Real-Time Dense Simultaneous Localization and Mapping using Monocular Cameras

by

W. Nicholas Greene

B.S.E., Princeton University (2010)

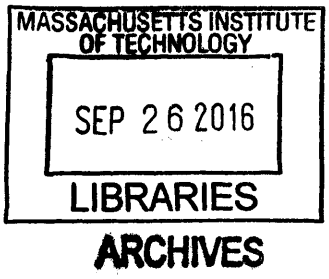
Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of

Master of Science in Aeronautics and Astronautics

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2016



©2016 William Nicholas Greene, All rights reserved.
The author hereby grants to MIT and The Charles Stark Draper Laboratory, Inc. permission to reproduce and to distribute publicly paper and electronic copies of this thesis document in whole or in any part medium now known or hereafter created.

Author ..... Signature redacted
Department of Aeronautics and Astronautics
August 18, 2016

Certified by ..... Signature redacted .....
Nicholas Roy
Associate Professor of Aeronautics and Astronautics
Thesis Supervisor

Certified by .... Signature redacted .....
Dr. Ted J. Steiner
Senior Member of the Technical Staff, Draper
Thesis Supervisor

Accepted by ..... Signature redacted .....
Paulo C. Lozano
Associate Professor of Aeronautics and Astronautics
Chair, Graduate Program Committee

Real-Time Dense Simultaneous Localization and Mapping using Monocular Cameras

by

W. Nicholas Greene

Submitted to the Department of Aeronautics and Astronautics
on August 18, 2016, in partial fulfillment of the
requirements for the degree of
Master of Science in Aeronautics and Astronautics

Abstract

Cameras are powerful sensors for robotic navigation as they provide high-resolution environment information (color, shape, texture, etc.), while being lightweight, low-power, and inexpensive. Exploiting such sensor data for navigation tasks typically falls into the realm of monocular simultaneous localization and mapping (SLAM), where both the robot's pose and a map of the environment are estimated concurrently from the imagery produced by a single camera mounted on the robot.

This thesis presents a monocular SLAM solution capable of reconstructing dense 3D geometry online without the aid of a graphics processing unit (GPU). The key contribution is a multi-resolution depth estimation and spatial smoothing process that exploits the correlation between low-texture image regions and simple planar structure to adaptively scale the complexity of the generated keyframe depthmaps to the quality of the input imagery. High-texture image regions are represented at higher resolutions to capture fine detail, while low-texture regions are represented at coarser resolutions for smooth surfaces. This approach allows for significant computational savings while simultaneously increasing reconstruction density and quality when compared to the state-of-the-art. Preliminary qualitative results are also presented for an adaptive meshing technique that generates dense reconstructions using only the pixels necessary to represent the scene geometry, which further reduces the computational requirements for fully dense reconstructions.

Thesis Supervisor: Nicholas Roy
Title: Associate Professor of Aeronautics and Astronautics

Thesis Supervisor: Dr. Ted J. Steiner
Title: Senior Member of the Technical Staff, Draper

Acknowledgments

First and foremost, I would like to thank my family for all their support and encouragement over the years, especially Dad for prodding me to get my foot in the MIT door. Thank you to Nick Roy for giving me the opportunity the join his group and for all the invaluable advice and guidance, especially before I became a grad student. Thank you to the Draper Fellow Program and the DARPA Fast Lightweight Autonomy Program for funding my work, as well as Paul DeBitetto, Pete Lommel, Ted Steiner, and all the members of Draper for providing additional mentorship and feedback. Thanks to Sanjeev Kulkarni at Princeton and Jonathan Su and all my old colleagues at Lincoln Laboratory for providing the academic foundation on which I now stand. Finally, thanks to all the members of the Robust Robotics Group and the FLA team for making research fun.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Despite the rapid progress in monocular SLAM over the last several years, state-of-the-art approaches are either too computationally expensive, too limited in scale, or too geometrically sparse to be successfully used for high-speed MAV navigation. This thesis presents research that addresses some of the shortcomings of sparse, dense, and semi-dense monocular SLAM methods and allows fully dense geometry to be estimated with the speed, efficiency, and scale of semi-dense methods. This chapter outlines the motivation behind the research (Section 1.1), a brief introduction to the monocular SLAM problem (Section 1.2), and provides an overview of the subsequent chapters (Section 1.3).

1.1 Motivation

The mobile robotics field has expanded rapidly over the last twenty years, driven by a combination of factors including advances in small, lightweight sensors, robust algorithms expressed in the language of probability theory to interpret those sensors, and powerful computing hardware to drive those algorithms. While robots have been used in industrial contexts, mobile robots are just now becoming viable for many non-industrial applications, including for autonomous driving [1], aerial photography [2], search and rescue [3], and intelligence, surveillance, and reconnaissance (ISR) [4].

One class of robotic platforms that has received a sizable amount of interest over

this time has been micro-aerial vehicles (MAVs) such as quadrotors, hexrotors, and fixed-wing aircraft, which typically weigh between 0.5-5kg and measure between 0.25-2m in diameter. Their speed and agility, coupled with their mechanical simplicity and well-understood dynamics models, make them ideal platforms for academic research as well as applications ranging from package delivery [5, 6] to planetary exploration [7, 8].

Despite a number of successes and continued promise, however, today's autonomous MAVs are severely limited in the speed and agility with which they can safely fly, especially when restricted to the unknown, GPS-denied, and comms-denied environments that occur in the real world. Without pre-built maps, external localization services such as GPS, or communications links like Wi-Fi, MAVs must perform all sensing and computation onboard, which is a significant challenge given the platform's size, weight, and power (SWaP) restrictions and which effectively constrains today's MAVs to navigating 2D slices of the world at speeds far below their dynamic capabilities.

Efficiently extracting information relevant to collision-free motion using only the low-SWaP sensors and computation that can be carried onboard these small robots is the overarching problem that motivates this work. While high-speed, GPS-denied navigation presents challenges at all levels of the autonomy pipeline (from low-level control to high-level task planning), this thesis focuses on improving the state-of-the-art in state estimation and perception using only a single commodity monocular camera as the primary sensor and a mobile computer without a high-SWaP discrete graphics processing unit (GPU) as the primary compute device. The central algorithm of this thesis, Multi-Level Mapping (MLM) [9], as well as an adaptive meshing approach, will be presented and shown to produce the dense geometric information needed for high-speed motion planning even with low-SWaP hardware.

1.2 Simultaneous Localization and Mapping

Any state estimation and perception system used on a mobile robot typically needs to provide two principal quantities for the purposes of collision-free navigation in an unknown environment: the *pose* (i.e. position and orientation) of the robot and a

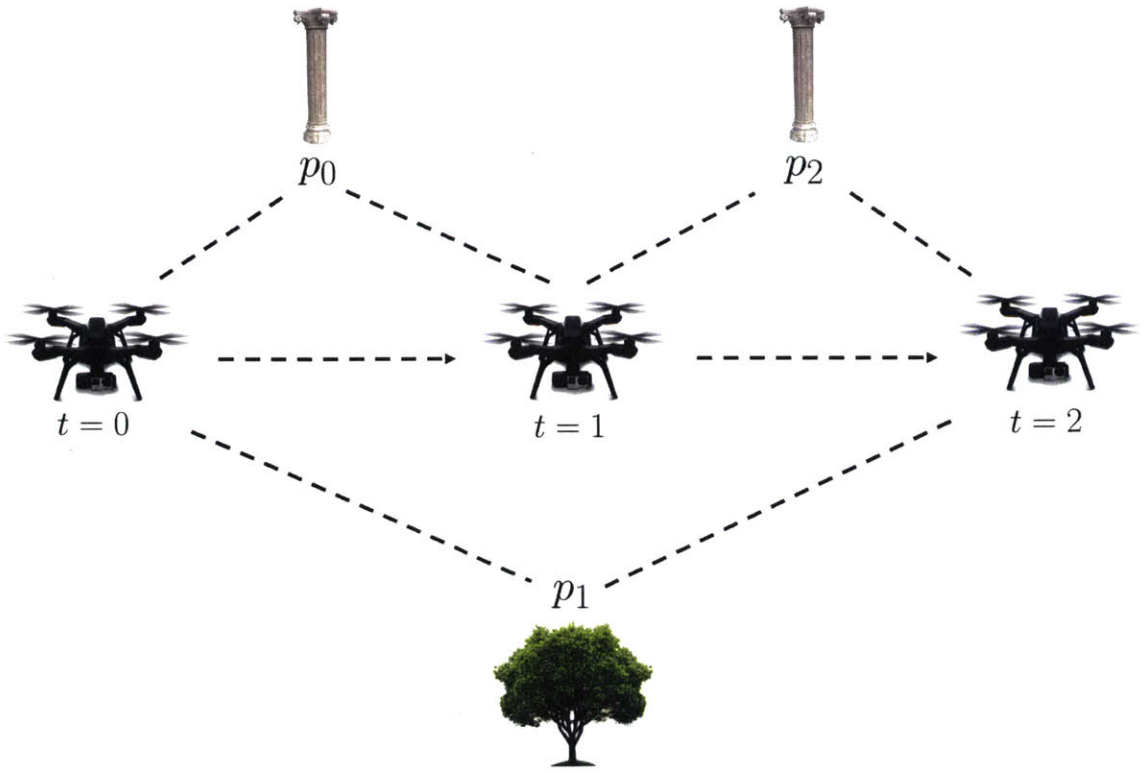


Figure 1-1: Simultaneous localization and mapping (SLAM) is a classical problem in robotics where a robot equipped with an imperfect sensor (e.g. a quadrotor with a camera) moves through an environment and observes various landmarks (e.g. pillars and trees) over time. The robot must fuse the noisy measurements of the landmarks to estimate its pose, while simultaneously mapping the environment. In the scenario depicted, a quadrotor with a camera moves through the world from time $t = 0$ to $t = 2$ and observes pillar landmarks p_0 and p_2 and tree landmark p_1.

map of the environment that encodes obstacles. Both quantities are usually needed to order to plan motions through the environment that do not collide with obstacles.

Simultaneous localization and mapping (SLAM) is the traditional formulation of this problem where a robot with imperfect sensors traverses an unknown environment with a set of landmarks. As the robot moves, it perceives the landmarks through its sensors and fuses these noisy measurements in order to *localize* itself with respect to those landmarks, which have to be estimated (or *mapped*) concurrently (see Figure 1-1). The chicken-and-egg nature of the problem (estimate the robot's pose requires a map, and estimating the map requires the robot's pose), has led to many strategies to compute both quantities simultaneously in real-time.

(a) 2D Scanning Lidar

(b) Active Structure-Light RGB-D Camera

(c) Passive Stereo Camera

(d) Passive Monocular Camera

Figure 1-2: A variety of sensors can be used for SLAM, including 2D scanning LIDARS [10], structured light RGB-D cameras [11], passive stereo cameras [12], and passive monocular cameras [13]. Each sensing modality has strengths and weaknesses, but this work concentrates on passive monocular cameras since they can be used both indoors and outdoors, can range to arbitrarily far objects given sufficient baseline, and are low SWaP, inexpensive, and ubiquitous.

1.2.1 Why Cameras?

A variety of sensing modalities may be used to drive a SLAM pipeline (see Figure 1-2), but traditional sensor suites typically include an inertial measurement unit (IMU) that measures the linear accelerations and angular rates experience by the robot and a range sensor such as an ultrasonic rangefinder or scanning LIDAR [10]. Scanning LIDARS, both the 2D and 3D varieties, have driven much of the work in SLAM over the last fifteen years, but are problematic for small, agile robots such as MAVs because they are either high-SWaP (the smallest 3D LIDARS typically weigh more than 1kg) or do not adequately observe the environment (2D LIDARS only perceive slices of the environment at a time, which is insufficient for MAVs undergoing severe pitch and roll angles).

Cameras, on the other hand, are attractive sensors for high-speed MAV navigation as they provide high-resolution environment information (color, shape, texture, etc.), while being lightweight, low-power, and inexpensive. Active or multi-camera variants such as structured-light and stereo cameras may also perceive depth information

directly, which is valuable for SLAM. This work, however, focuses on monocular (i.e. passive, single-lens) cameras for a number of reasons.

First, active sensors based on structured light or time-of-flight technology [11] have a limited detection range (typically $\sim$ 5m) and are essentially inoperable in sunlight due to electromagnetic interference. Passive stereo cameras [12] work outdoors, but their detection range is limited by the baseline between the two cameras, which is necessarily small on MAVs, and requires accurate calibration to be effective.

Passive monocular cameras [13], however, are robust to sunlight and can detect distant structure given sufficient translational motion between frames. Spurred by the rapid adoption of smartphone technology, they are also smaller, lighter, and cheaper than the aforementioned alternatives and nearly ubiquitous. The next sections outline the three dominant approaches to performing SLAM using monocular cameras without an onboard IMU.

1.2.2 Sparse Monocular SLAM

Sparse monocular SLAM is the most mature form of SLAM using monocular cameras and has its roots in the photogrammetry and computer vision communities as a related problem called *Structure-from-Motion* (SfM) [14, 15]. SfM can be considered an offline version of sparse monocular SLAM that generalizes to the case of multiple cameras, non-sequential images, and images taken at different times [16]. Given a set of images, SfM algorithms estimate the pose of the camera where each image was taken, along with a *sparse* point cloud of features which represent the map. Typically, salient features (e.g. corners or lines) [17, 18, 19] are detected in the images and associated across frames. Given these associations, the relative pose transform between the images can be inferred [20, 21], which then informs the 3D locations of the points that correspond to the features. Finding the optimal setting for both the poses and map points is usually accomplished using an approach called *bundle adjustment* [22, 23, 24, 16, 25, 26], which frames the problem as a sparse, nonlinear least squares objective that can be optimized efficiently using the Gauss-Newton or Levenberg-Marquardt algorithms [27, 28].

While early sparse monocular SLAM methods used the probabilistic filtering approaches common at the time for LIDAR-based SLAM [29], modern variants effectively perform online bundle adjustment by estimating the pose for a subset of images, while concurrently estimating a sparse feature-based point cloud map [30, 31]. Salient features are first detected in an image, and then associated with the features that comprise the current map, which allows the current pose of the camera to be estimated. The camera poses are then used to refine the map by minimizing a nonlinear least squares objective function.

1.2.3 Dense Monocular SLAM

While sparse monocular SLAM algorithms are still preferred when optimizing for the camera trajectory due to their ability to cleanly and efficiently close loops and minimize drift (even more so when fused with IMU information [32, 33, 34]), the sparse point cloud map representation poses problems if the map is to be used for online motion planning to avoid obstacles. Although the number and density of features may vary from algorithm to algorithm, they typically do not allow for free and occupied space to be differentiated.

Driven by the proliferation of massively parallel graphics processing units (GPUs) and general purpose GPU (GPGPU) programming languages such as CUDA and OpenCL, as well as interest from the virtual and augmented reality communities, *dense* monocular SLAM methods have shown promise in computing the type of maps required for high-speed, online motion planning [35, 36, 37, 38, 39, 40]. Leveraging the power of the GPU, dense approaches estimate depth for *every* pixel in each incoming frame into either dense point cloud or mesh representations that can be used to describe obstacles in extremely fine detail.

The roots of these approaches lie in the stereo and multi-view stereo (MVS) literature, where dense geometry is estimated from a set on images taken from known poses [41, 42, 43]. Rather than detecting a sparse set of salient features and associating them across images, MVS algorithms typically match patches of pixels in one image with those of another comparison image that lie along the *epipolar line* — the

projection of the ray of all possible depths onto the comparison image. The noisy depth estimates are then fused and spatially regularized.

1.2.4 Semi-Dense Monocular SLAM

Though there has been some work porting dense monocular SLAM methods to mobile devices [39], dense methods still prove to be too computationally expensive for low-SWaP MAVs, even with recent system-on-a-chip (SOC) GPU hardware [44]. Furthermore, state-of-the-art dense algorithms do not currently scale to the large environments that high-speed MAVs will likely need to fly through.

Semi-dense monocular SLAM approaches are currently an attractive compromise between the speed and scale of sparse methods and the density of dense methods [45, 46, 47, 48]. Rather than estimating depth for *every* input pixel, semi-dense approaches focus computational resources on the high-gradient pixels that carry the most information, while ignoring low-texture regions that carry very weak depth information. This allows semi-dense approaches, such as LSD-SLAM [47], to capture detailed geometry in high-texture areas, but still scale to large environments and run at camera frame-rate without the need for GPU acceleration.

1.3 Thesis Overview

Despite the speed and scale that semi-dense monocular SLAM algorithms provide, the resulting point cloud maps suffer from the same problems as those of sparse methods — namely that they do not allow free and occupied space to be differentiated and thus are not yet able to be used for high-speed MAV navigation. This thesis presents research that addresses some of the shortcomings of the aforementioned sparse, dense, and semi-dense monocular SLAM methods and allows fully dense geometry to be estimated with the speed, efficiency, and scale of semi-dense methods, bringing MAV-capable dense monocular SLAM one step closer to viability.

The central result of this thesis is a dense algorithm called Multi-Level Mapping (MLM) [9], which allows dense 3D geometry to be estimated online without the aid

of a graphics processing unit (GPU). The key contribution is a multi-resolution depth estimation and spatial smoothing process that exploits the correlation between low-texture image regions and simple planar structure to adaptively scale the complexity of the generated reconstruction to the quality of the input imagery. High-texture image regions are represented at higher resolutions to capture fine detail, while low-texture regions are represented at coarser resolutions for smooth surfaces. This approach allows for significant computational savings while increasing reconstruction density and quality compared to the state-of-the-art.

Preliminary results are also presented for an adaptive meshing technique that further reduces the computational requirements for fully dense reconstructions. The intuition behind the approach is that it is much cheaper to check whether a depth hypothesis is supported by the available imagery than it is to actually compute the correct depth. To that end, we maintain a piecewise-linear dense depth mesh whose vertices comprised a subset of the high-image gradient pixels. At each frame, the mesh is refined by computing the stereo matching cost for each pixel in the dense depthmap induced by the mesh. Vertices are then added to the mesh where the interpolated depthmap poorly fits the input imagery, such that depths are only computed for those pixels that are needed to represent the geometry in the scene, which may result in a significant speedup over the state-of-the-art.

The layout of the rest of this thesis is as follows. Chapter 2 presents an overview of the monocular SLAM problem with relevant related work and background information. Chapter 3 then dives into the MLM algorithm in detail, while Chapter 4 contains the qualitative and quantitative evaluation of the algorithm. Chapter 5 presents the adaptive depth meshing technique along with preliminary results. The thesis is concluded in Chapter 6.

Chapter 2

Monocular Simultaneous Localization and Mapping

This chapter gives a brief overview of the monocular SLAM problem and its solutions. Section 2.1 discusses the probabilistic formulation of the SLAM problem, its modern interpretation as a sparse factor graph, and the distinction between the backend factor graph solvers and frontend factor generators that compose most modern SLAM pipelines. Section 2.2 then goes into more detail on the SLAM backend, first covering early filtering-based approaches that marginalize out the past before detailing modern smoothing-based methods that solve for the entire state trajectory. From there, Section 2.3 covers different aspects of the the monocular SLAM frontend, from epipolar geometry, to visual odometry, to depth estimation and regularization. Finally, Section 2.4 presents full monocular SLAM solutions in detail, including sparse, dense, and semi-dense pipelines.

2.1 Probabilistic Formulation

Recall the scenario depicted in Figure 1-1. At each discrete timestep $i \in \mathbb{N}$, our robot moves through the world and observes a set of landmarks through its imperfect sensors. It must then fuse those noisy measurements to estimate its state in addition to the state of the map.

To begin, let us denote the robot (or camera) pose at time i by $\mathbf{x}_i \in \mathcal{X}$. Typically, we are interested in the 3D pose of the robot and thus $\mathcal{X} = \mathbb{SE}(3)$, the group of rigid body transformations. At times, however, we may need to generalize to other groups such as Sim(3), the group of similarity transforms [47, 9].

We assume that the robot motion can be described by a first-order Markov model such that the distribution of the current pose $\mathbf{x}_i$ is independent of the past given the previous pose $\mathbf{x}_{i-1}$. If we let $\mathbf{X} = (\mathbf{x}_0, \ldots, \mathbf{x}_T)$ denote the pose history up to time T and $\mathbf{U} = (\mathbf{u}_1, \ldots, \mathbf{u}_T)$ denote the odometry or control input history, then the distribution of the pose history given the odometry $p(\mathbf{X}|\mathbf{U})$ can be factored according to $p(\mathbf{X}|\mathbf{U}) = p(\mathbf{x}_0) \prod_{i=1}^{T} p(\mathbf{x}_i|\mathbf{x}_{i-1}, \mathbf{u}_i)$, where $p(\mathbf{x}_0)$ denotes the prior on the initial state and $p(\mathbf{x}_i|\mathbf{x}_{i-1}, \mathbf{u}_i)$ denotes the motion model parameterized by odometry $\mathbf{u}_i$.

Next, let $\mathbf{P} = (\mathbf{p}_1, \ldots, \mathbf{p}_M)$ for $\mathbf{p}_j \in \mathbb{R}^3$ refer to the set of M landmarks in the map (we are typically only interested in their 3D positions). We assume that they are stationary. Over time, the sensor aboard the robot will observe the landmarks and produce some number K noisy measurements $\mathbf{Z} = (\mathbf{z}_1, \ldots, \mathbf{z}_K)$ for $\mathbf{z}_k \in \mathcal{Z}$. The measurement space $\mathcal{Z}$ will vary from system to system, but could be as simple as the image domain $\Omega \subset \mathbb{R}^2$ such that each measurement $\mathbf{z}_k$ corresponds to the pixel coordinates of a detected landmark. We assume that measurement $\mathbf{z}_k$ depends only on the pose of the robot $\mathbf{x}_{i_k}$ at the time of observation and the particular landmark under observation $\mathbf{p}_{j_k}$ and can be modeled with likelihood function $p(\mathbf{z}_k|\mathbf{x}_{i_k}, \mathbf{p}_{j_k})$. (Note that we assume the association between measurement $\mathbf{z}_k$ and the pose $\mathbf{x}_{i_k}$ and landmark $\mathbf{p}_{j_k}$ is known.)

With these definitions in hand, the posterior distribution of the pose history $\mathbf{X}$ and landmarks $\mathbf{P}$ given measurements $\mathbf{Z}$ and odometry $\mathbf{U}$ can be written as:

$$p(\mathbf{X}, \mathbf{P}|\mathbf{Z}, \mathbf{U}) \propto p(\mathbf{x}_0) \prod_{i=1}^{T} p(\mathbf{x}_i|\mathbf{x}_{i-1}, \mathbf{u}_i) \prod_{k=1}^{K} p(\mathbf{z}_k|\mathbf{x}_{i_k}, \mathbf{p}_{j_k}) \tag{2.1}$$

The objective of SLAM will be to compute an estimate of the pose history $\hat{\mathbf{X}}$ and the landmark map $\hat{\mathbf{P}}$ from this posterior distribution. For example, by maximizing $p(\mathbf{X}, \mathbf{P}|\mathbf{Z}, \mathbf{U})$ over $\mathbf{X}$ and $\mathbf{P}$, we obtain the the *maximum a posteriori* (MAP) estimate.

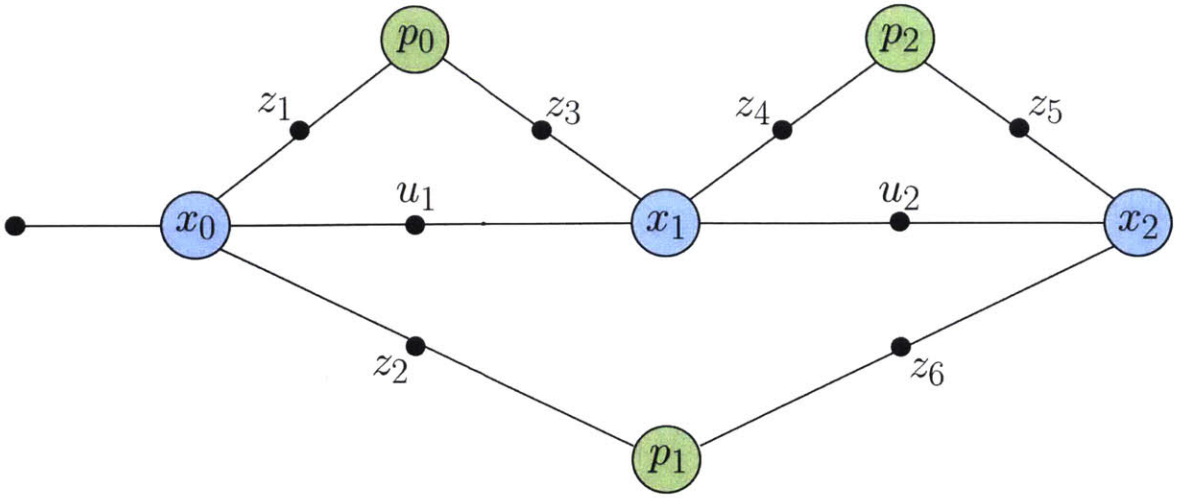


Figure 2-1: The posterior distribution of the SLAM problem as described in Equation (2.1) can be represented succinctly as a factor graph. The variable nodes $\mathbf{x}_0$, $\mathbf{x}_1$, $\mathbf{x}_2$ denote the robot state over time, while $\mathbf{p}_0$, $\mathbf{p}_1$, $\mathbf{p}_2$ signify the landmarks in the map. The factor nodes $\mathbf{u}_1$, $\mathbf{u}_2$ show odometry constraints, while the $\mathbf{z}_i$ factors represent noisy landmark observations. By emphasizing the factorization of the posterior distribution through independence relations, this interpretation reveals the sparse nature of the problem and insights into efficient solution strategies.

For further information on the probabilistic foundations of SLAM see [49, 50, 51].

2.1.1 Graphical Model

While the posterior distribution of the SLAM problem given in Equation (2.1) is useful in its own right, its graphical representation, particularly as a *factor graph* [52], can be more illuminating as it reveals more of the specific problem structure.

A factor graph $\mathcal{G} = (\mathcal{V}, \mathcal{F}, \mathcal{E})$ is a bipartite graph with variable nodes $v_i \in \mathcal{V}$, factor nodes $f_j \in \mathcal{F}$, and edges $e_{ij} \in \mathcal{E}$. Each edge e_{ij} connects a variable node and a factor node. As its name implies, a factor graph is used to model the factorization of a function over the variable nodes $f(\mathcal{V}) = \prod_j f_j(\mathcal{V}_j)$, where $\mathcal{V}_j \subseteq \mathcal{V}$. The specific factorization of f given by the $f_j \in \mathcal{F}$ dictates the graph structure. An edge e_{ij} connects variable v_i and factor f_j if $v_i \in \mathcal{V}_j$.

Looking to the posterior distribution $p(\mathbf{X}, \mathbf{P}|\mathbf{Z}, \mathbf{U})$ in Equation (2.1) again, we can see that it is a function of four sets of variables $p(\mathbf{X}, \mathbf{P}|\mathbf{Z}, \mathbf{U}) = f(\mathbf{X}, \mathbf{P}, \mathbf{U}, \mathbf{Z})$, and factorizes in a particular fashion given our conditional independence assump-

tions. We can therefore encode it as a factor graph with variable nodes $\mathcal{V} = (\mathbf{X}, \mathbf{P})$ and factor nodes $\mathcal{F} = (\mathbf{U}, \mathbf{Z})$. The odometry factors constrain particular pairs of pose variables, while the measurement factors constrain pairs of pose variables and landmark variables (see Figure 2-1).

Computing solutions to the SLAM problem can now be considered a special case of performing inference on probabilistic graphical models, where a number of relevant algorithms exist in the literature [52, 53]. Furthermore, it should be apparent that the graph structure induced by the SLAM problem is *sparse* — i.e. the number of edges in the graph $|\mathcal{E}|$ is far lower than that of a fully connected graph. This additional structure can be exploited for further computational savings, allowing for large SLAM problems to be solved quickly, in real-time [54, 55, 56].

2.1.2 Backend vs. Frontend

Given the graph interpretation of SLAM, a distinction can be made between *generating* the graph for a particular SLAM instance and *solving* that graph for a solution. We will refer to the former process – that of generating the SLAM graph from input data — as the SLAM *frontend* and refer to the latter process — that of finding a solution for a given graph — as the SLAM *backend.* Much of the current work in monocular SLAM deals more directly with the frontend, as generating factors is more specific to the particulars of the input data (images in this case). Nonetheless, it is worth covering a few of the influential SLAM backend solvers in more detail (Section 2.2) before venturing to the frontend (Section 2.3).

2.2 Backend

Before discussing modern smoothing-based SLAM solvers in Section 2.2.2, we will first detail the filter-based approaches that were popular during the early days of SLAM in Section 2.2.1.

2.2.1 Filter-based Approaches

Before the full implications of the graphical nature of SLAM was widely understood and appreciated, initial solutions modeled it as a *state estimation* problem to be solved via recursive Bayesian estimators or nonlinear *filters*, such as the extended Kalman filter (EKF), unscented Kalman filter (UKF) [57], and particle filter [58]. Posing SLAM as a state estimation problem was natural at the time, although we now know it to be a special case of the more general graphical interpretation.

The earliest filter-based SLAM solution is considered to be EKF-SLAM [59, 60], which (as its name implies) applied an EKF to compute the posterior distribution of the map $\mathbf{P}$ and the most recent pose $\mathbf{x}_T$ given the measurement history $\mathbf{Z} = (z_1, \ldots, z_T)$ and odometry history $\mathbf{U}$. Marginalizing out the past poses $\mathbf{X}_- = (\mathbf{x}_0, \ldots, \mathbf{x}_{T-1})$ from the posterior distribution in Equation (2.1) yields the familiar recursive Bayesian update formula:

$$p(\mathbf{x}_T, \mathbf{P}|\mathbf{Z}, \mathbf{U}) = \int_{\mathbf{X}_-} P(\mathbf{X}, \mathbf{P}|\mathbf{Z}, \mathbf{U}) d\mathbf{X}_- \tag{2.2}$$

$$= p(\mathbf{x}_T, \mathbf{P}|\mathbf{z}_T, \mathbf{Z}_-, \mathbf{U}) \tag{2.3}$$

$$\propto p(\mathbf{z}_T|\mathbf{x}_T, \mathbf{P}) p(\mathbf{x}_T, \mathbf{P}|\mathbf{Z}_-, \mathbf{U}) \tag{2.4}$$

$$= p(\mathbf{z}_T|\mathbf{x}_T, \mathbf{P}) \int_{\mathbf{x}_{T-1}} p(\mathbf{x}_T|\mathbf{x}_{T-1}, \mathbf{u}_T) p(\mathbf{x}_{T-1}, \mathbf{P}|\mathbf{Z}_-, \mathbf{U}_-) d\mathbf{x}_{T-1}, \tag{2.5}$$

where $\mathbf{Z}_- = (\mathbf{z}_1, \ldots, \mathbf{z}_{T-1})$ and $\mathbf{U}_- = (\mathbf{u}_1, \ldots, \mathbf{u}_{T-1})$. The act of marginalizing out past poses has the effect of eliminating those variables from the factor graph in 2-1 and introducing new factors between the remaining variables.

The following zero-mean, additive-Gaussian noise process and measurement models are assumed:

$$\mathbf{x}_i = f(\mathbf{x}_{i-1}, \mathbf{u}_i) + w_i, \quad w_i \sim \mathcal{N}(0, W) \tag{2.6}$$

$$\mathbf{z}_i = h(\mathbf{x}_{i_i}, \mathbf{p}_{j_i}) + v_i, , \quad v_i \sim \mathcal{N}(0, V). \tag{2.7}$$

When the prior $p(\mathbf{x}_0)$ is assumed to be Gaussian and the process and measurement

models above are differentiable, the Bayesian update equations can be computed in closed form (the well-known Kalman filter equations).

The first viable particle filter approach to SLAM is generally attributed to the FastSLAM algorithm [61, 62], which exploits *Rao-Blackwellization* to overcome the curse of dimensionality suffered by traditional particle filters. Typically, the number of samples required by a particle filter scales exponentially with the state space dimension (the dimension of the pose history $\mathbf{X}$ and map $\mathbf{P}$ in the SLAM case). Rao-Blackwellization allows significantly fewer particles to be used by partitioning the filter state such that the distribution of some state variables can be represented analytically.

In the SLAM context, the key insight is to apply the chain rule to the posterior such that the distribution over the landmarks factorizes:

$$p(\mathbf{X}, \mathbf{P}|\mathbf{Z}, \mathbf{U}) = p(\mathbf{X}|\mathbf{Z}, \mathbf{U})p(\mathbf{P}|\mathbf{X}, \mathbf{Z}, \mathbf{U}) \tag{2.8}$$

$$= p(\mathbf{X}|\mathbf{Z}, \mathbf{U}) \prod_{j=1}^{M} p(\mathbf{p}_j|\mathbf{X}, \mathbf{Z}, \mathbf{U}). \tag{2.9}$$

The conditional independence of the landmarks $\mathbf{P}$ given the trajectory $\mathbf{X}$ is readily apparent when the factor graph in Figure 2-1 is considered. Each landmark variable $\mathbf{p}_j \in \mathbf{P}$ can now be represented analytically with (for example) a Gaussian distribution, while only the pose variables need to be approximated with samples. Furthermore, the sampling the pose history $\mathbf{X}$ can be done incrementally by sampling a new pose for each particle at each timestep.

2.2.2 Smoothing-based Approaches

While the aforementioned filter-based approaches worked well for small-scale 2D problems, their limitations became apparent as SLAM moved to 3D, the process and measurement models involved (as described in Equation 2.6) became more and more nonlinear, and the scale of the problems under investigation grew with both the pose history and map size.

First and foremost, the cubic complexity of the EKF precludes large maps from being considered since the full covariance matrix over the current pose and landmarks has to be maintained and inverted. Furthermore, marginalizing out past poses has the side effect of introducing correlations between otherwise unrelated state variables, ensuring that the covariance matrix becomes dense over time. Marginalization also makes any errors introduced by the linearization process permanent, which can lead to inconsistent solutions [63].

Smoothing-based approaches, on the other hand, sidestep these issues by solving for the entire pose trajectory $\mathbf{X}$ along with the map $\mathbf{P}$. The key intuition is that when past poses are not marginalized away, the sparse structure of the SLAM problem, as indicated by the factor graph in Figure 2-1, can be maintained, allowing for solutions to be computed efficiently, despite the increase in the number of poses.

Consider the posterior distribution $p(\mathbf{X}, \mathbf{P}|\mathbf{Z}, \mathbf{U})$ from Equation (2.1) once more. If we assume the same process and measurement models as described in Equation (2.6), then maximizing the posterior can be framed as a nonlinear least squares problem:

$$\arg\max_{\mathbf{X},\mathbf{P}} p(\mathbf{X}, \mathbf{P}|\mathbf{Z}, \mathbf{U}) = \arg\min_{\mathbf{X},\mathbf{P}} \; -\log p(\mathbf{X}, \mathbf{P}|\mathbf{Z}, \mathbf{U}) \tag{2.10}$$

$$= \arg\min_{\mathbf{X},\mathbf{P}} \left\{ ||\mathbf{x}_0 - \mu_0||^2_{\Sigma_0} + \sum_{i=1}^{T} ||\mathbf{x}_i - f(\mathbf{x}_{i-1}, \mathbf{u}_i)||^2_W + \sum_{k=1}^{K} ||\mathbf{z}_k - h(\mathbf{x}_{i_k}, \mathbf{p}_{i_k})||^2_V \right\}. \tag{2.11}$$

When optimized using Gauss-Newton or Levenberg-Marquardt, the quadratic approximation to this objective can be described by $||\mathbf{A}\mathbf{x} - \mathbf{b}||^2$, for $\mathbf{x} = \begin{bmatrix} \mathbf{X}^T & \mathbf{P}^T \end{bmatrix}^T$. The *measurement Jacobian* $\mathbf{A}$ obtained by linearizing f and h is usually sparse, as is the *information matrix* $\mathbf{A}^T\mathbf{A}$, both of which can be derived from the factor graph in Figure 2-1.

This large, sparse nonlinear least squares problem can be solved efficiently in a number of ways [64, 65, 66, 67] and has a large crossover with SfM and bundle

adjustment [22, 23, 25, 16, 24, 28, 26]. The most recent approaches compute solutions to Equation (2.10) incrementally, which results in significant computational savings since the new factors that are added to the graph as the robot explores typically only affect nodes in the recent past [55, 56].

2.3 Frontend

Given a set of odometry factors $\mathbf{U}$ and landmark factors $\mathbf{Z}$, we can pass them off to a nonlinear least squares solver (such as iSAM [55, 56]) to compute the optimal pose trajectory $\mathbf{X}$ and map $\mathbf{P}$. Considerable effort must be exerted, however, to extract these factors from raw sensor data, which in the passive monocular camera case is a sequence of images. This section will outline the basic steps to computing odometry and landmark (or depth) factors from imagery.

2.3.1 Epipolar Geometry

The factors that we would like to extract from a given video sequence are typically defined by the *epipolar geometry* generated by the camera's extrinsic parameters (i.e. the camera poses) and intrinsic parameters (i.e. focal length and calibration) [15]. Consider the scenario depicted in Figure 2-2 with two images I_1 and I_2 from a video sequence with relative transform $\mathbf{T}_2^1$ between the camera poses. The 3D landmark $\mathbf{p}_1$ projects into I_1 as pixel $\mathbf{u}_1$, as do all points along the ray marked in red. This ray projects into I_2 as the line in green — the *epipolar line*. The pixel associated with $\mathbf{p}_1$ in I_2 must lie along this line. Now, if we knew $\mathbf{T}_2^1$, we would be able to compute the epipolar line and search for the pixel that matches $\mathbf{u}_1$ along the line. With the association we could then estimate the depth for $\mathbf{p}_1$. On the other hand, if we knew the association, we could then back out $\mathbf{T}_2^1$. The latter process of estimating the relative transformations between camera poses is called *visual odometry* and will be discussed in Section 2.3.2. The former process of computing depth from pixel associations is called *depth estimation* (or *stereopsis*) and will be detailed in Section 2.3.3. It should be noted that both problems rely on the ability to accurately and robustly associate

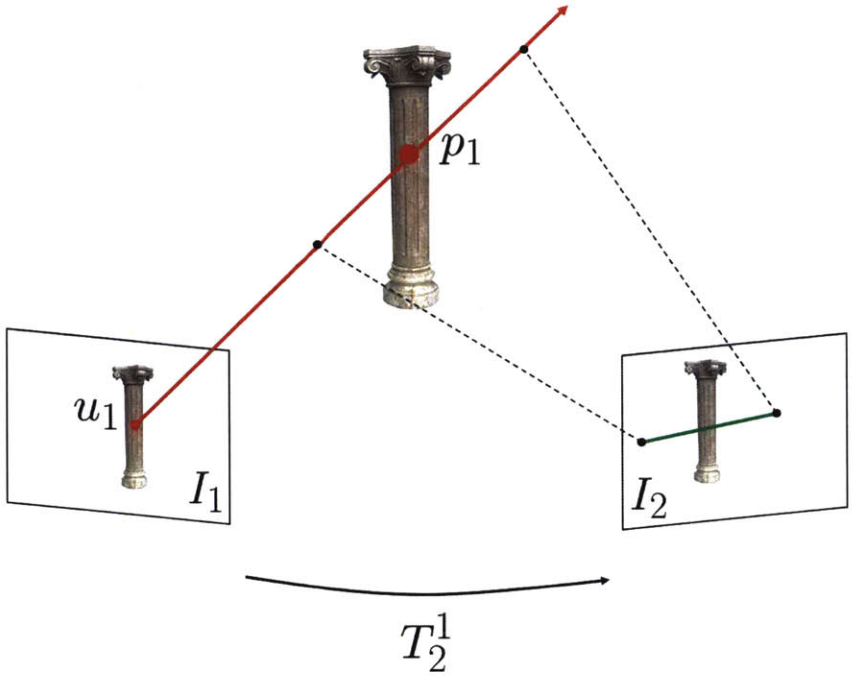


Figure 2-2: In this scenario we have two images I_1 and I_2 taken from two different poses, with the relative transformation between the two given by $\mathbf{T}_2^1$. The 3D point $\mathbf{p}_1$ projects into I_1 as pixel $\mathbf{u}_1$, as do all points along the ray in red. This ray projects into I_2 as the line in green — the *epipolar line*. The pixel in I_2 that is associated with $\mathbf{p}_1$ must lie along this line.

pixels across images.

2.3.2 Visual Odometry

Visual odometry entails estimating the relative pose of a moving camera (up to an unknown scale factor) using just its images. Once computed, the relative poses can then be added as odometry factors in a SLAM graph to optimize for the global trajectory. Techniques usually fall into either *feature-based* or *direct* methods.

Feature-based monocular visual odometry methods generally appeared much earlier than direct methods, with seminal work dating back to [68, 69, 70, 71, 72]. They rely on detecting a small number of salient features (e.g. corners, lines, or blobs) in the image stream that can be either tracked or matched across frames. The associated features can then be used to estimate the relative motion between the frames.

There is a large body of work in the computer vision literature on detecting,

tracking, and matching features across images. Feature detection typically involves efficiently selecting pixels in the image that are both distinctive and robust to scale, rotation, or lighting changes. The earliest detectors attempted to find corners in images by comparing patches around candidate pixels with slightly shifted versions. Moravec corners [70], for example, computed the sum-of-squared-differences (SSD) between the template patch and shifted patches along a set of cardinal directions, issuing a detection if the SSD was high for all directions (the mismatch between the patch and its shifted versions implies the pixel is distinctive). Harris corners [73] and Shi-Tomasi corners [74] improved upon this idea by computing a quadratic approximation to the SSD cost at the candidate pixel and labeling it a corner if the two eigenvalues of the Hessian (or some approximation to them) were both large. Rather than directly computing the Hessian of the SSD cost, FAST (Features from Accelerated Segment Test) [75] finds corners by performing a binary test between the candidate pixel and a set of its neighbors — if a contiguous number n of those neighbors are all brighter or darker than the center pixel, the candidate pixel is labeled a corner. Blobs, computed using the Difference-of-Gaussians (DoG) or Determinant-of-the-Hessian approaches, are also a popular features [17, 18].

Once a set of features are detected, they must be either tracked into the next frame (usually called sparse *optical flow*) or matched across different images. Feature tracking can be accomplished by performing a nonlinear least squares optimization to minimize the SSD between a patch around the feature in one image and its projection into the second image (usually called Lucas-Kanade registration) [76]. The famous Kanade-Lucas-Tomasi (KLT) feature tracker uses this least squares approach with Shi-Tomasi corners [77]. Feature matching typically involves computing a robust *descriptor* vector for the feature and performing brute-force, nearest-neighbor, or RANSAC [78] matching with outlier rejection. Just as there are many approaches to feature detection, there are many variations on computing robust (i.e. scale, rotation, illumination, noise invariant) descriptors. SIFT [17] and HoG [79] descriptors are based on histograms of gradient orientations around the feature, while SURF [18] features use the sum of Haar-like wavelet responses. BRIEF [80], ORB [19], and

Census [81] features all use a series of comparison tests between the feature and set of neighbors to construct a binary descriptor vector.

Once a set of features has been associated between two images, the 8-Point Algorithm of [68] or the later 5-Point Algorithm of [82, 83] can be used in conjunction with RANSAC to extract the relative transform between the two camera poses up to an unknown scale factor. (This scale ambiguity arises from the loss of absolute metric information that occurs when a 3D scene is projected onto a 2D image. Since multiple scenes may project as the same image, it is only possible to recover the scene geometry up to a scale factor.)

Consider Figure 2-2. Both the 8-Point and 5-Point algorithms exploit epipolar constraints to estimate the rigid body transform $\mathbf{T}_2^1 \in \mathbb{SE}(3)$, composed of rotation $\mathbf{R}_2^1 \in \mathbb{SO}(3)$ and translation $\mathbf{t}_2^1 \in \mathbb{R}^3$, between I_1 and I_2. Note that the vector $\mathbf{t}_2^1$ and the vector $\bar{\mathbf{u}}_1$ (the pixel $\mathbf{u}_1$ in homogeneous coordinates) define a plane — the *epipolar plane*. The normal vector of this plane is given by

$$\mathbf{t}_2^1 \times \bar{\mathbf{u}}_1 = \left[\mathbf{t}_2^1\right]_\times \bar{u}_1, \tag{2.12}$$

where if $\mathbf{t}_2^1 = \begin{bmatrix} x & y & z \end{bmatrix}^T$, the skew-symmetric matrix $\left[\mathbf{t}_2^1\right]_\times$ is defined as

$$\left[\mathbf{t}_2^1\right]_\times = \begin{bmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{bmatrix}. \tag{2.13}$$

This normal vector must be perpendicular to all vectors defined in the plane, including $\mathbf{p}_1 - \mathbf{t}_2^1 \propto \mathbf{R}_2^1 \bar{\mathbf{u}}_2$, where $\mathbf{u}_2$ is the projection of $\mathbf{p}_1$ into I_2. Thus,

$$\begin{aligned} \left(\mathbf{R}_2^1 \bar{\mathbf{u}}_2\right)^T \left[\mathbf{t}_2^1\right]_\times \bar{\mathbf{u}}_1 &= \bar{\mathbf{u}}_2^T \mathbf{R}_1^2 \left[\mathbf{t}_2^1\right]_\times \bar{\mathbf{u}}_1 && (2.14) \\ &= \bar{\mathbf{u}}_2^T \mathbf{E} \bar{\mathbf{u}}_1 && (2.15) \\ &= 0. && (2.16) \end{aligned}$$

The matrix $\mathbf{E} = \mathbf{R}_1^2 \left[\mathbf{t}_2^1\right]_\times$ is called the *essential matrix* and the constraint $\bar{\mathbf{u}}_2^T \mathbf{E} \bar{\mathbf{u}}_1 = 0$

must hold for all pixels $\mathbf{u}_1, \mathbf{u}_2$ that are associated with the same 3D point $\mathbf{p}_1$. Given a set of associated pixels, $\mathbf{E}$ can be estimated by solving the linear system of equations induced by the $\bar{\mathbf{u}}_2^T \mathbf{E} \bar{\mathbf{u}}_1 = 0$ constraints for each association. The rotation $\mathbf{R}_2^1$ and translation $\mathbf{t}_2^1$ can then be recovered using singular value decomposition (SVD). As their names suggest, the 8-Point and 5-Point algorithms require a minimum of 8 and 5 feature associations, respectively.

Rather than extracting and matching features, direct methods estimate the rigid body motion between the two camera poses by incrementally aligning the raw pixel values of the two images given depth information. The combination of depth information and an initial estimate of the transform $\mathbf{T}_2^1$ (e.g. the identity) allows pixels in I_1 to be projected into I_2. The transform is then iteratively refined by minimizing the pixel error between the projected (or *warped*) version of I_1 and I_2. This optimization is usually framed as a Lucas-Kanade [76] style nonlinear least squares problem:

$$\hat{\mathbf{T}}_2^1 = \underset{\mathbf{T} \in \mathbb{SE}(3)}{\arg\min} \sum_{\mathbf{u} \in \Omega_D} ||I_2(w(\mathbf{u}, D(\mathbf{u}), \mathbf{T})) - I_1(\mathbf{u})||^2, \tag{2.17}$$

where the warp function w projects pixel $\mathbf{u}$ into I_2 assuming depth $D(\mathbf{u})$ and relative pose $\mathbf{T}$. The set Ω_D is the set of pixels that have depths and can be a sparse sampling of pixels [45], the pixels in I_1 with high-gradient [46], or *every* pixel [35].

The objective in Equation (2.17) is usually minimized by performing Gauss-Newton or Levenberg-Marquardt steps [27], which repeatedly solve quadratic approximations to the cost by linearizing the residual $I_2(w(\mathbf{u}, D(\mathbf{u}), \mathbf{T})) - I_1(\mathbf{u})$ in $\mathbf{T}$. This approximation is typically only valid for small warps and so-called *coarse-to-fine* strategies often need to be employed to ensure a good solution [84, 85]. The optimization in 2.17 is first performed at the coarsest level of a power-of-two image pyramid, which removes high spatial frequency components from the images and allows a gross estimate of the transform to be obtained. That coarse solution is then used to initialize the optimization at the next, higher resolution level where it is refined.

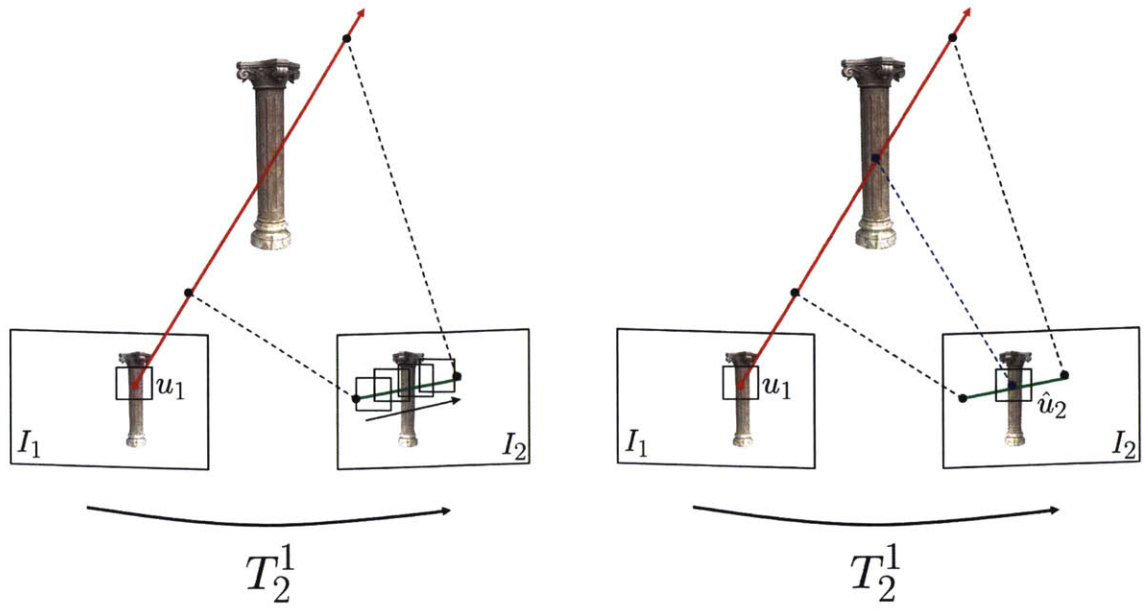
In addition, the L_2 norm used in Equation (2.17) may be overly sensitive to out-

liers, so it is often replaced by a robust error metric [86, 87] and optimized using *iteratively reweighted least squares* (IRLS), where a weighted version of Equation (2.17) is solved that approximates the solution using the robust norm [88, 89, 90]. For more detail on Lucas-Kanade optimization, see the tutorial presented in [91].

Direct visual odometry methods possess some notable advantages to sparse approaches. Since they rely on raw pixel intensities, no feature detection or extraction steps need to be performed, which can often be expensive. Not relying on salient features also means that more subtle (or general) information can be applied to the optimization — for example any pixel with gradient information can influence the optimization of Equation (2.17), while sparse methods rely on distinctive corner or blob detection. Furthermore, the feature matching process is usually prone to outliers and false matches, which can wreck havoc on the essential matrix estimation and is why RANSAC is usually employed. Direct methods, however, degrade gracefully with the presence of outliers — in addition to other forms of noise like motion blur and illumination changes– when robust statistics are leveraged. They do require depth information to be recovered, though, which is typically more difficult to obtain than the camera pose itself. The standard approaches to depth estimation will be detailed in Section 2.3.3.

2.3.3 Depth Estimation

Depth estimation (sometimes known as *3D reconstruction* or *stereopsis*) refers to the process of estimating the 3D structure of a scene (or the *depth* of the scene) given only a set of 2D images (in addition to the poses from which the images were captured and relevant camera calibration parameters). While depth information can be computed using some of the visual odometry building blocks from Section 2.3.2, such as feature detection, tracking, and matching (which is the basis for most sparse monocular SLAM systems), the emphasis here will be on computing *dense* depth representations that encode the surface geometry of the scene and allow for object modeling and novel view prediction. To differentiate this problem from dense monocular SLAM, remember that in the dense monocular SLAM case the camera poses are unknown



(a) Epipolar Search

(b) Triangulation Given Correspondence

Figure 2-3: Depth estimation from a set of images is fundamentally a pixel association or correspondence problem. With knowledge of the relative transform $\mathbf{T}_2^1$, the epipolar line corresponding to query pixel $\mathbf{u}_1$ can be computed in I_2. By searching for the pixel along this line that matches $\mathbf{u}_1$ as depicted in Figure 2-3a, the depth for $\mathbf{u}_1$ can be estimated via triangulation as in Figure 3-5.

and must be estimated (along with the scene depth) in real-time. A common dense monocular SLAM approach is to use spare methods to estimate the camera poses that are then used to infer the dense scene geometry [36, 39, 37]. Despite the prior knowledge of the camera poses, dense 3D reconstruction is still a difficult inference problem.

It is also worth noting that while estimating dense depth information is invaluable for understanding the 3D structure of an environment, the volume of data that this entails is problematic for backend SLAM solvers. Given the depths to every pixel in a camera image, the naive approach to incorporate this information into a SLAM graph would be to initialize a landmark for each of these pixels and add a corresponding landmark factor. When one considers the resolution and framerate of even the most modest cameras used for SLAM, however, the number of factors this approach would require will quickly overwhelm the state-of-the-art in sparse, incremental least squares solvers. For example, the results shown for the iSAM2 [56] solver typically involve

tens of thousands of factors. Compare this with a modest camera that produces VGA resolution images (640x480 pixels) at 30Hz, which would equate to roughly 300,000 landmark factors being inserted in the graph every 30 ms. The standard approach around this problem is to *remove* the map entirely from the graph and solve for just the pose trajectory (usually called *pose-graph* SLAM)[61, 62, 51]. The depth information can still be exploited, however, by incorporating it into the odometry factors (refer to dense visual odometry in Section 2.3.2).

The roots of the depth estimation problem lie predominantly in the stereo vision literature, where the special cases of two-frame rectified stereo depth estimation [92, 93, 94] and (offline) multi-view stereo depth estimation [41, 42, 43] have been studied extensively (see [95, 96, 97] for more information). As in the case of visual odometry (particularly the feature-based variant), depth estimation is fundamentally a data association or correspondence problem between the pixels of two images captured from different views. Consider Figure 2-3 where a two-frame depth estimation problem is set up. Suppose we wish to estimate the depth for pixel $\mathbf{u}_1$ in I_1. Since the camera poses are known, we can compute the epipolar line corresponding to $\mathbf{u}_1$ in I_2. The pixel in I_2 corresponding to $\mathbf{u}_1$ must lie along this line. We can thus *search* along this line for a pixel that matches $\mathbf{u}_1$ (usually a neighborhood around each candidate pixel is considered to make the matching more robust). The optimal match $\hat{\mathbf{u}}_2$ can then be triangulated with $\mathbf{u}_1$ to estimate the depth of the 3D point that projects onto the two pixels. Note that this process assumes that the neighborhood of pixels around $\mathbf{u}_1$ and $\hat{\mathbf{u}}_2$ are visually similar. This is typically called the *Lambertian* assumption, where 3D points remain visually similar when observed from multiple viewpoints.

Nearly all stereo depth estimation algorithms follow this basic approach of epipolar search, patch matching, and triangulation. There are a variety of matching costs that can be used to associate pixels including SSD, sum-of-absolute differences (SAD), normalized-cross-correlation (NCC), and binary matching costs such as the Census Transform [81] (all similar to the work in sparse feature matching).

In addition, the representation of the depth information is another important design choice as it often dictates how depth information from multiple compari-

son frames are fused. One standard approach is to define a 2D *depthmap*, *inverse* depthmap, or *disparity map* (they are usually interchangeable) [98, 99, 100]. A depthmap (or inverse depthmap) is a scalar function over the image domain $D : \Omega \rightarrow \mathbb{R}_+$ that maps each pixel location to a depth (or inverse depth). A disparity map is similar, but maps each pixel to disparity, which is the motion that a pixel undergoes when projected into another image. If the depth of a point is large compared to the baseline between the cameras, the pixel will not change much between frames and will thus exhibit low disparity. If the depth is small compared to the camera baselines, the pixel will undergo larger motion between frames and the resulting disparity will be large. Depth information can be fused in these representations using probabilistic filtering [46, 47, 45].

Voxel or grid-based approaches are another popular way to represent and fuse depth information. Each voxel can be used to represent the aggregated matching cost from each stereo comparison with high-cost voxels "carved" away [101, 102, 103]. Voxels can also represent the color of the scene (known as *voxel coloring*) [104] or the distance to the nearest surface [105, 106, 107, 41]. Triangular meshes [108] and other deformable models [109, 110] are also viable alternatives.

There is one important functional block that has been omitted from the discussion so far — namely *spatial regularization.* As one can imagine, the epipolar search, patch matching, triangulation, and fusion steps described above are all subject to noise, errors, and outliers, and when used in isolation typically generate poor quality reconstructions. (These approaches are usually referred to as *local* methods.) Furthermore, the depths in large textureless image regions are generally unobservable since the lack of gradient information cripples the patch matching process. (It is also worth mentioning that since depth estimation relies on the camera poses, and the poses generated from the visual odometry methods discussed in Section 2.3.2 are only known up to a scale factor, the depths produced are similarly scale ambiguous.)

Nonetheless, we have a good deal of prior information about the world that we can bring to bear to the problem. Firstly, we know that the world is locally smooth and that scene geometry can often be well described using 2D surfaces. In addition,

certain environments of interest (e.g. man-made environments) are approximately piecewise-planar. We can denoise the depth estimates computed above, or fill in the depths in textureless regions, to reflect this prior information. Depth estimation algorithms that attempt to do this are usually referred to as *global* methods.

The standard approach to applying spatial regularization to depth information is to pose the problem as *energy minimization* [109, 111, 112, 93] where the regularized solution must balance smoothness with fitting the input data:

$$E(D) = E_{smooth}(D) + \lambda E_{data}(D). \tag{2.18}$$

The scalar term $\lambda > 0$ controls the tradeoff between smoothness and data fit. Usually $E_{smooth}(D)$ is some functional that penalizes non-smoothness, for example some norm on the gradient ∇D (for discrete data, this could be approximated using forward or central differences).

One way to derive the energy terms above is to define a probability distribution over the depthmap pixels using a *Markov Random Field* (MRF) [53]. The Markov properties of the MRF allow smoothness constraints to be introduced quite easily by correlating neighboring depths. Inference can then be performed on the MRF to extract the optimal depthmap using min-flow/max-cut-type algorithms [113, 105, 114, 115, 116].

A faster (but less robust) approach can be employed for the rectified two-frame stereo case. Rectification is a common preprocessing step that warps the two images such that all epipolar lines are parallel and aligned along the image rows. The epipolar search can then be performed by walking along each image row independently. Dynamic Programming can then be used to find the optimal association between the left image row and the right image row [92, 117, 118, 119, 120].

The specific forms of the energy terms in Equation (2.18) can greatly impact the reconstruction quality. For most types of regularization problems (i.e. not depth regularization), a squared L_2 norm for both $E_{smooth}(D)$ and $E_{data}(D)$ would be a

natural first choice, for example:

$$E_{data}(D) = \int_{\Omega} ||D(\mathbf{u}) - g(\mathbf{u})||^2_{\Sigma} \, d\mathbf{u} \tag{2.19}$$

$$E_{smooth}(D) = \int_{\Omega} ||\nabla D(\mathbf{u})||^2_{\Sigma_0} \, d\mathbf{u}. \tag{2.20}$$

Note that these functions essentially correspond to the special case of Gaussian measurement model and Gaussian prior. There are two problems with this approach, however. First, when applied to $E_{data}(D)$, the quadratic nature of the squared L_2 norm makes the solution extremely sensitive to outliers or errors in the input data. Second, when applied to $E_{smooth}(D)$, the solution is unable to undergo large discontinuities, which are common for depth data (consider the edge between a near object and the faraway background).

An alternative choice for $E_{smooth}(D)$ that is able to enforce smoothness but capture large discontinuities was proposed by [121] and is commonly referred to as the *Total Variation* (TV) regularizer:

$$E_{smooth}(D) = \int_{\Omega} ||\nabla D(\mathbf{u})|| \, d\mathbf{u}. \tag{2.21}$$

By penalizing the L_2 norm of the gradient (as opposed to the square of the L_2 norm), the TV regularizer allows functions to undergo sharp discontinuities (technically it biases them to be piecewise constant). When combined with a squared L_2 norm in $E_{data}(D)$, this approach is referred to as the TV-L_2 or Rudin-Osher-Fatemi (ROF) model. One can replace the squared L_2 data term with an L_1 data term to add robustness to outliers, which yields the TV-L_1 model, or replace the L_2 norm in E_{smooth} with a Huber norm to yield the Huber-ROF model. A variety of optimization schemes for objectives of this type have been proposed, typically leveraging the Euler-Lagrange equation, non-smooth convex optimization, the proximal operator, or primal-dual approaches [93, 122, 106, 123, 124, 125, 126].

With an understanding of both filtering and smoothing approaches to backend optimization, as well as the generation of frontend factors through visual odometry

and depth estimation, we now transition to discussing full monocular SLAM pipelines in Section 2.4.

2.4 Full Systems

This section will highlight important monocular SLAM pipelines, which combine the building blocks from the preceding sections into fully functioning, real-time systems. We first discuss sparse approaches in Section 2.4.1, where a sparse point cloud map is solved for directly in the SLAM graph, before covering dense methods in Section 2.4.2 and semi-dense methods in Section 2.4.3, which separate the map reconstruction task from the underlying SLAM backend.

2.4.1 Sparse Methods

The first monocular SLAM system to operate in real-time at camera framerate is commonly attributed to [29], which used the EKF-SLAM backend described in Section 2.2.1 with a sparse set of point landmarks initialized using Shi-Tomasi corners [74] and tracked using an exhaustive SSD search inside an elliptical region determined by the landmarks' 3D uncertainty. While an important milestone, the approach suffered from the shortcomings of EKF-SLAM outlined in Section 2.2.2 (e.g. the cubic complexity of the EKF constrains the number of poses and landmarks that can be estimated). In order to maintain real-time pose estimation, the mapping component was scaled back such that only tens of landmarks are maintained by the filter at any point in time. Filter-based approaches based on FastSLAM[61, 62] were also explored by [127, 128].

The Parallel Tracking and Mapping (PTAM) algorithm developed by [30] sidestepped the constraints imposed by filter-based approaches by splitting the tracking (i.e. visual odometry) and mapping computations into separate threads that run in parallel at different rates. This approach was a significant departure from the state of monocular SLAM research at the time since it effectively allowed the least squares/bundle adjustment techniques from offline SfM [22, 23, 25, 16, 24, 28, 26]

to be used in a real-time context. With tracking and mapping split into separate threads, the least squares objective in Equation (2.10) can be solved efficiently in parallel. In the tracking thread, the current camera pose is computed relative to a past *keyframe* at framerate by holding the map points in Equation (2.10) fixed and removing those that are outside the current view from the optimization. The mapping thread can then optimize the full objective over the keyframe poses (a small subset of all available frames) and the landmarks at a lower rate, which allows many more landmarks to be considered. Compared to the tens of landmarks that the were supported with approach of [29], PTAM is able to map thousands of points in desktop-sized environments. The significance of this work is difficult to overstate as nearly all monocular SLAM systems since follow this parallel approach.

ORB-SLAM developed by [31] is a recent improvement to the general PTAM design that adds loop closure factors to constrain odometry drift along with several techniques to perform SLAM graph management. The choice of factors in graph-based SLAM has a significant impact on both tracking and mapping accuracy. In PTAM, for example, odometry factors are added to the graph whenever a new keyframe is initialized. If too few keyframes are created, tracking is likely to suffer since the overlap between the current image and the keyframe is small. However, if too many keyframes are created, then the backend least squares optimization can become prohibitively slow. This same argument can be applied to the creation of landmarks: too few and tracking may suffer, too many and the backend optimization be crawl to a halt. ORB-SLAM's solution to this problem is to liberally add both landmarks and keyframe factors to SLAM graph, but prune the graph over time such that only a small number of highly information points and poses remain. This allows for robust tracking even through erratic camera motion or low-texture regions, but fast loop closing and low drift as well.

Although technically not a full SLAM solution, the Semi-Direct Visual Odometry (SVO) algorithm of [45] is worth mentioning as it is able to estimate poses extremely efficiently (at up to 300Hz on a commodity laptop) and is one of the few approaches to produce experimental results from running onboard an MAV. It is "semi-direct" in

that it uses a combination of raw pixel intensity patches of a sparse set of keypoints in addition to point features.

2.4.2 Dense Methods

While the robust solutions to the sparse monocular SLAM problem such as ORB-SLAM and others have led many to believe it is largely solved, the problem of *dense* monocular SLAM is less mature, although key advances have been made in recent years. The Dense Tracking and Mapping (DTAM) algorithm of [35] combines the keyframe-based, parallel approach of PTAM with the horsepower of the GPU to achieve impressive real-time results with both dense visual odometry and dense depth estimation. Each keyframe in DTAM contains a 3D cost volume that samples a range of inverse depths per pixel. The stereo matching costs for each hypothetical inverse depth are aggregated across all subsequent frames. A smooth, minimum-cost surface is then extracted from the cost volume using a variational regularization approach by [126]. This smooth reconstruction is then used to densely track each new frame using the coarse-to-fine Lucas-Kanade algorithm described in Section 2.3.2. This approach has been extended in [129] which uses a non-local TV regularizer that biases the solution to be piecewise-planar (instead of piecewise constant) and is thus better able to interpolate over textureless regions like walls and floors..

The method of [38] takes the approach of densifying PTAM keyframes. For each keyframe, a multi-view version of the stereo method described in [94] is used to construct depthmaps which are then fused into a voxel-based representation called a *signed-distance function* (SDF). Each voxel of the SDF records the distance to the nearest surface. The surface itself can then be generated by extracting the zero-level set of the function. Similar to DTAM, variation regularization is applied to enforce surface smoothness.

REMODE developed by [37] uses a semi-direct visual odometry method [45] to estimate camera poses and estimates keyframe depthmaps using a Bayesian filter over each pixel that takes into account the probability of occlusions and outliers. The depthmap is then smoothed using a variational regularizer that is weighted by the

confidence of each depth.

The MonoFusion algorithm [36] also uses a sparse monocular SLAM pipeline to compute poses, but does not utilize keyframes for dense mapping. Instead, depthmaps are computed for each image by performing a modified version of PatchMatch Stereo [130] using variable baseline comparison images. The depthmaps are then fused into an SDF using the method of [107] before the surface is extracted via raycasting. Unlike the other methods mentioned, no explicit regularization is performed. The fusion of depthmaps from every live image (as opposed from a smaller set of keyframes) is enough to constraint the surface to be smooth. The MobileFusion algorithm [39] extends MonoFusion to run at 25 Hz on a commodity smartphone, but sacrifices the volume under reconstruction and other resolution parameters.

2.4.3 Semi-Dense Methods

Although the dense approaches outlined in Section 2.4.2 demonstrate impressive reconstruction results, they are computationally expensive, often require high-end GPUs to run in real-time, and are limited to small desk-sized or room-sized environments. Semi-dense methods sit in between sparse and dense approaches in terms of computational efficiency, reconstruction quality, and map scale.

The Large-Scale Direct-SLAM (LSD-SLAM) algorithm [46, 47] estimates keyframe depthmaps using a per-pixel probabilistic filter similar to [37], but only does so for the high-gradient pixels. Since low-texture image regions are ignored, a significant speedup can be obtained and a GPU is not required. Furthermore, since no volumetric fusion is attempted, the scale of the reconstructed environments can be increased substantially. Once generated, the point clouds produced from the semi-dense keyframe depthmaps are incrementally aligned using a least squares backend solver [54], but with the poses defined in Sim(3) instead of $\mathbb{SO}(3)$ to account for scale drift. New images are tracked using the direct visual odometry approach described in Section 2.3.2 over the high-gradient pixels.

A similar approach was proposed in [48] that uses ORB-SLAM[31] internally to compute poses, but then estimates a semi-dense depthmap for each keyframe. Each

keyframe depthmap is computed by performing a direct epipolar search in N neighboring keyframes and probabilistically fusing the N depth measurements measurements before applying an inter-keyframe consistency check to remove outliers.

Chapter 3

Multi-Level Mapping

Despite the speed and scale that the semi-dense monocular SLAM algorithms described in Chapter 2 provide, the resulting semi-dense point cloud maps do not accurately capture the scene geometry — especially in low-texture image regions — and thus are not yet applicable to high-speed MAV navigation. This chapter presents a novel algorithm called Multi-Level Mapping (MLM) [9] that allows fully dense geometry to be estimated online without the aid of a GPU, bringing MAV-capable dense monocular SLAM one step closer to viability.

The key contribution of MLM is a multi-resolution depth estimation and spatial smoothing process that exploits the correlation between low-texture image regions and simple planar structure to adaptively scale the resolution of the generated reconstruction to the quality of the input imagery. Image texture and depth are highly correlated, with texture changes typically signaling depth discontinuities that occur around objects in the scene. We exploit this correlation by making the reasonable assumption that low-texture image regions are approximately planar and can be accurately represented with more coarsely sampled depth estimates. High-texture image regions are thus represented at higher resolutions to capture fine detail, while low-texture regions are represented at coarser resolutions for smooth surfaces. This approach allows for significant computational savings while simultaneously increasing reconstruction density and quality when compared to the state-of-the-art.

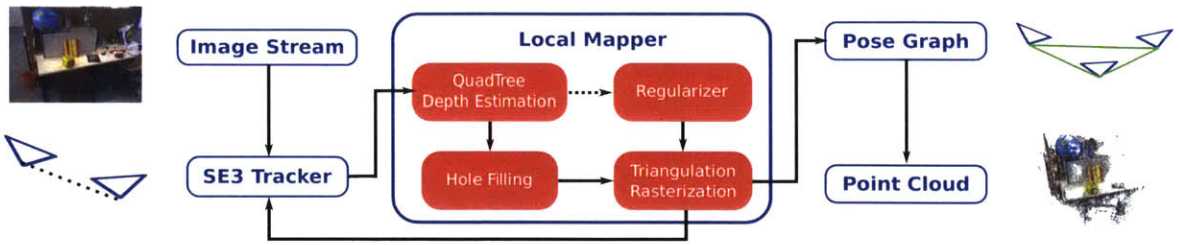


Figure 3-1: *MLM Pipeline*: Incoming images are first tracked in $\mathbb{SE}(3)$ relative to the current keyframe using dense, direct image alignment. Tracked frames are then passed to the Local Mapper, which estimates a quadtree-based, multi-resolution inverse depthmap using many short-baseline stereo computations. Holes in the depthmap are then filled before being interpolated back into the native image resolution using a simple software rasterization procedure. When a keyframe is finished, it is passed to a variational regularizer which removes outliers and smooths away noise. The keyframe is then inserted into a pose-graph defined on $\mathrm{Sim}(3)$ and is incrementally aligned to the other keyframes. Finally, the depthmaps are projected into 3D and visualized as colored point clouds.

3.1 Algorithm Outline

This section briefly outlines the MLM dense monocular SLAM algorithm (see Figure 3-1). As a reminder, we wish to robustly estimate both the pose of a moving monocular camera and a dense representation of the scene geometry using only the camera's image stream. The pipeline begins with a dense, direct visual odometry frontend (see Section 2.3.2) that tracks the camera's pose in $\mathbb{SE}(3)$ relative to a past *keyframe* (Section 3.3). Once tracked, this image is then used to refine the *inverse depthmap* of the keyframe (Section 3.4). Each keyframe is constructed using a variable-resolution data structure called a quadtree [131] such that the depth estimates can exploit low-spatial-frequency information and can be intelligently distributed over the keyframe to both capture fine detail and increase density. Holes in the variable-resolution keyframe depthmap are then filled to further increase density (Section 3.5), before the depthmap is projected back to the native image scale using a triangulation and rasterization procedure (Section 3.6) for future tracking and display. Before a new keyframe is created, a final round of spatial regularization is performed on the old keyframe depthmap (Section 3.7). The keyframe is then added to a pose-graph of keyframe depthmaps that are incrementally aligned over $\mathrm{Sim}(3)$

using a sparse nonlinear least squares solver [54] and then displayed as point clouds (Section 3.8).

3.2 Problem Formulation

In this section we will briefly outline the notation we will use to mathematically describe the MLM approach (Section 3.2.1) and state the dense monocular SLAM problem that it solves (Section 3.2.2).

3.2.1 Notation

We will represent grayscale images of size $m \times n$ pixels as scalar functions defined over a 2D domain. The "live" image at discrete time l will be represented by $I_l : \Omega \to \mathbb{R}_+$, where $\Omega \subset \mathbb{R}^2$ represents the image pixel domain.

We denote the pose of the current camera at time l with respect to keyframe k (up to the unobservable global scale factor) by transform

$$\mathbf{T}_l^k = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ 0 & 1 \end{bmatrix} \in \mathbb{SE}(3), \tag{3.1}$$

with rotation matrix $\mathbf{R} \in \mathbb{SO}(3)$ and translation $\mathbf{t} \in \mathbb{R}^3$. We let $\bar{\mathbf{x}} = \begin{bmatrix} \mathbf{x}^T & 1 \end{bmatrix}^T$ represent the homogeneous coordinates of vector $\mathbf{x}$ such that a point $\mathbf{p}_l \in \mathbb{R}^3$ in frame l can be transformed into frame k by $\bar{\mathbf{p}}_k = \mathbf{T}_l^k \bar{\mathbf{p}}_l$.

The matrix $\mathbf{K} \in \mathbb{R}^{3\times3}$ will represent the camera intrinsic parameters. The perspective projection function will be defined as $\pi\left(\begin{bmatrix} x & y & z \end{bmatrix}^T\right) = \begin{bmatrix} x/z & y/z \end{bmatrix}^T$. The projection of point $\mathbf{p}_l \in \mathbb{R}^3$ into camera k is therefore given by

$$\mathbf{u} = \pi(\mathbf{K}\mathbf{T}_l^k \bar{\mathbf{p}}_l), \tag{3.2}$$

where the de-homogenization is implied for notational clarity. We also define the inverse projection function $\mathbf{p} = \pi^{-1}(\mathbf{u}, d) = \bar{\mathbf{u}}/d$, which maps pixel $\mathbf{u}$ to 3D point $\mathbf{p}$

with *inverse* depth d.

We define keyframe K_k to be a tuple $(\mathbf{S}_k^W, I_k, D_k, V_k)$, where $D_k : \Omega \to \mathbb{R}_+$ is the inverse depthmap associated with image I_k (scaled to have a mean of 1) at the base image level, and $V_k : \Omega \to \mathbb{R}_+$ is the associated map of inverse depth variances. Note that only a subset of pixels $\Omega_k \subseteq \Omega$ will have valid inverse depth estimates due to the absence of texture information, image noise, and outliers. $\mathbf{S}_k^W = (\mathbf{T}_k^W, s_k) \in \mathrm{Sim}(3)$ is the pose $\mathbf{T}_k^W$ of the camera with respect to world frame W (taken to be that of the first keyframe) with scale factor $s_k > 0$ that scales the geometry in D_k appropriately.

We arrange the keyframes in a pose graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where vertices $\mathcal{V} = \{K_k\}$ is the set of keyframes and edges $\mathcal{E} = \{\mathbf{S}_i^j \in \mathrm{Sim}(3) : K_i, K_j \in \mathcal{V}\}$ is the set of odometry factors. Each $\mathbf{S}_i^j$ provides a measurement of the rigid body motion $\mathbf{T}_i^j$ and scale factor $s_i^j > 0$ that aligns the point clouds $\pi^{-1}(\Omega_i, D_i(\Omega_i))$ and $\pi^{-1}(\Omega_j, D_j(\Omega_j))$. The projection of all the keyframe point clouds into W will comprise our map.

3.2.2 Problem Statement

Here we state the dense monocular SLAM problem as described in Chapter 2. Given a sequence of images I_l from a moving camera, we would like to simultaneously estimate the pose of the camera along with a map of the surrounding environment that we represent as a graph of keyframes. Each keyframe will define an inverse depthmap that can be projected into 3D space as a point cloud that will denote structure in the environment. Our goal is thus to estimate online:

- The current camera pose $\mathbf{T}_l^k$ relative to the current keyframe K_k (Section 3.3)
- The inverse depthmap D_k for the current keyframe K_k (Sections 3.4 to 3.7)
- The optimal keyframe poses $\{\mathbf{S}_i^j\}$ for keyframe graph $\mathcal{G}$ (Section 3.8).

While the above quantities are interdependent, we follow the parallelized approach of PTAM [30] and decouple their computations, solving for each component in a separate thread.

Furthermore, we focus our attention on improving the quality of the depthmap D_k (which in turn affects the estimates $\mathbf{T}_l^k$ and $\mathbf{S}_k^W$). State-of-the-art approaches such as LSD-SLAM [47] only estimate depth for regions of Ω with high-image gradient, and are unable to interpolate through low-texture regions, resulting in point cloud maps with undesirable holes (that is $|\Omega_k| \ll |\Omega|$). Our primary contribution will be to increase the fraction of each keyframe with valid depth estimates (i.e. increase $|\Omega_k|$), while also increasing the accuracy of D_k, through a multi-resolution depth estimation process using quadtrees [131].

3.3 Tracking on $\mathbb{SE}(3)$

We use the coarse-to-fine image alignment method of [47] for tracking in $\mathbb{SE}(3)$ between keyframe K_k and the current image I_l, with the addition of a global illumination term to account for lighting variation between frames as in [132]. With the increased density of our keyframes, we also use all available pixels for tracking, not just those with high-image gradient at the finest image scale as in [47].

For each incoming frame I_l, we estimate $\mathbf{T}_l^k$ by minimizing the following robust nonlinear least squares objective:

$$E(\mathbf{T}_l^k) = \sum_{\mathbf{u} \in \Omega_k} \left\| \frac{r_p^2(\mathbf{u}, \mathbf{T}_l^k)}{\sigma^2_{r_p(\mathbf{p}, \mathbf{T}_l^k)}} \right\|_\epsilon \tag{3.3}$$

where $||\cdot||_\epsilon$ is the (scalar) Huber norm defined as

$$||x||_\epsilon = \begin{cases} \frac{x^2}{2\epsilon} & \text{if } |x| \leq \epsilon \\ |x| - \frac{\epsilon}{2} & \text{otherwise.} \end{cases} \tag{3.4}$$

The Huber norm is used to minimize the "staircasing" effect that is common with L_1 error metrics [126]. Near zero, the Huber norm acts like a squared-L_2 cost, which reduces the penalty for small variations. Away from zero, however, it acts like an L_1 cost, which reduces the effect of outliers on the solution.

The *photometric* residual r_p is the pixel intensity error incurred when pixel $\mathbf{u}$ in the keyframe image I_k is projected into the current image I_l assuming inverse depth $D_k(\mathbf{u})$ and relative pose $\mathbf{T}_l^k$. Its specific form is given by

$$r_p(\mathbf{u}, \mathbf{T}_l^k) = I_k(\mathbf{u}) - I_l(\pi(\mathbf{Kp})) - r_{1/2} \tag{3.5}$$

$$\mathbf{p} = \mathbf{T}_k^l \mathbf{K}^{-1} \pi^{-1}(\mathbf{u}, D_k(\mathbf{u})), \tag{3.6}$$

where $\mathbf{p}$ is the projected 3D point of pixel $\mathbf{u}$ from the keyframe into the new frame assuming inverse depth $D_k(\mathbf{u})$. The $r_{1/2}$ term is the median photometric residual across all pixels and serves to remove global illumination changes from the cost [132]. If the global illumination of the scene changes between I_k and I_l, the minimum photometric residual will no longer lie near zero, which can bias the solution. Removing the median residual, however, recenters the distribution of the errors around zero.

The $\sigma^2_{r_p(\mathbf{p},\mathbf{T}_l^k)}$ term is the variance of the photometric residual r_p obtained by propagating the inverse depth variance $V_k(\mathbf{u})$ into the residual space:

$$\sigma^2_{r_p(\mathbf{p},\mathbf{T}_l^k)} = 2\sigma_I^2 + \left(\frac{\partial r_p(\mathbf{u}, \mathbf{T}_l^k)}{\partial D_k(\mathbf{u})}\right)^2 V_k(\mathbf{u}), \tag{3.7}$$

where σ_I^2 is a user-set pixel intensity noise based on the characteristics of the camera.

We minimize Equation (3.3) by performing iteratively reweighted least squares (IRLS) (sometimes known as *M-estimation* or *robust regression*), which approximates the robust cost using a weighted nonlinear least squares objective. This approximation is then optimized using Gauss-Newton [27]. The weights are adjusted on each iteration such that the true, robust objective is minimized. We perform the optimization in a coarse-to-fine fashion, with solutions at coarser levels of the image pyramid used to initialize finer levels.

After convergence, we use the newly tracked frame $(I_l, \mathbf{T}_l^k)$ to update the depthmap D_k and variances V_k of K_k as described in the next sections.

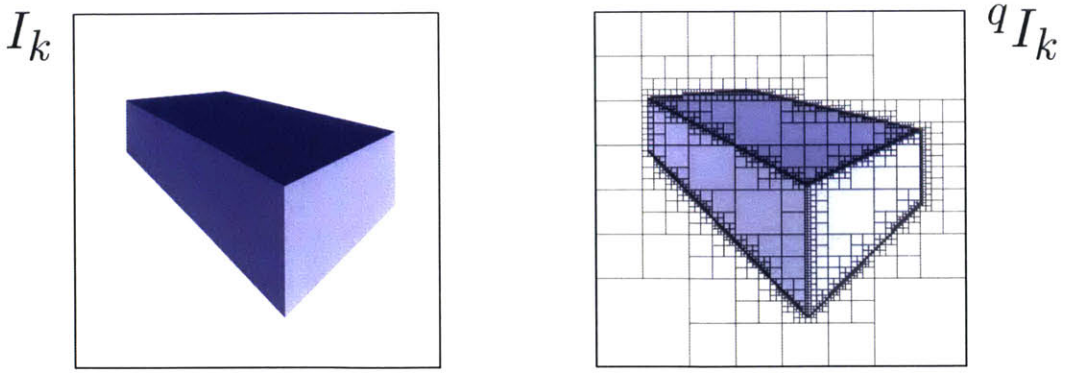


Figure 3-2: Each keyframe image I_k (left) is converted to a multi-resolution representation ${}^{q}I_k$ (right) using quadtrees [131]. I_k is first converted to a standard L-level power-of-two images pyramid ${}^{L}I_k$. If this pyramid is interpreted as a tree structure, clipping subtrees with similar pixel intensities yields the quadtree ${}^{Q}I_k$. The leaves of this tree are then extracted to yield ${}^{q}I_k$. This multi-resolution data structure compresses low-texture regions as single pixels by representing them at a coarser image resolution.

3.4 Depth Estimation using Quadtree Keyframes

With the pose of the current camera relative to previous keyframe $\mathbf{T}_l^k$ computed, the images (I_k, I_l) now form a stereo[1] pair that we use to update the inverse depthmap D_k. We follow the depth estimation method in [46], but perform stereo computations at multiple image scales to increase the density of the depthmap without sacrificing speed.

The approach in [46] may be considered a *single-level* approach because all stereo computations are performed at a single image scale (the native image resolution). As depicted in Figure 3-3, low-texture image regions are difficult to reconstruct using a single image scale because low spatial frequency information is ignored. Our *multi-level* approach, on the other hand, is able to estimate depth for low-texture pixels by performing the epipolar search at coarser image scales, where low spatial frequency information may be leveraged to guide the matching process (see Figure 3-4).

For each stereo pair (I_k, I_l) we compute standard L-level power-of-two image pyramids that we denote $({}^{L}I_k, {}^{L}I_l)$. We interpret these pyramids as tree data structures,

[1]Stereo in this case is across pairs of successive monocular images, rather than simultaneous images from binocular cameras. We follow [46, 47], and others in using the term "stereo" for this monocular image processing.

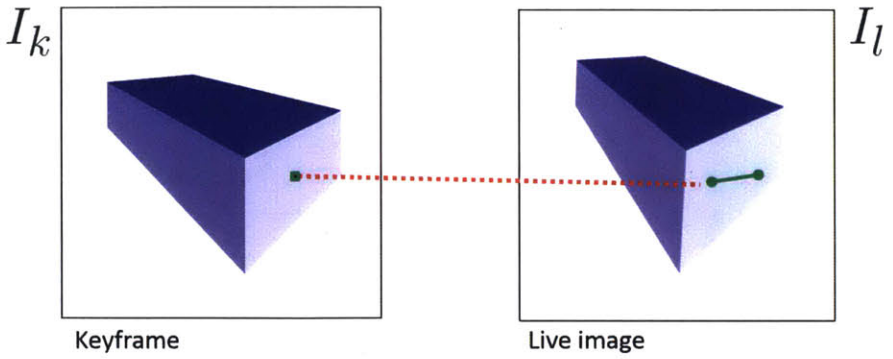


Figure 3-3: *Single-level Depth Estimation*: The depth estimation method of [46, 47] depicted above computes depth only for high-gradient pixels in each keyframe image I_k, which leaves undesirable holes in the reconstruction. Depth for low-texture regions (such as the indicated pixel on the left) are not able to be estimated because no gradient information is present in the epipolar search region in I_l (shown on the right in green) to indicate a match.

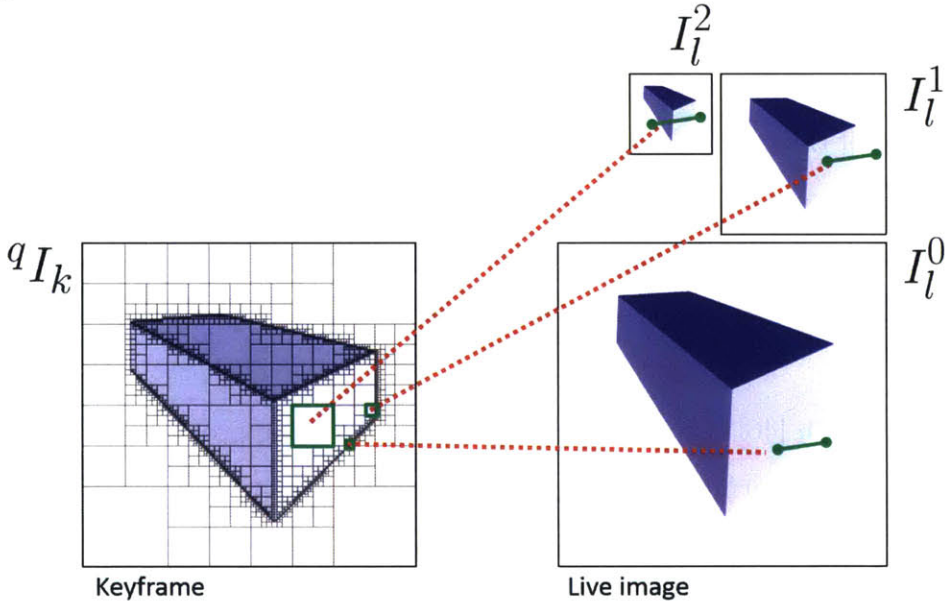


Figure 3-4: *Multi-level Depth Estimation*: We perform stereopsis at the image scale appropriate for the available texture. For each keyframe I_k we compute a quadtree-compressed representation ${}^qI_k$ based on the pixel intensities and define an inverse depthmap over the leaf nodes (left). For each newly tracked frame I_l, we compute its power-of-two image pyramid ${}^LI_l$ and perform small-baseline stereo computations between nodes in ${}^qI_k$ and pixels in ${}^LI_l$ at the image scale corresponding to the given node (right). Compare the epipolar search region in I_l^2 for the highlighted low-texture pixel with that shown in Figure 3-3. By performing the search at a coarser image scale, lower frequency information is available to constrain the match.

with each parent pixel (or *node*) connected to four child nodes at a finer image scale. We then transform ${}^{L}I_k$ into a quadtree ${}^{Q}I_k$ by identifying sub-trees of ${}^{L}I_k$ with similar pixel intensities and *clipping* them from the tree [131]. We extract the *leaf* nodes of ${}^{Q}I_k$ and refer to them as ${}^{q}I_k$ (see Figure 3-2).[2] If we let $I_k^l : \Omega^l \rightarrow \mathbb{R}_+$ represent the lth image level in the full pyramid ${}^{L}I_k$, node $i \in {}^{q}I_k$ comprises a pixel location $\mathbf{u}_i \in \Omega^{l_i}$ with intensity $I_k^{l_i}(\mathbf{u}_i)$, where l_i is the pyramid level index. We then define a corresponding inverse depthmap ${}^{q}D_k$ with variances ${}^{q}V_k$ and perform updates on this representation before projecting the depths back to the full-resolution D_k (see Section 3.6).

To update the depth ${}^{q}D_k(i)$ for node $i \in {}^{q}I_k$, we check the magnitude of the image gradient $\nabla I_k^{l_i}(\mathbf{u}_i)$. If the magnitude falls below a threshold (e.g. 5), we skip the update. If the gradient is sufficient, we search along the epipolar line defined by $\mathbf{T}_l^k$ in image $I_l^{l_i}$ for a matching pixel $\mathbf{u}_i^*$ (see Figure 3-4).[3] The number of disparities searched along this line corresponds to the $\pm 2\sigma$ inverse depth interval induced from the prior inverse depth variance ${}^{q}V_k(\mathbf{u}_i)$. Matches are determined using sum-of-squared-differences (SSD) along a 5-sample window with a ratio test to ensure a unique match.

If a match $\mathbf{u}_i^*$ is found along the epipolar line, we compute the depth $\mathbf{z}_i$ that the association $(\mathbf{u}_i, \mathbf{u}_i^*)$ induces via triangulation. We then compute a variance σ_i^2 for this "measurement" according to the noise model in [46], which is composed of two terms based on the two primary sources of error in the measurement: (1) error in the epipolar line due to imperfect pose information and (2) error in the match $\mathbf{u}_i$ due to noise in the raw pixel values along the epipolar line. The new inverse depth measurement $(\mathbf{z}_i, \sigma_i^2)$ is then fused with the prior inverse depth estimate ${}^{q}D_k(i)$ and

[2] An equivalent construction of ${}^{q}I_k$ would be to take I_k and recursively merge pixel neighborhoods with similar intensities into single pixels defined at coarser image scales.

[3] Note that the comparison is performed between the quadtree leaf nodes of the keyframe image and the full image pyramid for the incoming frame. We do not need to compute a quadtree representation for the incoming frame.

${}^qV_k(i)$ via the standard Bayesian update equations:

$$ {}^qD_k(i) \leftarrow \frac{\mathbf{z}_i {}^qV_k(i) + {}^qD_k(i)\sigma_i^2}{{}^qV_k(i) + \sigma_i^2} \tag{3.8} $$

$$ {}^qV_k(i) \leftarrow \frac{{}^qV_k(i)\sigma_i^2}{{}^qV_k(i) + \sigma_i^2}. \tag{3.9} $$

We also keep track of the number of successful observations for each node and do not initiate the epipolar search if this number drops below a threshold (that is we mark the node as "invalid").

Our approach performs a similar number of stereo computations per keyframe as that of [46], but is able to more effectively "cover" the keyframe with inverse depth estimates by representing lower-texture image regions with coarser resolution pixels. We perform stereo computations at the image scale appropriate for the available texture, by which we are able to increase the density of the keyframe depthmap D_k after projecting the depths in ${}^qD_k$ back to full-resolution. Image texture and depth are highly correlated, with texture changes typically signaling depth discontinuities that occur around objects in the scene. We exploit this correlation by making the reasonable assumption that low-texture image regions are approximately planar and can be accurately represented with more coarsely sampled depth estimates.

After the inverse depths for each node in the multi-level depthmap are updated, we attempt to fill the holes in the depthmap created by invalid nodes as described in Section 3.5.

3.5 Hole-Filling

Although the multi-level depth estimation approach described in Section 3.4 aims to recover depth even in low-texture image regions, the epipolar search as shown in Figure 3-4 may still fail to produce a match due to image noise and outliers, in addition to the limitations imposed by a finite epipolar search region and finite number of pyramid levels. If the stereo computation at node $i \in {}^qI_k$ has failed repeatedly, however, we may still infer its depth by considering its spatial neighbors

(leveraging the assumption that the world is smooth and that the inverse depths of spatial neighbors are correlated). If the number of successful stereo observations for the spatial neighbors of i exceeds a threshold, we re-initialize ${}^{q}D_k(i)$ to be the mean of the estimates of its neighbors, weighted by the variance of each estimate, and attempt the stereo search again at the next incoming frame. This hole-filling procedure helps to increase the density of ${}^{q}D_k$ and ensure that as many pixels as possible have depth estimates. After a round of hole-filling, we project ${}^{q}D_k$ back to the full-resolution D_k, as described in Section 3.6, which is then passed to the tracking frontend described in Section 3.3.

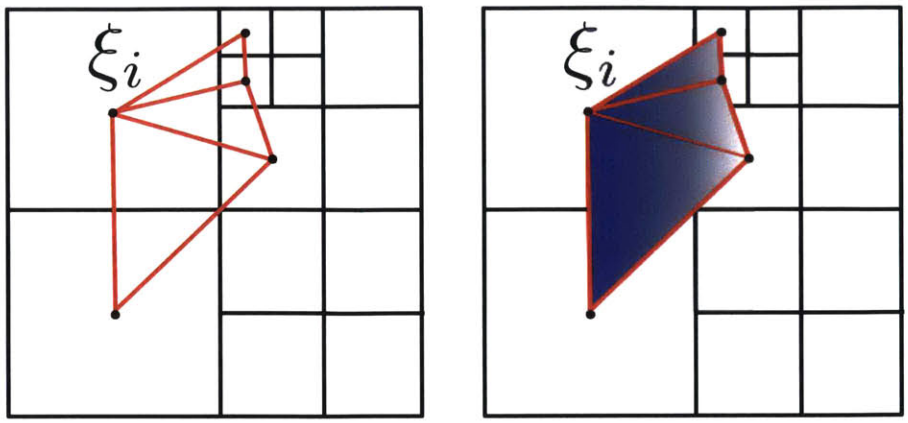
3.6 Triangulation and Rasterization

We perform depthmap updates on the variable resolution representation ${}^{q}D_k$ to enable spatial regularization and hole filling, but need to track the next incoming image and display point clouds using the best full-resolution keyframe depthmap we can infer so far. Thus, at each timestep, we take the current variable resolution keyframe depthmap ${}^{q}D_k$ and recover the native resolution depthmap D_k.

For each pixel $\mathbf{u} \in \Omega$ at native resolution, we approximate $D_k(\mathbf{u})$ by linearly interpolating among the depth estimates at nearby quadtree leaves in ${}^{q}D_k$ using a simple triangulation and rasterization scheme. This kind of interpolation is not strictly necessary, but assigning the same depth to all full-resolution pixels corresponding to a leaf node in ${}^{q}D_k$ leads to unnecessarily quantized or "blocky" depthmaps (this is effectively a piecewise constant approximation versus a piecewise linear approximation). The interpolation scheme then raises the question of which quadtree leaves are neighbors to a particular full-resolution pixel $\mathbf{u}$, and we use a simple triangulation scheme to determine the neighbors [133].

Given a triangulation of ${}^{q}D_k$, we linearly interpolate between the multi-level depth[4] estimates using software rasterization accelerated by single-instruction-multiple-data (SIMD) instructions to fill in D_k (see Figure 3-5). We compute a new interpo-

[4]Note that we linearly interpolate the *depths* associated with each node, not the inverse depths.



(a) Triangulation (b) Rasterization

Figure 3-5: We triangulate the locations of our multi-resolution quadtree inverse depthmap using a single pass over the full-resolution pixels (a). We then interpolate the depth values between node locations using software rasterization to generate a piecewise linear depthmap (b).

lated depthmap for every mapping iteration to pass to the $\mathbb{SE}(3)$ tracker. However, we use a variational regularizer described in the next section to remove outliers and smooth away noise before $\mathrm{Sim}(3)$ alignment and point cloud display.

3.7 Spatial Regularization

The multi-resolution inverse depthmap ${}^{q}D_k$ computed in Section 3.4 may be corrupted by noise as well as outliers from false-matches that can degrade map quality. Before we finalize a keyframe and pass its depthmap to the pose-graph optimization backend, we apply a variational regularizer to remove the outliers and smooth away noise (see the overview of spatial regularization approaches in Section 2.3.3).

Assuming the quadtree-compressed keyframe image ${}^{q}I_k$ contains N nodes, we first arrange ${}^{q}D_k$ into a vector in $\mathbb{R}^N_+$:

$$\mathbf{z} = \begin{bmatrix} {}^{q}D_k(1) & \dots & {}^{q}D_k(N) \end{bmatrix}^T \in \mathbb{R}^N_+. \tag{3.10}$$

We then let the vector $\boldsymbol{\xi} \in \mathbb{R}^N_+$ denote the regularized solution and minimize the

following convex objective function:

$$E(\boldsymbol{\xi}) = TV_\epsilon(\boldsymbol{\xi}) + \lambda||\mathbf{W}(\boldsymbol{\xi} - \mathbf{z})||_1, \tag{3.11}$$

where $TV_\epsilon(\boldsymbol{\xi})$ is the Total Variation-Huber norm, λ is a scale-factor that sets the influence of the L_1 data-term (in experiments set to 0.005), and $\mathbf{W}$ is a diagonal weighting matrix. The Total Variation-Huber norm promotes smooth solutions while preserving edges and the weighted L_1 data-term reduces the effect of outliers in $\mathbf{z}$.

The minimization of Equation (3.11) is performed using the first-order primal-dual algorithm of [126], which will be described shortly. While this approach is typically implemented on a GPU [35, 37, 38], we find that when running in a separate thread and operating on our quadtree-compressed depthmap, our version runs sufficiently fast for real-time operation on a CPU. After a new keyframe is triggered (based on the euclidean and angular distance to the last keyframe), we run our regularizer for a fixed number of iterations on the outgoing depthmap before passing it to the tracker (Section 3.3) and pose-graph optimizer (Section 3.8).[5]

As described in Section 2.3.3, the Total Variation-Huber norm for a continuous (and differentiable) scalar function over the image domain $f : \Omega \to \mathbb{R}$ is defined as

$$TV_\epsilon(f) = \int_\Omega ||\nabla f(\mathbf{u})||_\epsilon \; d\mathbf{u}, \tag{3.12}$$

where the (isotropic) multi-dimensional Huber norm is defined as

$$||\mathbf{x}||_\epsilon = \begin{cases} \frac{||\mathbf{x}||_2^2}{2\epsilon} & \text{if } ||\mathbf{x}||_2 \leq \epsilon \\ ||\mathbf{x}||_2 - \frac{\epsilon}{2} & \text{otherwise.} \end{cases} \tag{3.13}$$

If the vector of inverse depth estimates $\boldsymbol{\xi} \in \mathbb{R}_+^N$ is interpreted as N point samples of an underlying continuous function $\xi : \Omega \to \mathbb{R}_+$, the discrete analog of Equation 3.12

[5]We find that running the regularizer after the depth estimation process is complete produces better results than applying the regularization in parallel, before the depths have converged.

is given by

$$TV_\epsilon(\boldsymbol{\xi}) = \sum_{i=1}^{N} ||\nabla\xi(\mathbf{u}_i)||_\epsilon. \tag{3.14}$$

Note, however, that the set of gradient vectors $\begin{bmatrix}\nabla\xi(\mathbf{u}_1) & \dots & \nabla\xi(\mathbf{u}_N)\end{bmatrix} \in \mathbb{R}^{2N}$ can be approximated as the output of a linear operator $\mathbf{D} : \mathbb{R}^N \to \mathbb{R}^{2N}$ such that

$$\mathbf{D}\boldsymbol{\xi} \approx \begin{bmatrix}\nabla\xi(\mathbf{u}_1) & \dots & \nabla\xi(\mathbf{u}_N)\end{bmatrix} \in \mathbb{R}^{2N}. \tag{3.15}$$

The discrete Total Variation-Huber norm can then be written as

$$TV_\epsilon(\boldsymbol{\xi}) = \sum_{i=1}^{N} || \, [\mathbf{D}\boldsymbol{\xi}]_i \, ||_\epsilon \tag{3.16}$$

$$= ||\mathbf{D}\boldsymbol{\xi}||_\epsilon, \tag{3.17}$$

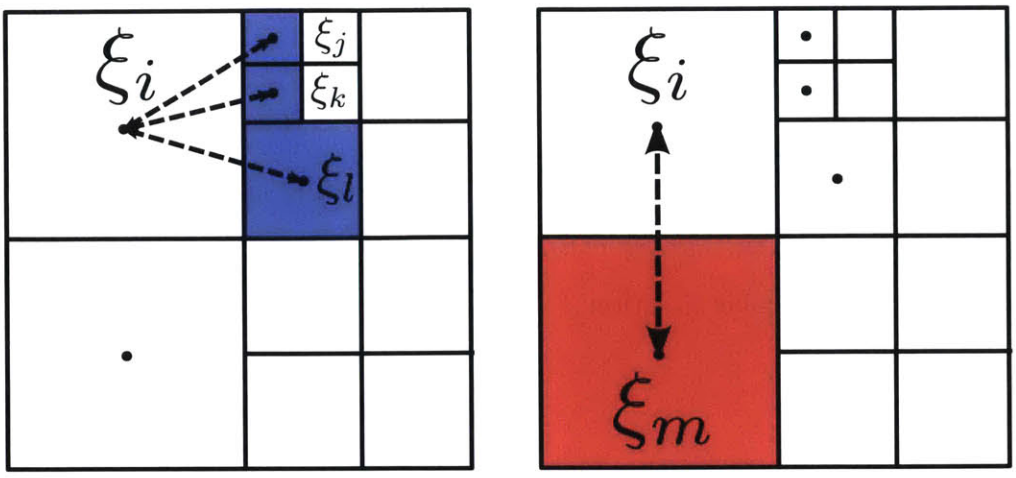
where the last line follows a common abuse of notation (the Huber norm is applied to each individual gradient vector in $\mathbb{R}^2$, not the set of N such vectors in $\mathbb{R}^{2N}$).

If the sampling of ξ to form $\boldsymbol{\xi}$ were uniform over the 2D image domain Ω, $\mathbf{D}$ could simply be the sparse matrix that encodes the standard horizontal and vertical forward differences:

$$\nabla\xi(\mathbf{u}_{i,j}) \approx [\mathbf{D}\boldsymbol{\xi}]_{i,j} \tag{3.18}$$

$$= \begin{bmatrix}\boldsymbol{\xi}_{i+1,j} - \boldsymbol{\xi}_{i,j} \\ \boldsymbol{\xi}_{i,j+1} - \boldsymbol{\xi}_{i,j}\end{bmatrix}. \tag{3.19}$$

In our case, we must modify $\mathbf{D}$ such that the horizontal and vertical derivatives are approximated on our variable-scale $\boldsymbol{\xi}$. As shown in Figure 3-6, each element of $\boldsymbol{\xi}$ corresponds to a square region of pixels at full-resolution. We approximate the derivative in each direction by simply averaging the forward differences between a



(a) Horizontal Neighbors (b) Vertical Neighbors

Figure 3-6: *Discrete derivative computation*: We approximate the discrete derivative at node ξ_i in our multi-level inverse depthmap using forward-differences with the node's (a) horizontal and (b) vertical neighbors.

node and the nodes bordering its square region:

$$\nabla\xi(\mathbf{u}_i) \approx \begin{bmatrix} \frac{\sum_{j=1}^{|\mathcal{N}^h(\boldsymbol{\xi}_i)|} \mathcal{N}_j^h(\boldsymbol{\xi}_i)}{|\mathcal{N}^h(\boldsymbol{\xi}_i)|} - \boldsymbol{\xi}_i \\ \frac{\sum_{j=1}^{|\mathcal{N}^v(\boldsymbol{\xi}_i)|} \mathcal{N}_j^v(\boldsymbol{\xi}_i)}{|\mathcal{N}^v(\boldsymbol{\xi}_i)|} - \boldsymbol{\xi}_i, \end{bmatrix} \tag{3.20}$$

where $\mathcal{N}^h(\boldsymbol{\xi}_i)$ and $\mathcal{N}^v(\boldsymbol{\xi}_i)$ denote the horizontal and vertical neighbors of $\boldsymbol{\xi}_i$, respectively. The differences should technically be weighted by their distance from $\mathbf{u}_i$, but we found that it did not affect the approximation significantly. Note again that this computation can be encoded in a sparse matrix operator.

The diagonal weighting matrix $\mathbf{W} = \text{diag}\,(w_1, \ldots, w_N)$ incorporates the depth uncertainty into the data term and is defined as

$$w_i = \begin{cases} 0 & \text{if no stereo match found} \\ \frac{1}{\sqrt{{}^q V_k(i)}} & \text{otherwise.} \end{cases} \tag{3.21}$$

With this set of weights, the regularized solution will be penalized more severely for deviating from depth estimates with low uncertainty. In the case where no stereo match has been found for a node, the weight placed on the data term is 0, meaning

the smoothing term will dominate.

To derive the update equations to minimize (3.11), we first compute a dual representation of $TV_\epsilon(\boldsymbol{\xi})$ called the *convex conjugate* (also known as the *Legendre-Fenchel transform*) [134]. For a scalar function $f : \mathbb{R}^N \to \mathbb{R}$, the convex conjugate function $f^* : \mathbb{R}^N \to \mathbb{R}$ is defined as

$$f^*(\mathbf{q}) = \max_{\mathbf{x}} \mathbf{q}^T\mathbf{x} - f(\mathbf{x}), \tag{3.22}$$

for dual variable $\mathbf{q} \in \mathbb{R}^N$. This operation essentially describes f in terms of its supporting hyperplanes. Note that f^* is convex (no matter the convexity of f) because it is the pointwise maximum of a set of affine functions in $\mathbf{q}$. However if f is convex and *closed* (i.e. its sublevel sets are closed), then it can be reconstructed from its conjugate function:

$$f(\mathbf{x}) = \max_{\mathbf{q}} \mathbf{x}^T\mathbf{q} - f^*(\mathbf{q}). \tag{3.23}$$

In this case, one can see that f is obtained by taking the conjugate of the conjugate function f^*.

The convex conjugate for the Huber norm $f(\mathbf{x}) = ||\mathbf{x}||_\epsilon$ can be derived by computing the conjugate functions for its squared-L_2 and L_2 components separately. Taking the squared-L_2 component first, when $||\mathbf{x}||_2 \leq \epsilon$ we have

$$f^*(\mathbf{q}) = \max_{\mathbf{x}} \mathbf{q}^T\mathbf{x} - \frac{1}{2\epsilon}||\mathbf{x}||_2^2. \tag{3.24}$$

After setting the derivative of $\mathbf{q}^T\mathbf{x} - \frac{1}{2\epsilon}||\mathbf{x}||_2^2$ to zero, the maximum is obtained at $\mathbf{x}^* = \epsilon\mathbf{q}$. After substituting for $\mathbf{x}$, this yields

$$f^*(\mathbf{q}) = \frac{\epsilon}{2}\mathbf{q}^T\mathbf{q} \text{ for } ||\mathbf{q}||_2 \leq 1. \tag{3.25}$$

Now considering the L_2 component when $||\mathbf{x}||_2 > \epsilon$, we have

$$f^*(\mathbf{q}) = \max_{\mathbf{x}} \mathbf{q}^T\mathbf{x} - ||\mathbf{x}||_2 + \frac{\epsilon}{2}. \tag{3.26}$$

While $\mathbf{q}^T\mathbf{x} - ||\mathbf{x}||_2 + \epsilon/2$ is not differentiable, we may still easily reason about its maximum. Ignoring the constant $\epsilon/2$ for a moment, note that if $||\mathbf{q}||_2 < 1$, then the plane defined by $\mathbf{q}^T\mathbf{x}$ is a supporting hyperplane of $||\mathbf{x}||_2$ with the contact point at $\mathbf{x} = 0$. This implies that $\mathbf{q}^T\mathbf{x} \leq ||\mathbf{x}||_2$ for all $\mathbf{x} \in \mathbb{R}^N$, with equality only attained at $\mathbf{x} = 0$, and thus

$$\max_{\mathbf{x}} \mathbf{q}^T\mathbf{x} - ||\mathbf{x}||_2 + \epsilon/2 = \epsilon/2. \tag{3.27}$$

If $||\mathbf{q}||_2 > 1$, however, then $\mathbf{q}^T\mathbf{x}$ can be arbitrarily greater than $||\mathbf{x}||_2$ and

$$\max_{\mathbf{x}} \mathbf{q}^T\mathbf{x} - ||\mathbf{x}||_2 + \epsilon/2 = \infty. \tag{3.28}$$

Thus the conjugate function for the L_2 component when $||\mathbf{x}||_2 > \epsilon$ is given by

$$f^*(\mathbf{q}) = \delta_Q(\mathbf{q}) + \frac{\epsilon}{2} \text{ for } ||\mathbf{q}||_2 > 1, \tag{3.29}$$

where the indicator function $\delta_Q(\mathbf{q})$ for $Q = \{\mathbf{q} : ||\mathbf{q}||_2 \leq 1\}$ is defined as

$$\delta_Q(\mathbf{q}) = \begin{cases} 0 & \text{if } \mathbf{q} \in Q \\ \infty & \text{otherwise.} \end{cases} \tag{3.30}$$

Now combining the conjugate functions for the squared-L_2 and L_2 components we finally have

$$f^*(\mathbf{q}) = \frac{\epsilon}{2}\mathbf{q}^T\mathbf{q} + \delta_Q(\mathbf{q}), \tag{3.31}$$

which can be used to reconstruct the Huber norm:

$$||\mathbf{x}||_\epsilon = \max_{\mathbf{q}} \mathbf{q}^T\mathbf{x} - \frac{\epsilon}{2}\mathbf{q}^T\mathbf{q} - \delta_Q(\mathbf{q}). \tag{3.32}$$

The Total Variation-Huber norm of $\boldsymbol{\xi}$ is simply a sum of N such functions:

$$TV_\epsilon(\boldsymbol{\xi}) = \sum_{i=1}^{N} \max_{\mathbf{q}_i} \mathbf{q}_i^T [\mathbf{D}\mathbf{x}]_i - \frac{\epsilon}{2}\mathbf{q}_i^T\mathbf{q}_i - \delta_{Q_i}(\mathbf{q}_i) \tag{3.33}$$

$$= \max_{\mathbf{q}} \mathbf{q}^T\mathbf{D}\boldsymbol{\xi} - \frac{\epsilon}{2}\mathbf{q}^T\mathbf{q} - \delta_Q(\mathbf{q}) \tag{3.34}$$

using the dual variable $\mathbf{q} = \begin{bmatrix}\mathbf{q}_1^T & \dots & \mathbf{q}_N^T\end{bmatrix}^T \in \mathbb{R}^{2N}$ and set $Q = \{\mathbf{q} \in \mathbb{R}^{2N} : ||\mathbf{q}_i||_2 \leq 1 \text{ for } i = 1, \dots, N\}$.

Substituting this representation into the cost in Equation (3.11), our objective can be written in primal-dual form

$$\min_{\boldsymbol{\xi}} \max_{\mathbf{q}} \ \mathbf{q}^T\mathbf{D}\boldsymbol{\xi} + G(\boldsymbol{\xi}) - F^*(\mathbf{q}), \tag{3.35}$$

where we follow the notation of [126] and define

$$G(\boldsymbol{\xi}) = ||\mathbf{W}(\boldsymbol{\xi} - \mathbf{z})||_1 \tag{3.36}$$

$$F^*(\mathbf{q}) = \frac{\epsilon}{2}\mathbf{q}^T\mathbf{q} + \delta_Q(\mathbf{q}). \tag{3.37}$$

The intuition behind the optimization approach outlined in [126] will be to perform gradient ascent steps in $\mathbf{q}$ followed by gradient descent steps in $\boldsymbol{\xi}$. However, since neither G nor F^* is differentiable, we will need to generalize the gradient steps via the *proximal operator* [135]. For a proper convex function $f : \mathbb{R}^N \to \mathbb{R}$ that is also closed, the proximal operator is defined as

$$\text{prox}_{\lambda f}(\mathbf{y}) = \arg\min_{\mathbf{x}} f(\mathbf{x}) + \frac{1}{2\lambda}||\mathbf{x} - \mathbf{y}||_2^2. \tag{3.38}$$

When f is differentiable, this operation is essentially a gradient descent step with

step size λ:

$$\text{prox}_{\lambda f}(\mathbf{y}) \approx \mathbf{y} - \lambda \nabla f(\mathbf{x}). \tag{3.39}$$

Following [126], the primal-dual update steps are then given by:

$$\mathbf{q}^{n+1} = \text{prox}_{\alpha_\mathbf{q}, F^*}(\mathbf{q}^n + \alpha_\mathbf{q} \mathbf{D}\bar{\boldsymbol{\xi}}) \tag{3.40}$$

$$\boldsymbol{\xi}^{n+1} = \text{prox}_{\alpha_{\boldsymbol{\xi}}, G}(\boldsymbol{\xi}^n - \alpha_{\boldsymbol{\xi}} \mathbf{D}^T \mathbf{q}^{n+1}) \tag{3.41}$$

$$\bar{\boldsymbol{\xi}}^{n+1} = \boldsymbol{\xi}^{n+1} + \theta(\boldsymbol{\xi}^{n+1} - \boldsymbol{\xi}^n). \tag{3.42}$$

for step sizes $\alpha_{\boldsymbol{\xi}}, \alpha_\mathbf{q} > 0$. The last line denotes an extragradient step with step size $\theta \in [0, 1]$.

The proximal operators for G and F^* in Equation (3.35) are given pointwise by

$$\text{prox}_{\alpha_{\boldsymbol{\xi}}, G}(x_i) = \begin{cases} x_i - \lambda w_i \alpha_{\boldsymbol{\xi}} & \text{if } x_i - z_i > \lambda \alpha_{\boldsymbol{\xi}} w_i \\ x_i + \lambda w_i \alpha_{\boldsymbol{\xi}} & \text{if } x_i - z_i < -\lambda \alpha_{\boldsymbol{\xi}} w_i \\ z_i & \text{if } |x_i - z_i| \leq \lambda \alpha_{\boldsymbol{\xi}} w_i \end{cases} \tag{3.43}$$

$$\text{prox}_{\alpha_\mathbf{q}, F^*}(y_i) = \frac{\frac{y_i}{1+\alpha_\mathbf{q}\epsilon}}{\max\{1, ||\frac{y_i}{1+\alpha_\mathbf{q}\epsilon}||_2\}}. \tag{3.44}$$

After convergence, we copy $\boldsymbol{\xi}$ back to ${}^qD_k$ and rasterize the solution to obtain the smoothed depthmap D_k, which is then incrementally aligned with overlapping keyframes in our pose-graph optimization backend and displayed (see Section 3.8).

3.8 Pose-graph Optimization on $\mathbf{Sim}(3)$

Similar to the $\mathbb{SE}(3)$ tracking front-end described in Section 3.3, we use the robust image and depth alignment method of [47] for generating constraints in our Sim(3) pose-graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ in order to estimate the optimal keyframe poses $\mathbf{S}_k^W$ and point cloud.

When keyframe K_k is finalized, it is added to the vertex set $\mathcal{V}$ and a set of potential

neighbor keyframes $\mathcal{C} \subseteq \mathcal{V} \setminus K_k$ is generated through a search of predecessors in $\mathcal{V}$ and an appearance-based loop-closure detector [136].

For each $K_j \in \mathcal{C}$, transforms $\mathbf{S}_k^j, \mathbf{S}_j^k \in \mathrm{Sim}(3)$ linking K_k and K_j are computed using the Sim(3) tracking method described below. If $\mathbf{S}_k^j$ and $\mathbf{S}_j^k$ are consistent with each other, they are added to the constraint set $\mathcal{E}$. $\mathcal{G}$ is then refined online using a sparse Levenberg-Marquardt least squares solver in the software package `g2o` [54] to produce the optimal keyframe pose $\mathbf{S}_k^W$, which we use to project D_k into the world frame W.

Tracking on Sim(3) is achieved using iteratively reweighted least squares (IRLS) as in Section 3.3, with the cost function from 3.3 modified to include the residual r_d between the two inverse depthmaps D_j and D_k:

$$E(\mathbf{S}_k^j) = \sum_{\mathbf{u} \in \Omega_k} \left\| \frac{r_p^2(\mathbf{u}, \mathbf{S}_k^j)}{\sigma^2_{r_p(\mathbf{p}, \mathbf{S}_k^j)}} + \frac{r_d^2(\mathbf{u}, \mathbf{S}_k^j)}{\sigma^2_{r_d(\mathbf{p}, \mathbf{S}_k^j)}} \right\|_\epsilon \tag{3.45}$$

$$r_d(\mathbf{u}, \mathbf{S}_k^j) = [\mathbf{p}]_z^{-1} - D_j(\pi(\mathbf{K}\mathbf{p})) \tag{3.46}$$

$$\mathbf{p} = \mathbf{S}_k^j \mathbf{K}^{-1} \pi^{-1}(\mathbf{u}, D_k(\mathbf{u})). \tag{3.47}$$

Here we can see that the inverse depth residual is computed by projecting pixel $\mathbf{u}$ into frame j using inverse depth $D_k(\mathbf{u})$ and $\mathbf{S}_k^j$ and computing the difference between the new inverse depth in this frame and that given by D_j. The variance for the residual is computed by propagating the inverse depth variances through the residual function:

$$\sigma^2_{r_d(\mathbf{u}, \mathbf{S}_k^j)} = J_k^2 V_k(\mathbf{u}) + J_j^2 V_j(\pi(\mathbf{K}\mathbf{p})) \tag{3.48}$$

$$J_k = \frac{\partial r_d(\mathbf{u}, \mathbf{S}_k^j)}{\partial D_k(\mathbf{u})} \tag{3.49}$$

$$J_j = \frac{\partial r_d(\mathbf{u}, \mathbf{S}_k^j)}{\partial D_j(\pi(\mathbf{p}))}. \tag{3.50}$$

3.9 Summary

This chapter described the MLM dense monocular SLAM algorithm in detail. MLM maintains a map composed of a set of keyframe inverse depthmaps arranged in a

graph. Each inverse depthmap is defined on a multi-resolution data structure called a quadtree [131] that allows the depth estimation process to exploit the correlation between low-texture image regions and simple planar structure to adaptively scale the complexity of the depthmap to the quality of the input imagery. High-texture image regions are represented at higher resolutions to capture fine detail, while low-texture regions are represented at coarser resolutions for smooth surfaces.

Given a live image stream, incoming images are tracked in $\mathbb{SE}(3)$ relative to the current keyframe using dense, direct image alignment (Section 3.3) before being used to refine the multi-resolution inverse depthmap of the current keyframe (Section 3.4). Holes in the variable-resolution keyframe depthmap are then filled to further increase density (Section 3.5), before the depthmap is projected back to the native image scale using a triangulation and rasterization procedure (Section 3.6) for future tracking and display. Before a new keyframe is created, a final round of spatial regularization is performed on the old keyframe depthmap (Section 3.7). The keyframe is then added to the graph of keyframe depthmaps that are incrementally aligned over Sim(3) using a sparse nonlinear least squares solver [54] and then displayed as point clouds (Section 3.8).

This approach allows for significant computational savings while simultaneously increasing reconstruction density and quality when compared to the state-of-the-art. A qualitative and quantitative evaluation of MLM compared to LSD-SLAM [47] will be presented in Chapter 4.

Chapter 4

Evaluation

In this chapter, the tracking and reconstruction performance of the MLM algorithm described in Chapter 3 is evaluated qualitatively using video captured from a hand-held camera as well as quantitatively on publicly available benchmark datasets [137]. The MLM implementation is based off the LSD-SLAM source code.[1] All processing was done in real time on a consumer laptop with an Intel Core i7 processor with 4 hyperthreaded cores and 8 GB of RAM. All metrics were computed using the raw inverse depthmaps, however, a maximum depth threshold of 1-2 units[2] was used for display purposes.

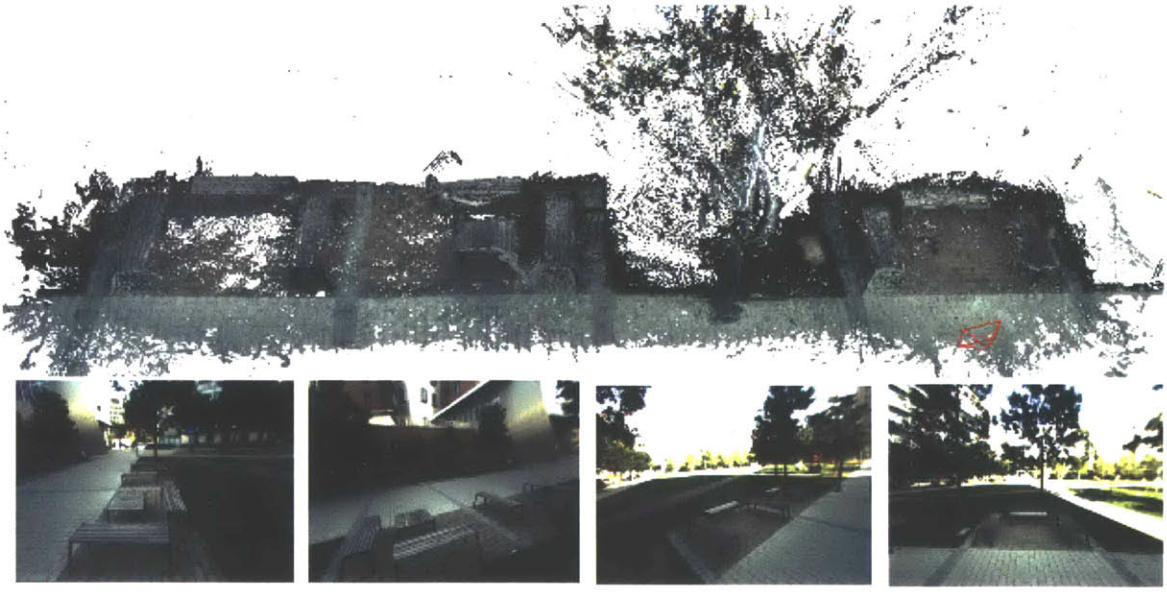
4.1 Qualitative Evaluation

Several 2-4 minute video sequences were captured using a global shutter Point Grey Firefly color camera with a resolution of 640x480 pixels and 30 Hz frame-rate. The first small-scale sequence was recorded in a roughly 2 m^2 office area and is shown in Figure 4-1a. Note the accurate reconstructions of high-texture regions around the desk. The second sequence was recorded in a 50 m^2 outdoor area near a set of benches shown in Figure 4-1b. The final sequence was captured inside a 50 m^2 laboratory test

[1] `https://github.com/tum-vision/lsd_slam`

[2] As described in Chapter 3, the inverse depthmaps are normalized to have a mean of 1. The scale factor of each Sim(3) keyframe pose is responsible for scaling the point clouds such that they align in the world frame.

(a) Desk dataset.



(b) Bench dataset.

Figure 4-1: We demonstrate the quality of our reconstructions on several datasets recorded using a handheld camera: (a) shows the final point cloud map from a small desk scene, while (b) shows the map from an outdoor bench area. All processing was performed in real-time on a consumer laptop.

space and is shown in Figure 4-2. These examples show how our multi-level approach is able to interpolate through low-texture regions such as the ground and walls.

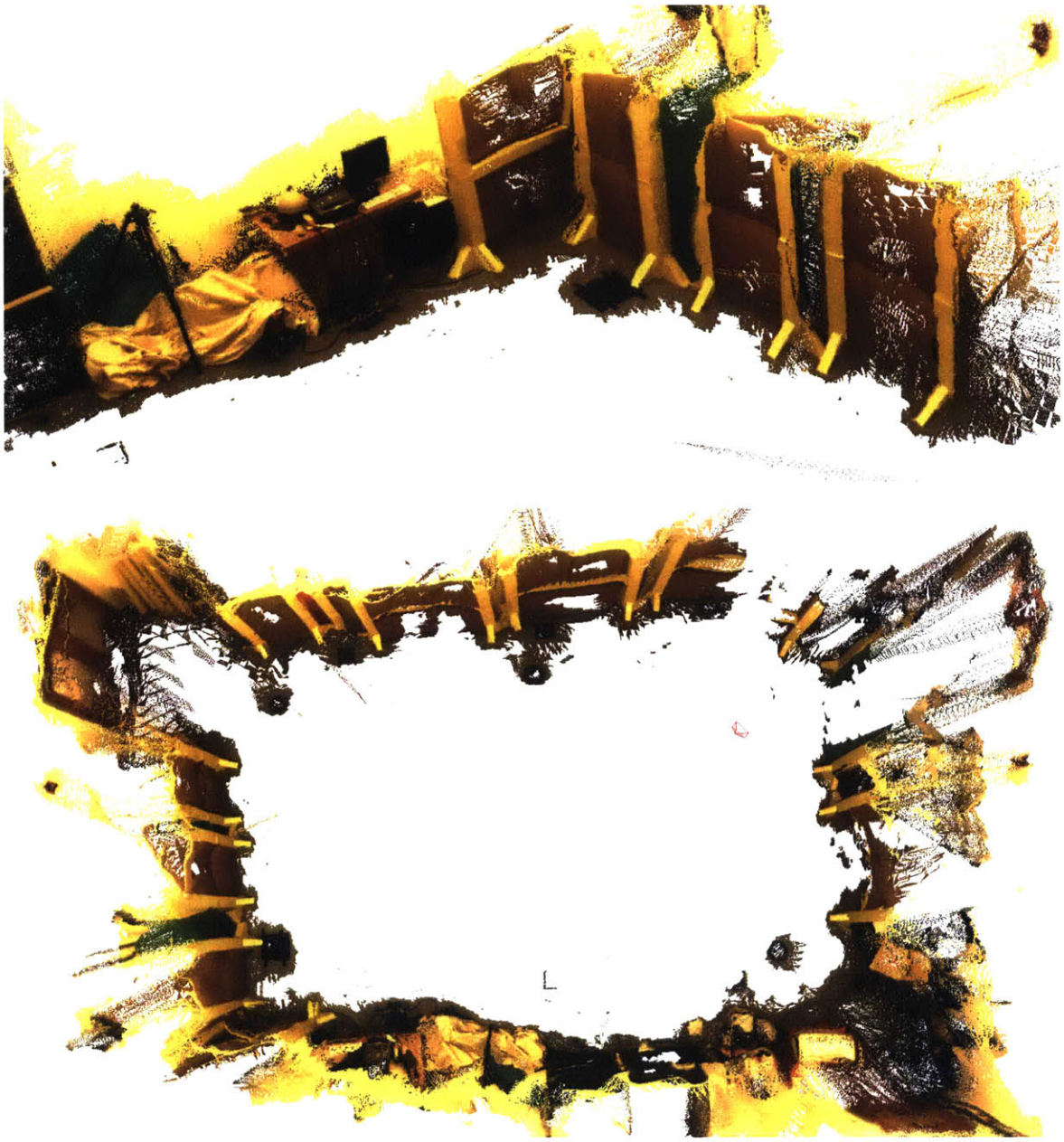


Figure 4-2: Our multi-level depth estimation and spatial regularization process enables dense point cloud maps to be generated online without GPU acceleration. This point cloud was generated in real-time from approximately two minutes of 30 Hz video around a laboratory workspace.

Tracking Accuracy [RMSE]				
	Pos. [m]		Angle [deg]	
Dataset	LSD-SLAM	MLM	LSD-SLAM	MLM
fr2/desk	2.1	**0.15**	88	**4.8**
fr3/long_office_household	2.1	**0.65**	68	**11**
fr3/nostructure_texture_near_withloop	0.28	**0.22**	3.7	**2.6**
fr3/structure_texture_far	0.22	**0.17**	2.3	**0.83**

Table 4.1: The addition of a robust illumination term in our $\mathbb{SE}(3)$ tracker, coupled with the increased depthmap density, results in improved tracking performance. LSD-SLAM had particular trouble with `fr2/desk` and `fr3/long_office_household` and would frequently lose track of the camera or fail to detect a loop closure, resulting in increased error. The results for each dataset are computed across 10 trials.

4.2 Quantitative Evaluation

We ran our pipeline on four video sequences from the TUM RGB-D SLAM Benchmarks [137], which were captured using a hand-held Microsoft Kinect [11] in a variety of environments with pose ground truth provided by a motion capture system. We used the depth frames as a proxy for depth ground truth and compare the performance of MLM against LSD-SLAM [47] using a variety of metrics. We used the first depth frame to initialize our system in order to set the global scale factor and ignored the first 5 keyframes to screen out initialization effects while the poses and depths converge. Furthermore, as both our pipeline and LSD-SLAM are heavily multi-threaded, performance can vary from run to run based on differences in keyframe selection and loop closure detection, so we report metrics averaged over 10 trials for each dataset.

We first evaluate the accuracy of the inverse depth estimates of the two systems by computing the *relative inverse depth error*. For each valid inverse depth estimate in each depthmap, we compute the inverse depth error relative to the corresponding pixel from the Kinect depthmap and normalize this error by the Kinect inverse depth. We then average this error across all pixels with depths and across all trials. As shown in Figure 4-4, MLM's inverse depth estimates have lower relative error than LSD-SLAM's for all four benchmark datasets.

Next, we consider the *density* of the inverse depthmaps. We compute the fraction

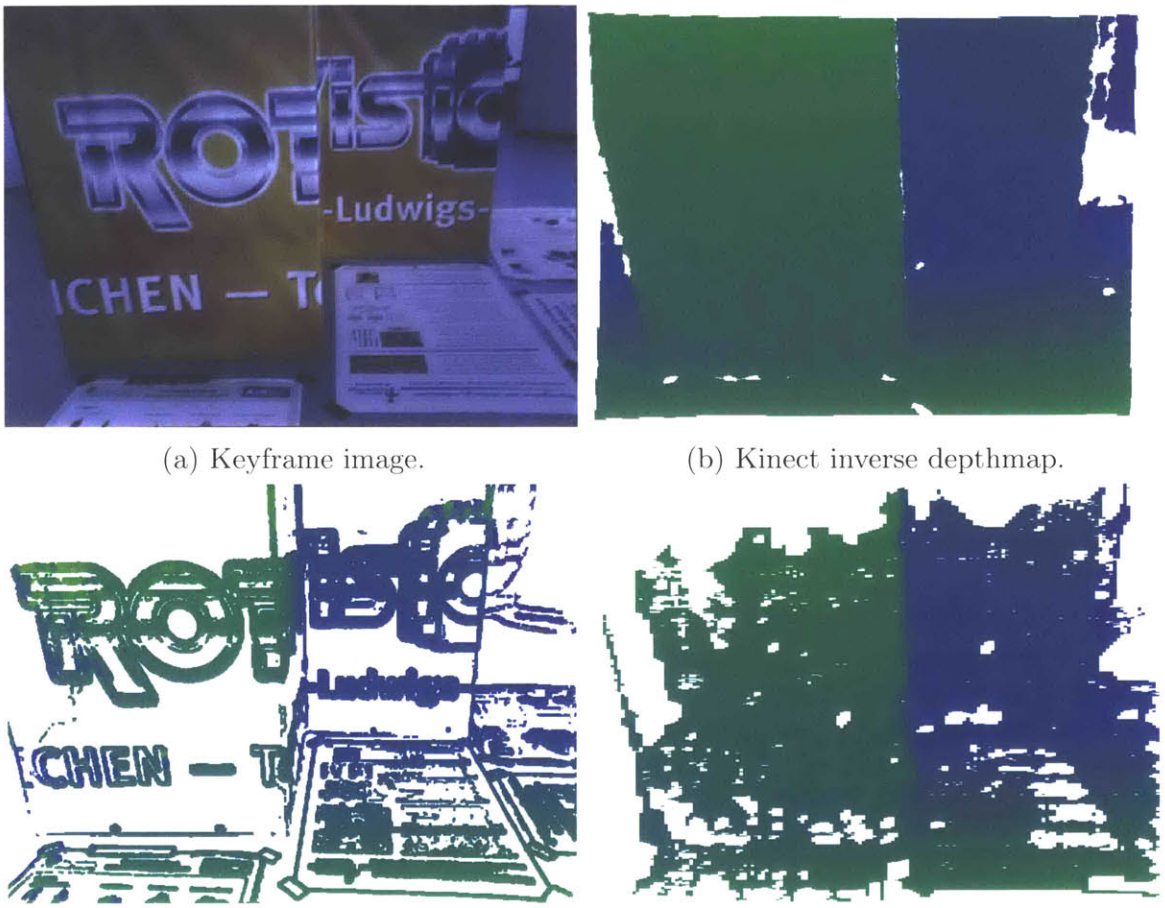
Average Time per Mapping Update [ms]		
Dataset	LSD-SLAM	MLM
fr2/desk	16	17
fr3/long_office_household	13	16
fr3/nostructure_texture_near_withloop	12	15
fr3/structure_texture_far	14	17

Table 4.2: The multi-resolution depth estimation approach of MLM allows for significantly more depth estimates per keyframe, while maintaining real-time operation at 30 Hz. Here, the average run-time per keyframe update, including depth estimation, hole-filling, and rasterization, over 10 trials is presented for each dataset.

of pixels in each depthmap that have inverse depth estimates that are within 10 percent of the corresponding Kinect values (in other words, we only count "accurate" inverse depths in these tallies) and then average these values across all depthmaps and all trials. As shown in Figure 4-5, MLM has *denser* (and more accurate) inverse depthmaps than LSD-SLAM for all four datasets (see Figure 4-7 for examples of the final reconstructions for the four datasets). Reconstructions for one of the datasets (`fr3/structure_texture_far`) are also shown in detail in Figure 4-3 and Figure 4-6. Figure 4-3 compares the depthmap density of LSD-SLAM against MLM with the corresponding Kinect inverse depthmap as a reference. Note that MLM is able to accurately reconstruct the scene in the absence of gradient information. Figure 4-6, in turn, shows how the increased density of each individual depthmap leads to a higher-fidelity point cloud across all keyframes.

The increased reconstruction density of the MLM keyframes also allows for more accurate pose estimates as shown in Table 4.1. With denser, more accurate keyframe depthmaps (in addition to a robust global illumination term), MLM achieves lower tracking error in both position and orientation than LSD-SLAM for all datasets considered.

These improvements come without significantly increasing runtime relative to LSD-SLAM as shown in Table 4.2. By leveraging image data from both high and low spatial frequencies, MLM is able to estimate denser, more accurate reconstructions and more accurate poses with nearly the same runtime as LSD-SLAM.



(a) Keyframe image.

(b) Kinect inverse depthmap.

(c) LSD-SLAM

(d) MLM

Figure 4-3: Our algorithm significantly increases the number of accurate inverse depth estimates per keyframe compared to LSD-SLAM [47], a state-of-the-art algorithm for semi-dense monocular SLAM. Here we compare inverse depthmaps for a single keyframe from LSD-SLAM (c) and MLM (d) on a publicly available benchmark dataset [137]. The corresponding color image for the keyframe is shown in (a) and the Kinect inverse depthmap used as ground truth is shown in (b). Green indicates near depths and blue indicates far depths. Note the increased reconstruction density and quality of MLM when compared to LSD-SLAM.

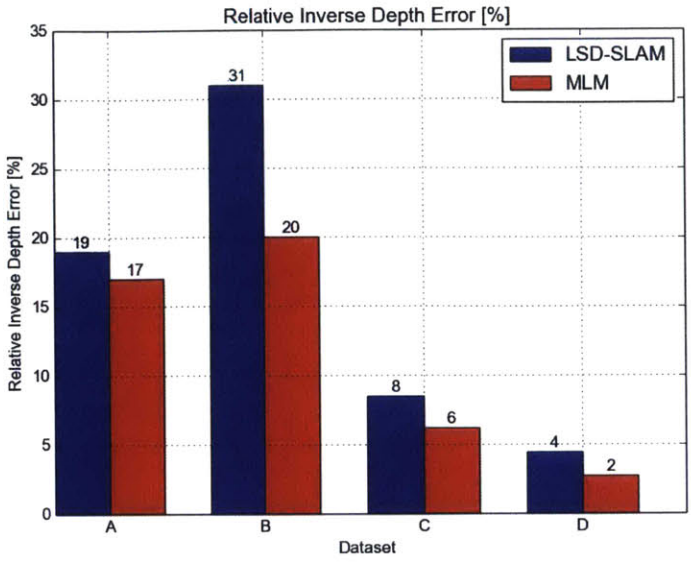


Figure 4-4: For each keyframe, we compute the inverse depth error relative to the corresponding value from the Kinect across all pixels with valid estimates. The results from each dataset are averaged across all keyframes and across 10 trials and are displayed above each bar. Our multi-level approach achieves more accurate depth estimates on all four datasets.

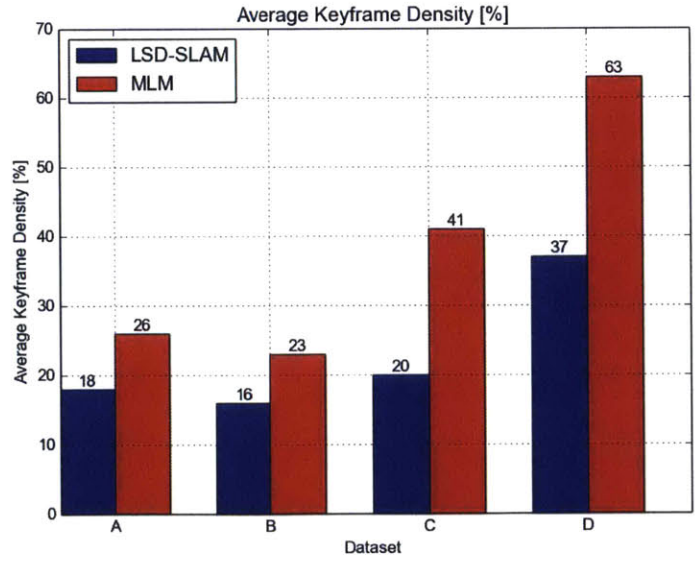
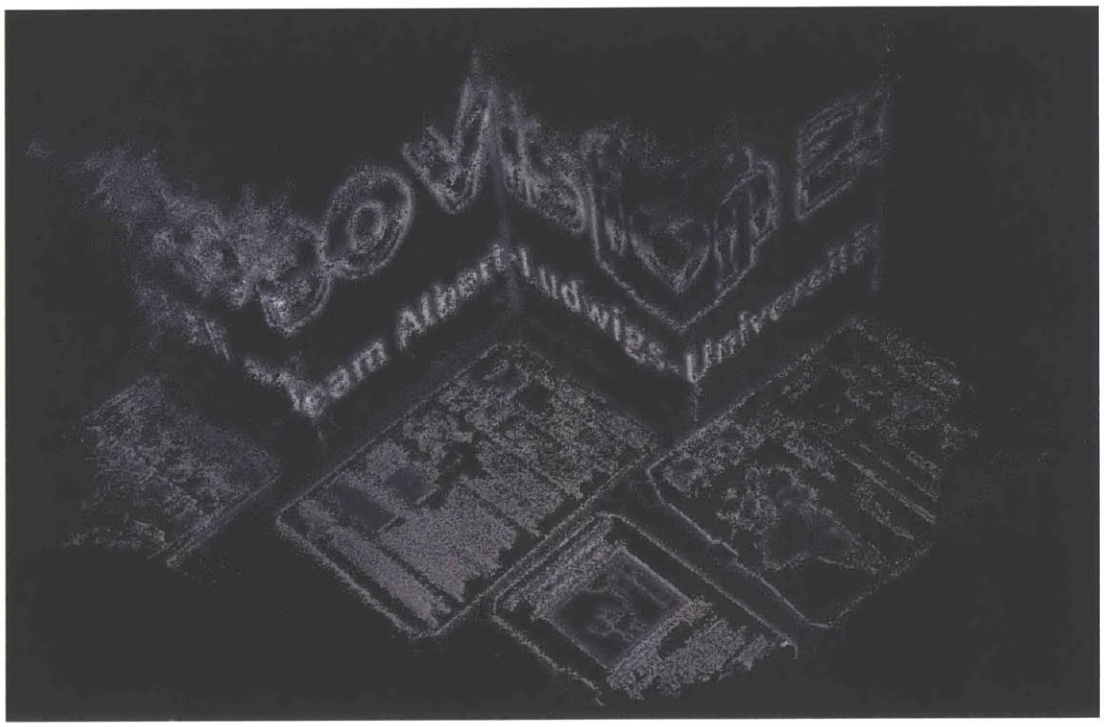


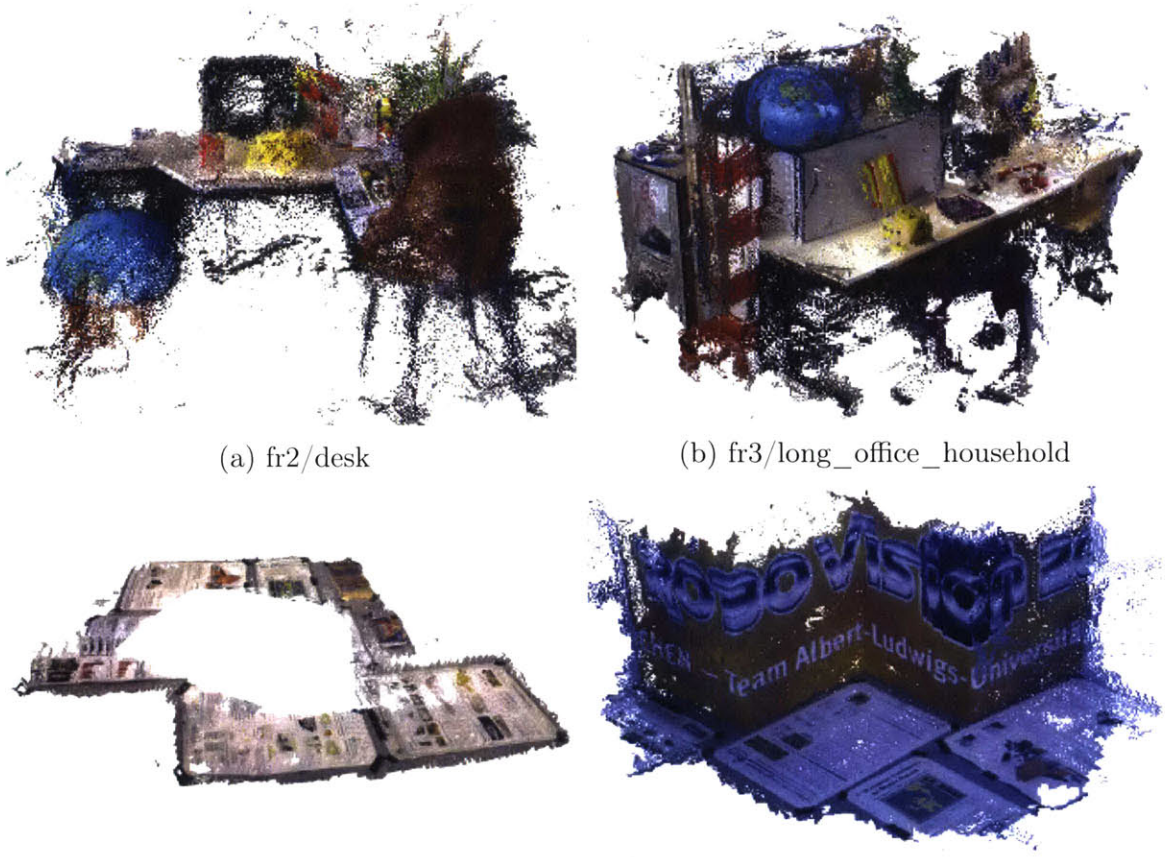
Figure 4-5: We compute the fraction of pixels in each keyframe with inverse depth estimates that are within 10 percent of the corresponding values from the Kinect depth frames. The results for each dataset are averaged across all keyframes and across 10 trials and are displayed above each bar. Our multi-level approach significantly increases the number of accurate inverse depths per keyframe.



(a) LSD-SLAM Point Cloud.

(b) MLM Point Cloud.

Figure 4-6: Here we compare the final point clouds from LSD-SLAM (a) and MLM (b) on a publicly available benchmark dataset [137]. Note how MLM is able to intelligently interpolate depth information through low texture regions.



(a) fr2/desk

(b) fr3/long_office_household

(c) fr3/nostruct._tex_near_w.l.

(d) fr3/structure_texture_far

Figure 4-7: We validate our monocular SLAM pipeline on publicly available benchmark datasets captured using a Microsoft Kinect [137], which demonstrates our algorithm's ability to produce dense reconstructions through low-texture image regions.

4.3 Discussion

The results of the previous section show that MLM is able to significantly improve upon the state-of-the-art in CPU-only monocular SLAM in terms of both reconstruction accuracy and density by intelligently distributing computational resources to image regions with high-texture while estimating low-texture regions at coarser resolutions.

The increased inverse depthmap density, coupled with global illumination compensation, also improves tracking performance in both position and orientation. Note that the benchmark videos in [137] are challenging for pure monocular SLAM systems due to the automatic settings (gain, exposure, brightness, etc.) and rolling shutter of the Kinect's RGB camera as well as camera motion. LSD-SLAM had particular trouble with the first two datasets (`fr2/desk` and `fr3/long_office_household`) and frequently lost track of the camera or failed to detect a loop closure, leading to increased tracking error over the 10 trials.

MLM appears to generate the best reconstructions across the different datasets when the geometric structure is mainly planar and fronto-parallel, as in the laboratory workspace (Figure 4-2), `fr3/nostructure_texture_near_withloop`, and `fr3/structure_texture_far`. This is most likely because these scenes play to the strengths of the variational regularizer described in Section 3.7, which biases the inverse depthmaps to be piecewise constant.

Fine geometric structure (e.g. the tree in Figure 4-1b) and depth discontinuities (e.g. the desk lamps in Figure 4-1a) are not reconstructed as cleanly, however. Fine geometric structure may be problematic because the depth signal from these regions may fall below the noise floor dictated by the sensor noise and texture strength, encouraging more false-matches. Depth discontinuities appear to be *smeared* in some cases, suggesting the regularization strength is too high. Weighting the regularizer by the image gradient (as done in [35]) may help in these cases.

Chapter 5

Adaptive Depth Meshing

This chapter presents an alternative dense reconstruction method to the multi-resolution approach of MLM discussed in Chapter 3 and Chapter 4. MLM is able to produce dense reconstructions by varying the detail of the geometry based on the available image texture. Since low-texture image regions do not carry significant depth signal and are often correlated with simple geometric structure, MLM represents these regions with more coarsely sampled depth estimates, allowing it to generate dense inverse depthmaps with the speed of semi-dense methods.

MLM, however, always represents high-texture image regions with finely sampled depth estimates — even if the corresponding geometry is actually simple. Consider, for example, a floor covered with highly textured carpet or tiles. Although the number of depth estimates needed to accurately reconstruct the floor is quite small (e.g. the four corners of the room), MLM will run an expensive epipolar search for every pixel with high gradient across the floor. The approach discussed in this chapter, however, directly adapts the reconstruction to the estimated scene geometry such that simple geometric structures are efficiently represented with only a small number of depth estimates that can be estimated quickly, no matter the texture.

The intuition behind the method is that given the computational load of depth estimation, SWaP constrained platforms should carefully choose which pixels to estimate depth for so as to not waste resources on points which add no value to the overall reconstruction. This approach was inspired by the recent work of [138], which

Algorithm 1 Update depthmesh with new image.

function UPDATEDEPTHMESH($V_{k-1}, \mathbf{T}_k^W, I_k$)
    // Update each vertex.
    for $v_i \in V_{k-1}$ **do**
        // Track vertex into new frame.
        $v_i^- \leftarrow \text{TRACK}(v_i, \mathbf{T}_k^W, I_k)$

        // Compute depth using track history.
        $(z_i, \sigma_{z_i}^2) \leftarrow \text{DEPTH}(\mathbf{u}_i^-, h_i)$

        // Fuse depth.
        $(\xi_i, \sigma_i^2) \leftarrow \text{FUSE}((\xi_i^-, \sigma_i^{2-}), (z_i, \sigma_{z_i}^2))$
    end for

    // Refine mesh.
    $V_k \leftarrow \text{REFINEMESH}(V_k)$
end function

estimates a semi-dense disparity map from stereo images, iteratively refining the map by adding new disparity estimates where the matching fit is poor.

The approach presented in this chapter robustly tracks high gradient pixels between consecutive frames using semi-dense optical-flow based technique that allows for wide-baseline depth measurements (Section 5.2). These high-gradient pixels are then interpreted as the vertices of a dense, piecewise linear, triangular depthmesh (Section 5.3) that is iteratively refined to capture the geometry of the scene with a small number of vertices (far fewer than either MLM or LSD-SLAM [47]) (Section 5.4), which may result in a significant speedup over the state-of-the-art. For now, we assume that the camera poses are known and concentrate on the depth estimation problem exclusively.

5.1 Algorithm Outline

The main update routine for the approach is summarized in Algorithm 1. Given an image sequence $I_k : \Omega \rightarrow \mathbb{R}_+$ (over image domain $\Omega \subset \mathbb{R}^2$) from a moving camera with known pose $\mathbf{T}_k^W \in \mathbb{SE}(3)$ in world frame W, at each timestep k we wish to

estimate the inverse depthmap $D_k : \Omega \to \mathbb{R}_+$, which we will approximate using a piecewise-planar, triangular, inverse depth *mesh*. The vertices of the mesh will be denoted by V_k. Each vertex $v_i \in V_k$ is defined at a pixel location $\mathbf{u}_i \in \Omega$ in the current frame and at locations $h_i = \{\mathbf{u}_i^j\}$ for past frames $j < k$, with an inverse depth estimate $\xi_i \in \mathbb{R}_+$ and inverse depth variance $\sigma_i^2 \in \mathbb{R}_+$. Assuming that the scene geometry is smooth, we produce a dense inverse depthmap that approximates the scene by linearly interpolating the inverse depths between the mesh vertices.

For each new tracked image $(I_k, \mathbf{T}_k^W)$, the vertices from the previous timestep V_{k-1} are tracked into the new image using a Lucas-Kanade-style [76] least squares optimization. The inverse depth estimate for each vertex is subsequently updated based on the result of this tracking. A wide-baseline depth measurement $\mathbf{z}_i$ with uncertainty $\sigma_{\mathbf{z}_i}^2$ is then produced for each vertex v_i using a past vertex location using the history h_i. The noisy measurement is then fused with the vertex's inverse depth estimate using a standard Bayesian update as in MLM [9] and LSD-SLAM [46, 47] (see Section 3.4). Finally, the updated vertex set is refined by adding new vertices where the reconstruction fit is poor.

5.2 Pixel Tracking

For each vertex $v_i \in V_{k-1}$, we first *predict* where its corresponding pixel $\mathbf{u}_i$ projects into the new image I_k given the current inverse depth estimate ξ_i and the current pose $\mathbf{T}_k^W$:

$$\mathbf{u}_i^- = \pi(\mathbf{K}\mathbf{T}_W^k\mathbf{T}_{k-1}^W\mathbf{K}^{-1}(\bar{\mathbf{u}}_i/\xi_i)), \tag{5.1}$$

where $\mathbf{u}_i^-$ is the predicted location.

We next refine $\mathbf{u}_i^-$ by minimizing the following nonlinear least squares cost function along the epipolar line $\mathbf{e}_i \in \mathbb{R}^2$ (see Section 2.3.1):

$$E(\Delta) = \frac{1}{2}\sum_{\mathbf{v}\in\mathcal{W}}\left(I_k(\mathbf{u}_i^- + \mathbf{v} + \mathbf{e}_i\Delta) - I_{k-1}(\mathbf{u}_i + \mathbf{v})\right)^2 + \frac{1}{2\sigma_i^2}\Delta^2 \tag{5.2}$$

where the $\mathbf{v} \in \mathcal{W}$ are an equally spaced window of samples along the epipolar line and $\Delta \in \mathbb{R}$ is the amount that $\mathbf{u}_i^-$ moves along the epipolar line (usually called the *optical flow*). In addition to constraining the pixel motion to lie along $\mathbf{e}_i$, the second term in Equation (5.2) also regularizes the motion, with the regularization strength governed by the vertex's inverse depth variance σ_i^2. The more confident that an inverse depth estimate is (i.e. the lower the variance), the more costly large pixel motions become.

We optimize Equation (5.2) iteratively using the inverse compositional Lucas-Kanade formulation [139, 140]. Given a current estimate of the motion Δ (initially zero), we find the optimal increment δ that minimizes

$$\begin{aligned}\tilde{E}(\delta) = \frac{1}{2} \sum_{\mathbf{v} \in \mathcal{W}} \left(I_{k-1}(\mathbf{u}_i + \mathbf{v} + \mathbf{e}_i \delta) - I_k(\mathbf{u}_i^- + \mathbf{v} + \mathbf{e}_i \Delta) \right)^2 \\ + \frac{1}{2\sigma_i^2} (\Delta + \delta)^2.\end{aligned} \tag{5.3}$$

Note that the roles of I_k and I_{k-1} are reversed in the inverse compositional formulation so that the Hessian of the cost function does not need to be computed at every iteration [91].

We optimize $\tilde{E}$ by performing coarse-to-fine Gauss-Newton steps by linearizing $I_{k-1}(\mathbf{u}_i + \mathbf{v} + \mathbf{e}_i \delta)$ about $\delta = 0$:

$$I_{k-1}(\mathbf{u}_i + \mathbf{v} + \mathbf{e}_i \delta) \approx I_{k-1}(\mathbf{u}_i + \mathbf{v}) + \nabla I_{k-1}(\mathbf{u}_i + \mathbf{v}) \mathbf{e}_i \delta, \tag{5.4}$$

and solving the linear system induced by substituting this approximation into Equation (5.3) for the optimal motion increment δ. This increment is then used to update the gross motion $\Delta \leftarrow \Delta - \delta$. (Note the sign difference of the increment δ with the inverse compositional formulation.)

Once the optimization has converged, we update the pixel location of the vertex $\mathbf{u}_i^- \leftarrow \mathbf{u}_i^- + \mathbf{e}_i \Delta$ and update the the inverse depth accordingly to obtain the predicted estimate (ξ_i^-, σ_i^{2-}). We will denote this *predicted* vertex by v_i^-.

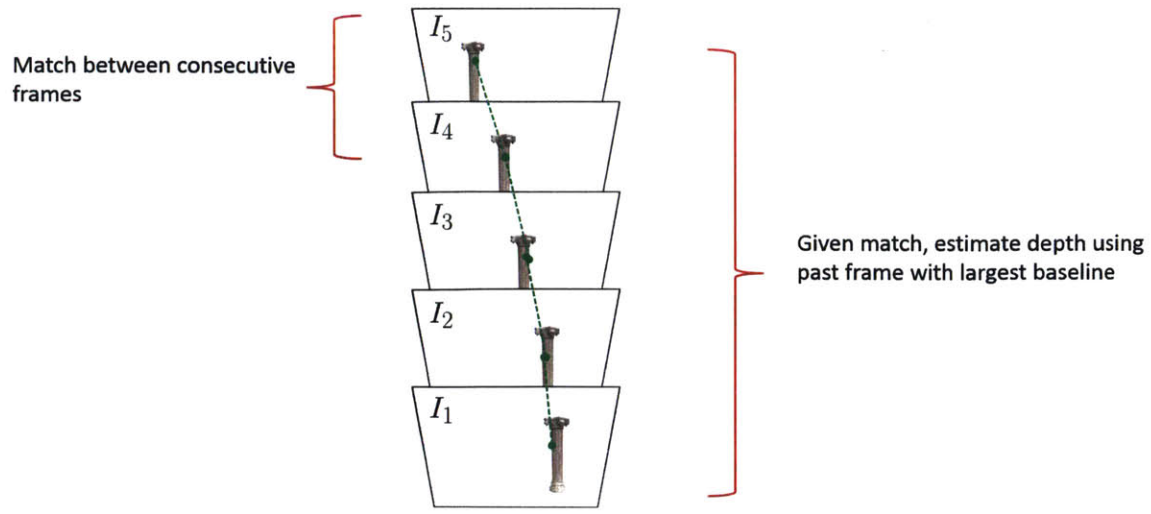


Figure 5-1: The scenario depicted above shows a pixel (green) tracked between a succession of images. Tracking is facilitated by leveraging small baseline images. For example, between I_4 and I_5. On the other hand, wide-baseline depth estimation is enabled by leveraging comparison images further into the past, such as I_5 and I_1.

5.3 Depth Estimation and Fusion

With the vertex v_i^- tracked into the current I_k, we compute a new wide-baseline depth measurement z_i by triangulating $\mathbf{u}_i^-$ with the oldest pixel $\mathbf{u}_i^{j_0}$ in the history of detections h_i.

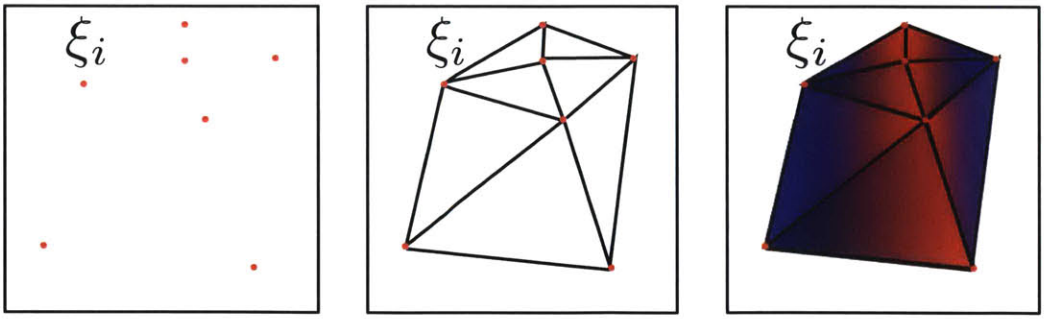
We then compute a measurement variance $\sigma_{z_i}^2$ for this value using the noise model in [47], and fuse $(z_i, \sigma_{z_i}^2)$ with (ξ_i^-, σ_i^{2-}) using a standard Bayesian update:

$$\xi_i \leftarrow \frac{\sigma_{z_i}^2 \xi_i^- + \sigma_i^{2-} z_i}{\sigma_i^{2-} + \sigma_{z_i}^2} \tag{5.5}$$

$$\sigma_i^2 \leftarrow \left(\frac{1}{\sigma_i^{2-}} + \frac{1}{\sigma_{z_i}^2} \right)^{-1} \tag{5.6}$$

before setting the updated vertex v_i.

Estimating inverse depths in this manner allows the pixel association problem to be decoupled from the actual depth computation (see Figure 5-1). Pixel association or tracking is typically more robust when the baseline between the two images is small (because the images are more similar), while depth estimation is typically more robust when the baseline is large (because it allows for better triangulation). By separating



(a) Vertices (b) Triangulation (c) Rasterization

Figure 5-2: A dense depthmap is computed from a sparse set of vertices V_k (a). A triangular mesh of the vertices is computing using a Delaunay triangulation that maximizes the minimum angle across all triangles (b). The inverse depths inside the convex hull of vertices are computed by interpolating between the inverse depth of the vertices using a simple rasterization scheme.

the two tasks, we can associate pixels using a small-baseline comparison image (i.e. the previous frame), but triangulate using a wide-baseline comparison image (i.e. the oldest element in the history).

5.4 Mesh Refinement

With the updated vertex set V_k, we now mesh the vertices (Section 5.4.1) and regularize the inverse depth estimates using the connectivity of the mesh assuming the scene geometry is smooth (Section 5.4.2). Finally, new vertices are added to the mesh in areas that are not well reconstructed (Section 5.4.3).

5.4.1 Delaunay Triangulation

We use the open-source library `Triangle` [141] to compute a *Delaunay triangulation* of the vertices V_k. A *triangulation* of a set of 2D points partitions the convex hull of the points into triangular regions such that the corners of the triangles lie at the points. The set of such triangles is typically called a *mesh*. Triangulations are not unique [133]. The Delaunay triangulation [142] generates triangles such that the minimum angle across all the triangles in the mesh is maximized (i.e. "skinny" triangles are discouraged). Once the inverse depth vertex set V_k is triangulated, we

linearly interpolate the inverse depths inside each triangle using a simple rasterization scheme to achieve a dense depthmap D_k (see Figure 5-2).

5.4.2 Regularization

In order to smooth the inverse depths and remove outliers we use a simple two-stage approach. First, gross outlier inverse depths are removed by applying a median filter to each vertex $v_i \in V_k$ using its neighbors computed by the Delaunay triangulation. More specifically, for each vertex v_i, we fetch the inverse depths of its neighbors based on the triangulation. If the inverse depth estimate ξ_i is far from the median value of the neighbors, we replace ξ_i with the median, otherwise we leave the value untouched. We then apply a low-pass filter using the same approach: we replace the depth estimate ξ_i for vertex v_i with the mean of its neighbors.

5.4.3 Mesh Augmentation

New vertices are added to the mesh at each iteration of the algorithm while respecting a user-defined density parameter $p \in (0, 1)$. Small p will reduce the density of vertices in the mesh, while large p will increase the density. The specific value is set to maintain real-time operation.

To add vertices, we first create a binary mask of the current depthmap using the current vertex set V_k. If an $N \times N$ region of pixels contains a vertex, the region will be set to false. We then detect FAST corners and high-gradient pixels in the uncovered regions at multiple image levels and initialize a vertex at the detected location with probability p.

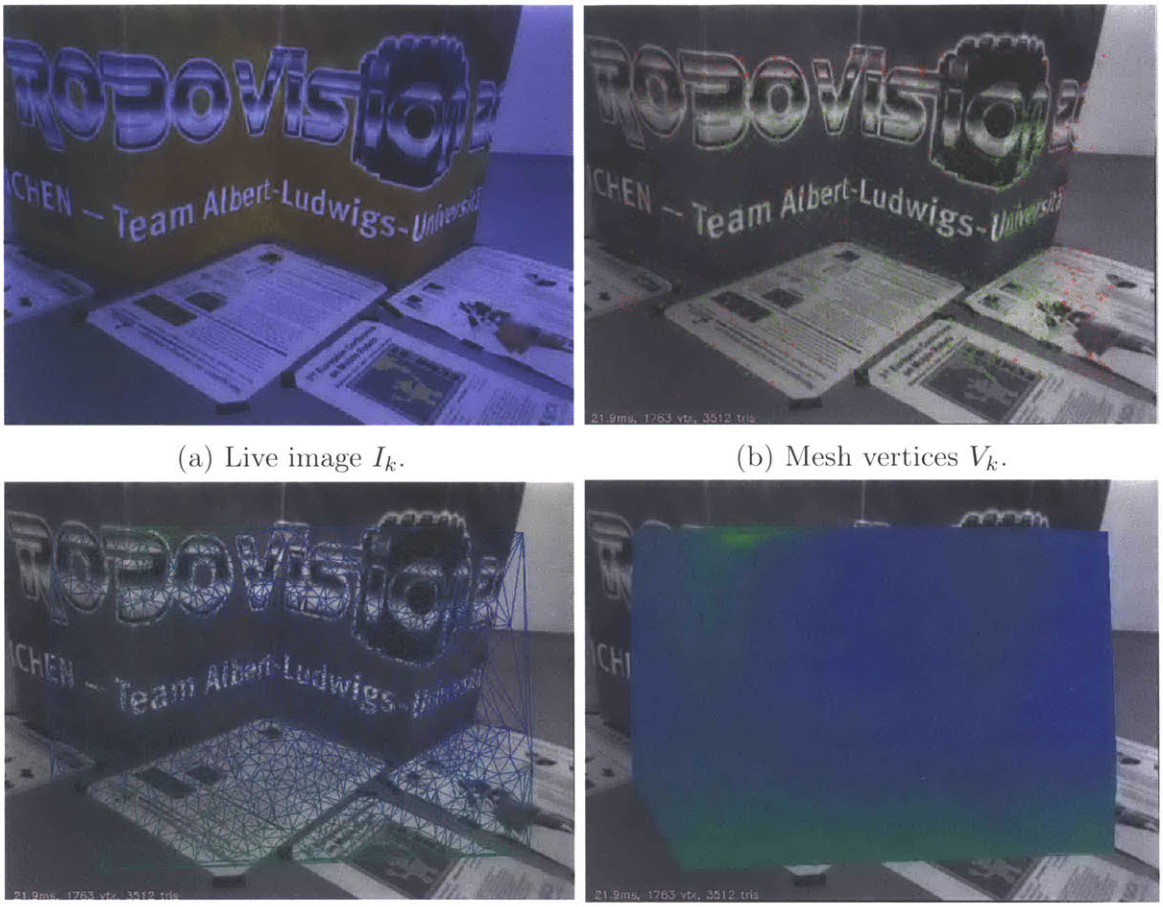
New vertices are also added based on the matching cost generated by the depthmap D_k induced by V_k. For each pixel $\mathbf{u} \in \Omega$ that is not a vertex and has a depth in D_k, we compute the sum-of-squared differences between a window of pixels around $\mathbf{u}$ in I_k and around the projection of $\mathbf{u}$ into the previous frame assuming depth $D_k(\mathbf{u})$. If the matching cost exceeds a threshold, we add it to the vertex set with probability p.

5.5 Preliminary Results

We present preliminary qualitative reconstruction results using the `fr3/structure_texture_far` dataset from the TUM RGB-D SLAM benchmarks [137] with ground truth pose estimates from a motion capture system. Our implementation is written in C++. All results were produced on a consumer desktop with an Intel Core i7 processor with 4 hyperthreaded cores and 16 GB of RAM.

Figure 5-3 shows a snapshot of the algorithm at a particular point in the dataset using $p = 0.001$ and 5 pyramid levels for tracking. The input image is shown in the upper left, while the tracked vertices V_k are shown in the upper right. The colors correspond to track age, with red points indicating new tracks that have yet to converge. The result of the Delaunay triangulation is shown in the bottom left and the dense interpolated inverse depthmap is shown in the bottom right. The colormap indicates depth, with blue indicating far depths and green indicating near depths. Note the completeness of the depthmap, despite the fact that the number of vertices tracked ranges from 1500 to 2500, which is an order of magnitude fewer than that of MLM [9] and LSD-SLAM [47].

Figure 5-4 shows the resulting point cloud induced from the dense inverse depthmap for this instance in time and compares it to the final point cloud from MLM. Note the reduced outliers and noise in the reconstruction due to the implicit regularization afforded by the mesh, as well as the comparable reconstruction density. With additional improvements, this approach should offer dense reconstructions with significantly lower computational requirements than the state-of-the-art.



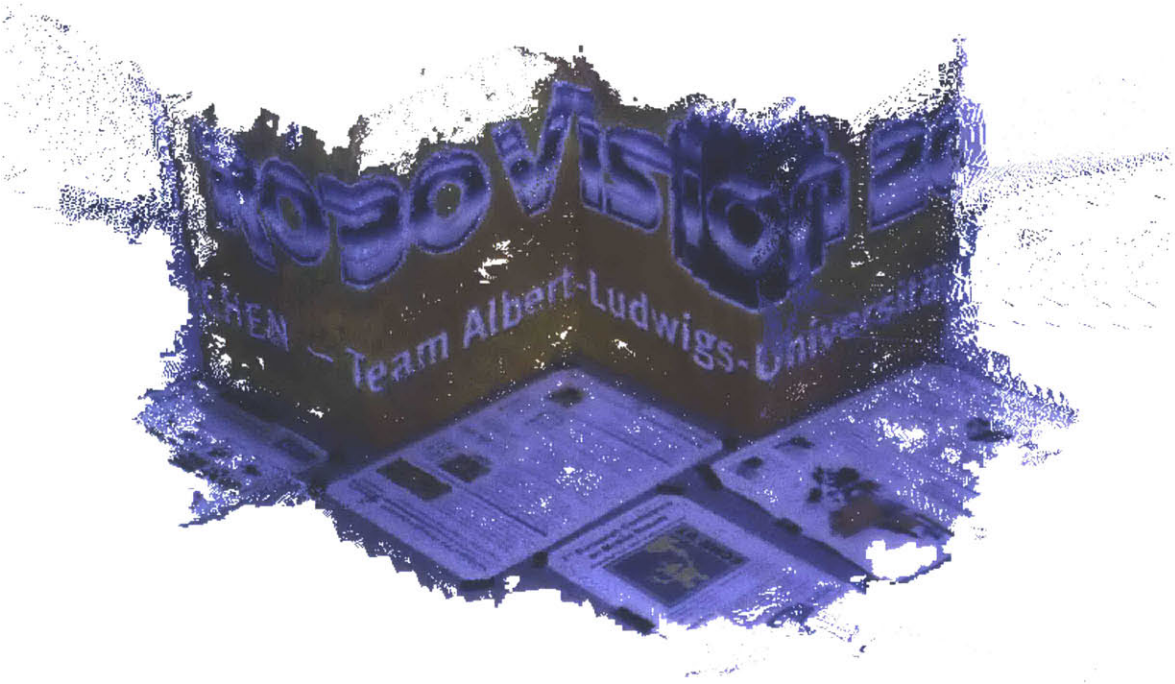
(a) Live image I_k.

(b) Mesh vertices V_k.

(c) Delaunay triangulation.

(d) Interpolated Inverse Depthmap D_k.

Figure 5-3: We show preliminary qualitative reconstruction results on a publicly available benchmark dataset [137]. (a) shows the live image at this point in the dataset. (b) shows the tracked mesh vertices V_k colored by track age (red indicates a young track). The Delaunay triangulation of the vertices is depicted in (c). The dense inverse depthmap (blue indicates far, green indicates near) induced by the triangulation is shown in (d). Note the completeness and fidelity of the depthmap despite the small number of vertices tracked (1500-2500), an order of magnitude lower than MLM [9] and LSD-SLAM [47].



(a) MLM

(b) Meshing approach

Figure 5-4: Here we compare the point clouds from MLM (a) and a single frame from our meshing approach (b). Note the decreased noise in the meshing point cloud due to the implicit regularization afforded by the mesh, as well as the increased density despite estimating inverse depths for far fewer points.

5.6 Discussion

The adaptive depth meshing algorithm described in the previous sections is a promising approach to dense reconstruction that intelligently balances reconstruction density and fidelity with the number of stereo computations. Additional work is needed, however, to make the pixel tracking more efficient and depth fusion more robust. Although the point cloud shown in Figure 5-4 is encouraging, there are slight biases in the depth estimates (e.g. the poster on the floor) that can most likely be corrected. For example, the depth noise model can be augmented to account for outlier measurements directly as in [45]. The biases could also be due to errors in the input poses such that the image textures are not correctly aligned. Refining the incoming poses using the current mesh (such that the map and poses are consistent) could reduce this effect. A more intelligent regularization scheme based on total variation could also be employed to increase the map fidelity, although some care would need to be taken in order to apply it directly to the mesh representation. More work is also needed to generalize the reconstruction performance to disparate datasets, minimize the computational load to run on low-SWaP hardware, and evaluate the accuracy of the approach.

Further computational savings could also be made by not only *augmenting* the mesh for regions with high photometric error, but also *pruning* vertices from the mesh that are redundant. Most environments may be accurately described with only a small number of 3D points that correspond to the corners of the geometric structure. The pixels that these points project to can be thought of the minimal set of vertices needed to accurately reconstruct the environment. The mesh refinement method described could be modified such that the vertices tracked are whittled down to only this minimal set. In addition to the increased efficiency that this would provide, it would also add additional smoothness benefits from the implicit regularization afforded by the mesh. A machine learning approach could also be employed to *learn* which 2D pixels in the input imagery correspond to this minimal set of 3D points, which may make the approach robust to different types of imagery and camera motion.

Chapter 6

Conclusion

Despite the rapid progress in real-time monocular SLAM over the last several years, state-of-the-art approaches are either too computationally expensive, too limited in scale, or too geometrically sparse to be successfully used for high-speed MAV navigation. This thesis presented research that addresses some of these shortcomings and allows fully dense geometry to be estimated with the speed, efficiency, and scale of semi-dense methods.

A novel dense monocular SLAM algorithm called Multi-Level Mapping (MLM) [9] was presented in Chapter 3 that allows dense 3D geometry to be estimated online without the aid of a graphics processing unit (GPU). The key contribution is a multi-resolution depth estimation and spatial smoothing process that exploits the correlation between low-texture image regions and simple planar structure to adaptively scale the complexity of the generated reconstruction to the quality of the input imagery. High-texture image regions are represented at higher resolutions to capture fine detail, while low-texture regions are represented at coarser resolutions for smooth surfaces. This approach allows for significant computational savings while simultaneously increasing reconstruction density and quality when compared to the state-of-the-art.

MLM was evaluated in Chapter 4 both qualitatively on hand-held camera data, as well as quantitatively on publicly available benchmark datasets [137]. The quantitative analysis shows that MLM improves upon the state-of-the-art in CPU-only monocular SLAM in terms of both tracking and mapping performance. MLM pro-

duced more accurate pose estimates than LSD-SLAM [47] across all datasets. It also significantly increased the accuracy and density of the produced depthmaps without sacrificing speed, leading to higher-fidelity reconstructions produced online.

Preliminary qualitative results were also presented for an adaptive meshing technique in Chapter 5 that may further reduce the computational requirements for fully dense monocular SLAM. The intuition behind the approach is that it is much more efficient to check whether a depth hypothesis is supported by the available imagery than it is to actually compute the correct depth. To that end, we maintain a piecewise-linear dense depth mesh whose vertices comprise a subset of the high image gradient pixels. At each frame, the mesh is refined by computing the stereo matching cost for each pixel in the dense depthmap induced by the mesh. Vertices are then added to the mesh where the interpolated depthmap poorly fits the input imagery, such that depths are only computed for those pixels that are needed to represent the geometry in the scene, which offers a significant computation speedup that brings MAV-capable dense monocular SLAM one step closer to viability.

Bibliography

[1] Keith Naughton. Google's Driverless-Car Czar on Taking the Human Out of the Equation. http://www.bloomberg.com/features/2016-john-krafcik-interview-issue. Accessed: 2016-08-11.

[2] Evan Ackerman. Skydio's Camera Drone Finally Delivers on Autonomous Flying Promises. http://spectrum.ieee.org/automaton/robotics/drones/skydio-camera-drone-autonomous-flying. Accessed: 2016-08-04.

[3] Steve Strunsky. Fugitive who fled police in Passaic River sewer pipe may be trapped, cops say. http://www.nj.com/essex/index.ssf/2016/08/passaic_river_manhunt_intensifies_after_suspect_es.html. Accessed: 2016-08-04.

[4] Scott Shane and David E. Sanger. Drone Crash in Iran Reveals Secret U.S. Surveillance Effort. http://www.nytimes.com/2011/12/08/world/middleeast/drone-crash-in-iran-reveals-secret-us-surveillance-bid.html. Accessed: 2016-08-11.

[5] Farhad Manjoo. Think Amazon's Drone Delivery Idea is a Gimmick? Think Again. http://www.nytimes.com/2016/08/11/technology/think-amazons-drone-delivery-idea-is-a-gimmick-think-again.html. Accessed: 2016-08-11.

[6] Steve Dent. Feds give Google OK to test Project Wing drone deliveries. https://www.engadget.com/2016/08/02/feds-give-google-ok-to-test-project-wing-drone-deliveries. Accessed: 2016-08-11.

[7] Kelsey D. Atherton. NASA is Testing a Drone for Mars. http://www.popsci.com/nasa-has-mars-plane-concept. Accessed: 2016-08-04.

[8] Elizabeth Landau. Helicopter Could Be 'Scout' for Mars Rovers. http://www.jpl.nasa.gov/news/news.php?feature=4457.

[9] W. Nicholas Greene, Kyel Ok, Peter Lommel, and Nicholas Roy. Multi-Level Mapping: Real-time Dense Monocular SLAM. In *Proc. ICRA*, 2016.

[10] Hokuyo Automatic Co. LTD. Hokuyo UTM-30LX. https://www.hokuyo-aut.jp/02sensor/07scanner/utm_30lx.html. Accessed: 2016.07.20.

[11] Microsoft. Kinect for Windows Sensor Components and Specifications. https://msdn.microsoft.com/en-us/library/jj131033.aspx. Accessed: 2016.07.20.

[12] PointGrey. Bumblebee2 1394a. https://www.ptgrey.com/bumblebee2-firewire-stereo-vision-camera-systems. Accessed: 2016.07.20.

[13] PointGrey. Firefly MV. https://www.ptgrey.com/firefly-mv-usb2-cameras. Accessed: 2016.07.20.

[14] Olivier Faugeras, Quang-Tuan Luong, and Theo Papadopoulo. *The geometry of multiple images: the laws that govern the formation of multiple images of a scene and some of their applications.* MIT Press, 2004.

[15] Richard Hartley and Andrew Zisserman. *Multiple View Geometry.* Cambridge University Press, 2003.

[16] Grant Schindler, Frank Dellaert, and Sing Bing Kang. Inferring temporal order of images from 3d structure. In *Proc CVPR*, 2007.

[17] David G Lowe. Object recognition from local scale-invariant features. In *Proc. ICCV*, 1999.

[18] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (SURF). *Comp. Vis. and Image Understanding*, 2008.

[19] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. ORB: An efficient alternative to SIFT or SURF. In *Proc. ICCV*, 2011.

[20] Davide Scaramuzza and Friedrich Fraundorfer. Visual Odometry: Part 1: The First 30 Years and Fundamentals. *IEEE Robotics & Automation Magazine*, 2011.

[21] Friedrich Fraundorfer and Davide Scaramuzza. Visual Odometry: Part 2: Matching, Robustness, Optimization, and Applications. *IEEE Robotics & Automation Magazine*, 2012.

[22] Bill Triggs, Philip F McLauchlan, Richard I Hartley, and Andrew W Fitzgibbon. Bundle adjustmentâĂŤa modern synthesis. In *International Workshop on Vision Algorithms*, 1999.

[23] Frank Dellaert, Steven M Seitz, Charles E Thorpe, and Sebastian Thrun. Structure from motion without correspondence. In *Proc. CVPR*, 2000.

[24] Sameer Agarwal, Noah Snavely, Ian Simon, Steven M Seitz, and Richard Szeliski. Building rome in a day. In *Proc. ICCV*, 2009.

[25] Noah Snavely, Steven M Seitz, and Richard Szeliski. Photo tourism: exploring photo collections in 3d. In *Transactions on Graphics (TOG)*, 2006.

[26] Kurt Konolige and Willow Garage. Sparse sparse bundle adjustment. In *Proc. BMVC*, 2010.

[27] Jorge Nocedal and Stephen Wright. *Numerical optimization.* Springer Science & Business Media, 2006.

[28] Manolis IA Lourakis and Antonis A Argyros. Sba: A software package for generic sparse bundle adjustment. *Transactions on Mathematical Software (TOMS)*, 2009.

[29] Andrew J Davison. Real-time simultaneous localisation and mapping with a single camera. In *Proc. ICCV*, 2003.

[30] Georg Klein and David Murray. Parallel tracking and mapping for small AR workspaces. In *Proc. ISMAR*, 2007.

[31] Raul Mur-Artal, JMM Montiel, and Juan D Tardos. Orb-slam: a versatile and accurate monocular slam system. *Trans. on Robotics*, 2015.

[32] Anastasios I Mourikis and Stergios I Roumeliotis. A multi-state constraint kalman filter for vision-aided inertial navigation. In *Proc. ICRA*, 2007.

[33] Christian Forster, Luca Carlone, Frank Dellaert, and Davide Scaramuzza. Imu preintegration on manifold for efficient visual-inertial maximum-a-posteriori estimation. In *RSS*, 2015.

[34] Allen D Wu, Eric N Johnson, Michael Kaess, Frank Dellaert, and Girish Chowdhary. Autonomous flight in gps-denied environments using monocular vision and inertial sensors. *Journal of Aerospace Information Systems*, 2013.

[35] Richard A Newcombe, Steven J Lovegrove, and Andrew J Davison. DTAM: Dense tracking and mapping in real-time. In *Proc. ICCV*, 2011.

[36] Vivek Pradeep, Christoph Rhemann, Shahram Izadi, Christopher Zach, Michael Bleyer, and Steven Bathiche. MonoFusion: Real-time 3D reconstruction of small scenes with a single web camera. In *Proc. ISMAR*, 2013.

[37] Matia Pizzoli, Christian Forster, and Davide Scaramuzza. REMODE: Probabilistic, monocular dense reconstruction in real time. In *Proc. ICRA*, 2014.

[38] Gottfried Graber, Thomas Pock, and Horst Bischof. Online 3D reconstruction using convex optimization. In *Proc. ICCV workshop*, 2011.

[39] Peter Ondruska, Pushmeet Kohli, and Shahram Izadi. MobileFusion: Real-time volumetric surface reconstruction and dense tracking on mobile phones. In *Trans. on Visualization and Computer Graphics*, 2015.

[40] Andreas Wendel, Michael Maurer, Gottfried Graber, Thomas Pock, and Horst Bischof. Dense reconstruction on-the-fly. In *Proc. CVPR*, 2012.

[41] Michael Goesele, Brian Curless, and Steven M Seitz. Multi-view stereo revisited. In *Proc. CVPR*, 2006.

[42] Yasutaka Furukawa and Jean Ponce. Accurate, dense, and robust multiview stereopsis. *Trans. PAMI*, 2010.

[43] Vu Hoang Hiep, Renaud Keriven, Patrick Labatut, and Jean-Philippe Pons. Towards high-resolution large-scale multi-view stereo. In *Proc. CVPR*, 2009.

[44] NVIDIA. Jetson TK1. `https://www.nvidia.com/object/jetson-tk1-embedded-dev-kit.html`. Accessed: 2016.07.20.

[45] Christian Forster, Matia Pizzoli, and Davide Scaramuzza. SVO: Fast semi-direct monocular visual odometry. In *Proc. ICRA*, 2014.

[46] Jakob Engel, Jurgen Sturm, and Daniel Cremers. Semi-dense visual odometry for a monocular camera. In *Proc. ICCV*, 2013.

[47] Jakob Engel, Thomas Schöps, and Daniel Cremers. LSD-SLAM: Large-scale direct monocular slam. *Proc. ECCV*, 2014.

[48] Raúl Mur-Artal and Juan D Tardós. Probabilistic semi-dense mapping from highly accurate feature-based monocular slam. *Prov. RSS*, 2015.

[49] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic robotics*. MIT press, 2005.

[50] Hugh Durrant-Whyte and Tim Bailey. Simultaneous localization and mapping: Part I. *IEEE Robotics & Automation Magazine*, 2006.

[51] Tim Bailey and Hugh Durrant-Whyte. Simultaneous localization and mapping: Part II. *IEEE Robotics & Automation Magazine*, 2006.

[52] Frank R Kschischang, Brendan J Frey, and H-A Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 2001.

[53] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT Press, 2009.

[54] Rainer Kümmerle, Giorgio Grisetti, Hauke Strasdat, Kurt Konolige, and Wolfram Burgard. g2o: A general framework for graph optimization. In *Proc. ICRA*, 2011.

[55] Michael Kaess, Ananth Ranganathan, and Frank Dellaert. iSAM: Incremental smoothing and mapping. *Trans. on Robotics*, 2008.

[56] Michael Kaess, Hordur Johannsson, Richard Roberts, Viorela Ila, John J Leonard, and Frank Dellaert. iSAM2: Incremental smoothing and mapping using the bayes tree. *IJRR*, 2011.

[57] Eric A Wan and Rudolph Van Der Merwe. The unscented kalman filter for nonlinear estimation. In *Adaptive Systems for Signal Processing, Communications, and Control Symposium 2000. AS-SPCC. The IEEE 2000*, 2000.

[58] Arnaud Doucet, Nando De Freitas, and Neil Gordon. An introduction to sequential monte carlo methods. In *Sequential Monte Carlo methods in practice.* 2001.

[59] P Cheeseman, R Smith, and M Self. A stochastic map for uncertain spatial relationships. In *Proc. ISRR*, 1987.

[60] MWM Gamini Dissanayake, Paul Newman, Steve Clark, Hugh F Durrant-Whyte, and Michael Csorba. A solution to the simultaneous localization and map building (slam) problem. *IEEE Transactions on robotics and automation*, 2001.

[61] Michael Montemerlo, Sebastian Thrun, Daphne Koller, and Ben Wegbreit. FastSLAM: A factored solution to the simultaneous localization and mapping problem. In *AAAI*, 2002.

[62] Michael Montemerlo, Sebastian Thrun, Daphne Koller, and Ben Wegbreit. FastSLAM 2.0 : An improved particle filtering algorithm for simultaneous localization and mapping that provably converges. In *Proc. IJCAI*, 2003.

[63] Simon J Julier and Jeffrey K Uhlmann. A counter example to the theory of simultaneous localization and map building. In *Proc. ICRA*. IEEE, 2001.

[64] Feng Lu and Evangelos Milios. Globally consistent range scan alignment for environment mapping. *Autonomous Robots*, 1997.

[65] Sebastian Thrun and Michael Montemerlo. The graph slam algorithm with applications to large-scale mapping of urban structures. *IJRR*, 2006.

[66] Kurt Konolige, Giorgio Grisetti, Rainer Kümmerle, Wolfram Burgard, Benson Limketkai, and Regis Vincent. Efficient sparse pose adjustment for 2d mapping. In *Proc. IROS*, 2010.

[67] Frank Dellaert and Michael Kaess. Square root sam: Simultaneous localization and mapping via square root information smoothing. *IJRR*, 2006.

[68] H Christopher Longuet-Higgins. A computer algorithm for reconstructing a scene from two projections. *Nature*, 1981.

[69] Christopher G Harris and JM Pike. 3d positional integration from image sequences. *Image and Vision Computing*, 1988.

[70] Hans P Moravec. Obstacle avoidance and navigation in the real world by a seeing robot rover. Technical report, Stanford University, 1980.

[71] Larry Matthies and STEVENA Shafer. Error modeling in stereo navigation. *Journal on Robotics and Automation*, 1987.

[72] Larry Henry Matthies. Dynamic stereo vision. Technical report, 1989.

[73] Chris Harris and Mike Stephens. A combined corner and edge detector. In *Alvey Vision Conference*, 1988.

[74] Jianbo Shi and Carlo Tomasi. Good features to track. In *Proc. CVPR*, 1994.

[75] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In *Proc. ECCV*, 2006.

[76] Bruce D Lucas, Takeo Kanade, et al. An iterative image registration technique with an application to stereo vision. In *IJCAI*, 1981.

[77] Carlo Tomasi and Takeo Kanade. Detection and tracking of point features. Technical report, Carnegie Mellon University, 1991.

[78] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 1981.

[79] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Proc. CVPR*. IEEE, 2005.

[80] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. Brief: Binary robust independent elementary features. In *Proc. ECCV*, 2010.

[81] Ramin Zabih and John Woodfill. Non-parametric local transforms for computing visual correspondence. In *Proc. ECCV*, 1994.

[82] David Nistér. An efficient solution to the five-point relative pose problem. *Trans. PAMI*, 2004.

[83] David Nistér, Oleg Naroditsky, and James Bergen. Visual odometry. In *Proc. CVPR*, 2004.

[84] Padmanabhan Anandan. A computational framework and an algorithm for the measurement of visual motion. *IJCV*, 1989.

[85] James R Bergen, Patrick Anandan, Keith J Hanna, and Rajesh Hingorani. Hierarchical model-based motion estimation. In *European conference on computer vision*, 1992.

[86] Frank R Hampel, Elvezio M Ronchetti, Peter J Rousseeuw, and Werner A Stahel. *Robust statistics: the approach based on influence functions.* 2011.

[87] Guoying Li. Robust regression. *Exploring data tables, trends, and shapes*, 1985.

[88] Alireza Bab-Hadiashar and David Suter. Robust optic flow computation. *IJCV*, 1998.

[89] Michael J Black and Paul Anandan. The robust estimation of multiple motions: Parametric and piecewise-smooth flow fields. *Computer Vision and Image Understanding*, 1996.

[90] Ee Ping Ong and Michael Spann. Robust optical flow computation based on least-median-of-squares regression. *IJCV*, 1999.

[91] Simon Baker and Iain Matthews. Lucas-kanade 20 years on: A unifying framework. *IJCV*, 2004.

[92] Henry Harlyn Baker. Depth from edge and intensity based stereo. Technical report, DTIC Document, 1982.

[93] Stuart Geman and Donald Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *Trans. PAMI*, 1984.

[94] Robert T Collins. A space-sweep approach to true multi-image matching. In *Proc. CVPR*, 1996.

[95] Daniel Scharstein and Richard Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *IJCV*, 2002.

[96] Steven M Seitz, Brian Curless, James Diebel, Daniel Scharstein, and Richard Szeliski. A comparison and evaluation of multi-view stereo reconstruction algorithms. In *Proc. CVPR*, 2006.

[97] Richard Szeliski. *Computer vision: algorithms and applications.* Springer Science & Business Media, 2010.

[98] Richard Szeliski. A multi-view approach to motion and stereo. In *Proc. CVPR*, 1999.

[99] Sing Bing Kang, Richard Szeliski, and Jinxiang Chai. Handling occlusions in dense multi-view stereo. In *Proc. CVPR*, 2001.

[100] Pau Gargallo and Peter Sturm. Bayesian 3d modeling from images using multiple depth maps. In *Proc. CVPR*, 2005.

[101] Steven M Seitz and Charles R Dyer. Photorealistic scene reconstruction by voxel coloring. *IJCV*, 1999.

[102] Kiriakos N Kutulakos and Steven M Seitz. A theory of shape by space carving. *IJCV*, 2000.

[103] Adrian Broadhurst, Tom W Drummond, and Roberto Cipolla. A probabilistic framework for space carving. In *Proc. ICCV*, 2001.

[104] W Bruce Culbertson, Thomas Malzbender, and Greg Slabaugh. Generalized voxel coloring. In *International Workshop on Vision Algorithms*, 1999.

[105] Sébastien Roy and Ingemar J Cox. A maximum-flow formulation of the n-camera stereo correspondence problem. In *Proc. ICCV*, 1998.

[106] Olivier Faugeras and Renaud Keriven. *Variational Principles, Surface Evolution, PDE's, Level Set Methods and the Stereo problem.* IEEE, 2002.

[107] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In *Proc. on Computer Graphics and Interactive Techniques*, 1996.

[108] Pascal Fua and Yvan G Leclerc. Object-centered surface reconstruction: Combining multi-image stereo and shading. *IJCV*, 1995.

[109] Demetri Terzopoulos. Regularization of inverse visual problems involving discontinuities. *Trans. PAMI*, 1986.

[110] Demetri Terzopoulos and Dimitri Metaxas. Dynamic 3d models with local and global deformations: Deformable superquadrics. In *Proc. ICCV*, 1990.

[111] Michael J Black and Anand Rangarajan. On the unification of line processes, outlier rejection, and robust statistics with applications in early vision. *IJCV*, 1996.

[112] Daniel Scharstein and Richard Szeliski. Stereo matching with nonlinear diffusion. *IJCV*, 1998.

[113] Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast approximate energy minimization via graph cuts. *Trans. PAMI*, 2001.

[114] Hiroshi Ishikawa and Davi Geiger. Occlusions, discontinuities, and epipolar lines in stereo. In *Proc. ECCV*, 1998.

[115] Olga Veksler. *Efficient graph-based energy minimization methods in computer vision.* PhD thesis, Cornell University, 1999.

[116] Vladimir Kolmogorov and Ramin Zabih. Computing visual correspondence with occlusions using graph cuts. In *Proc. ICCV*, 2001.

[117] Yuichi Ohta and Takeo Kanade. Stereo by intra-and inter-scanline search using dynamic programming. *Trans. PAMI*, 1985.

[118] Peter N Belhumeur and David Mumford. A bayesian treatment of the stereo correspondence problem using half-occluded regions. In *Proc. CVPR*, 1992.

[119] Peter N Belhumeur. A bayesian approach to binocular steropsis. *IJCV*, 1996.

[120] Davi Geiger, Bruce Ladendorf, and Alan Yuille. Occlusions and binocular stereo. *IJCV*, 1995.

[121] Leonid I Rudin, Stanley Osher, and Emad Fatemi. Nonlinear total variation based noise removal algorithms. *Physica D: Nonlinear Phenomena*, 1992.

[122] David Mumford and Jayant Shah. Optimal approximations by piecewise smooth functions and associated variational problems. *Communications on Pure and Applied Mathematics*, 1989.

[123] Thomas Pock, Daniel Cremers, Horst Bischof, and Antonin Chambolle. An algorithm for minimizing the mumford-shah functional. In *Proc. ICCV*, 2009.

[124] Yu Nesterov. Smooth minimization of non-smooth functions. *Mathematical Programming*, 2005.

[125] Ernie Esser, Xiaoqun Zhang, and Tony F Chan. A general framework for a class of first order primal-dual algorithms for convex optimization in imaging science. *Journal on Imaging Sciences*, 2010.

[126] Antonin Chambolle and Thomas Pock. A first-order primal-dual algorithm for convex problems with applications to imaging. *Journal of Mathematical Imaging and Vision*, 2011.

[127] Ethan Eade and Tom Drummond. Scalable monocular slam. In *Proc. CVPR*, 2006.

[128] Ethan Eade and Tom Drummond. Edge landmarks in monocular slam. In *Proc. BMVC*, 2006.

[129] Pedro Piniés, Lina Maria Paz, and Paul Newman. Dense mono reconstruction: Living with the pain of the plain plane. In *Proc. ICRA*, 2015.

[130] Michael Bleyer, Christoph Rhemann, and Carsten Rother. Patchmatch stereo-stereo matching with slanted support windows. In *Proc. BMVC*, 2011.

[131] Raphael A. Finkel and Jon Louis Bentley. Quad Trees: A data structure for retrieval on composite keys. *Acta Informatica*, 1974.

[132] Tiago Gonçalves, Andrew Comport, et al. Real-time direct tracking of color images in the presence of illumination variation. In *Proc. ICRA*, 2011.

[133] Mark De Berg, Marc Van Kreveld, Mark Overmars, and Otfried Cheong Schwarzkopf. *Computational Geometry.* Springer, 2000.

[134] Stephen Boyd and Lieven Vandenberghe. *Convex optimization.* Cambridge University Press, 2004.

[135] Neal Parikh, Stephen P Boyd, et al. Proximal algorithms. *Foundations and Trends in Optimization*, 2014.

[136] Arren Glover, William Maddern, Michael Warren, Stephanie Reid, Michael Milford, and Gordon Wyeth. OpenFABMAP: An open source toolbox for appearance-based loop closure detection. In *Proc. ICRA*, 2012.

[137] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A benchmark for the evaluation of RGB-D slam systems. In *Proc. IROS*, 2012.

[138] Sudeep Pillai, Srikumar Ramalingam, and John J. Leonard. High-Performance and Tunable Stereo Reconstruction. In *Proc. ICRA*, 2016.

[139] Gregory D Hager and Peter N Belhumeur. Efficient region tracking with parametric models of geometry and illumination. *Trans. PAMI*, 1998.

[140] Simon Baker and Iain Matthews. Equivalence and efficiency of image alignment algorithms. In *Proc. CVPR*, 2001.

[141] Jonathan Richard Shewchuk. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In *Applied Computational Geometry: Towards Geometric Engineering*. 1996. From the First ACM Workshop on Applied Computational Geometry.

[142] Jonathan Richard Shewchuk. Delaunay refinement algorithms for triangular mesh generation. *Computational Geometry*, 2002.