be the number of elements in the observation δ_{ab}; typically $m' = m$. Then $\delta_{ab}(\mathbf{x})$, $\mathbf{f}_{ab}(\mathbf{x})$, $\mathbf{r}_{ab}(\mathbf{x})$, and $\tilde{\mathbf{r}}_{ab}(\mathbf{x})$ are all $m' \times 1$ vectors, Ω_{ab} is $m' \times m'$, and $\epsilon_{ab}(\mathbf{x})$ and $\tilde{\epsilon}_{ab}(\mathbf{x})$ are scalars. The Jacobian $J_{ab}(\mathbf{x})$ is $m' \times mn$. Defining $n' = |\mathcal{E}|$ be the number of edges in the graph, then Ω is $m'n' \times m'n'$, $\mathbf{r}(\mathbf{x})$ is $m'n' \times 1$, and $J(\mathbf{x})$ is $m'n' \times mn$.

6.3 State spaces

As mentioned earlier, the choice of state space can have a significant impact upon the results. In this section three different state spaces are described and their strengths and weaknesses are outlined. In all cases, $m' = m = 3$.

6.3.1 Global state space (GSS)

The most natural choice for state space is the global pose, that is, the pose of the robot in a global coordinate system:

$$\mathbf{x}_i \equiv \mathbf{p}_i = \begin{bmatrix} x_i & y_i & \theta_i \end{bmatrix}^T. \tag{6.8}$$

The use of a global state space (GSS) leads to a simple formulation of the energy of the system and subsequently a sparse Jacobian. However, since the GSS representation directly solves for the global poses, each node is only affected by the nodes to which it is directly connected. This causes slow convergence since changes will be propagated slowly and can easily be trapped in a local minimum if the initial conditions are poor.

6.3.2 Incremental state space (ISS)

Olson et al. [61] propose using the incremental state space, in which the state is the difference between consecutive poses:

$$\mathbf{x}_i \equiv \mathbf{p}_i - \mathbf{p}_{i-1} = \begin{bmatrix} x_i - x_{i-1} \\ y_i - y_{i-1} \\ \theta_i - \theta_{i-1} \end{bmatrix}, \qquad i = 1, \ldots, n, \tag{6.9}$$

with $\mathbf{x}_0 \equiv \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T$.

With an incremental state space, the ith pose is given by the sum of all states up to and including i:

$$\mathbf{p}_i = \sum_{k=0}^{i} \mathbf{x}_k. \tag{6.10}$$

This state space allows changes to be propagated through the system quickly because changing one state affects the global pose of all nodes past it. However, the coupling between the different elements has been lost, so that a change in orientation for one node does not directly affect the positions of the other nodes.

6.3.3 Relative state space (RSS)

Another alternative is to use a relative state space:

$$\mathbf{x}_i \equiv \begin{bmatrix} x'_i & y'_i & \theta'_i \end{bmatrix}^T, \tag{6.11}$$

with $\mathbf{x}_0 \equiv \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T$. The parameters x'_i, y'_i, and θ'_i describe the relative transformation between the $(i-1)$th and ith poses, specifically the ith pose in the $(i-1)$th coordinate frame. Assuming a righthand coordinate system with positive angles describing counterclockwise rotation:

$$\mathbf{p}_i = \mathbf{p}_{i-1} + \underbrace{\begin{bmatrix} \cos\theta_{i-1} & -\sin\theta_{i-1} & 0 \\ \sin\theta_{i-1} & \cos\theta_{i-1} & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{R(\theta_{i-1})} \underbrace{\begin{bmatrix} x'_i \\ y'_i \\ \theta'_i \end{bmatrix}}_{\mathbf{x}_i} \tag{6.12}$$

$$= \sum_{k=1}^{i} R(\theta_{k-1})\mathbf{x}_k, \tag{6.13}$$

where

$$\theta_b = \sum_{k=1}^{b} \theta'_k \tag{6.14}$$

is the global orientation, as mentioned earlier. Defining ${}^a\mathbf{p}_b$ as the relative pose between a and b, $a < b$, that is, pose b in coordinate frame a, it is not difficult to verify that

$$ {}^{a}\mathbf{p}_b = R^T(\theta_a)(\mathbf{p}_b - \mathbf{p}_a) = \sum_{k=a+1}^{b} R({}^{a}\theta_{k-1})\mathbf{x}_k \tag{6.15} $$

since ${}^{0}\mathbf{p}_b = \mathbf{p}_b$ and ${}^{0}\theta_b = \theta_b$, where

$$ {}^{a}\theta_b = \theta_b - \theta_a = \sum_{k=a+1}^{b} \theta'_k \tag{6.16} $$

is the angle of frame b with respect to frame a. Note that $\theta_a = -{}^{a}\theta_0$, so that $R^T(\theta_a) = R({}^{a}\theta_0)$.

6.4 Approach

In general, the solution to (6.7) is found by repeatedly computing $\mathbf{r}(\tilde{\mathbf{x}})$ and $J(\tilde{\mathbf{x}})$ for the current estimate, solving the equation for $\Delta\mathbf{x}$, then adding $\Delta\mathbf{x}$ to the current estimate to yield the estimate for the next iteration. The process is repeated until the system converges (i.e., $\|\Delta\mathbf{x}\| \leq \tau$, where τ is a threshold).

The standard Gauss-Newton approach is to solve the equation directly in each iteration, leading to

$$ \Delta\mathbf{x} = M^{-1}J^T(\tilde{\mathbf{x}})\Omega\mathbf{r}(\tilde{\mathbf{x}}) \tag{6.17} $$

where $M = J^T(\tilde{\mathbf{x}})\Omega J(\tilde{\mathbf{x}})$ is a $3n \times 3n$ *preconditioning matrix.* Instead, a two-step approach is proposed that first uses a variation of stochastic gradient descent in the relative state space (POReSS), followed by Gauss-Seidel in the global state space (Graph-Seidel). Although either of these is itself a standalone solution, results show that the two exhibit complementary characteristics. The former is better at quickly getting near the global minimum even with poor initial conditions but can take many iterations to reach convergence, while the latter is better at performing detailed refinements of the estimate but requires a starting point near the global minimum.

6.4.1 Non-stochastic gradient descent

Stochastic gradient descent (SGD) is a standard iterative method for finding the minimum of a function. SGD repeatedly updates the state based on a single constraint between nodes a and b:

$$\Delta \mathbf{x} = \lambda_{ab} M^{-1} J_{ab}^T(\tilde{\mathbf{x}}) \Omega_{ab} \mathbf{r}_{ab}(\tilde{\mathbf{x}}), \tag{6.18}$$

where $\lambda_{ab} \equiv \lambda / |b - a|$, λ is a scalar learning rate, and where the order of the constraints is chosen randomly. The proposed approach, like that of Olson et al. [61], is the same as SGD except that the order is deterministic, with constraints selected in decreasing order of the number of nodes they affect. For that reason, this approach is referred to as *non-stochastic gradient descent*.

Using a relative state space, the residual corresponding to a constraint between nodes a and b is given by

$$\mathbf{r}_{ab}(\mathbf{x}) = \delta_{ab} - {}^{a}\mathbf{p}_{b}. \tag{6.19}$$

The Jacobian is obtained by differentiating the right side of (6.15) with respect to the states:

$$J_{ab} = \begin{bmatrix} \cdots & \mathbf{0} & {}^{a}_{b}B_{a+1} & {}^{a}_{b}B_{a+2} & \cdots & {}^{a}_{b}B_{b} & \mathbf{0} & \cdots \end{bmatrix}, \tag{6.20}$$

where

$$
{}^a_bB_i \equiv \frac{\partial}{\partial \mathbf{x}_i}{}^a\mathbf{p}_b = \begin{bmatrix} \cos{}^a\theta_{i-1} & -\sin{}^a\theta_{i-1} & {}^a_b\alpha_i \\ \sin{}^a\theta_{i-1} & \cos{}^a\theta_{i-1} & {}^a_b\beta_i \\ 0 & 0 & 1 \end{bmatrix} \tag{6.21}
$$

$$
\begin{bmatrix} {}^a_b\alpha_i \\ {}^a_b\beta_i \end{bmatrix} \equiv \sum_{k=i}^{b} \begin{bmatrix} -x'_k \sin{}^a\theta_{k-1} - y'_k \cos{}^a\theta_{k-1} \\ x'_k \cos{}^a\theta_{k-1} - y'_k \sin{}^a\theta_{k-1} \end{bmatrix} \tag{6.22}
$$

if $a+1 \leq i \leq b$, or ${}^a_bB_i \equiv \mathbf{0}$ otherwise, where $\mathbf{0}$ is a vector of zeros. As in the incremental state space approach of Olson et al. [61], there is no need to ever compute the Jacobian explicitly. A full derivation of the Jacobian can be seen in Appendix C.1

In the general case the Jacobian is neither sparse nor linear. Plugging (6.20) into (6.18) yields a linear system that is difficult to compute due to the $M^{-1} = (J^T(\tilde{\mathbf{x}})\Omega J(\tilde{\mathbf{x}}))^{-1}$ term. Following Olson et al. [61], instead of explicitly computing this matrix all but the diagonal elements are ignored:

$$
M \approx \operatorname{diag}(J^T(\tilde{\mathbf{x}})\Omega J(\tilde{\mathbf{x}})) \tag{6.23}
$$

$$
= \operatorname{diag}\underbrace{\left(\sum_{(a,b)\in\mathcal{E}} {}^a\mathbf{m}_b\right)}_{\mathbf{m}}, \tag{6.24}
$$

where $\operatorname{diag}(\mathbf{v}) \equiv \sum_i \mathbf{e}_i\mathbf{e}_i^T\mathbf{v}\mathbf{e}_i^T$ creates a diagonal matrix from a vector, where $\mathbf{e}_i$ is a vector of all zeros except a 1 in the ith element; and

$$
{}^a\mathbf{m}_b \equiv \begin{bmatrix} \cdots & 0 & {}^a_b\gamma^T_{a+1} & {}^a_b\gamma^T_{a+2} & \cdots & {}^a_b\gamma^T_b & 0 & \cdots \end{bmatrix}^T, \tag{6.25}
$$

where

$$ {}^{a}_{b}\gamma_i \equiv diag({}^{a}_{b}B_i^T \, \Omega_{ai} \, {}^{a}_{b}B_i), \tag{6.26}$$

where $diag(A) \equiv \sum_i \mathbf{e}_i^T A \mathbf{e}_i \mathbf{e}_i$ extracts the diagonal of a matrix.

When the constraint is between two consecutive nodes, $b = a+1$, the Jacobian reduces to a very simple form:

$$J_{ab} = \begin{bmatrix} \cdots & \mathbf{0} & I_{\{3\times3\}} & \mathbf{0} & \cdots \end{bmatrix}, \tag{6.27}$$

where $I_{\{3\times3\}}$ is the 3×3 identity matrix. Note that the vast majority of constraints in a typical pose optimization problem are between consecutive nodes, so that this simple form yields a tremendous speedup. Note also that as the preconditioned matrix is being constructed, the computation is simpler in the case of consecutive nodes:

$$ {}^{a}\mathbf{m}_b \equiv \begin{bmatrix} \cdots & \mathbf{0} & diag(\Omega_{ab}) & \mathbf{0} & \cdots \end{bmatrix}^T. \tag{6.28}$$

Plugging the simplified Jacobian of (6.27) into (6.18), approximating M by its diagonal elements, and taking the Moore-Penrose pseudoinverse, resulting in

$$\begin{aligned} \Delta\mathbf{x} &= \lambda_{ab} M^{\dagger} \begin{bmatrix} \ldots & \mathbf{0} & diag(\Omega_{ab}) & \mathbf{0} & \ldots \end{bmatrix}^T \mathbf{r}_{ab}(\tilde{\mathbf{x}}) && (6.29) \\ &\approx \lambda_{ab} \begin{bmatrix} \ldots & \mathbf{0} & I_{\{3\times3\}} & \mathbf{0} & \ldots \end{bmatrix}^T \mathbf{r}_{ab}(\tilde{\mathbf{x}}) && (6.30) \\ &= \lambda_{ab} \mathbf{r}_{ab}(\tilde{\mathbf{x}}), && (6.31) \end{aligned}$$

where the second line is equal in the case of a diagonal Ω_{ab}. Thus it can be seen that in the case of consecutive nodes, only one state needs to be modified, and the update is extremely simple. In the general case, the complexity of computing $J_{ab}^T \Omega_{ab} \mathbf{r}_{ab}$ is

Algorithm 3 POReSS

```
▷ Precompute $M$
$\mathbf{m} \leftarrow$ zeros$(3n, 1)$
for $(a, b) \in \mathcal{E}$ do                                ▷Note: $a < b$
    $\mathbf{m} \leftarrow \mathbf{m} + {}^{a}\mathbf{m}_{b}$
end for
▷ Minimize
while not converged do
    for $(a, b) \in \mathcal{E}$ do                            ▷Note: $a < b$
        $\mathbf{r} \leftarrow (\delta_{ab} - {}^{a}\mathbf{p}_{b}) \bmod_{\theta} 2\pi$
        if $b == a + 1$ then
            $\mathbf{x}_b \leftarrow \mathbf{x}_b + \lambda \mathbf{r}$
        else
            $\mathbf{r} \leftarrow \Omega_{ab}\mathbf{r}$
            for $i \leftarrow a + 1$ to $b$ do
                $\Delta \leftarrow R({}^{a}\theta_{i-1}) \begin{bmatrix} r_x \\ r_y \\ 0 \end{bmatrix} + r_\theta \begin{bmatrix} {}^{a}_{b}\alpha_i \\ {}^{a}_{b}\beta_i \\ 1 \end{bmatrix}$
                $\mathbf{x}_i \leftarrow \mathbf{x}_i + \frac{\lambda}{b-a} \begin{bmatrix} \Delta_x / m_{3i-2} \\ \Delta_y / m_{3i-1} \\ \Delta_\theta / m_{3i} \end{bmatrix}$
            end for
        end if
    end for
end while
```

proportional to the number of nodes between a and b. Pseudocode for POReSS can be seen in Algorithm 3, where $\mathbf{m} \equiv \sum {}^{a}\mathbf{m}_b \equiv \begin{bmatrix} m_1 & \cdots & m_n \end{bmatrix}^T$ is the vector such that $M \approx \text{diag}(\mathbf{m})$, $\mathbf{r} = \begin{bmatrix} r_x & r_y & r_\theta \end{bmatrix}^T$, $\Delta = \begin{bmatrix} \Delta_x & \Delta_y & \Delta_\theta \end{bmatrix}^T$, and mod_θ computes the modulo of the last element of the vector while leaving the other elements unchanged. To improve readability the pseudocode does not include all the optimizations used in the implementation of POReSS. A more complete derivation of Graph-Seidel can be found in Appendix C.2.

6.4.2 Graph-Seidel

While the use of non-stochastic gradient descent on a Relative State Space allows for a quick optimization of a pose graph, it does a poor job of converging to an exact solution. This is addressed in the second phase of the optimization process. In this phase an implementation of Gauss-Seidel is used that is optimized for a graph, which is refered to as *Graph-Seidel*. Graph-Seidel is better suited to finding exact solutions and can perform well given adequate initial conditions.

In the Graph-Seidel optimization a relative state space is not used but instead a global state space is used. This change is made because the second phase provides a refinement to the original optimization, and small changes to a state should not have large implications on the entire system.

The residual is the same as before, but now left side of (6.15) is used, which is combined with (6.19) to yield

$$\mathbf{r}_{ab}(\mathbf{x}) = \delta_{ab} - R^T(\theta_a)(\mathbf{p}_b - \mathbf{p}_a) \tag{6.32}$$

To simplify the math we define

$$\begin{aligned} \mathbf{r}'_{ab}(\mathbf{x}) &\equiv R(\theta_a)\mathbf{r}_{ab}(\mathbf{x}) && (6.33) \\ &= \mathbf{p}_a - \mathbf{p}_b + R(\theta_a)\delta_{ab} && (6.34) \\ \Omega'_{ab} &\equiv R(\theta_a)\Omega_{ab}R^T(\theta_a). && (6.35) \end{aligned}$$

and note that $\epsilon(\mathbf{x})$ does not change when substituting $\mathbf{r}'_{ab}$ for $\mathbf{r}_{ab}$, and Ω'_{ab} for Ω_{ab}.

The trick to a simple and fast algorithm is to assume that $R(\theta_a)$ is constant when taking the derivative $\frac{\partial\epsilon(\mathbf{x})}{\partial\mathbf{p}_i}$. The key insight is that, if $R(\theta_a)$ is constant, then $\epsilon(\mathbf{x})$ is convex in the states. As a result, the system does not not need to linearized at all but instead the derivatives of the system can be used to solve directly for the states, then iterate by updating $R(\theta_a)$. Even though the assumption is not true, it does not affect the result because $R(\theta_a)$ is recomputed in each iteration. Employing this assumption, setting the derivative to zero yields

$$\sum_{(a,i)\in\mathcal{E}_i^{in}} \Omega'_{ai}\mathbf{r}'_{ai}(\mathbf{x}) \quad = \sum_{(i,b)\in\mathcal{E}_i^{out}} \Omega'_{ib}\mathbf{r}'_{ib}(\mathbf{x}), \tag{6.36}$$

where

$$\mathcal{E}_i^{in} \equiv \{(a,b) : (a,b)\in\mathcal{E} \text{ and } b = i\} \tag{6.37}$$

$$\mathcal{E}_i^{out} \equiv \{(a,b) : (a,b)\in\mathcal{E} \text{ and } a = i\} \tag{6.38}$$

are the set of edges into and out of, respectively, node i.

Since GSS is being used, note that $\mathbf{x} = \mathbf{p}$ and $\mathbf{x}_i = \mathbf{p}_i$ for $i = 1,\ldots,n$. Rearranging terms yields

$$\begin{bmatrix} \overline{\Omega}_1 & -\overline{\Omega}_{12} & \cdots & -\overline{\Omega}_{1n} \\ -\overline{\Omega}_{21} & \overline{\Omega}_2 & \cdots & -\overline{\Omega}_{2n} \\ \vdots & & \ddots & \vdots \\ -\overline{\Omega}_{n1} & -\overline{\Omega}_{n2} & \cdots & \overline{\Omega}_n \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_n \end{bmatrix} = \begin{bmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \\ \vdots \\ \mathbf{v}_n \end{bmatrix}, \tag{6.39}$$

where, assuming not a multigraph,

$$\overline{\Omega}_{ab} \equiv \overline{\Omega}_{ba} \equiv \begin{cases} \Omega'_{ab} & \text{if } \delta_{ab} \text{ exists} \\ \Omega'_{ba} & \text{if } \delta_{ba} \text{ exists} \\ \mathbf{0}_{\{3\times3\}} & \text{otherwise} \end{cases} \tag{6.40}$$

$$\overline{\Omega}_a \equiv \sum_b \overline{\Omega}_{ab} \tag{6.41}$$

$$\mathbf{v}_i \equiv \underbrace{\sum_{(a,i)\in\mathcal{E}_i^{in}} R(\theta_a)\Omega_{ai}\delta_{ai}}_{\text{edges in}} - \underbrace{\sum_{(i,b)\in\mathcal{E}_i^{out}} R(\theta_i)\Omega_{ib}\delta_{ib}}_{\text{edges out}}\,. \tag{6.42}$$

This can be solved using Gauss-Seidel iterations of the form:

$$\mathbf{x}_i^{(k+1)} = \overline{\Omega}_i^{-1}\left(\mathbf{v}_i + \sum_{j<i} \overline{\Omega}_{ij}\mathbf{x}_j^{(k+1)} + \sum_{j>i} \overline{\Omega}_{ij}\mathbf{x}_j^{(k)}\right), \tag{6.43}$$

where k is the iteration number. To improve convergence, successive over-relaxation (SOR) is employed. Note that the matrix remains constant, while the vector $\mathbf{v}$ must be recomputed each iteration. As before, the pseudocode in Algorithm 4 does not show any optimizations used in the implementation of Graph-Seidel.

6.5 Full POReSS Experimental Results

The proposed algorithm was ran on three synthetic data sets and one real data set to evaluate its performance. Two of the three synthetic datasets were graphs with a number of interconnections, shown in Figures 6.2 and 6.5. In addition, the proposed approach is also run on a large, simple, single-loop graph of varying sizes (up to 40 million nodes and constraints).

Algorithm 4 Graph-Seidel

```
while not converged do
    ▷ Compute $\mathbf{v}$
    for $i \leftarrow 1$ to $n$ do
        $\mathbf{v}_i \leftarrow \mathbf{0}_{\{3\times 1\}}$
        for $(a,b) \in \mathcal{E}$ do
            if $a == i$ then
                $\mathbf{v}_i \leftarrow \mathbf{v}_i + \overline{\Omega}_{ab} R(\theta_a) \delta_{ab}$
            else if $b == i$ then
                $\mathbf{v}_i \leftarrow \mathbf{v}_i - \overline{\Omega}_{ab} R(\theta_a) \delta_{ab}$
            end if
        end for
    end for
    ▷ Minimize
    for $i \leftarrow 1$ to $n$ do
        $\mathbf{w} \leftarrow \mathbf{0}_{\{3\times 1\}}$
        for $(a,b) \in \mathcal{E}$ do
            if $a == i$ then
                $\mathbf{w} \leftarrow \mathbf{w} + \overline{\Omega}_{ab} \mathbf{x}_b$
            else if $b == i$ then
                $\mathbf{w} \leftarrow \mathbf{w} + \overline{\Omega}_{ab} \mathbf{x}_a$
            end if
        end for
        $\mathbf{x}_i \leftarrow \overline{\Omega}_i^{-1} (\mathbf{v}_i + \mathbf{w})$
    end for
end while
```

6.5.1 Complex Graphs

Even in the case of complex graphs with many interconnections and a very corrupted input, the POReSS algorithm is able to get to a close solution in as little as 1 step, allowing Graph-Seidel to take over and optimize to an exact solution. In these graphs proposed approach is compared to two of the state-of-the-art algorithms, TORO [26, 25, 27] and g2o [48]. These approaches are compared by viewing the runtime and the number of iterations needed to reach an optimized graph, as well as the final residual of the optimized graph.

In the first graph, obtained from [48], the proposed approach is able to achieve a recognizable result, cutting the residual almost in half, in only one iteration of POReSS. See Figure 6.2. From the chart in Figure 6.3 it can be seen that the POReSS algorithm is already close to its local minimum after 1 iteration, even though the graph is far from an optimal solution. Once POReSS has reached a local minimum Graph-Seidel takes over to refine the graph. Although it takes several iterations of Graph-Seidel to optimize the graph each iteration runs in a trivial amount of time, allowing the system to get within 98% of an optimized solution in under 0.1 seconds and to a final solution in 0.25 seconds. The proposed algorithm was not able to get to as optimal of a solution in comparison to g2o. However, examining Figure 6.4, it can be seen that the proposed approach was able to reach its optimal solution approximately at the same time g2o achieved its optimal solution.

The second graph, obtained from [39], Figure 6.5, is an even larger complex graph with more interconnections. In this graph TORO was able to get closer to an optimal solution in 1 step, however it was unable to reach an optimal solution. Even though POReSS did not get as close to an optimal solution as TORO in its first few iterations it was able to get close enough for Graph-Seidel to arrive at a more optimal solution.

6.5.2 Large Simple Graph

In the context of optimizing a trajectory online, or incrementally, an optimization process would only really need to consider the most recent loop closure in order to get close to an exact solution, this close solution could then be refined later offline. However, as the number of nodes in one loop gets larger the run time of the optimization process also increases. To compare the proposed approach to the state of the art, a simple square with one loop closing constraint between the first and last nodes is generated noise to the rotational component at the corners is added, this graph can be seen in Figure (6.6). The number of nodes per edge from 4000 to 40 million and plotted the runtime of an iteration for POReSS, g2o, TORO, and Olson (as described in [61]), see Figure 6.7. Every technique successfully converged to a solution in 5 iterations regardless of the number of nodes, provided the algorithm was able to run.

Regardless of the number of nodes per side, the proposed approach had the shortest run time. Both TORO and g2o eventually ran out of memory after 4000 and 400,000 nodes respectively, and were unable to perform the optimization. Olson's algorithm was able to run on the graph of 4 million and 40 million nodes, but was terminated after 10^4 seconds (≈ 3 hours). POReSS was able to successfully optimize a graph of up to 40 million nodes requiring just over two seconds per iteration.

6.5.3 Real Data Set

The final experiment was on real data obtained from [6]. The data was collected from a vehicle driving around a parking lot making multiple loops. This graph

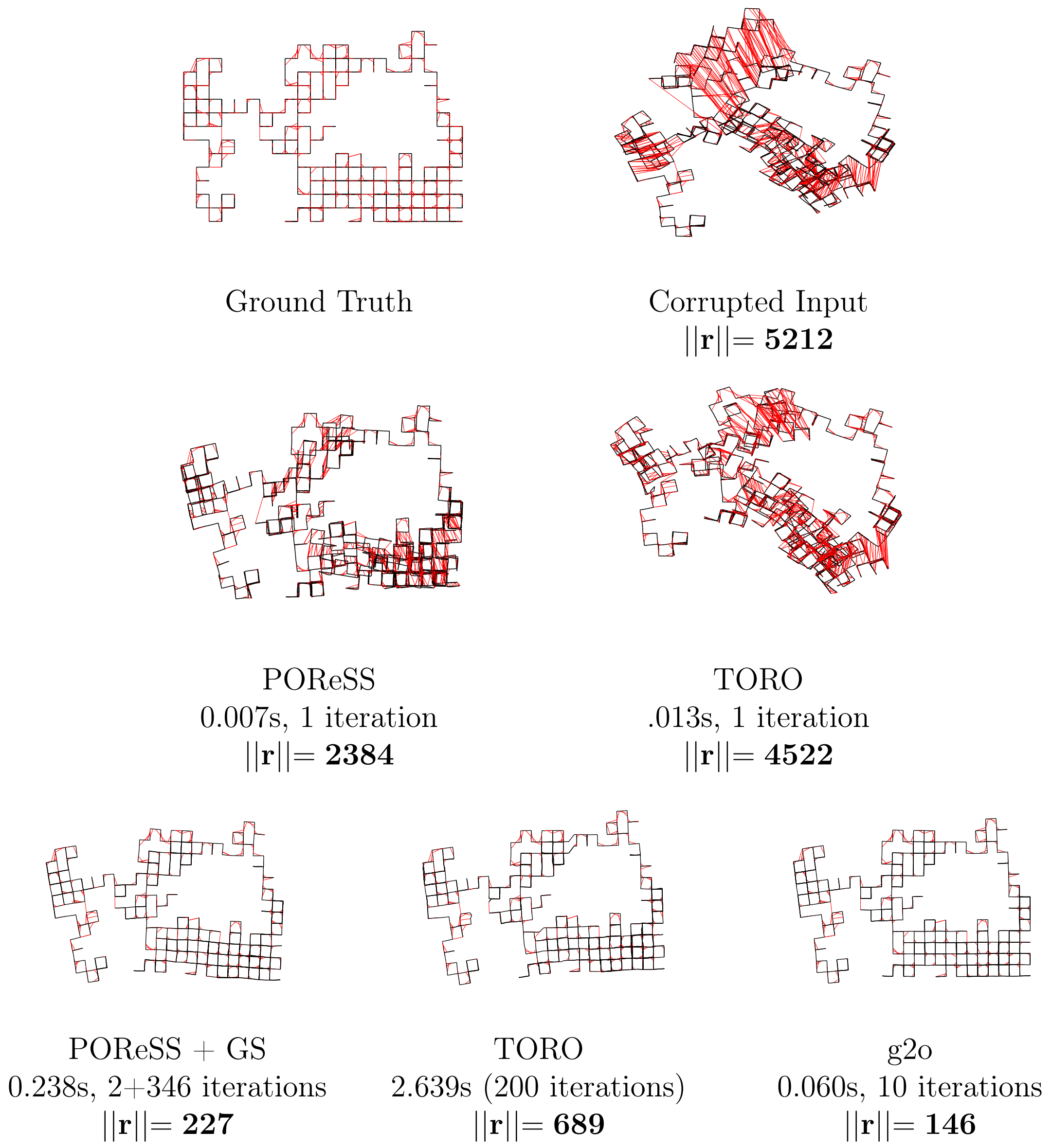


Figure 6.2: Optimizations of the Manhattan World data set released with g2o. This graph contains 3500 nodes and 5600 constraints. First row shows the ground truth and the corrupted input. Second row shows the proposed approach after 1 iteration of POReSS in comparison to 1 iteration of TORO. Third row shows the final optimization after convergence of the proposed approach in comparison to TORO and g2o. The number of iterations in the proposed approach are shown as POReSS iterations + GS iterations.

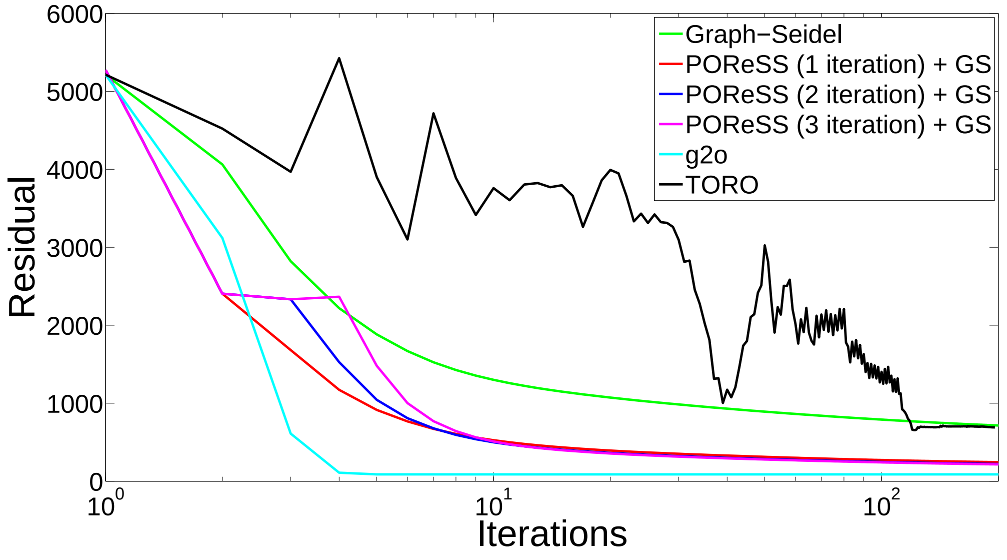


Figure 6.3: The number of iterations vs. the residual of the Manhattan World dataset released with g2o when using varying optimization techniques. Note the x-axis is on a logarithmic scale.

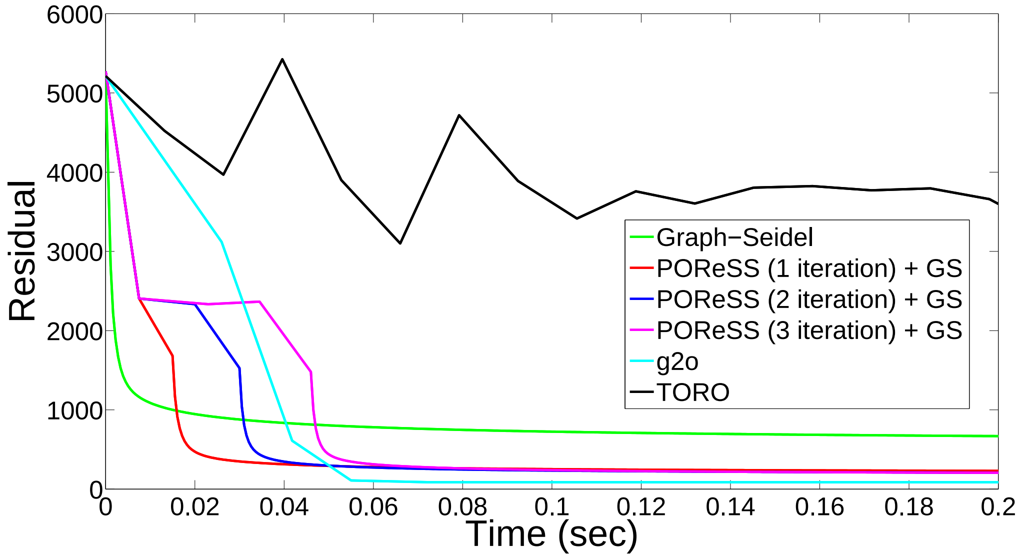
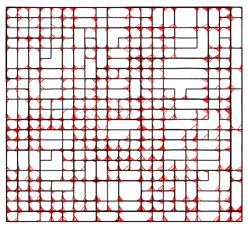
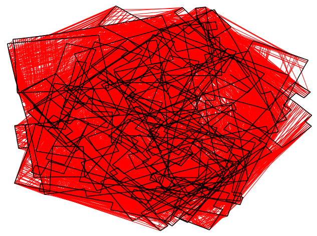


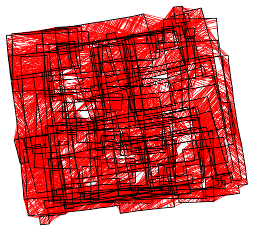
Figure 6.4: The runtime vs. the residual of varying optimization techniques when optimizing the Manhattan World dataset released with g2o. The full runtime of TORO is not shown and is cut off at 0.2 seconds which is ≈ 40 iterations.



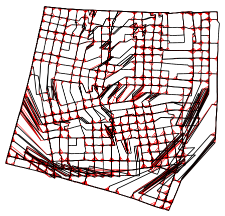
Ground Truth



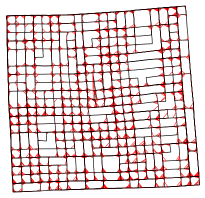
Corrupted Input
$||\mathbf{r}||=$ **309792**



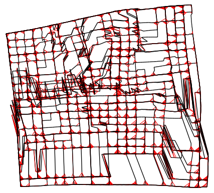
POReSS
1 iteration (0.620s)
$||\mathbf{r}||=$ **130900**



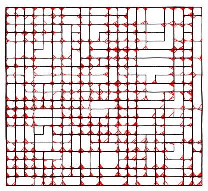
TORO
1 iteration (.067s)
$||\mathbf{r}||=$ **9260**



POReSS + GS
3 + 773 iterations (3.33s)
$||\mathbf{r}||=$ **1209**



TORO
200 iterations (13.586s)
$||\mathbf{r}||=$ **7058**



g2o
10 iterations (1.812s)
$||\mathbf{r}||=$ **512**

Figure 6.5: The dataset provided by [39] containing 10,000 nodes and 30,000 constraints. The number of iterations in the proposed approach is shown as POReSS iterations + GS iterations.

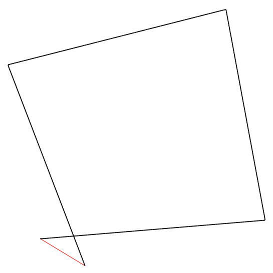


Figure 6.6: A simple graph of a single loop with noise added

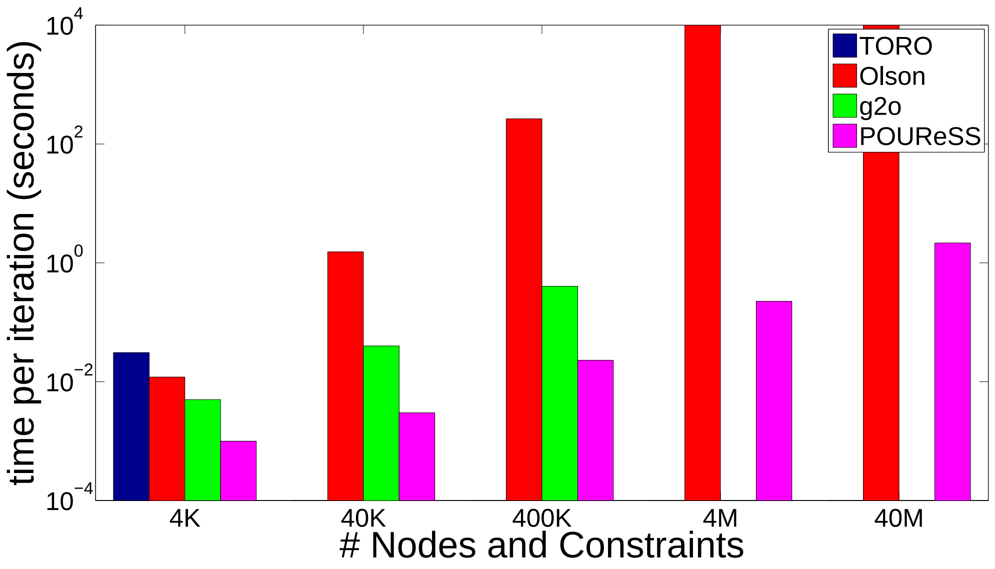


Figure 6.7: The amount of time required for each optimization algorithm to optimize a single loop of varying number of constraints. If no bar is shown for an optimization technique for a number of nodes it means the optimization technique failed to run on the graph with that number of nodes. All timings were cut off at 10^4 seconds.

can be seen in Figure 6.8. The proposed approach was the only optimization technique of the three to optimize the graph close to the ground truth. Unlike the two synthetic data sets, this data set required POReSS to run for 12 iterations before switching to Graph-Seidel for the final optimization.

6.6 Rotational Optimization

Previous work [66] has shown that the majority of errors in a trajectory are a result of rotational drift, and if this rotational drift can be removed then in some datasets there is little need for loop closure. The latter statement is not necessarily true for all datasets however the first statement is almost invariably true for real datasets. In this section an appraoch is proposed to reduce the error caused by rotational drift by using a variant on the proposed POReSS algorithm that only considers the rotational component. The resultant graph will then be used as a starting point for the unmodified Graph-Seidel algorithm to make the final refinements to the graph. To avoid confusion the original POReSS system that uses both the rotational and position components is referred to as Full POReSS and the POReSS considering only the rotational component as rPOReSS.

By only optimizing the rotational components the x and y components of the state effectively become constants, which greatly decreases the complexity of the derivation because the non-linearity of the rotational component effecting the position component is removed. The state goes from

$$\mathbf{x}_i \equiv \left[\begin{array}{ccc} x'_i & y'_i & \theta'_i \end{array} \right]^T \tag{6.44}$$

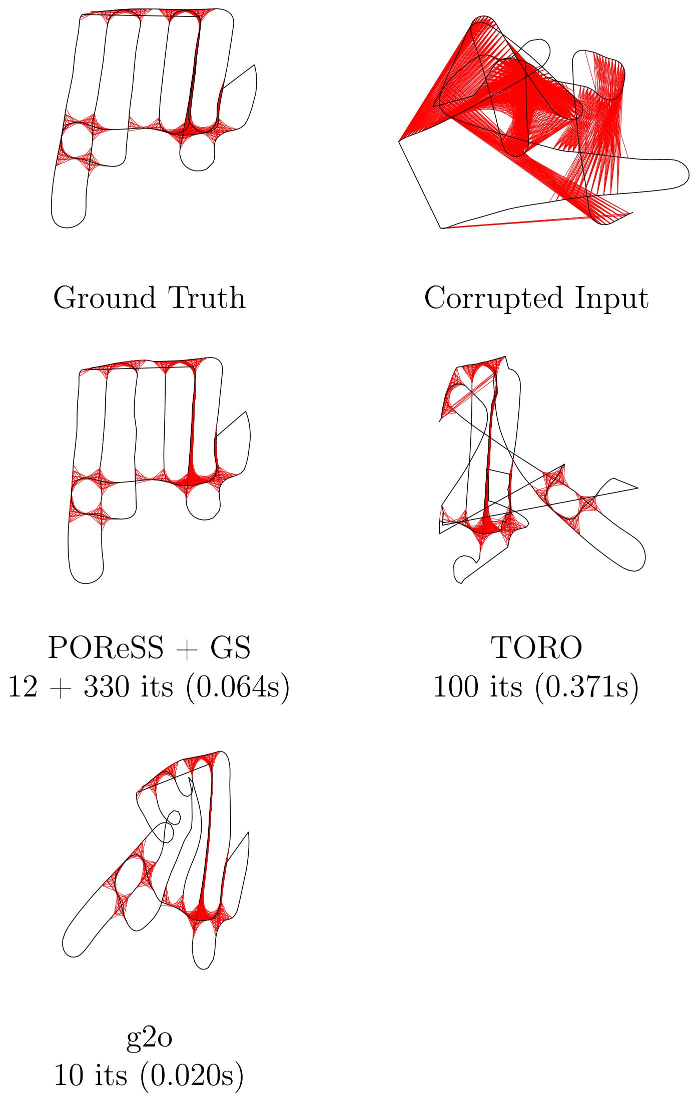


Figure 6.8: Graph of vehicle driving around a parking lot obtained from [6]. This graph contains 407 nodes and 1625 constraints. In this real-world data set the proposed approach was the only approach of the three that was able to optimize the graph to a state that was resembling of the ground truth. The number of iterations in the proposed approach are shown as POReSS iterations + GS iterations.

to

$$\mathbf{x}_i \equiv \begin{bmatrix} \theta'_i \end{bmatrix}^T \tag{6.45}$$

which results in a relative pose between a and b as

$${}^{a}\mathbf{p}_b = \sum_{i=a}^{b-1} \theta'_i \tag{6.46}$$

This mirrors the relative pose seen in (6.15) but is now linear. However (6.13), the computation of the global poses, is unmodified. This retains the advantage of Full POReSS's ability to effect the overall global state of the system with small changes to the state. In addition the equation for the update of the states, (6.18) is modified to reflect the existance of only one non-constant state per node. This allows the for replacement of the inverted covariance matrix of the measurment between nodes a and b, Ω_{ab}, to be replaced with the inverted variance of the rotational component of the measurmenet, ω_{ab}.

$$\Delta\mathbf{x} = \lambda M^{-1} J_{ab}^T(\tilde{\mathbf{x}})\omega_{ab}\mathbf{r}_{ab}(\tilde{\mathbf{x}}), \tag{6.47}$$

Much like in the derivation for Full POReSS the Jacobian of the system is formed by differentiating (6.46) with respect to the states resulting in a very simple Jacobian.

$$J_{ab} = \begin{bmatrix} \cdots & 0 & \underbrace{1}_{a} & \cdots & \underbrace{1}_{b-1} & 0 & \cdots \end{bmatrix} \tag{6.48}$$

As in Full POReSS this Jacobian never needs to be explicitly computed.

Mirroring the derivation for Full POReSS it is easy to see in the case of subsequent nodes that when plugging J_{ab} into (6.47) yields the following for the update to the state of node a

$$\Delta\mathbf{x} = \lambda\mathbf{r}_{ab}(\tilde{\mathbf{x}}) \tag{6.49}$$

Algorithm 5 rPOReSS

```
//Precompute M
$M$ = Zeroes(N, 1)
for $(i, j) \epsilon \mathcal{E}$ do
    if $i + 1 == j$ then
        $M = M +^i \mathbf{m}_j$
    end if
end for
//Minimize
while not converged do
    for $(i, j) \epsilon \mathcal{E}$ do
        $r = mod2pi(\delta_{ij} - {}^i\mathbf{p}_j)$
        if $j - i > 1$ then
            for $k = i + 1; k < j; k{+}{+}$ do
                $X_k = X_k + \frac{\lambda}{j-i}\omega_{ij} r M_k^{-1}$
            end for
        else
            $X_k = X_k + \lambda r$
        end if
    end for
end while
```

In the general case $M^{-1} \approx \text{diag}(J^T(\tilde{\mathbf{x}})\omega J(\tilde{\mathbf{x}}))^{-1}$ which is easy to compute given the simplistic structure of the Jacobian.

$$M = \overline{\text{diag}} \left(\sum_{(a,b)\epsilon\mathcal{E}} {}^a\mathbf{m}_b \right) \tag{6.50}$$

where

$${}^a\mathbf{m}_b = \begin{bmatrix} \cdots & 0 & \underbrace{\omega_{ab}}_{a} & \cdots & \underbrace{\omega_{ab}}_{b-1} & 0 & \cdots \end{bmatrix} \tag{6.51}$$

Pseudocode for rPOReSS can be see in Algorithm 5, to improve readability it does not show any of the optimizations used in the implementation.

6.7 rPOReSS Experimental Results

To test rPOReSS it was directly compared against the Full POReSS approach on both real and synthetic data sets of varying interconnectivity. Two synthetic datasets can be seen in Figures 6.9 and 6.10. For the full explanation of the these two datasets and comparison against other state of the art algorithms see Section 6.5.1.

Two real world datasets can be seen in Figures 6.11 and 6.13. Figure 6.11 compares rPOReSS to Full POReSS as well as g2o, plots of the residuals can be seen in Figure 6.12. These two figures shows g2o reaching an incorrect solution and Full POReSS diverging away from the solution while rPOReSS is able to succesfully reach an optimal solution. The second real world data set can be seen in Figure 6.13 where rPOReSS is directly compared to Full POReSS, a full explanation of this graph and comparison to other SOA algorithms can be seen in Section 6.5.3.

6.8 Incremental Two Phase

Previous sections of this chapter have assumed a batch optimization process. However, it is desirable to be able to perform graph optimizations incrementally for online systems. In this section an incremental implementation of the rPOReSS system which is referred to as irPOReSS is proposed. Several incremental approaches like [40, 73] restrain the incremental optimization process to a small window, only looking at the most recent additions to the graph. While this decreases the runtime it can cause also hinder the performance of the optimization, especially in highly interconnected graphs. An incremental approach that only considers the most recent edges, the edges between current loop closure and the previous loop closure, is also proposed. However, to address the performance issues caused by only optimizing

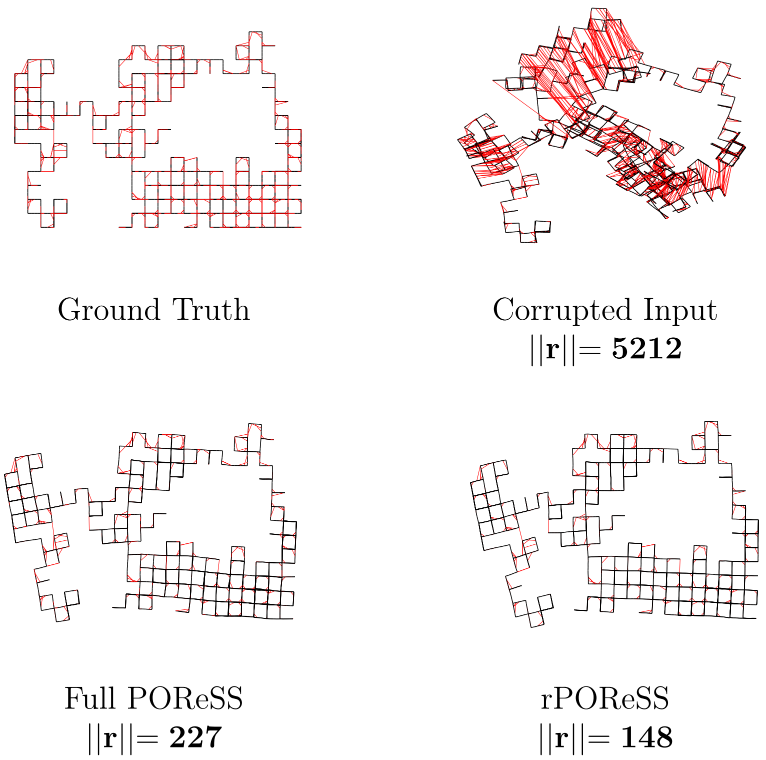

Ground Truth
Corrupted Input
$||\mathbf{r}||= \mathbf{5212}$
Full POReSS
$||\mathbf{r}||= \mathbf{227}$
rPOReSS
$||\mathbf{r}||= \mathbf{148}$

Figure 6.9: Comparison of rPOReSS to Full POReSS on the Manhattan World dataset released with g2o. While visually their does not appear to be much of difference between the two optimized graphs, the final residual of rPOReSS is smaller than that of Full POReSS (65%). While the overall structure is almost not noticeably different, the lower residual indicates that rPOReSS is able to yield a more accurate optimization both locally and globally.

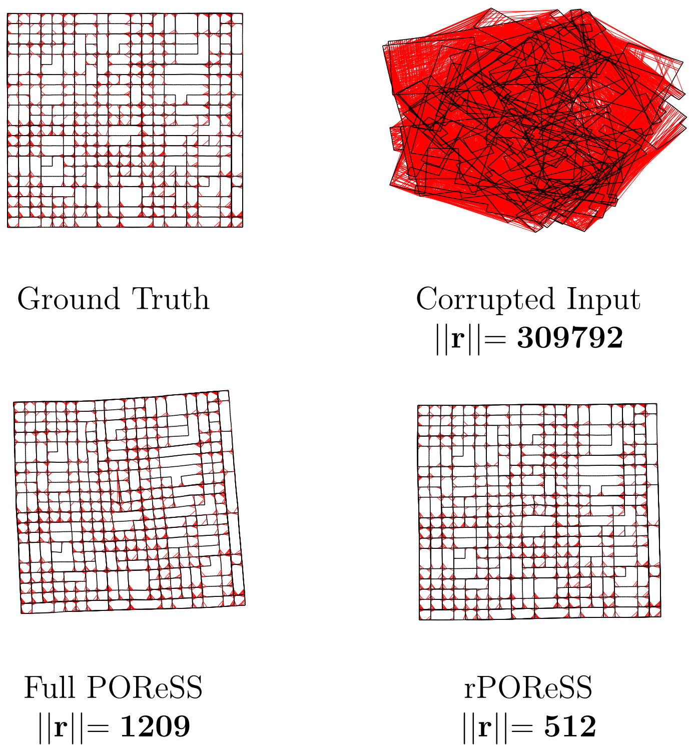


Figure 6.10: Comparison of rPOReSS to Full POReSS on the dataset by [39] containing 10,000 nodes and 30,000 constraints. In this comparison, not only is the residual smaller (42%) the overall structure more closely resembles the ground truth.

.

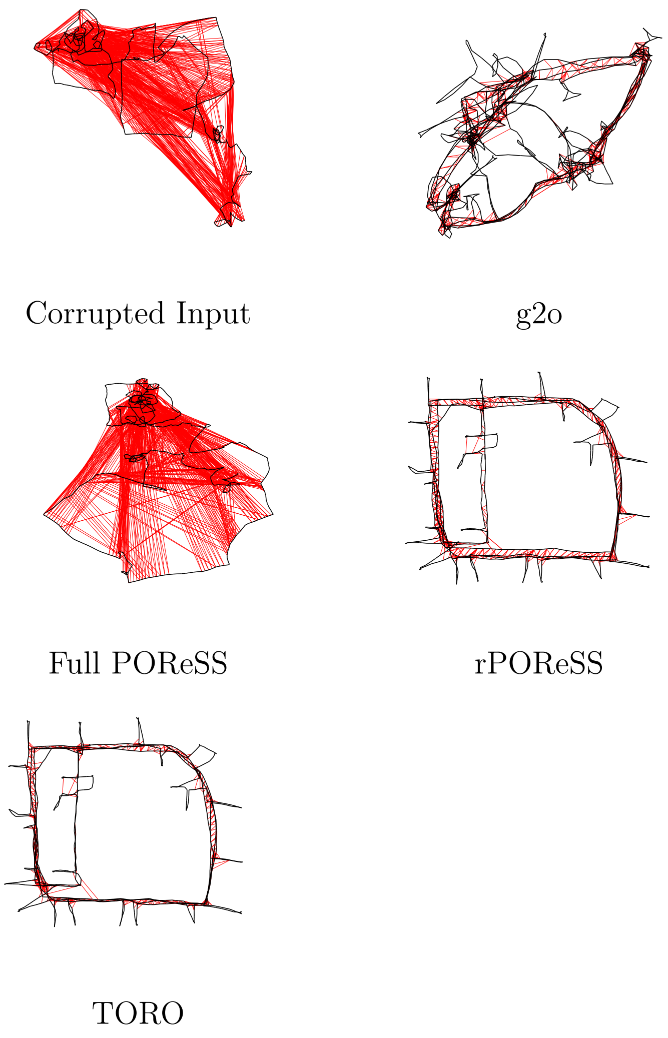


Figure 6.11: Comparison of rPOReSS, Full POReSS, g2o, and TORO on the Seattle Intel Research Lab provided by Dirk Hähnel. The initial starting condition was highly corrupted to provide a challenging experiment. Both g2o and Full POReSS could not correctly optimize the graph. Full POReSS never reaches a stable state and the system diverges. rPOReSS is able to successfully optimize the graph, even with a poor initial condition.

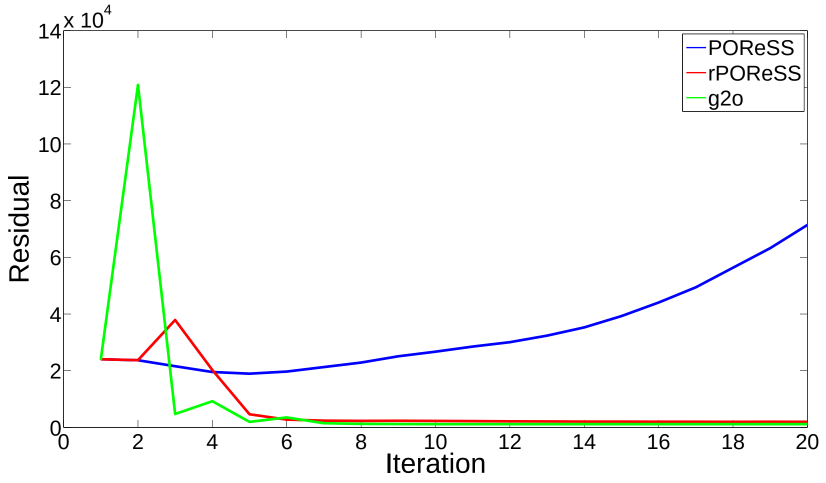


Figure 6.12: Plot of the residual against the number of iterations for POReSS, rPOReSS, and g2o on the dataset seen in Figure 6.11. This plot shows that POReSS is unable to optimize the system and actually becomes unstable and diverges from the solution. rPOReSS and g2o both reach an optimized solution, however viewing the optimized graphs in Figure 6.11 it is clear that the solution reached by g2o is incorrect while the solution reached by rPOReSS is more optimal.

the most the recent edges, a batch optimization on the currently constructed graph every T frames, where T is some predefined number, is also run. This allows the incremental approach to take advantage of the Markov property of sparsly connected graphs, while ignoring this property in highly interconnected graphs.

Since the irPOReSS system is used to incrementally construct the graph it does not provide a refined optimization. The output from the irPOReSS can be used as an initial starting point for the Graph-Seidel algorithm to fully refine the optimization. However, Graph-Seidel is only used whenever a fully refined optimization is desired. If Graph-Seidel is used midway through the incremental process it will make the small adjustements to refine the optimization, however whenever the next iteration of irPOReSS is run, since irPOReSS uses a relative state space, it will undo all refinements made by Graph-Seidel effectively wasting computation time. The use of

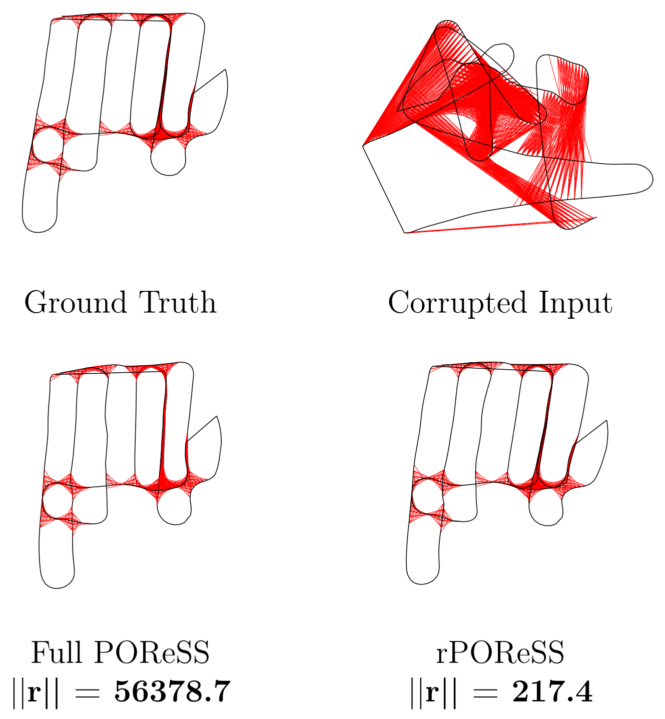
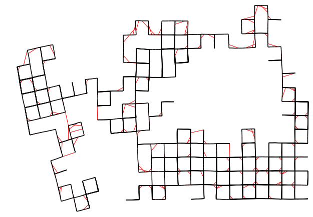

Ground Truth
Corrupted Input
Full POReSS
$||\mathbf{r}|| = \mathbf{56378.7}$
rPOReSS
$||\mathbf{r}|| = \mathbf{217.4}$

Figure 6.13: Comparison of rPOReSS and Full POReSS on data obtained from [6].

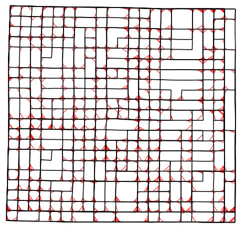
Algorithm 6 irPOReSS

```
graph = EmptyGraph();
counter = 0;
start_edge = 0;
while true do
    edge = GetNextEdge();
    if edge == NULL then
        break;
    end if
    if edge.a + 1 == edge.b then
        graph.AddEdge(edge);
    else
        graph.AddEdge(edge);
        rPOReSS(graph, start_edge)
        start_edge = graph.n_edges();
        counter++
        if counter % T == 0 then
            rPOReSS(graph, 0);
        end if
    end if
end while
Graph-Seidel(graph);
```

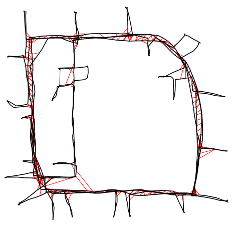
Graph-Seidel is reserved until the end when no more iterations of irPOReSS are to be run. Psuedocode for irPOReSS can be seen in Algorithm 6 and results comparing for irPOReSS can be seen in Figure 6.14.



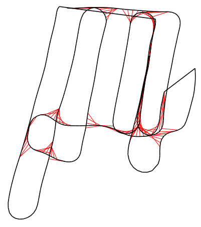
$||\mathbf{r}|| = \mathbf{108}$



$||\mathbf{r}|| = \mathbf{397}$



$||\mathbf{r}|| = \mathbf{21}$



$||\mathbf{r}|| = \mathbf{105}$

Figure 6.14: Optimized graphs and their residuals using irPOReSS.

Chapter 7

Conclusion

7.1 Camera Tracking

A direct technique to robustly estimate camera motion using an RGBD sensor, even when there is little geometric information in the scene (e.g., flat walls) has been presented. The proposed algorithm has been incorporated into the KinectFusion algorithm, enabling camera tracking not only through flat regions but also in areas where the majority of depth readings are beyond the boundaries of the TSDF volume. This improvement was achieved by aligning the color projections of the point clouds between concurrent frames. These projections are aligned by incrementally warping one color projection onto the other one using Lucas-Kanade. The final alignment of the color projections allows for the creation of a correspondence map between two point clouds. This correspondence map replaces the projective data association technique implemented in the variant of ICP in KinectFusion which inhibited camera tracking in sparse geometric feature environments. As seen in the experiments the correspondence map obtained using the proposed LKDA technique allows KinectFusion to maintain tracking in environment that have limited geometric features, as well as in environments that extend beyond the TSDF volume.

In addition four variations of the LKDA algorithm have been examined: with and without color, and translation and affine warps between images. It was found that adding a color channel to LKDA does not improve results significantly to warrant the extra computational cost. However, it was found that using an affine warp enables camera tracking in all environments in which the system was tested. The use of the LKDA does not hinder the real-time performance in the KinectFusion algorithm due to its ability to be implemented on a GPU to run in real-time [74, 18]. Future work will be aimed at extending this approach to large-scale environments.

7.2 Data Storage and Integration

The use of two separate representations of 3D environments, 3D occupancy grids and TSDFs has been examined. The 3D occupancy grid is a simpler representation of an environment allowing for easier integration of new data but lacks the ability to marginalize out noise and discretizes the environment. TSDFs provide a continuous representation of the environment but require more data to store a reconstruction but have the added benefit of easily removing noise from a reconstruction.

Both of these representations were integrated into an octree framework in order to extend both approaches to large scale environments. It has been shown that the use of octrees greatly reduces the amount of memory required to store a 3D reconstruction of an environment by only allocating memory for occupied space and not for free space. Although the general representation of a TSDF requires more memory than a 3D occupancy grid it has been shown that the ability of the TSDF to remove noise results in reconstructions that require less memory.

7.3 Loop Closing

A two phase optimization process for solving the PoseSLAM problem was presented. The first phase uses a relative state space (POReSS). It has been shown through extensive experiments on complex synthetic graphs, large synthetic graphs, and a real-world data set that, surprisingly, the opposite is true.

While it was shown that POReSS is able to reach a local minimum quickly, its inability to reach a global minimum was also seen. By switching to an algorithm more suitable for fine adjustments to a graph in the second phase of the proposed optimization process, namely Graph-Seidel, the proposed approach was able to reach an optimized graph that closely resembles the ground truth graph. Graph-Seidel, when seeded with the output from POReSS, decreases the residual further than that of TORO and achieves results comparable to g2o. In fact, on at least one real-world dataset it achieved results better than that of g2o.

Additionally, two variations on the initially proposed POReSS algorithm, namely rPOReSS and IR-POReSS, have been introduced. It was shown that by removing only the rotational drift, rPOReSS, and using that optimization as a starting point for Graph-Seidel lower residuals could be acheived even in the presence of a very poor initial estimates. An incremental approach, IR-POReSS, was also proposed to perform graph optimizations online.

One bottleneck to the POReSS approach, and the two proposed variations, is the same bottleneck when using an incremental state space, constraints that affect multiple states. Others have shown that a tree representation of the graph allows for a quicker update of the states. While POReSS does not require this update, it does require a composition of relative transformations between two nodes at either endpoint of a constraint. Future work should focus on running POReSS on a tree representation of the graph in order to speed up this composition as well as its extension to 3D.

Appendices

Appendix A

Point Cloud Alignment Error Metrics

A.1 Point-to-Plane Error Metric

Given point clouds, P and Q, correspondence between the two clouds, and a transformation, T, from P to Q the point-to-plane error metric measure the distance between a point, p_i, and the plane formed by the normal, n_i, of its corresponding point, q_i. The cost function between clouds P and Q is

$$E = \sum_{i=1}^{N} \left([Tp_i - q_i] \cdot n_i \right)^2 \tag{A.1}$$

. Finding the value of T that minimizes this cost function will bring P and Q into alignment, provided the correspondences are correct, where T is a Euclidean transformation.

$$T = \begin{bmatrix} C_\theta C_\phi & -C_\phi S_\psi + S_\phi S_\theta C_\phi & S_\phi S_\psi + C_\psi S_\theta C_\phi & t_x \\ C_\theta C_\psi & C_\phi C_\psi + S_\phi S_\theta S_\phi & -S_\phi C_\psi + C_\phi S_\theta S_\psi & t_y \\ -S_\theta & S_\phi C_\theta & C_\phi C_\theta & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.2}$$

where $S_{\theta,\phi,\psi}$ and $C_{\theta,\phi,\psi}$ are equivalent to $sin(\theta, \phi, \psi)$ and $cos(\theta, \phi, \psi)$ respectively.

Since T is non-linear and there is no direct solution to find the value for T that minimizes the error, the value of T has to be iterativelly estimated by incrementally composing smaller transformations.

$$T = T^{(k)}T^{(k-1)} \dots T^{(1)} \tag{A.3}$$

where $T^{(k)}$ is the incremental transformation calculated in the k^{th} iteration. Each iteration of the minimization process computes an incremental transformation, $\tilde{T}$, that is composed with the current estimate, ${}^{k}_{1}T$, to yield a value for T. Breaking T into a composition of its current estimate and an incremental transformation yields the following cost function.

$$E = \sum_{i=1}^{N} \left(\left[\tilde{T}\hat{p}_i - q_i \right] \cdot n_i \right)^2 \tag{A.4}$$

where $\hat{p}_i =^k_1 Tp_i$. By making the assumption that in each iteration the incremental transformation will be small, $\tilde{T}$ can be linearized by making an approximation of $\tilde{T}$ for small rotations. For small rotations $\theta, \psi, \phi \approx 0$ yielding $cos(\theta), cos(\psi), cos(\phi) \approx 1$ and $sin(\theta), sin(\psi), sin(\phi) \approx 0$.

$$\tilde{T} \approx \begin{bmatrix} 1 & -\psi + \phi\theta & \phi\psi + \theta & t_x \\ \psi & 1 + \phi\theta\phi & -\phi + \theta\psi & t_y \\ -\theta & \phi & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \approx \begin{bmatrix} 1 & -\psi & \theta & t_x \\ \psi & 1 & -\phi & t_y \\ -\theta & \phi & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{A.5}$$

Inserting $\tilde{T}$ into equation (A.4) yields the following linearized cost function.

$$E = \sum_{i=1}^{N} \left(\left[\begin{bmatrix} 1 & -\psi & \theta & t_x \\ \psi & 1 & -\phi & t_y \\ -\theta & \phi & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \hat{p}_i^x \\ \hat{p}_i^y \\ \hat{p}_i^z \\ 1 \end{bmatrix} - \begin{bmatrix} q_i^x \\ q_i^y \\ q_i^z \\ 1 \end{bmatrix} \right] \cdot \begin{bmatrix} n_i^x \\ n_i^y \\ n_i^z \\ 0 \end{bmatrix} \right)^2 \tag{A.6}$$

$$E = \sum_{i=1}^{N} \left(\begin{bmatrix} \hat{p}_i^x - q_i^x + t_x - \psi \hat{p}_i^y + \theta \hat{p}_i^z \\ \hat{p}_i^y - q_i^y + t_y - \phi \hat{p}_i^z + \psi \hat{p}_i^x \\ \hat{p}_i^z - q_i^z + t_z + \phi \hat{p}_i^y - \theta \hat{p}_i^x \\ 0 \end{bmatrix} \cdot \begin{bmatrix} n_i^x \\ n_i^y \\ n_i^z \\ 0 \end{bmatrix} \right)^2 \tag{A.7}$$

$$\begin{aligned} E = \sum_{i=1}^{N} (n_i^x \hat{p}_i^x - n_i^x q_i^x + n_i^x t_x - n_i^x \psi \hat{p}_i^y + n_i^x \theta \hat{p}_i^z + n_i^y \hat{p}_i^y - n_i^y q_i^y + n_i^y t_y - n_i^y \phi \hat{p}_i^z \\ + n_i^y \psi \hat{p}_i^x + n_i^z \hat{p}_i^z - n_i^z q_i^z + n_i^z t_z + n_i^z \phi \hat{p}_i^y - n_i^z \theta \hat{p}_i^x)^2 \end{aligned} \tag{A.8}$$

$$\begin{aligned} E = \sum_{i=1}^{N} ((n_i^x \hat{p}_i^z - n_i^z \hat{p}_i^x)\theta + (n_i^y \hat{p}_i^x - n_i^x \hat{p}_i^y)\psi + (n_i^z \hat{p}_i^y - n_i^y \hat{p}_i^z)\phi + n_i^x t_x + n_i^y t_y \\ + n_i^z t_z + n_i^x \hat{p}_i^x + n_i^y \hat{p}_i^y + n_i^z \hat{p}_i^z - n_i^x q_i^x - n_i^y q_i^y - n_i^z q_i^z)^2 \end{aligned} \tag{A.9}$$

Reforming the above energy equation into a linear system $Ax = b$ the values for $\theta, \phi, \psi, t_x, t_y, t_z$ can be found by solving for $x = (A^T A)^{-1} A^T b$ where

$$A_{\{Nx6\}} = \begin{bmatrix} n_0^x \hat{p}_0^z - n_0^z \hat{p}_0^x & n_0^z \hat{p}_0^y - n_0^y \hat{p}_0^z & n_0^y \hat{p}_0^x - n_0^x \hat{p}_0^y & n_0^x & n_0^y & n_0^z \\ n_1^x \hat{p}_1^z - n_1^z \hat{p}_1^x & n_1^z \hat{p}_1^y - n_1^y \hat{p}_1^z & n_1^y \hat{p}_1^x - n_1^x \hat{p}_1^y & n_1^x & n_1^y & n_1^z \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ n_N^x \hat{p}_N^z - n_N^z \hat{p}_N^x & n_N^z \hat{p}_N^y - n_N^y \hat{p}_N^z & n_N^y \hat{p}_N^x - n_N^x \hat{p}_N^y & n_N^x & n_N^y & n_N^z \end{bmatrix} \tag{A.10}$$

$$A^T A_{\{6x6\}} = \sum_{i=1}^{N} \begin{bmatrix} (n_i \times \hat{p}_i)(n_i \times \hat{p}_i)^T & (n_i \times \hat{p}_i) n_i^T \\ n_i (n_i \times \hat{p}_i)^T & n_i n_i^T \end{bmatrix} \tag{A.11}$$

$$x = \begin{bmatrix} \theta & \phi & \psi & t_x & t_y & t_z \end{bmatrix}^T \tag{A.12}$$

$$b_{\{Nx1\}} = \begin{bmatrix} n_0^x \hat{p}_0^x + n_0^y \hat{p}_0^y + n_0^z \hat{p}_0^z - n_0^x q_0^x - n_0^y q_0^y - n_0^z q_0^z \\ n_1^x \hat{p}_1^x + n_1^y \hat{p}_1^y + n_1^z \hat{p}_1^z - n_1^x q_1^x - n_1^y q_1^y - n_1^z q_1^z \\ \vdots \\ n_N^x \hat{p}_N^x + n_N^y \hat{p}_N^y + n_N^z \hat{p}_N^z - n_N^x q_N^x - n_N^y q_N^y - n_N^z q_N^z \end{bmatrix} \tag{A.13}$$

$$A^T b_{\{6x1\}} = \sum_{i=1}^{N} \begin{bmatrix} (n_i \times \hat{p}_i)((\hat{p}_i - q_i)^T n_i) \\ n_i((\hat{p}_i - q_i)^T n_i) \end{bmatrix} \tag{A.14}$$

Inserting the values in x into $\tilde{T}$ yields a linearized transformation that can be composed with ${}^k_1T$ to yield a more refined estimate of T. However the value calculated for $\tilde{T}$ is not a Euclidean transformation because the rotational compoenent is not a rotation matrix. Composing a Euclidean transformation with a non-Euclidean transformation yields a non-Euclidean transformation, so $\tilde{T}$ needs to be taken to the nearest Euclidean transformation.

$$\tilde{T} = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \tag{A.15}$$

$$M = R\left(\frac{\mathbf{e_1 e_1^T}}{\sqrt{\lambda_1}} + \frac{\mathbf{e_2 e_2^T}}{\sqrt{\lambda_2}} + \frac{\mathbf{e_3 e_3^T}}{\sqrt{\lambda_3}} \right) \tag{A.16}$$

$$\tilde{T} = \begin{bmatrix} M & t \\ 0 & 1 \end{bmatrix} \tag{A.17}$$

where λ_i and $\mathbf{e_i}$ is the i^{th} eigenvalue and eigenvector, respectively, for R^TR

A.2 Point-to-Point Error Metric

Given point clouds, P and Q, correspondence between the two clouds, and a transformation, T, from P to Q the point-to-point error metric measure the distance between a point, p_i, and its corresponding point, q_i. The cost function between clouds P and Q is

$$E = \sum_{i=1}^{N} \left([Tp_i - q_i]\right)^2 \tag{A.18}$$

. Finding the value of T that minimizes this cost function will bring P and Q into alignment, provided the correspondences are correct, where T is a Euclidean transformation. P and Q are of the form

$$P = \begin{bmatrix} p_0^x & p_0^y & p_0^z \\ p_1^x & p_1^y & p_1^z \\ \vdots & \vdots & \vdots \\ p_N^x & p_N^y & p_N^z \end{bmatrix} \tag{A.19}$$

The value for T that minimizes this cost function can be directly solved using Constrained Orthogonal Procrustes Analysis.

$$t = \hat{P} - \hat{Q} \tag{A.20}$$

$$A = \bar{P}^T \bar{Q} \tag{A.21}$$

$$U \Sigma V^T = A \tag{A.22}$$

$$R = U \Sigma' V^T \tag{A.23}$$

$$\Sigma' = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & sign(det(UV^T)) \end{bmatrix} \tag{A.24}$$

$$T = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \tag{A.25}$$

where $\hat{P}$ and $\hat{Q}$ are the centroids of point cloud P and Q respectivally and $\bar{P}$ and $\bar{Q}$ are the centralized point clouds P and Q respectively.

Appendix B

Graph Optimization Techniques

B.1 Gauss-Seidel Method

Gauss-Seidel is an iterative technique used for solving the standard linear system

$$Ax = b \tag{B.1}$$

Instead of naivly solving for $x = A^{-1}b$, which could take a lot of computation time if A is large, if A is positive-definite or diagnoally domninant matrix Gauss-Seidel can solve for x by decomposing A into a lower triangular matrix, L, and strictly uper triangular matrix, U.

$$\begin{aligned} (L+U)x &= b & \text{(B.2)} \\ Lx &= b - Ux & \text{(B.3)} \end{aligned}$$

Since Gauss-Seidel is an iterative technique the values for x in the $k^{th}+1$ iteration, $x^{(k+1)}$, are solved for using values of x in the k^{th} iteration, $x^{(k)}$

$$\begin{aligned} Lx^{(k+1)} &= b - Ux^{(k)} && \text{(B.4)} \\ x^{(k+1)} &= L^{-1}(b - Ux^{(k)}) && \text{(B.5)} \end{aligned}$$

Since L is a lower triangular matrix the above equation is solved by forward substitution to yield the following iterative technique

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j<i} a_{ij} x_j^{(k+1)} - \sum_{j>i} a_{ij} x_j^{(k)} \right) \tag{B.6}$$

where, in each iteration of k, i increments from 1 to n, the number of rows, and j increments from 1 to n for each iteration of i.

B.1.1 Forward Substitution

For a linear system in the form $Lx = b$ where L is a lower triangular matrix x can be solved quickly using forward substitution

$$\begin{aligned} l_{1,1}x_1 &= b_1 && \text{(B.7)} \\ l_{2,1}x_1 + l_{2,2}x_2 &= b_2 && \text{(B.8)} \\ &\vdots && \text{(B.9)} \\ l_{m,1}x_1 + l_{m,2}x_2 + \ldots l_{m,m}x_m &= b_m && \text{(B.10)} \end{aligned}$$

$$
\begin{aligned}
x_1 &= \frac{b_1}{l_{1,1}} && \text{(B.11)}\\
x_2 &= \frac{b_2 - l_{2,1}x_1}{l_{2,2}} && \text{(B.12)}\\
&\vdots && \text{(B.13)}\\
x_m &= \frac{b_m - \sum_{i=1}^{m-1} l_{m,i}x_i}{l_{m,m}} && \text{(B.14)}
\end{aligned}
$$

B.2 Stochastic Gradient Descent

Stochastic gradient descent is a method to minimize a cost function which is typically of the following form.

$$F(\zeta) = \sum_{i=1}^{N} F_i(\zeta) \tag{B.15}$$

Where $F()$ is the cost function, ζ is the set of parameters to be estimated, and F_i is the cost of the i^{th} observation. A standard gradient descent algorithm would compute the gradient of F with respect to parameters, $\nabla F(\zeta)$ and the parameters would be updated by making a step along the direction of the gradient.

$$\zeta = \zeta + \alpha \nabla F(\zeta) \tag{B.16}$$

where α is some learning rate. This process would be repeated until convergence.

Stochastic Gradient Descent approximates $\nabla F(\zeta)$ by only considering one observation at a time.

$$\nabla F(\zeta) \approx \nabla F_i(\zeta) \tag{B.17}$$

$$\zeta = \zeta + \alpha F_i(\zeta). \tag{B.18}$$

The observation to be considered is randomly selected from the set of observations and is not considered again until all other observations have been used. This process is repeated until convergence.

Appendix C

Graph Optimization Derivations

This chapter contains a detailed derivation of the Jacobian used in the POReSS algorithm as well as the derivation of the linear system used in the Graph-Seidel algorithm, both of which are discussed in Chapter 6.

C.1 Differentiating RSS Relative Pose

$$
\begin{aligned}
{}^{a}\mathbf{p}_b &= \sum_{k=a+1}^{b} R({}^{a}\theta_{k-1})\mathbf{x}_k && \text{(C.1)}\\
&= \sum_{k=a+1}^{b} \begin{bmatrix} \cos {}^{a}\theta_{k-1} & -\sin {}^{a}\theta_{k-1} & 0 \\ \sin {}^{a}\theta_{k-1} & \cos {}^{a}\theta_{k-1} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x'_k \\ y'_k \\ \theta'_k \end{bmatrix} && \text{(C.2)}\\
&= \sum_{k=a+1}^{b} \begin{bmatrix} x'_k \cos {}^{a}\theta_{k-1} - y'_k \sin {}^{a}\theta_{k-1} \\ x'_k \sin {}^{a}\theta_{k-1} + y'_k \cos {}^{a}\theta_{k-1} \\ \theta'_k \end{bmatrix} && \text{(C.3)}
\end{aligned}
$$

Taking derivatives yields, when $a+1 \leq i \leq b$,

$$\frac{\partial}{\partial x_i}{}^a\mathbf{p}_b = \begin{bmatrix} \cos {}^a\theta_{i-1} \\ \sin {}^a\theta_{i-1} \\ 0 \end{bmatrix} \tag{C.4}$$

$$\frac{\partial}{\partial y_i}{}^a\mathbf{p}_b = \begin{bmatrix} -\sin {}^a\theta_{i-1} \\ \cos {}^a\theta_{i-1} \\ 0 \end{bmatrix} \tag{C.5}$$

$$\frac{\partial}{\partial \theta_i}{}^a\mathbf{p}_b = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} + \sum_{k=i}^{b} \begin{bmatrix} -x'_k \sin {}^a\theta_{k-1} - y'_k \cos {}^a\theta_{k-1} \\ x'_k \cos {}^a\theta_{k-1} - y'_k \sin {}^a\theta_{k-1} \\ 0 \end{bmatrix} \tag{C.6}$$

or $\begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T$ otherwise for each. The summation limit comes from the fact that ${}^a\theta_{k-1}$ depends only upon $\theta'_{a+1}, \ldots, \theta'_{k-1}$. Therefore ${}^a\mathbf{p}_b$ consists of a summation of terms like $f_{a+1:a} + f_{a+1:a+1} + f_{a+1:a+2} + f_{a+1:a+3} + \cdots + f_{a+1:b-1}$, where $f_{a+1:a} = 0$. All the terms depend upon θ_{a+1}, so differentiating with respect to θ_i when $i = a+1$ yields all the terms. But the first term does not depend upon θ_{a+2}, so differentiating with respect to θ_i when $i = a+2$ yields all but the first term. And so forth.

Concatenating these:

$${}^a_bB_i \equiv \frac{\partial}{\partial \mathbf{x}_i}{}^a\mathbf{p}_b = \begin{bmatrix} \cos {}^a\theta_{i-1} & -\sin {}^a\theta_{i-1} & {}^a_b\alpha_i \\ \sin {}^a\theta_{i-1} & \cos {}^a\theta_{i-1} & {}^a_b\beta_i \\ 0 & 0 & 1 \end{bmatrix} \tag{C.7}$$

where

$$\begin{bmatrix} {}^a_b\alpha_i \\ {}^a_b\beta_i \end{bmatrix} = \sum_{k=i}^{b} \begin{bmatrix} -x'_k \sin {}^a\theta_{k-1} - y'_k \cos {}^a\theta_{k-1} \\ x'_k \cos {}^a\theta_{k-1} - y'_k \sin {}^a\theta_{k-1} \end{bmatrix} \tag{C.8}$$

By definition,

$$J_{ab} \equiv \begin{bmatrix} {}^a_bB_1 & {}^a_bB_2 & \cdots & {}^a_bB_n \end{bmatrix} \tag{C.9}$$

C.2 Graph-Seidel Derivation

Combining (6.15) and (6.19) yields

$$\mathbf{r}_{ab}(\mathbf{x}) = \delta_{ab} - R^T(\theta_a)(\mathbf{p}_b - \mathbf{p}_a) \tag{C.10}$$

From (6.2) we have

$$\begin{aligned} \epsilon(\mathbf{x}) &\equiv \sum_{(a,b)\in\mathcal{E}} \mathbf{r}^T_{ab}(\mathbf{x})\Omega_{ab}\mathbf{r}_{ab}(\mathbf{x}) && \text{(C.11)} \\ &= \sum_{(a,b)\in\mathcal{E}} \mathbf{r}^T_{ab}(\mathbf{x})R^TR\Omega_{ab}R^TR\mathbf{r}_{ab}(\mathbf{x}) && \text{(C.12)} \\ &= \sum_{(a,b)\in\mathcal{E}} (\mathbf{r}'_{ab})^T(\mathbf{x})\Omega'_{ab}\mathbf{r}'_{ab}(\mathbf{x}) && \text{(C.13)} \end{aligned}$$

for any orthogonal matrix R, where

$$\begin{aligned} \mathbf{r}'_{ab}(\mathbf{x}) &\equiv R\mathbf{r}_{ab}(\mathbf{x}) && \text{(C.14)} \\ \Omega'_{ab} &\equiv R\Omega_{ab}R^T. && \text{(C.15)} \end{aligned}$$

If we let $R = R(\theta_a)$, then we have

$$\mathbf{r}'_{ab}(\mathbf{x}) \;=\; \mathbf{p}_a - \mathbf{p}_b + R(\theta_a)\delta_{ab}. \tag{C.16}$$

Now the trick is to assume, for the moment, that $R(\theta_a)$ is constant. This assumption is not true, of course, because we will also be solving for θ_a, but it simplifies the math (and algorithm) considerably, and as we shall see in a moment does not affect the result because we recompute $R(\theta_a)$ each iteration anyway.

The key insight is that, if $R(\theta_a)$ is constant, then $\epsilon(\mathbf{x})$ is convex in the states. As a result, we do not need to linearize the system at all but instead can simply take derivatives to solve directly for the states, then iterate by updating $R(\theta_a)$. This leads to a very simple and fast algorithm.

We have

$$\begin{aligned}
\epsilon(\mathbf{x}) &= \sum_{(a,b)\in\mathcal{E}} \epsilon_{ab}(\mathbf{x}) && \text{(C.17)}\\
&= \sum_{(a,b)\in\mathcal{E}} \underbrace{(\mathbf{r}'_{ab})^T(\mathbf{x})\Omega'_{ab}\mathbf{r}'_{ab}(\mathbf{x})}_{\mathbf{g}_{ab}(\mathbf{x})} && \text{(C.18)}\\
&= \sum_{(a,i)\in\mathcal{E}_i^{in}} \mathbf{g}_{ai}(\mathbf{x}) + \sum_{(i,b)\in\mathcal{E}_i^{out}} \mathbf{g}_{ib}(\mathbf{x}) + \sum_{(a,b)\in\mathcal{E}_i^{\neg}} \mathbf{g}_{ab}(\mathbf{x}), && \text{(C.19)}
\end{aligned}$$

for any i, where

$$\begin{aligned}
\mathcal{E}_i^{in} &\equiv \{(a,b) : (a,b)\in\mathcal{E} \text{ and } b = i\} && \text{(C.20)}\\
\mathcal{E}_i^{out} &\equiv \{(a,b) : (a,b)\in\mathcal{E} \text{ and } a = i\} && \text{(C.21)}\\
\mathcal{E}_i^{\neg} &\equiv \{(a,b) : (a,b)\in\mathcal{E} \text{ and } a \neq i \text{ and } b \neq i\} && \text{(C.22)}
\end{aligned}$$

so that $\mathcal{E} = \mathcal{E}_i^{in} \bigcup \mathcal{E}_i^{out} \bigcup \mathcal{E}_i^{\neg}$ for any i.

Assuming $R(\theta_a)$ is constant, taking derivatives yields

$$\begin{aligned}\frac{\partial \mathbf{g}_{ai}(\mathbf{x})}{\partial \mathbf{p}_i} &= -\Omega'_{ai}\mathbf{r}'_{ai}(\mathbf{x}) && \text{(C.23)}\\ \frac{\partial \mathbf{g}_{ib}(\mathbf{x})}{\partial \mathbf{p}_i} &= \Omega'_{ib}\mathbf{r}'_{ib}(\mathbf{x}) && \text{(C.24)}\\ & && \text{(C.25)}\end{aligned}$$

Putting these together:

$$\frac{\partial \epsilon(\mathbf{x})}{\partial \mathbf{p}_i} = -\sum_{(a,i)\in\mathcal{E}_i^{in}} \Omega'_{ai}\mathbf{r}'_{ai}(\mathbf{x}) + \sum_{(i,b)\in\mathcal{E}_i^{out}} \Omega'_{ib}\mathbf{r}'_{ib}(\mathbf{x}) \qquad \text{(C.26)}$$

Setting the derivative to zero:

$$\begin{aligned}\sum_{(a,i)\in\mathcal{E}_i^{in}} \Omega'_{ai}\mathbf{r}'_{ai}(\mathbf{x}) &= \sum_{(i,b)\in\mathcal{E}_i^{out}} \Omega'_{ib}\mathbf{r}'_{ib}(\mathbf{x}) && \text{(C.27)}\\ \sum_{(a,i)\in\mathcal{E}_i^{in}} \Omega'_{ai}(\mathbf{p}_a - \mathbf{p}_i + R(\theta_a)\delta_{ai}) &= \sum_{(i,b)\in\mathcal{E}_i^{out}} \Omega'_{ib}(\mathbf{p}_i - \mathbf{p}_b + R(\theta_i)\delta_{ib}) && \text{(C.28)}\\ & && \text{(C.29)}\end{aligned}$$

or

$$\begin{aligned}&\left(\sum_{(a,i)\in\mathcal{E}_i^{in}} \Omega'_{ai} + \sum_{(i,b)\in\mathcal{E}_i^{out}} \Omega'_{ib}\right)\mathbf{p}_i && \text{(C.30)}\\ -&\left(\sum_{(a,i)\in\mathcal{E}_i^{in}} \Omega'_{ai}\right)\mathbf{p}_a - \left(\sum_{(i,b)\in\mathcal{E}_i^{out}} \Omega'_{ib}\right)\mathbf{p}_b && \text{(C.31)}\\ =&\sum_{(a,i)\in\mathcal{E}_i^{in}} \Omega'_{ai}R(\theta_a)\delta_{ai} - \sum_{(i,b)\in\mathcal{E}_i^{out}} \Omega'_{ib}R(\theta_i)\delta_{ib} && \text{(C.32)}\end{aligned}$$

Since we are using a GSS, we have $\mathbf{x} = \mathbf{p}$, and therefore

$$\begin{bmatrix} \overline{\Omega}_1 & -\overline{\Omega}_{12} & \cdots & -\overline{\Omega}_{1n} \\ -\overline{\Omega}_{21} & \overline{\Omega}_2 & \cdots & -\overline{\Omega}_{2n} \\ \vdots & & \ddots & \vdots \\ -\overline{\Omega}_{n1} & -\overline{\Omega}_{n2} & \cdots & \overline{\Omega}_n \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_n \end{bmatrix} = \begin{bmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \\ \vdots \\ \mathbf{v}_n \end{bmatrix}, \tag{C.33}$$

where, assuming we do not have a multigraph,

$$\overline{\Omega}_{ab} \equiv \overline{\Omega}_{ba} \equiv \begin{cases} \Omega'_{ab} & \text{if } \delta_{ab} \text{ exists} \\ \Omega'_{ba} & \text{if } \delta_{ba} \text{ exists} \\ \mathbf{0}_{\{3\times3\}} & \text{otherwise} \end{cases} \tag{C.34}$$

$$\overline{\Omega}_a \equiv \sum_b \overline{\Omega}_{ab} \tag{C.35}$$

$$\mathbf{v}_i \equiv \underbrace{\sum_{(a,i)\in\mathcal{E}_i^{in}} R(\theta_a)\Omega_{ai}\delta_{ai}}_{\text{edges in}} - \underbrace{\sum_{(i,b)\in\mathcal{E}_i^{out}} R(\theta_i)\Omega_{ib}\delta_{ib}}_{\text{edges out}}. \tag{C.36}$$

This linear system, which is the form $Ax = b$, can be solved using Gauss-Seidel since the combination of all elements in $\overline{\Omega}_{ab} \geq 0$ and equation (C.35) makes A diagonally dominant, $|a_{ii}| \geq \sum_{j\neq i}|a_{ij}|$ for all i.

Bibliography

[1] S. Agarwal, N. Snavely, S. M. Seitz, and R. Szeliski. Bundle adjustment in the large. In *Proceedings of the European Conference on Computer Vision*, 2010.

[2] C. Audras, A. I. Comport, M. Meilland, and P. Rives. Real-time dense RGB-D localisation and mapping. In *Australian Conference on Robotics and Automation*, Dec. 2011.

[3] S. Baker and I. Matthews. Lucas-Kanade 20 years on: A unifying framework. *International Journal of Computer Vision*, 56(3):221–255, 2004.

[4] B. Benoit and R. Chaoui. Three-dimensional ultrasound with maximal mode rendering: a novel technique for the diagnosis of bilateral or unilateral absence or hypoplasia of nasal bones in second-trimester screening for Down syndrome. *Journal of the International Society of Ultrasound in Obstetrics and Gynecology*, 25(1):19–24, Jan. 2005.

[5] G. Blais and M. Levine. Registering multiview range data to create 3D computer objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 17(8):820–824, Aug. 1995.

[6] J.-L. Blanco, F.-A. Moreno, and J. Gonzalez. A collection of outdoor robotic datasets with centimeter-accuracy ground truth. *Autonomous Robots*, 27(4):327–351, Nov. 2009.

[7] V. Blanz and T. Vetter. A morphable model for the synthesis of 3D faces. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '99, pages 187–194, 1999.

[8] M. Botsch, A. Wiratanaya, and L. Kobbelt. Efficient high quality rendering of point sampled geometry. In *Thirteenth Eurographics Workshop on Rendering*, 2002.

[9] M. Byröd and K. Astroöem. Conjugate gradient bundle adjustment. In *Proceedings of the European Conference on Computer Vision*, 2010.

[10] I. Carlbom, D. Terzopoulos, and K. Harris. Computer-assisted registration, segmentation, and 3D reconstruction from images of neuronal tissue sections. *IEEE Transactions on Medical Imaging*, 13(2):351–362, 1994.

[11] Y. Chen and G. Medioni. Object modeling by registration of multiple range images. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 3, pages 2724–2729, Apr. 1991.

[12] B. Curless and M. Levoy. A volumetric method for building complex models from range images. In *Proceedings of SIGGRAPH*, pages 303–312, 1996.

[13] A. Davison, I. Reid, N. Molton, and O. Stasse. MonoSLAM: Real-time single camera SLAM. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(6):1052–1067, 2007.

[14] F. Dellaert and M. Kaess. Square root SAM: Simultaneous location and mapping via square root information smoothing. *International Journal of Robotics Research*, 25(12):1181–1204, Dec. 2006.

[15] L. Douadi, M.-J. Aldon, and A. Crosnier. Pair-wise registration of 3D/color data sets with ICP. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 663–668, Oct. 2006.

[16] S. Druon, M. Aldon, and A. Crosnier. Color constrained ICP for registration of large unstructured 3D color data sets. In *IEEE International Conference on Information Acquisition*, pages 249 – 255, Aug 2006.

[17] T. Duckett, S. Marsland, and J. Shapiro. Learning globally consistent maps by relaxation. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, Apr. 2000.

[18] B. Duvenhage, J. P. Delport, and J. de Villiers. Implementation of the Lucas-Kanade image registration algorithm on a GPU for 3D computational platform stabilisation. In *Proceedings of the 7th International Conference on Computer Graphics, Virtual Reality, Visualisation and Interaction in Africa (AFRIGRAPH)*, pages 83–90, 2010.

[19] A. Elfes. Occupancy grids: A probabilistic framework for robot perception and navigation. *Journal of Robotics and Automation*, RA-3(3):249–265, June 1987.

[20] F. Endres, J. Hess, N. Engelhard, J. Sturm, D. Cremers, and W. Burgard. An evaluation of the RGB-D SLAM system. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, May 2012.

[21] A. Flint, C. Mei, I. Reid, and D. Murray. Growing semantically meaningful models for visual SLAM. In *Computer Vision and Pattern Recognition CVPR*, 2010.

[22] U. Frese, P. Larsson, and T. Duckett. A multilevel relaxation algorithm for simultaneous localisation and mapping. *IEEE Transactions on Robotics*, 21(2):196–207, Apr. 2005.

[23] Y. Furukawa, B. Curless, S. Seitz, and R. Szeliski. Manhattan-world stereo. In *Computer Vision and Pattern Recognition CVPR*, June 2009.

[24] Y. Furukawa, B. Curless, S. M. Seitz, and R. Szeliski. Reconstructing building interiors from images. In *Proceedings of the International Conference on Computer Vision*, Sept. 2009.

[25] G. Grisetti, S. Grzonka, C. Stachniss, P. Pfaff, and W. Burgard. Efficient estimation of accurate maximum likelihood maps in 3D. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3472–3478, Oct. 2007.

[26] G. Grisetti, C. Stachniss, and W. Burgard. Nonlinear constraint network optimization for efficient map learning. *IEEE Transactions on Intelligent Transportation Systems*, 10(3):428–439, Sept. 2009.

[27] G. Grisetti, C. Stachniss, S. Grzonka, and W. Burgard. A tree parameterization for efficiently computing maximum likelihood maps using gradient descent. In *Proceedings of Robotics: Science and Systems (RSS)*, June 2007.

[28] J.-S. Gutmann, M. Fukuchi, and M. Fujita. A floor and obstacle height map for 3d navigation of a humanoid robot. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, pages 1066–1071, 2005.

[29] J.-S. Gutmann and K. Konolige. Incremental mapping of large cyclic environments. In *IEEE Intl. Symp. on Computational Intelligence in Robotics and Automation (CIRA)*, pages 318–325, November 2000.

[30] D. Häehnel, W. Burgard, D. Fox, and S. Thrun. An efficient FastSLAM algorithm for generating maps of large-scale cyclic environments from raw laser range measurements. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2003.

[31] 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. *International Journal of Robotics Research*, 31(5):647–663, Apr. 2012.

[32] A. Hoover, D. Goldgof, and K. Bowyer. Egomotion estimation of range camera using the space envelope. *IEEE Transactions on Systems, Man, and Cybernetics*, 33(4):717–721, 2003.

[33] A. Howard, M. J. Matarić, and G. Sukhatme. Relaxation on a mesh: A formalism for generalized localization. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2001.

[34] A. Huang, A. Bachrach, P. Henry, M. Krainin, D. Maturana, D. Fox, and N. Roy. Visual odometry and mapping for autonomous flight using an RGB-D camera. In *International Symposium on Robotics Research (ISRR)*, Aug. 2011.

[35] G. Q. Huang, A. B. Rad, and Y. K. Wong. Online SLAM in dynamic environments. In *Proceedings of the 12th International Conference on Advanced Robotics*, pages 262–267, July 2005.

[36] B. Huhle, M. Magnusson, W. Strasser, and A. J. Lilienthal. Registration of colored 3D point clouds with a kernel-based extension to the normal distributions transform. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, pages 4025 – 4030, May 2008.

[37] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. Davison, and A. Fitzgibbon. KinectFusion: Real-time 3D reconstruction and interaction using a moving depth camera. In *Proceedings of the 24th ACM Symposium on User Interface Software and Technology*, pages 559–568, 2011.

[38] J. H. Joung, K. H. An, J. W. Kang, M. J. Chung, and W. Yu. 3D environment reconstruction using modified color ICP algorithm by fusion of a camera and a 3D laser range finder. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3082–3088, Oct. 2009.

[39] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. Leonard, and F. Dellaert. iSAM2: Incremental smoothing and mapping using the bayes tree. *International Journal of Robotics Research*, 31:217–236, Feb. 2012.

[40] M. Kaess, A. Ranganathan, and F. Dellaert. iSAM: Fast incremental smoothing and mapping with efficient data association. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 2007.

[41] M. Kaess, A. Ranganathan, and F. Dellaert. iSAM: Incremental smoothing and mapping. In *IEEE Transactions on Robotics*, 2008.

[42] C. Kerl, J. Sturm, and D. Cremers. Dense visual SLAM for RGB-D cameras. In *Proceedings of the International Conference on Intelligent Robot Systems (IROS)*, Nov. 2013.

[43] C. Kerl, J. Sturm, and D. Cremers. Robust odometry estimation for RGB-D cameras. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, May 2013.

[44] J. Klaess, J. Stueckler, and S. Behnke. Efficient mobile robot navigation using 3D surfel grid maps. In *Robotics; Proceedings of ROBOTIK 2012; 7th German Conference on*, pages 1–4, 2012.

[45] K. Konolige. Large-scale map-making. In *Proceedings of the National Conference on Artificial Intelligence*, July 2004.

[46] K. Konolige, G. Grisetti, R. Kummerle, W. Burgard, B. Limketkai, and R. Vincent. Sparse pose adjustment for 2D mapping. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2010.

[47] D. Krakow, J. Williams, M. Poehl, D. L. Rimoin, and L. D. Platt. Use of three-dimensional ultrasound imaging in the diagnosis of prenatal-onset skeletal dysplasias. *Journal of the International Society of Ultrasound in Obstetrics and Gynecology*, 21(5):467–472, May 2003.

[48] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. g2o: A general framework for graph optimization. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, pages 3607–3613, May 2011.

[49] K.-L. Low. Linear least-squares optimization for point-to-plane ICP surface registration. Technical Report TR04-004, University of North Carolina, Feb. 2004.

[50] F. Lu and E. Milios. Globally consistent range scan alignment for environment mapping. *Autonomous Robots*, 4:333–349, 1997.

[51] B. D. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence*, pages 674–679, 1981.

[52] T. K. Marks, A. Howard, M. Bajracharya, G. W. Cottrell, and L. Matthies. Gamma-SLAM: Using stereo vision and variance grid maps for SLAM in unstructured environments. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 2008.

[53] M. Montemerlo and S. Thrun. Large-scale robotic 3-D mapping of urban structures. In *Proceedings of the International Symposium on Experimental Robotics (ISER)*, June 2004.

[54] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. FastSLAM: A factored solution to the simultaneous localization and mapping problem. In *Proceedings of the AAAI National Conference on Artificial Intelligence*, 2002.

[55] H. Moravec and A. E. Elfes. High resolution maps from wide angle sonar. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 116–121, Mar. 1985.

[56] B. Morisset, R. B. Rusu, A. Sundaresan, K. Hauser, M. Agrawal, J. claude Latombe, and M. Beetz. Leaving flatland: Toward real-time 3D navigation.

[57] F. Mourgues, F. Devemay, and E. Coste-Maniere. 3D reconstruction of the operating field for image overlay in 3D-endoscopic surgery. In *Augmented Reality, 2001. Proceedings. IEEE and ACM International Symposium on*, pages 191–192, 2001.

[58] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohli, J. Shotton, S. Hodges, and A. Fitzgibbon. KinectFusion: Real-time dense surface mapping and tracking. In *Proceedings of the 10th IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 127–136, 2011.

[59] R. A. Newcombe, S. J. Lovegrove, and A. J. Davison. DTAM: Dense tracking and mapping in real-time. In *Proceedings of the International Conference on Computer Vision*, pages 2320–2327, 2011.

[60] Z. Noh, M. Sunar, and Z. Pan. A review on augmented reality for virtual heritage system. In *Learning by Playing. Game-based Education System Design and Development*, volume 5670 of *Lecture Notes in Computer Science*, pages 50–61. Springer Berlin Heidelberg, 2009.

[61] E. Olson, J. Leonard, and S. Teller. Fast iterative optimization of pose graphs with poor initial estimates. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, pages 2262–2269, May 2006.

[62] E. Olson, J. Leonard, and S. Teller. Spatially-adaptive learning rates for online incremental SLAM. In *Robotics: Science and Systems*, June 2007.

[63] K. Pathak, A. Birk, N. Vaskevicius, M. Pfingsthorn, S. Schwertfeger, and J. Poppinga. Online three-dimesional SLAM by registration of large planar surface segments and closed-form pose-graph relaxation. *Journal of Field Robotics*, 27(1):52–84, Jan. 2010.

[64] K. Pathak, A. Birk, N. Vaskevicius, and J. Poppinga. Fast registration based on noisy planes with unknown correspondences for 3-D mapping. *IEEE Transactions on Robotics and Automation*, 26(3):424–441, June 2010.

[65] B. Peasley and S. Birchfield. Replacing projective data association with Lucas-Kanade for KinectFusion. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, May 2013.

[66] B. Peasley, S. Birchfield, A. Cunningham, and F. Dellaert. Accurate on-line 3D occupancy grids using Manhattan world constraints. In *Proceedings of the*

IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Oct. 2012.

[67] P. Pfaff, R. Triebel, and W. Burgard. An efficient extension to elevation maps for outdoor terrain mapping and loop closing. *International Journal of Robotics Research*, 2007.

[68] A. Ranganathan, M. Kaess, and F. Dellaert. Loopy SAM. In *Proceedings of the International Joint Conference on Artificial Intelligence*, Jan. 2007.

[69] S. Rusinkiewicz, O. Hall-Holt, and M. Levoy. Real-time 3D model acquisition. *ACM Transactions on Graphics (SIGGRAPH)*, 21(3):438–446, July 2002.

[70] S. Rusinkiewicz and M. Levoy. Efficient variants of the ICP algorithm. In *Third International Conference on 3D Digital Imaging and Modeling (3DIM)*, June 2001.

[71] R. B. Rusu and S. Cousins. 3D is here: Point Cloud Library (PCL). In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, May 2011.

[72] A. Sahbani, S. El-Khoury, and P. Bidaud. An overview of 3D object grasp synthesis algorithms. *Robotics and Autonomous Systems*, 60(3):326–336, Mar. 2012.

[73] G. Sibley, L. Matthies, and G. Sukhatme. A sliding window filter for incremental slam. In D. Kragic and V. Kyrki, editors, *Unifying Perspectives in Computational and Robot Vision*, volume 8 of *Lecture Notes in Electrical Engineering*, pages 103–112. Springer US, 2008.

[74] S. N. Sinha, J.-M. Frahm, M. Pollefeys, and Y. Genc. GPU-based video feature tracking and matching. In *EDGE Workshop on Edge Computing Using New Commodity Architectures*, May 2006.

[75] R. Smith and P. Cheeseman. On the representation and estimation of spatial uncertainty. *International Journal of Robotics Research*, 5:56–68, 1987.

[76] R. Smith, M. Self, and P. Cheeseman. Estimating uncertain spatial relationships in robotics. In I. J. Cox and G. T. Wilfong, editors, *Autonomous Robot Vehicles*, pages 167–193. Springer-Verlag, 1990.

[77] F. Steinbrücker, J. Sturm, and D. Cremers. Real-time visual odometry from dense RGB-D images. In *Workshop on Live Dense Reconstruction with Moving Cameras (at ICCV)*, Nov. 2011.

[78] J. Strom and E. Olson. Occupancy grid rasterization in large environments for teams of robots. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2011.

[79] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A benchmark for the evaluation of RGB-D SLAM systems. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 573–580, Oct. 2012.

[80] S. Thrun. Learning occupancy grid maps with forward sensor models. *Autonomous Robots*, 15(2):111–127, 2003.

[81] S. Thrun. Robotic mapping: a survey. In *Exploring artificial intelligence in the new millennium*, pages 1–35. Morgan Kaufmann, Inc., 2003.

[82] C. Tomasi and T. Kanade. Detection and tracking of point features. Technical Report CMU-CS-91-132, Carnegie Mellon University, Apr. 1991.

[83] T. Tykkälä, C. Audras, and A. I. Comport. Direct iterative closest point for real-time visual odometry. In *Second International Workshop on Computer Vision in Vehicle Technology*, Nov. 2011.

[84] A. Uckermann, C. Elbrechter, R. Haschke, and H. Ritter. 3D scene segmentation for autonomous robot grasping. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 1734–1740, 2012.

[85] B. Wang, L. Jiang, J. Li, and H. Cai. Grasping unknown objects based on 3D model reconstruction. In *Advanced Intelligent Mechatronics. Proceedings, 2005 IEEE/ASME International Conference on*, pages 461–466, 2005.

[86] T. Whelan, H. Johannsson, M. Kaess, J. J. Leonard, and J. McDonald. Robust real-time visual odometry for dense RGB-D mapping. In *Proceedings of the International Conference on Robotics and Automation (ICRA)*, May 2013.

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

[88] K. M. Wurm, A. Hornung, M. Bennewitz, C. Stachniss, and W. Burgard. OctoMap: A probabilistic, flexible, and compact 3D map representation for robotic systems. In *Proceedings of the ICRA Workshop on Best Practice in 3D Perception and Modeling for Mobile Manipulation*, May 2010.