Leveraging Prior Information for Real-time Monocular Simultaneous Localization and Mapping

by

W. Nicholas Greene

Submitted to the Department of Aeronautics and Astronautics
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
February 2021

© Massachusetts Institute of Technology 2021. All rights reserved.

Author .....
Department of Aeronautics and Astronautics
January 25, 2021

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

Certified by .....
John Leonard
Professor of Mechanical and Ocean Engineering

Certified by .....
Luca Carlone
Assistant Professor of Aeronautics and Astronautics

Certified by .....
Andrew Davison
Professor of Robot Vision, Imperial College London

Accepted by .....
Zoltan Spakovszky
Chair, Graduate Program Committee

Leveraging Prior Information for Real-time Monocular Simultaneous Localization and Mapping

by

W. Nicholas Greene

Submitted to the Department of Aeronautics and Astronautics
on January 25, 2021, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

Monocular cameras are powerful sensors for a variety of computer vision tasks since they are small, inexpensive, and provide dense perceptual information about the surrounding environment. Efficiently estimating the pose of a moving monocular camera and the 3D structure of the observed scene from the images alone is a fundamental problem in computer vision commonly referred to as monocular simultaneous localization and mapping (SLAM). Given the importance of egomotion estimation and environmental mapping to many applications in robotics and augmented reality, the last twenty years have seen dramatic advances in the state of the art in monocular SLAM. Despite the rapid progress, however, several limitations remain that prevent monocular SLAM systems from transitioning out of the research laboratory and into large, uncontrolled environments on small, resource-constrained computing platforms.

This thesis presents research that attempts to address existing problems in monocular SLAM by leveraging different sources of prior information along with targeted applications of machine learning. First, we exploit the piecewise planar structure common in many environments in order to represent the scene using compact triangular meshes that will allow for faster reconstruction and regularization. Second, we leverage the semantic information encoded in large datasets of images to constrain the unobservable scale of motion of the monocular solution to the true, metric scale without additional sensors. Lastly, we compensate for known viewpoint changes when associating pixels between images in order to allow for robust, learning-based depth estimation across disparate views.

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

Thesis Committee Member: John Leonard
Title: Professor of Mechanical and Ocean Engineering

Thesis Committee Member: Luca Carlone
Title: Assistant Professor of Aeronautics and Astronautics

Thesis Committee Member: Andrew Davison
Title: Professor of Robot Vision, Imperial College London

Acknowledgments

Attending MIT has been a dream of mine for as long as I can remember. As a kid, I knew two things: (1) I wanted to be an engineer and (2) I wanted to go to MIT. Now at the end of my six and a half years here, I sometimes find it hard to believe it actually happened. I feel so incredibly lucky and privileged to have been able to study at this amazing institution. While I'm very proud of the research I've done during my time here, it would not have been possible without the help and support of a great many people.

First and foremost, I would like to thank Nick Roy for giving me the opportunity to join his group and for all the invaluable advice and guidance, especially before I became a grad student. It's fair to say my life might have turned out drastically different if Nick hadn't taken the time to respond to my cold email asking if I could somehow get involved in his group's research after hours. Nick both holds his students to an incredibly high bar and genuinely cares about their well-being beyond research. Having him as an adviser, I've always felt I've had both the freedom to pursue work that interested me and support when I wasn't sure what to do next.

Second, thank you to all the members of the Robust Robotics Group, CSAIL, and the broader MIT community for making research fun. From late night coding sessions, to flight tests in the field, to random boba breaks and game nights, I can't imagine a better group of people to spend so much time with and learn from.

Lastly, thank you to my thesis committee John Leonard, Luca Carlone, and Andrew Davison and thesis readers Valentin Peretroukhin and Gregory Stein for asking challenging questions and pushing my work to be better. Thank you to Sophia Hasenfus, Bryt Bradley, and Beth Marois for keeping my graduation on track. Thank you to all my teachers, from pre-school to college, who nurtured and encouraged me over the years. A special 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. And finally, thank you to my family for all your support and encouragement.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Cameras are powerful sensors for a variety of tasks since they are small, inexpensive, and provide dense perceptual information about the surrounding environment. Jointly estimating the position and orientation (or *pose*) of one or more moving cameras in real time, along with the 3D structure of the observed scene, is a fundamental problem in computer vision commonly referred to as visual simultaneous localization and mapping (SLAM). Given the importance of pose estimation and mapping to many different applications, the last twenty years has seen dramatic advances in the state of the art in visual SLAM, 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. As a result, many emerging technologies that are enabled by visual SLAM are now beginning to transition out of the laboratory and into the wild.

Self-driving cars are one of the most prominent examples of this trend. Vehicles from Google [150], Apple [94], Uber [115], and others are instrumented with multiple cameras that when combined offer improved situational awareness around the vehicle. These cameras are used to estimate the vehicle's egomotion, localize the vehicle on known roads and highways, detect lane markers and signs, track other cars and pedestrians, and map obstacles that must be avoided. While other sensing modalities are often used in conjunction with cameras, image data is of fundamental importance and may even be *sufficient* for full autonomy. Tesla's self-driving vehicle,

for example, relies primarily on cameras for its Autopilot technology [214]. While autonomous cars are large enough to carry a multitude of sensors and ample computation to drive visual SLAM systems, the consequences of failure when operating a vehicle on public roads means that these SLAM solutions must run in real-time with low latency and must be extremely robust to errors, sensor noise, and adverse environmental conditions that stress even state-of-the-art approaches.

Another class of robotic platforms that has received a sizable amount of interest over time has been micro-aerial vehicles (MAVs or *drones*) such as quadrotors, hexrotors, and fixed-wing aircraft, which typically weigh between 0.5–5 kg and measure between 0.25–1 m 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 aerial photography [2], search and rescue [206], package delivery [134, 46], planetary exploration [7, 119], and intelligence, surveillance, and reconnaissance (ISR) [195]. Unlike self-driving cars, autonomous MAVs are limited in terms of their size, weight, and power (SWaP), which simultaneously makes visual SLAM techniques extremely useful and extremely challenging. A fast-flying MAV must process camera data quickly enough to stabilize the attitude of the vehicle, register its pose with respect to obstacles, and further map those obstacles so they can be avoided at speed.

Augmented reality (AR) and virtual reality (VR) are other arenas where visual SLAM techniques have proven to be of critical importance. In both AR and VR, a person's view of a scene is replaced by a digital version generated from cameras at a similar pose to the person's eyes, for example via a handheld phone or head-mounted glasses. The images presented to the user may have virtual assets added to the scene, such as information overlays, game characters, or heads-up-displays (AR), or may have the scene replaced entirely with a virtual one (VR). The user, however, should be able to interact with the scene (both real and virtual portions) naturally, without breaking immersion. While VR has straightforward applications in the video game industry [85], the potential for AR systems to improve navigation [121], assembly [120], and telepresence [184] continues to grow. All of these tasks rely on visual SLAM to

track the pose of the mounted cameras and estimate the scene geometry so that it can be augmented or replaced. Similar to autonomous MAVs, AR and VR have strict SWaP constraints, especially when the user must carry the sensors and compute.

Monocular SLAM is a variant of visual SLAM where only a *single* camera is utilized to drive the SLAM process. This arrangement has a number of unique advantages compared to multi-camera equivalents, including lower SWaP, simpler hardware, and easier calibration. Monocular SLAM can also be considered the most fundamental form of visual SLAM, with most multi-camera variations designed to build on top of the monocular SLAM foundation. In addition, monocular imagery is much more common in the wild due to the nearly ubiquitous usage in the consumer space.

Early attempts at real-time monocular SLAM solutions leveraged the Extended Kalman Filter (EKF) [105, 104, 208] and were able estimate the values of tens of camera poses and landmarks by aggressively marginalizing out past states [32, 41]. As it became more understood that optimizing over the entire camera trajectory could actually lead to faster and more accurate results by exploiting the sparsity of the underlying least squares problem [44, 103, 102], offline techniques from the Bundle Adjustment and Structure-from-Motion (SfM) communities [222, 45, 201, 4] were adapted to the real-time case [110, 145, 65]. The years since have witnessed increasing focus on speed [158, 65], map density [58, 56, 55, 153, 80], and fusion with inertial sensors for metrically-scaled outputs [143, 124, 64, 202, 173].

Despite the rapid progress, however, several limitations remain that prevent monocular SLAM systems from transitioning out of the research laboratory and into large, uncontrolled environments on small, resource-constrained computing platforms [25, 42]. For instance, mapping an environment with sufficient *density* to adequately describe obstacles, such as by recovering a *depth* value for every pixel in an input image, is an extremely computationally intensive operation requiring sizable computation when only passive monocular imagery (and not, for example, active stereo or LIDAR measurements) is available. Producing poses and maps that are *metrically-scaled* so that the SLAM solution may be used by a robot or human cannot be accomplished with monocular imagery without additional sensors, which must be

calibrated, time-synchronized, and expertly-tuned. Finally, the appearance of objects changes drastically when observed from the disparate viewpoints encountered in monocular video, which makes the process of associating pixels from different images — a necessary component of monocular SLAM — all the more intractable.

In this thesis, we argue that there exist untapped sources of prior information that can be leveraged to address these challenges. Indeed, current approaches to monocular SLAM and monocular depth estimation often do not utilize all the information available to them to produce solutions. In particular, these algorithms do not operate in completely arbitrary environments. The world — from indoor and outdoor environments to urban and natural scenes — exhibits a number of structural regularities that are not always taken into account when building solutions. For instance, some geometric primitives are incredibly common (e.g., smooth surfaces), while others are incredibly rare (e.g., fractals). The geometry, color, and texture of a scene are all highly correlated with each other: roads are usually flat and grey, stop signs are usually red, trees are usually green, and so on. Semantically meaningful objects like cars, people, and doors have strong priors on their size, shape, and orientation. Furthermore, the appearance of these objects obey certain, well-defined rules in how they manifest in images under different views and lighting. When this additional structure is considered, we contend that there is considerable opportunity to improve monocular SLAM and monocular depth estimation. This thesis will identify and exploit specific latent sources of prior information that, coupled machine learning techniques, resolve the inadequacies alluded to above and allow for significant improvements in accuracy and speed above the state of the art. In the rest of this chapter, we will briefly describe the monocular SLAM problem in Section 1.1 before outlining its limitations and our proposed solutions in Section 1.2, Section 1.3, and Section 1.4.

1.1 Monocular SLAM

Any state estimation and perception system used on a mobile platform typically needs to provide two principal quantities to be of use: the *pose* (i.e., position and

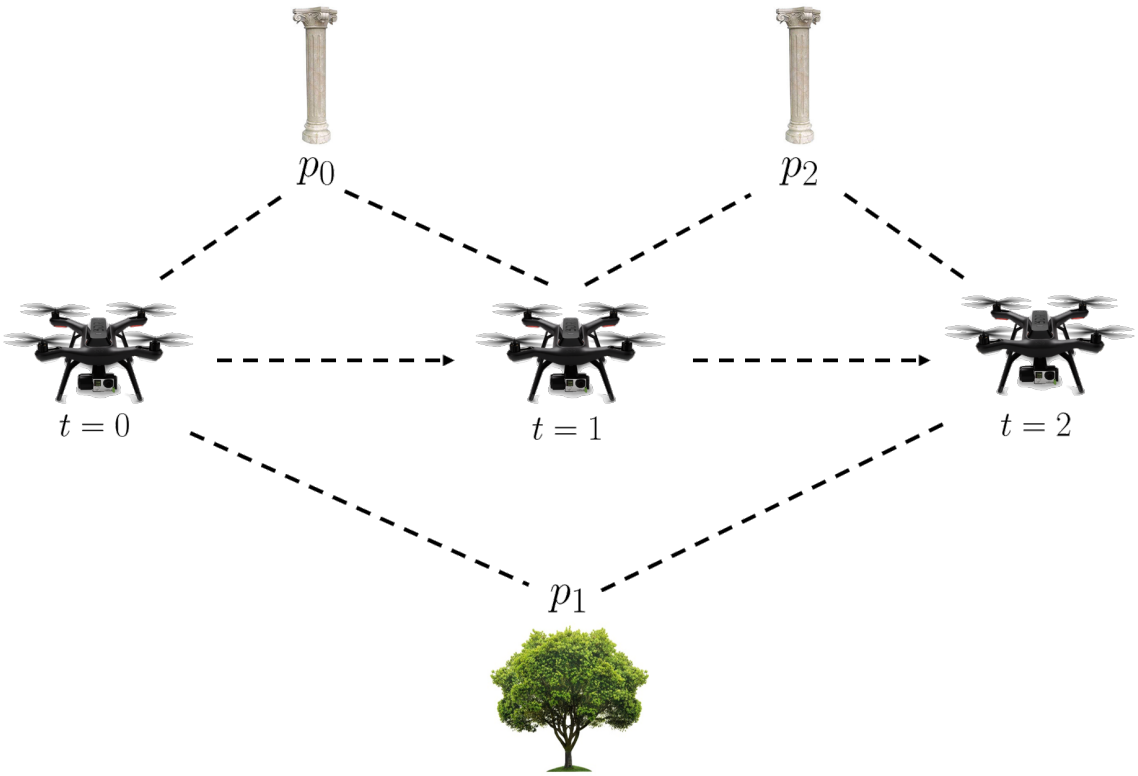


Figure 1-1: Simultaneous localization and mapping (SLAM) is a classical problem in robotics where a robot equipped with an imperfect sensor (e.g., a quadrotor with a camera) moves through an environment and observes various landmarks (e.g., pillars and trees) over time. The robot must fuse the noisy measurements of the landmarks to estimate its pose, while simultaneously estimating where the landmarks are in order to build a map. 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.

orientation) of the platform and a *map* of the environment that encodes the geometry of the scene (including when the scene changes over time). Both quantities are usually needed in order to, for example, plan motions through the environment that do not collide with obstacles, provide feedback to a low-level control loop to stabilize the platform, or overlay virtual assets on the scene in a convincing manner.

Simultaneous localization and mapping (SLAM) is the traditional formulation of this problem where a mobile platform (e.g., a robot) with imperfect sensors traverses an unknown environment with a set of landmarks. As the platform 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*)

(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 [131], structured light RGB-D cameras [139], passive stereo cameras [170], and passive monocular cameras [171]. Each sensing modality has strengths and weaknesses, but this thesis 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.

concurrently (see Figure 1-1). Note that there is a "chicken and egg" aspect to the SLAM problem: estimating the platform's pose requires a map, and estimating the map requires the platform's pose. Many strategies have been proposed to overcome this conundrum so that both pose and map can be computed simultaneously in real time.

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 experienced by the platform and a range sensor such as an ultrasonic rangefinder or scanning LIDAR [131]. Scanning LIDARS, both the 2D and 3D varieties, drove much of the research in the early days of SLAM, but are problematic for small, agile robots such as MAVs and handheld devices 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 when the platform pitches or rolls frequently).

Cameras, on the other hand, are attractive sensors for small mobile platforms as they provide high-resolution environment information (color, shape, texture, etc.), while being lightweight, low-power, and inexpensive. They also provide important non-geometric, *semantic* information such as the presence of objects of interest like people, cars, or signs. Furthermore, active or multi-camera variants such as structured-light and stereo cameras can perceive depth information directly, which is valuable for SLAM. This thesis, 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 [139] have a limited detection range (typically ~ 5m) and are essentially inoperable in sunlight due to electromagnetic interference. Passive stereo cameras [170] work outdoors, but their detection range is limited by the baseline between the two cameras, which is necessarily small on our platforms of interest, and require accurate calibration to be effective. Passive monocular cameras [171], however, are robust to sunlight and can detect distant structure given sufficient translational motion between frames. Spurred by the rapid adoption of consumer smartphones, they are also smaller, lighter, and cheaper than the aforementioned alternatives and nearly ubiquitous. We will refer to the SLAM problem where a monocular camera is the primary sensor as *monocular SLAM*.

There are many variants of monocular SLAM, each with strengths and weaknesses depending on the task of interest. For example, in many scenarios, it may be advantageous to factor the monocular SLAM problem into separate components: one for estimating the camera poses and one for recovering a *dense* map of the scene given those poses. We will refer to this second component as *monocular depth estimation*, as we typically wish to represent the scene using a *depthmap*: an image where a depth value is provided for every pixel. Monocular depth estimation has its roots in the stereo and multi-view stereo (MVS) literature, where dense geometry is estimated from a set on images taken from known poses, typically in an offline setting [35, 77, 68, 91]. Since the poses of the cameras are assumed known (or computed in a separate process), monocular depth estimation is fundamentally a *data*

association problem. If one can associate a pixel in one camera image with a pixel in another image taken from a different viewpoint, one can use the knowledge of the poses to *triangulate* the depth of the point. In traditional stereo, two cameras are rigidly mounted a short distance apart, whereas in multi-view stereo, the cameras can be arbitrarily placed or produced by a single monocular camera moving through the scene.

Although monocular SLAM and monocular depth estimation have received substantial interest in the robotics and computer vision literature, there remain key deficiencies that limit these algorithms in practice, especially on small, resource-constrained platforms [25, 42]. This thesis will focus on three such limitations: (1) estimating dense geometry is computationally expensive (Section 1.2), (2) metric scale is unobservable when only monocular imagery is used (Section 1.3), and (3) pixel matching is significantly affected by viewpoint changes (Section 1.4). We will then present solutions to these problems that are enabled by exploiting different sources of prior information along with targeted applications of machine learning.

1.2 Dense Depth Estimation is Expensive

Dense monocular depth estimation as described in the previous section is a computationally expensive process. Ideally, we wish to estimate a depth value for *every* pixel in a particular reference image given a set of neighboring images. Even for low resolution images, this is a tall order. For instance, a standard VGA image of size 480×640 pixels requires 307,200 separate depth estimates. Estimating depth for higher resolution images is even more daunting. A 1080p image of size 1080×1920 pixels requires over 2 million depth estimates, while a 4K image of size 2160×3840 pixels requires over 8 million depth estimates. Even with powerful GPU acceleration, producing depthmaps of at these scales is computationally intensive. Small, resource-constrained platforms such as MAVs and mobile phones have even less computation available. In addition, generating so many depth estimates increases the likelihood of noise that can corrupt the geometry. *Spatial regularization* must therefore be applied

to remove wildly inconsistent depth estimates and denoise the depths. In many cases, this spatial regularization step can be more involved than producing the depths in the first place.

We note, however, that the *geometry* of most environments of interest is usually much simpler than high-definition images would otherwise suggest. Consider the geometry of a city composed of a series of buildings. Each building typically has 5 exposed sides. Each side is mostly flat. Each building usually rests on a flat groundplane. To first order, therefore, the geometry of a city can be well-described as a finite set of planes or *piecewise planar*. It is unlikely that all 8 million depth estimates in a 4K image – or even all 300 thousand depth estimates in a VGA image – are necessary to encode the geometry for this type of scene. Indeed, if each plane is composed of a set of triangles (as is common in computer graphics), and each triangle requires only 3 different depth values (one for each triangle vertex), it is likely that far fewer depth values would be needed to reconstruct the scene than the total number of pixels in each image. More complicated environments, like forests, parks, or indoor scenes, may require more planes and triangles than a city environment would need to adequately capture the geometry, but a piecewise planar representation is a very good approximation, especially for many important tasks like obstacle avoidance and AR.

We therefore propose leveraging this prior information that many scenes of interest can be represented using planes to improve the dense monocular depth estimation process. We contend that current dense depth estimation methods, which estimate a depth value for *every* pixel in the input image, drastically *oversample* scenes relative to their inherent geometric complexity, which increases both the resources required to generate a solution and the regularization needed to produce a plausible reconstruction. By more intelligently sampling the scene using triangular meshes such that planar surfaces can be represented using only a small number of depth estimates, we can not only more compactly represent the scene geometry, but can take advantage of the graph structure of the mesh to apply more sophisticated regularization techniques with lower computational cost than existing approaches.

1.3 Metric Scale is Unobservable

The second limitation of monocular SLAM that we aim to address relates to producing *metrically-scaled* SLAM solutions. By metrically-scaled, we mean that the units used to express the camera poses and scene geometry can be converted to *physical* units that correspond to the real scene. Some sensing modalities, such as LIDAR or structured-light, may directly observe metric quantities, but monocular cameras do not. Monocular cameras are fundamentally *bearing sensors* that only observe the *angle* of incident light on the sensor plane, not the metric range to the structure from which the light was reflected. This is why it is impossible to tell the difference between a small object close to the camera, and a large object far away from the camera. Imagine, for example, a real car and a to-scale model of the same car. Controlling for other objects in the scene and lighting, the images produced by placing the car model close to the camera and the real car far away from the camera will be identical.

Algorithms that rely on these monocular images, therefore, have no way of directly observing the metric scale of the environment under observation. In particular, monocular SLAM methods can only estimate the camera's trajectory and environment map up to an arbitrary scale factor. Additional information must be exploited to resolve the metric scale of the solution. While assuming the camera lies at a fixed altitude is reasonable for some applications (e.g., when the camera is mounted on a vehicle), the current most popular technique is to fuse the camera with an inertial measurement unit (IMU), which measures linear acceleration and angular velocities. Since the linear accelerations are metric quantities, these visual-inertial SLAM approaches can recover fully metric solutions. However, significant flaws remain. First, since the metric scale information comes directly from the linear accelerometers, the platform must undergo high-acceleration maneuvers to expose the scale, which can be dangerous for some applications like MAV navigation. Second, any errors in the acceleration signal are doubly-integrated into the position of the camera, meaning the SLAM solution can quickly diverge from the true pose. Lastly, visual-inertial approaches are difficult to implement, calibrate, and require expert tuning to work

well.

Instead, we will leverage the prior *semantic* information encoded in large datasets of images to constrain the unobservable scale of motion of the monocular solution to the true, metric scale without additional sensors or aggressive maneuvers. We first implement a conventional monocular SLAM pipeline over a small sliding window of keyframes to generate accurate, but unscaled camera poses. We then train a small neural network to regress metric depth from the monocular images given calibrated stereo frames as training data. Unlike existing learning-based approaches [75, 76, 164, 84, 169], our technique leverages the insight that when used to estimate scale, these learned predictions need only be coarse in image space. This allows us to shrink our network to the point that performing inference in real time on a standard CPU becomes computationally tractable. We then fuse the unscaled keyframe poses and the metric depth measurements from the network into a global, scale factor graph that, when optimized, produces an accurate, fully-metric SLAM solution.

1.4 Pixel Matching is Affected by View

The final problem we will consider in this thesis concerns a different aspect of dense monocular depth estimation than the one described in Section 1.2. Recall that monocular depth estimation is fundamentally a problem of data association: assuming the camera poses for the input images are known, correctly associating two pixels in two different images allows the corresponding 3D point to be triangulated. Correctly associating these pixels is a challenging task, however, primarily because a scene can appear drastically different when observed from different viewpoints. Individual objects and structure can be scaled, rotated, or even occluded between views. Relating individual pixels after these transformations is not straightforward. Traditional stereo depth estimation methods try to control the ill-effects of this phenomenon by rigidly mounting cameras such that the respective sensor planes are axis-aligned and parallel, effectively removing any scale or rotation between the images. In monocular depth estimation and multi-view stereo, however, the input cameras can be arbitrarily placed

and pixel correspondences must be established across these disparate viewpoints.

One way to mitigate this correspondence problem is to *learn* alternative representations of the image data that aggregate global context information such that each pixel is more distinctive and therefore easier to match across frames. Learning-based depth estimation and MVS solutions use stacks of convolutional neural networks (CNNs) to extract features from each image, match them across different views, and triangulate depths for each pixel. They often then smooth or otherwise refine the resulting depthmaps. While impressive results have been presented using this paradigm, these learning-based MVS methods do not properly leverage all information available to improve the feature matching process. In particular, state-of-the-art methods from Im et al. [96], Wang and Shen [227], Huang et al. [95], and Yao et al. [237] all extract features from each input image without considering the poses of the cameras that produced them (which are assumed known) and without accounting for the warping that occurs to the images due to the scene being observed from different viewpoints. This means that the respective neural networks must learn feature representations that exhibit scale, rotation, and perspective invariance, which can be difficult to achieve given limited training data.

We instead propose a novel feature extraction network that properly accounts for the known viewpoint changes, thereby reducing the burden on the network to learn scale and rotation invariant features. By compensating for the known viewpoint changes during the feature extraction process itself, the network can learn features that are *specific* to the given placement of the cameras and appropriately modified *by construction*, which improves feature matching. Doing this compensation naively, however, can be computationally demanding. We therefore, improve on the strategy in two key ways. First, we compute our features in an *incremental* fashion, greatly reducing the number of convolutional layers than must be evaluated during extraction. Second, we leverage the refinement techniques described in Khamis et al [108] and generalize them from the rectified stereo domain to the monocular depth estimation setting, which allows us to compute features that are significantly downsampled from the input image resolution, providing significant speed improvements. The combina-

tion of these two techniques leads to increased depth estimation performance across disparate views, while being dramatically faster than the state of the art.

1.5 Contributions

We have now briefly outlined three key limitations that prevent monocular SLAM and monocular depth estimation methods from transitioning out of the research laboratory and into large, uncontrolled environments on small, resource-constrained platforms. Chapter 2, will provide a more in-depth overview of the monocular SLAM field, while the remaining chapters in this thesis will describe aforementioned problems in greater detail and present solutions that are each enabled by novel applications of prior information and targeted uses of machine learning. We believe these improvements take monocular SLAM and monocular depth estimation one step further towards wide acceptance in real-world applications.

In summary, this thesis makes the following contributions:

- A method for dense monocular depth estimation that exploits the piecewise planar structure of many environments to greatly accelerate the depth estimation process (Chapter 3).
- A learning-based approach for estimating the metric inverse depth of point landmarks from monocular images (Chapter 4).
- A monocular SLAM formulation that leverages scale measurements from a trained CNN to estimate metrically-scaled egomotion and structure (Chapter 4).
- A novel MVS network that can efficiently recover depths from cameras undergoing unconstrained motion (Chapter 5).
- A generalization of the learning-based stereo architecture of Khamis et al [108] to the multi-view depth estimation setting (Chapter 5).
- A fast, incremental approach to feature extraction that compensates for the known camera geometry (Chapter 5).

1.6 Publications

Sections of Chapter 1 and Chapter 2 were originally presented in Greene [79]. The work presented in Chapter 3 was originally published in Greene and Roy [81]. The work presented in Chapter 4 was originally published in Greene and Roy [82]. Lastly, the work presented in Chapter 5 will appear in Greene and Roy [83].

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 feature tracking, to visual odometry. Section 2.4 presents full monocular SLAM solutions in detail, including sparse, dense, and semi-dense pipelines. Finally Section 2.5 describes the dense monocular depth estimation problem, where the camera poses are assumed known and the objective is to densely map the observed environment.

2.1 Probabilistic Formulation

Recall the scenario depicted in Figure 1-1. At each discrete timestep $i \in \mathbb{N}$, our platform 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. For simplicity, we will assume that the platform is a mobile robot, but note that it could just as easily be an autonomous car, smartphone, VR goggles, or any other mobile device. We will also assume, for the moment, that the sensor is a generic one that can observe landmarks, but will focus on the monocular camera case in Section 2.3 onward.

To begin, let us denote the robot 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 [203, 56, 80].

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, then the distribution of the pose history $p(\mathbf{X})$ can be factored according to $p(\mathbf{X}) = p(\mathbf{x}_0) \prod_{i=1}^{T} p(\mathbf{x}_i|\mathbf{x}_{i-1})$, where $p(\mathbf{x}_0)$ denotes the prior on the initial state and $p(\mathbf{x}_i|\mathbf{x}_{i-1})$ denotes the distribution of possible states $\mathbf{x}_i$ conditioned on the previous pose $\mathbf{x}_i$ that is governed by an appropriate physics-based process or motion model, possibly parameterized by a control input.

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 the world is static such that the landmarks do not move, although generalizations can be made for dynamic scenes. Over time, the sensor onboard 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 $\mathbf{Z}$ can be written as:

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

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 $p(\mathbf{X}, \mathbf{P}|\mathbf{Z})$ over $\mathbf{X}$ and $\mathbf{P}$, we obtain the the *maximum a posteriori* (MAP) estimate. For further information on the probabilistic foundations of SLAM see [217, 50, 9, 25].

2.1.1 SLAM as a Factor Graph

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* [116], 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 $p(\mathbf{X}, \mathbf{P}|\mathbf{Z})$ in Equation (2.1) again, we can see that it is a function of three sets of variables $p(\mathbf{X}, \mathbf{P}|\mathbf{Z}) = f(\mathbf{X}, \mathbf{P}, \mathbf{Z})$, and factorizes in a particular fashion given our conditional independence assumptions. We can therefore encode it as a factor graph with variable nodes $\mathcal{V} = (\mathbf{X}, \mathbf{P})$ and factor nodes $\mathcal{F}$ composed of the prior on the initial pose $p(\mathbf{x}_0)$, the motion model terms $p(\mathbf{x}_i|\mathbf{x}_{i-1})$, and the measurement terms $p(\mathbf{z}_k|\mathbf{x}_{i_k}, \mathbf{p}_{j_k})$. In our running robot example, we can incorporate motion factors $\mathbf{u}_i$ to constrain the relative motion between particular pairs of pose variables, for example from odometry or a physical motion model given knowledge of the control variables, while the measurement factors constrain pairs of pose variables and landmark variables (see Figure 2-1).

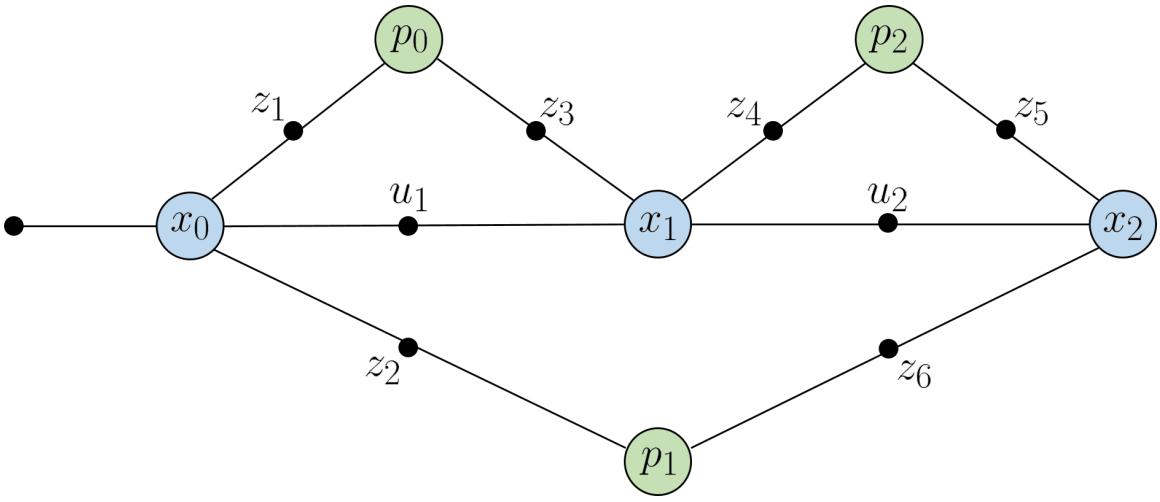


Figure 2-1: The posterior distribution of the SLAM problem as described in Equation (2.1) can be represented succinctly as a factor graph. The variable nodes $\mathbf{x}_0$, $\mathbf{x}_1$, $\mathbf{x}_2$ denote the robot state over time, while $\mathbf{p}_0$, $\mathbf{p}_1$, $\mathbf{p}_2$ signify the landmarks in the map. The factor nodes $\mathbf{u}_1$, $\mathbf{u}_2$ show relative motion 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.

In addition, as mentioned previously, we assume that the association between poses, landmarks, and observations are known, which is represented in the graph by the edges between variables nodes and factor nodes. Furthermore, note that both types of factors are not required to produce a valid solution and often only a single type is used in practice. For instance, traditional SfM and bundle adjustment pipelines do not use relative motion factors, while pose graph SLAM methods such as LSD-SLAM [56] do not use landmark factors by encoding depth information into the motion factors themselves.

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 [116, 111]. 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 many cases in real time [117, 44, 103, 102, 177].

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 *optimizing* the function represented by 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 thesis). 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 were 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) [226], particle filter [49], sparse extended information filter (SEIF) [219, 225], and delayed-state filter [60, 61]. Posing SLAM as a state estimation problem was natural initially, although it is now considered to be a special case of the more general graphical interpretation.

Early filter-based SLAM solutions used EKFs to compute the posterior distribution of the map $\mathbf{P}$ and the most recent pose $\mathbf{x}_T$ given the measurement history $\mathbf{Z} = (z_1, \ldots, z_T)$. These EKF-SLAM methods assumed a small number of point landmarks or beacons with known data association [122, 123, 200, 47]. 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:

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

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

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

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

where $\mathbf{Z}_- = (\mathbf{z}_1, \ldots, \mathbf{z}_{T-1})$. 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 generally assumed:

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

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

When the prior $p(\mathbf{x}_0)$ is assumed to be Gaussian and the process and measurement models are differentiable, the Bayesian update equations can be computed in closed form (the well-known Kalman filter equations and nonlinear variants). Non-differentiable process and measurements models can also be supported using the UKF [226].

Particle filters [49] relax the Gaussian assumption of Kalman Filter-based methods and represent the state using a finite set of samples. These samples are propagated in time using the process model $f(\mathbf{x}_{i-1})$ and then weighted and resampled using the measurement model $h(\mathbf{x}_{i_i}, \mathbf{p}_{j_i})$. Particle filters were first applied for localization given a known map in Dellaert et al. [43], Fox et al. [66], and Thrun et al. [218]. One of the earliest particle filter approach to full SLAM is the FastSLAM algorithm of Montemerlo et al. [140, 141], which exploits *Rao-Blackwellization* to overcome the curse of dimensionality suffered by traditional particle filters. Typically, the num-

ber 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 conditioned on samples from the marginal of the remaining state variables.

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

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

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

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 computational 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 with each new measurement. 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 [101].

Smoothing-based approaches, on the other hand, sidestep these issues by solving for the entire pose trajectory $\mathbf{X}$ (or a spatially distributed subset of the trajectory) along with the map $\mathbf{P}$. Retaining all the variables in $\mathbf{X}$ may seem to substantially increase the computational complexity of the estimation, but 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 still be maintained, allowing for solutions to be computed efficiently, despite the increase in the number of poses.

Strasdat et al. [204, 205] directly analyze the strengths and weaknesses of filters versus smoothing-based methods in the context of monocular SLAM, with the filtering-based approach of Eade and Drummond [53] and the smoothing-based approach of Klein and Murray [110] as reference algorithms. Using a combination of simulated and real imagery, they compare the accuracy of each type of approach in relation to the computational cost required to generate a solution. Except when computational resources are extremely limited, smoothing-based solutions are judged to be preferred over filtering-based solutions.

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

$$\begin{aligned}
\arg\max_{\mathbf{X},\mathbf{P}} p(\mathbf{X}, \mathbf{P}|\mathbf{Z}) &= \arg\min_{\mathbf{X},\mathbf{P}} \; -\log p(\mathbf{X}, \mathbf{P}|\mathbf{Z}) && (2.10)\\
&= \arg\min_{\mathbf{X},\mathbf{P}} \left\{ ||\mathbf{x}_0 - \mu_0||^2_{\Sigma_0} + \sum_{i=1}^{T} ||\mathbf{x}_i - f(\mathbf{x}_{i-1})||^2_W \right.\\
&\qquad \left. + \sum_{k=1}^{K} ||\mathbf{z}_k - h(\mathbf{x}_{i_k}, \mathbf{p}_{i_k})||^2_V \right\}. && (2.11)
\end{aligned}$$

When optimized using Gauss-Newton or Levenberg-Marquardt [156], 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 [132, 220, 114, 44] and has a large crossover with Structure-from-Motion (SfM) and bundle adjustment [222, 45, 201, 189, 4, 129, 113]. 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 [103, 102, 177, 232].

More sophisticated techniques from the optimization literature, such as convex relaxation, Lagrangian duality, and semi-definite programming, are also being explored to produce SLAM solutions with specific guarantees on optimality [176, 29, 175, 27, 28], especially in the presence of outlier measurements or incorrect data associations [235, 234]. Note that the optimization problem in Equation (2.10) is generally non-convex and thus convergence is only guaranteed to a *local* minimum, which may be arbitrarily bad based on the initialization point. These new approaches reformulate the optimization using the techniques mentioned to either *bound* the solution quality or *certify* when the solution is optimal.

2.3 Frontend

Given a set of relative motion factors and landmark measurement factors, we can pass them off to a nonlinear least squares solver (such as iSAM [103, 102]) 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 compute relative motion and landmark factors from monocular imagery. As previously mentioned, both types of factors are not required to produce a valid solution and often only a single type is used in practice.

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 principal point) [89]. Consider the scenario depicted in Figure 2-2 with two images I_1 and I_2 from a video sequence with relative transform $\mathbf{T}_2^1 \in \mathbb{SE}(3)$ 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 2.3.3. The former process of computing depth from pixel associations is called *depth estimation* (or *stereopsis*) and will be detailed in Section 2.5. It should be noted that both problems rely on the ability to accurately and robustly associate pixels across images.

2.3.2 Visual Landmark Factors

Visual landmark factors constrain the camera poses $\mathbf{X}$ and landmarks $\mathbf{P}$ using observations of the landmarks in the image data. These factors generally fall into two broad categories: *feature-based* methods, which estimate the camera motion as a function of the *reprojection error* of the landmark observations from frame to frame, or *direct* methods, which estimate the camera motion as a function of the *photometric* error of the landmark observations from frame to frame.

Feature-based methods appeared much earlier than direct methods, with seminal work dating back to the 1980s [128, 88, 142, 136, 137]. Factors are generated by computing a small number of salient *features* (e.g., corners, lines, or blobs) that can be reliably detected in the image stream and either tracking or matching these

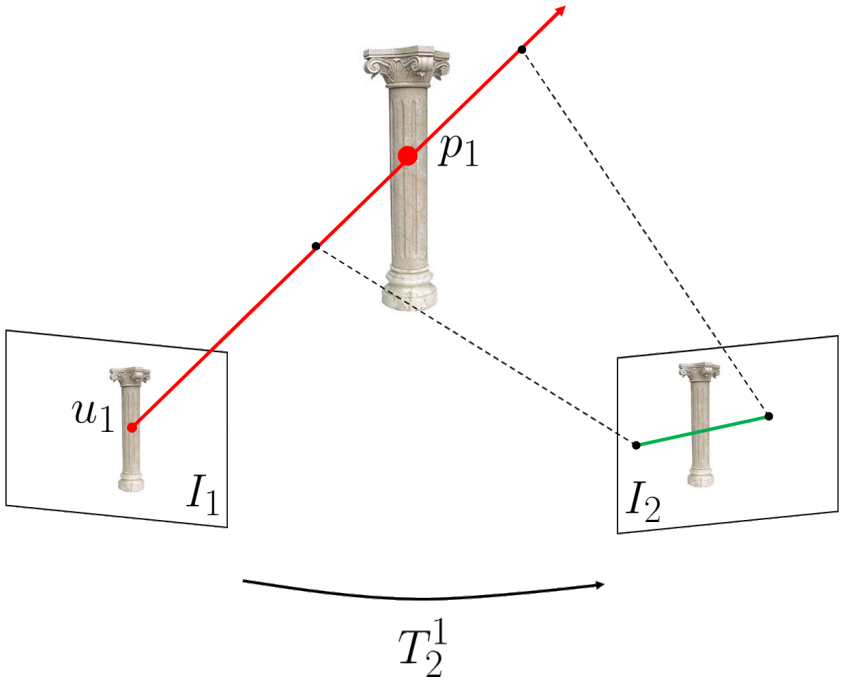


Figure 2-2: In this scenario we have two images I_1 and I_2 taken from two different poses, with the relative transformation between the two given by $\mathbf{T}_2^1$. The 3D point $\mathbf{p}_1$ projects into I_1 as pixel $\mathbf{u}_1$, as do all points along the ray in red. This ray projects into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

features across frames. Note that this process of tracking a feature from frame to frame implicitly assumes that the same feature can be reliably detected — essentially solving the data association problem. A single landmark is then initialized for each feature track and a factor is created that measures the discrepancy between the observed feature positions and the projection of the landmark into the image domain at a particular pose – otherwise known as *reprojection error*.

If landmark $\mathbf{p}_j$ is observed in an image taken from pose $\mathbf{x}_i$, the reprojection error is given by:

$$e_{reproj}(\mathbf{x}_i, \mathbf{p}_j) = ||\mathbf{z}_k - \pi(K\mathbf{x}_i^{-1}\mathbf{p}_j)||, \tag{2.12}$$

where $\mathbf{z}_k \in \Omega$ is the tracked feature position and $K \in \mathbb{R}^{3\times3}$ is the camera intrinsic matrix. The function $\pi(x, y, z) = (x/z, y/z)$ denotes perspective projection. Note that here $\mathbf{p}_j$ is parameterized by its 3D location in Cartesian coordinates, but other

landmark forms are possible, such as the *inverse depth* parameterization, which uses a pixel ray in a given image coupled with an *inverse depth* value to represent the point [33].

There is a large body of work in the computer vision literature on detecting, tracking, and matching sparse 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 [142], for example, compute 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 [87] and Shi-Tomasi corners [198] 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) detectors [180] find 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 popular feature detectors [130, 13].

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 tracking) [133]. The famous Kanade-Lucas-Tomasi (KLT) feature tracker uses this least squares approach with Shi-Tomasi corners [221]. Feature matching typically involves computing a robust *descriptor* vector for the feature and performing brute-force, nearest-neighbor, or RANSAC [63] 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 [130] and HoG [40] descriptors are based on histograms of gradient orientations around the feature, while SURF [13] features use the sum of Haar-like wavelet responses. BRIEF [26], ORB [182], and Census [238] features all use a series of comparison tests between the feature and a set of neighbors to construct a binary descriptor vector.

Direct landmark factors were developed more recently and aim to do away with explicit feature detection and tracking or matching [55, 229]. They use similar principles to factors based on features, but minimize *photometric error* rather than reprojection error. Whereas reprojection error measures the consistency of the camera poses and landmarks with tracked feature locations, photometric error measures the consistency of the poses and landmarks with the raw image intensities.

Suppose that landmark $\mathbf{p}_j$ is observed in two images: image I_i taken from pose $\mathbf{x}_i$ and I_k taken from pose $\mathbf{x}_k$. We will interpret the images $I_i, I_k : \Omega \to \mathbb{R}$ as scalar-valued functions from the image domain $\Omega \subset \mathbb{R}^2$ to $\mathbb{R}$. The photometric error relating $\mathbf{x}_i$, $\mathbf{p}_j$, and $\mathbf{x}_k$ is then given by:

$$e_{photo}(\mathbf{x}_i, \mathbf{p}_j, \mathbf{x}_k) = ||I_i(\pi(\mathbf{x}_i^{-1}\mathbf{p}_j)) - I_k(\pi(\mathbf{x}_k^{-1}\mathbf{p}_j))||. \tag{2.13}$$

It is common to sum the photometric error in a small patch of pixels.

Direct visual landmarks methods possess some notable advantages to feature-based 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, while feature-based methods rely on distinctive corner or blob detection. Furthermore, the feature matching process (i.e., detecting the same features reliably in each image), is usually prone to outliers and false matches, which can wreack havoc on the factor graph solution and is why RANSAC [63] is usually employed to remove false matches. Direct methods, on the other hand, may degrade more gracefully with

the presence of outliers, in addition to other forms of noise like motion blur. Care must be taken, however, to ensure that the appearance and lighting of the scene does not change over time to ensure that the photometric error is minimized at the true poses and landmark values.

2.3.3 Visual Odometry Factors

Visual odometry factors constrain the relative motion between the camera poses $\mathbf{X}$ using primarily the image data, although a physics-based model over plausible camera motion may sometimes be used to augment the visual information. Like visual landmark factors, they can be characterized as either feature-based or direct.

Feature-based visual odometry factors rely on the same detection and tracking strategies outlined in Section 2.3.2. Once a set of features has been associated between two images, the 8-Point Algorithm of [128] or the later 5-Point Algorithm of [154, 155] can be used in conjunction with RANSAC [63] 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 $\mathbf{T}_2^1 \in \mathbb{SE}(3)$ — the rigid body transform of camera 2 relative to camera 1, composed of rotation $\mathbf{R}_2^1 \in \mathbb{SO}(3)$ and translation $\mathbf{t}_2^1 \in \mathbb{R}^3$ — from observed images 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

$$\mathbf{t}_2^1 \times \bar{\mathbf{u}}_1 = \left[\mathbf{t}_2^1\right]_\times \bar{\mathbf{u}}_1, \tag{2.14}$$

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

$$\left[\mathbf{t}_2^1\right]_\times = \begin{bmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{bmatrix}. \tag{2.15}$$

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,

$$\begin{aligned} (\mathbf{R}_2^1 \bar{\mathbf{u}}_2)^T \left[\mathbf{t}_2^1\right]_\times \bar{\mathbf{u}}_1 &= \bar{\mathbf{u}}_2^T \mathbf{R}_1^2 \left[\mathbf{t}_2^1\right]_\times \bar{\mathbf{u}}_1 && (2.16) \\ &= \bar{\mathbf{u}}_2^T \mathbf{E} \bar{\mathbf{u}}_1 && (2.17) \\ &= 0. && (2.18) \end{aligned}$$

The matrix $\mathbf{E} = \mathbf{R}_1^2 [\mathbf{t}_2^1]_\times$ 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 (up to scale) 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. Once $\mathbf{T}_2^1$ has been estimated, a factor can be added to the graph that compares $\mathbf{T}_2^1$ to the value given directly by the current estimates for $\mathbf{x}_1$ and $\mathbf{x}_2$.

Rather than extracting and tracking or 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. Depth information is required for the subset of pixels used to constrain the poses, however. 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 photometric error between the projected (or *warped*) version of I_1 and I_2. This optimization is usually framed as a Lucas-Kanade [133] style nonlinear least squares

problem:

$$\hat{\mathbf{T}}_2^1 = \underset{\mathbf{T}\in\mathbb{SE}(3)}{\arg\min} \sum_{\mathbf{u}\in\Omega_D} ||I_2(w(\mathbf{u}, D(\mathbf{u}), \mathbf{T})) - I_1(\mathbf{u})||^2, \tag{2.19}$$

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 [65], the pixels in I_1 with high-gradient [58], or *every* pixel [153].

The objective in Equation (2.19) is usually minimized by performing Gauss-Newton or Levenberg-Marquardt steps [156], 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 [6, 16]. The optimization in 2.19 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 (2.19) may be overly sensitive to outliers, so it is often replaced by a robust error metric [86, 125] and optimized using *iteratively reweighted least squares* (IRLS), where a weighted version of Equation (2.19) is solved that approximates the solution using the robust norm [8, 17, 160]. For more detail on Lucas-Kanade optimization, see the tutorial presented in [11].

Direct visual odometry methods have similar benefits as direct visual landmark factors over their feature based counterparts. They do require depth information to be recovered, however, which is typically more difficult to obtain than the camera pose itself. The standard approaches to depth estimation will be detailed in Section 2.5.

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. Visual-inertial methods are then detail in Section 2.4.4 followed by learning-based approaches in Section 2.4.5.

2.4.1 Sparse Methods

Sparse monocular SLAM methods maintain a sparse point cloud map and generally use factors that minimize feature-based reprojection error. The first sparse monocular SLAM system to operate in real-time at camera framerate is commonly attributed to Davison [41], which used the EKF-SLAM backend described in Section 2.2.1 with a sparse set of point landmarks initialized using Shi-Tomasi corners [198] 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 computational 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[140, 141] were also explored by Eade and Drummond [52, 51].

The Parallel Tracking and Mapping (PTAM) algorithm developed by Klein and Murray[110] side-stepped 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 allowed the least squares/bundle adjustment techniques from offline SfM [222, 45, 201, 189, 4, 129, 113] 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.

ORB-SLAM developed by Mur-Artal et al. [145, 147] improved upon the general PTAM design by adding loop closure factors to constrain 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, relative motion 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 may crawl to a halt. ORB-SLAM's solution to this problem is to liberally add both landmarks and keyframe factors to the SLAM graph, but prune the graph over time such that only a small number of highly informative 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.

The Direct Sparse Odometry (DSO) method of Engel et al. [55] uses direct visual landmark factors that minimize photometric error over a sparse set of map points. Since it does not rely on feature detection or tracking, it can incorporate information from all image regions that have intensity gradients, rather than only corners. It also integrates a full photometric calibration into the optimization, which accounts for exposure time, lens vignetting, and non-linear response functions. The Semi-Direct Visual Odometry (SVO) algorithm of Forster et al. [65] 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 minimizes a mix of photometric and reprojection factors.

2.4.2 Dense Methods

Dense monocular SLAM methods differ from sparse methods in the density of points estimated in the map $\mathbf{P}$. While a few hundred map points may be observable in any given camera image in a sparse system, thousands to hundreds of thousands of points may be observable in each camera image in a dense system.

For example, the Dense Tracking and Mapping (DTAM) algorithm of Newcombe et al. [153] combines the keyframe-based, parallel approach of PTAM with the horsepower of the GPU to estimate a depth value for every pixel in each keyframe image. Each DTAM keyframe 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 Chambolle and Pock [31]. This smooth reconstruction is then used to densely track each new frame using the coarse-to-fine Lucas-Kanade algorithm described in Section 2.3.3. This approach has been extended in Pinies et al. [166] which uses a non-local total variation (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 Graber et al. [78] takes the approach of densifying PTAM keyframes. For each keyframe, a multi-view version of the stereo method described by Collins [35] 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. The approach of Newcombe and Davison [152] also densifies PTAM keyframes, building coarse meshes from sparse map points and then refining the meshes using correspondences generated from a separate dense optical flow process.

REMODE developed by Pizzoli et al. [167] uses a semi-direct visual odometry

method [65] 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 of Pradeep et al. [172] 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 [19] using variable baseline comparison images. The depthmaps are then fused into an SDF using the method of Curless and Levoy [37] 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 constrain the surface to be smooth. The MobileFusion algorithm [159] 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 of Engel et al. [58, 56] estimates keyframe depthmaps using a per-pixel probabilistic filter similar to Pizzoli et al. [167], 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 [117], 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 2.3.3 over the high-gradient pixels.

A similar approach was proposed in Mur-Artal and Tardós [146] that uses ORB-SLAM [145] 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.

2.4.4 Visual-Inertial Methods

Visual-inertial monocular SLAM methods fuse the monocular camera with measurements from an IMU. The IMU allows the SLAM system to maintain tracking through degraded visual environments, especially low-texture scenes where image gradients provide no information to localize the camera. The IMU also provides linear acceleration measurements that allow the metric scale of the SLAM solution to be inferred.

The Multi-State Constraint Kalman Filter (MSCKF) from Mourikis and Roumeliotis [143] tracks sparse features through the image stream then utilizes an EKF over a sliding window of keyframes to infer the camera poses from the feature tracks and IMU measurements. By projecting the filter residuals onto the null space of the measurement Jacobian matrix, the MSCKF does not require the landmark positions to be included in the filter state, greatly improving speed.

Smoothing-based visual-inertial methods such as the approach from Leutenegger et al. [124] insert relative motion factors computed from the IMU measurements into the SLAM factor graph between keyframe poses. *Pre-integrating* the IMU measurements correctly between keyframe poses so that the variables in the graph can be easily re-linearized was explored in the methods of Forster et al. [64] and Qin et al [173]. The approach of Forster et al. [64] also adapted the techniques of Mourikis and Roumeliotis [143] to the smoothing setting, improving speed by cleverly removing the map points from the optimization. The method of Rosinal et al. [179, 178] extends the visual-inertial paradigm by adding mesh reconstruction and semantic labeling to

the SLAM map.

2.4.5 Learning-based Methods

SLAM methods that rely on machine learning techniques have increased in popularity in recent years, driven by the rapid application of deep learning to a variety of computer vision tasks. Unlike the aforementioned approaches, learning-based SLAM systems attempt to *predict* or *regress* the camera poses and map using models trained from offline data, generally relying less on the epipolar constraints induced by live imagery. For example, the method of Wang et al. [230] uses a recurrent convolutional neural network (RCNN) [48] that extracts features from stacks of input images using a convolutional neural network (CNN) before passing the features through two recurrent, long-short-term memory (LSTM) [93] subnetworks that directly regress the image poses.

The DeMoN method of Ummenhofer [223] uses a chain of encoder-decoder networks that takes in a pair of consecutive images and outputs a depthmap of the first image and the relative pose of the second. The encoder-network chain is composed of an initial network that computes optical flow, depth, and camera motion, followed by a series of upsampling and refinement networks. It also regresses surface normals and confidence metric as intermediate values. Supervised losses using groundtruth depthmaps, optical flow, surface normals, and camera poses drive the training optimization. The imagery used to train the network has become a standard benchmark dataset for depth estimation problems.

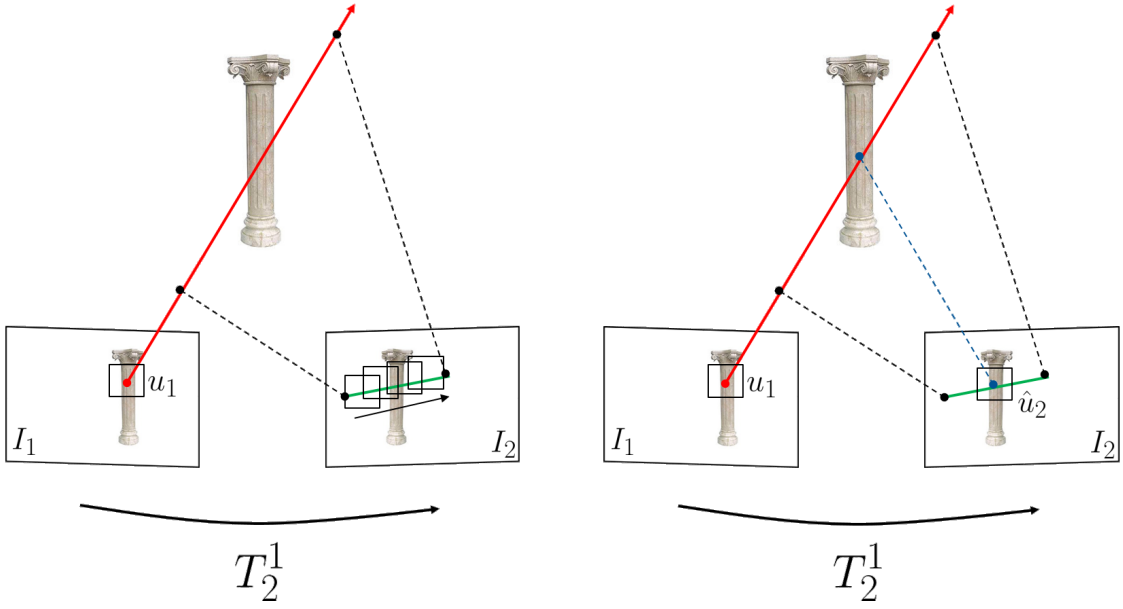
While supervised methods like DeMoN offer impressive performance, obtaining extensive groundtruth labels is difficult for large datasets. A growing trend in this space is to leverage unsupervised (or self-supervised) training losses so that more varied imagery can be used as training data. The UnDeepVO method of Li et al. [126], the SfMLearner method of Zhou et al. [241], the Monodepth2 method of Godard et al. [76], the SuperDepth method of Pillai et al. [164], and the PackNet-SfM method of Guizilini et al. [84] all exploit this concept. Rather than relying on groundtruth depthmaps, these approaches take as input a series of consecutive images and predict

the depths and relative camera motion using stacks of CNNs. The depthmaps and poses are then used to warp each image into the other coordinate frames. The photometric error between the warped images and the real images can then be used to infer the network weights.

Recently, techniques that combine deep learning and traditional geometric SLAM have been investigated. Zhan et al. [240] train two networks to estimate depth and optical flow from a pair of images that are then used to improve conventional visual odometry. The method of Bloesch et al. [20] replaces the point landmarks in the traditional monocular SLAM framework with a learned, low-dimensional feature vector (or *code*) that represents an entire, dense depthmap. The SLAM graph consisting of camera poses and depthmap codes is then optimized using classical techniques, allowing for both dense and probabilistic inference. Czarnowski et al. [38] built upon this technique to include local and global loop closures, keyframing, map maintenance, and relocalization. The network from Tang and Tang [212] similarly uses a compressed depthmap representation, but incorporates it into a fully differentiable variant of bundle adjustment that can be optimized end-to-end.

2.5 Dense Monocular Depth Estimation

A significant portion of this thesis investigates a variation of the monocular SLAM problem that we will refer to as dense monocular *depth estimation*. In this setting, we attempt to estimate the 3D structure of a scene (or the *depth* of the scene) given the images from a moving monocular camera. Crucially, for this modification of the monocular SLAM problem we will assume that the intrinsics and extrinsics (i.e., poses) of the camera are known *a priori* or computed in a separate process such as through one of the sparse monocular SLAM pipelines outlined in Section 2.4.1. While some depth information can be computed using those sparse techniques, 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



(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 2-3b.

dense monocular SLAM case the camera poses are unknown and must be estimated along with the scene depth. There is crossover between the two problems given their similarities, of course, but we will reserve the term "dense monocular depth estimation" for techniques that do not estimate or modify the camera poses.

Despite the additional simplifying assumptions, dense monocular depth estimation is still a difficult inference problem for many of the same reasons as dense monocular SLAM. Estimating a depth value for *every* pixel in an input image requires serious computational horsepower and sophisticated algorithms to be tractable. Furthermore, associating pixels across images taken from disparate viewpoints, which forms the backbone of monocular depth estimation, is a non-trivial task, especially when it must be attempted for every input pixel. The following sections will briefly outline the traditional techniques to tackling these issues, including spatial regularization in Section 2.5.2 and learning-based methods in Section 2.5.3.

2.5.1 Depth Estimation via Stereopsis

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 [10, 74, 35] and multi-view stereo (MVS) depth estimation [77, 68, 91] have been studied extensively (see [188, 193, 210] for more information). While rectified stereo methods traditionally leverage multiple cameras that are rigidly mounted and triggered with specialized hardware, MVS is very aligned with our problem of dense monocular depth estimation. In fact, dense monocular depth estimation may be considered a specific variant of MVS, where additional emphasis is placed on processing images from a single moving camera in real-time, while traditional MVS generally focuses more on imagery from multiple cameras and is more often an offline process.

As argued previously, 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 posed. 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 or patch 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 the pixels of 3D points remain visually similar when observed from multiple viewpoints. 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 [238] (all similar to the work in sparse feature matching).

Nearly all MVS and dense depth estimation algorithms are built on these foundational blocks of epipolar search, patch matching, and triangulation. In addition, the

representation of the depth information is another important design choice as it often dictates how depth information from multiple comparison frames are fused. One standard approach is to define a 2D *depthmap*, *inverse* depthmap, or *disparity map* (they are usually interchangeable) [209, 106, 71]. A depthmap (or inverse depthmap) is a scalar function over the image domain $D : \Omega \to \mathbb{R}_+$ 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 [58, 56, 65].

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 [194, 118, 23]. Voxels can also represent the color of the scene (known as *voxel coloring*) [36] or the distance to the nearest surface [181, 62, 37, 77]. Triangular meshes [67] and other deformable models [215, 216] are also viable alternatives.

An important form of MVS that combines both depthmap and voxel-based techniques was presented by Collins [35]. Dubbed Plane Sweep Stereo, this method reformulates the epipolar search, patch matching, and triangulation paradigm to efficiently compute dense depthmaps from multiple images. Rather than perform an epipolar search for each pixel individually, Plane Sweep first designates a privileged reference frame from all the input images. A cost volume in this coordinate frame is then defined by sampling a set of depth hypotheses over all the pixels. When all pixels from the reference image are considered, each depth hypothesis induces a plane in the reference coordinate frame. Each neighboring image is then *projected* onto each plane using a *homography* transformation. A cost volume is then produced by taking the difference between each projected image and the reference image. The depth sample that minimizes the matching costs can then be extracted for each pixel, producing

a dense depthmap. Fusing information from multiple images can be achieved simply by averaging the cost volumes generated by each reference image - comparison image pair.

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. Furthermore, the depths in large textureless image regions are generally unobservable since the lack of gradient information cripples the patch matching process.

Traditional image filtering techniques, such as low-pass filters or median filters, are sometimes applied to the resulting depthmaps to reduce noise and remove outliers. These filters can also be generalized to 3D and applied to the matching cost volume before extracting depths, which is sometimes referred to as *cost aggregation* [193]. Since filters of this type only incorporate information locally around each pixel or voxel, these denoising techniques are usually referred to as *local* methods.

More sophisticated regularization schemes have been developed, however, that attempt to enforce certain priors on the overall structure of the scene, such as smoothness, planarity, or other properties. Rather than only consider local information to produce smoothed reconstructions, these methods optimize over all the input data, such as a the entire noisy depthmap or cost volume, and hence are generally referred to as *global* methods. A brief overview of these techniques is presented in the following section.

2.5.2 Spatial Regularization

The standard approach to applying global spatial regularization to depth information is to pose the problem as *energy minimization* [215, 18, 187, 74] where the regularized solution must balance smoothness with fitting the input data. We define an energy functional $E(D)$ for a depthmap $D : \Omega \to \mathbb{R}_+$, by combining two cost functionals: a

smoothness cost $E_{smooth}(D)$ and a data fitting cost $E_{data}(D)$:

$$E(D) = E_{smooth}(D) + \lambda E_{data}(D). \tag{2.20}$$

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). The data fit term $E_{data}(D)$ typically is a measures the difference between the smoothed solution D and the noisy input data.

One way to derive the energy terms above is to define a probability distribution over the depthmap pixels using a *Markov Random Field* (MRF) [111]. 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 [21, 181, 98, 224, 112].

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 [10, 157, 15, 14, 73].

The specific forms of the energy terms in Equation (2.20) 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:

$$E_{data}(D) = \int_{\Omega} ||D(\mathbf{u}) - g(\mathbf{u})||_{\Sigma}^2 \, d\mathbf{u} \tag{2.21}$$

$$E_{smooth}(D) = \int_{\Omega} ||\nabla D(\mathbf{u})||_{\Gamma}^2 \, d\mathbf{u}, \tag{2.22}$$

where $g(\mathbf{u})$ denotes the noisy input data.

Note that these functions essentially correspond to the special case of Gaussian measurement model (with covariance Γ) and Gaussian prior (with covariance Σ). 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 Rudin et al. [183] and is commonly referred to as the *Total Variation* (TV) regularizer:

$$E_{smooth}(D) = \int_{\Omega} ||\nabla D(\mathbf{u})||\ d\mathbf{u}. \tag{2.23}$$

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 [74, 144, 62, 168, 151, 59, 31].

2.5.3 Learned Depth Estimation

Section 2.5.1 outlined ways that depth could be estimated from monocular imagery by exploiting epipolar constraints to find pixel correspondences that can then be triangulated. Section 2.5.2 then described ways that prior structure could be applied to the depth estimation problem to denoise estimates, remove outliers, and improve reconstruction accuracy. In recent years, many researchers have explored replacing

these components with deep neural networks that can be learned from data. For example, rather than using the raw image intensities when comparing pixel patches, more distinctive features can be learned that exploit higher-level semantic information and global context to improve the matching process. Rather than using simple low-pass filters or even mathematical priors defined *a priori* to denoise depths, filters can be learned directly from training data that more accurately enforce real world constraints.

Learning was first applied to the rectified stereo depth estimation problem. Zbontar and LeCun [239], for example, proposed a network to regress matching costs from small image patches, before using the costs in a classical pipeline. Mayer et al. [138] developed a network that directly regresses disparity using stacks of convolutions and deconvolutions. Kendall et al. [107] aggregate global context in the stereo cost volume using 3D convolutions. Khamis et al. [108] similarly use 3D convolutions to aggregate information in the cost volume, but significantly reduce the spatial resolution of the volume for speed before applying image-guided refiners to upsample the resulting disparities. (We will see in Chapter 5 some of the limitations of the end-to-end approaches that do not leverage known structure in the problem.)

Learning-based approaches to MVS depth estimation often follow the Plane Sweep paradigm described previously [35]. Yao et al. [237], for example, extract features per image, transform them into the reference volume using a differentiable warp operation, then regularize the costs using multi-scale 3D convolutions. Im et al. [96] compute matching costs similarly, but refine the costs for each depth hypothesis using the reference image features. Wang and Shen [227] compute a multi-view cost volume using classical techniques, but then regress the depths using an encoder-decoder network. Huang et al. [95], on the other hand, estimate depthmaps on 64×64 pixel patches before tiling the results to the input resolution.

There is also substantial interest in predicting depth directly from single monocular images instead of considering epipolar constraints induced from multiple images. Eigen et al. [54] use a two-scale convolutional network that regresses depth using supervised training. Similar to the learned SLAM methods outlined in Section 2.4.5,

self-supervised approaches have also been developed that enforce photometric consistency such as the method of Garg et al. [70], Godard et al. [75], and Poggi et al. [169]. These methods are trained using calibrated stereo images (i.e., two images taken from known poses), but only require a single image – *without* a known pose – to generate depths at test time, broadening the applicability of these systems to tasks where estimating pose may be difficult or intractable.

With an understanding of the state of both monocular SLAM and monocular depth estimation, the subsequent chapters in this thesis will outline specific limitations that prevent monocular SLAM and monocular depth estimation systems from generalizing to more uncontrolled environments. We will then present solutions to these problems that are enabled by untapped prior information along with targeted applications of machine learning.

Chapter 3

Fast Lightweight Mesh Estimation

This chapter describes a fast mesh reconstruction method that exploits the planar structure of many environments of interest to dramatically accelerate the depth estimation process. As argued in previous chapters, estimating dense 3D geometry from 2D images taken from a moving monocular camera is a fundamental problem in computer vision with a wide range of applications in robotics and augmented reality (AR). Though the visual tracking component of monocular *simultaneous localization and mapping* (SLAM) has reached a certain level of maturity over the last ten years [143, 64, 110, 65, 58, 145, 153], efficiently reconstructing dense environment representations for autonomous navigation or AR on small size-weight-and-power (SWaP) constrained platforms (such as mobile robots and smartphones) is still an active research front. Current approaches either transmit information to a ground-station for processing [231, 5, 39], sacrifice density [57, 56, 65, 145], run at significantly reduced framerates [191, 192], or limit the reconstruction volume to small scenes [159] or to past keyframes [80], all of which restrict their utility in practice, especially for mobile robot navigation. In this chapter, we propose a novel monocular depth estimation pipeline that enables dense geometry to be efficiently computed at upwards of 230 Hz using less than one Intel i7 4820K CPU core — a small enough footprint to fit completely onboard an autonomous micro-aerial vehicle (MAV), with sufficient accuracy to enable closed-loop motion planning using the reconstructions.

Our key insight is to recognize that for many environments and applications,

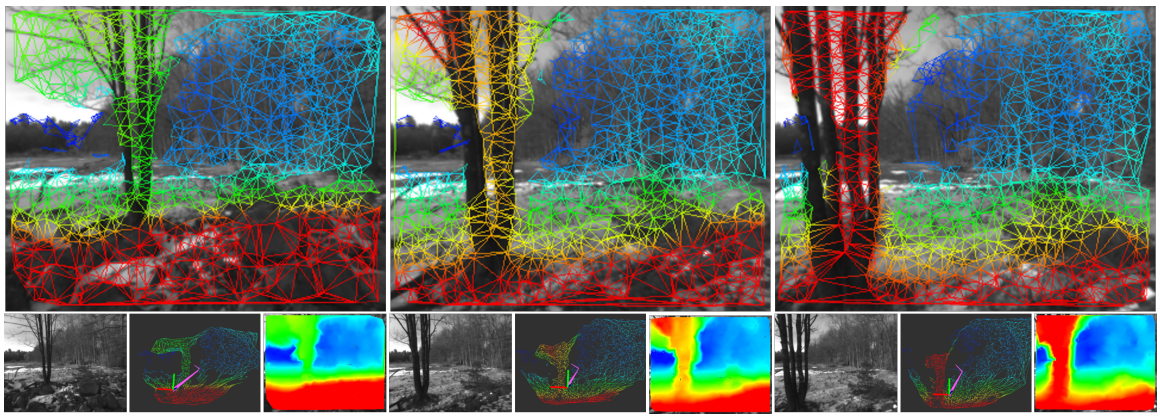


Figure 3-1: *Fast Lightweight Mesh Estimation* – FLaME generates 3D mesh reconstructions from monocular images in real-time onboard computationally constrained platforms. The key to the approach is a graph-based variational optimization framework that allows for the mesh to be efficiently smoothed and refined. The top row of images (from left to right) show the meshes computed onboard a small autonomous robot flying at 3.5 meters-per-second as it avoids a tree. The bottom row shows the current frame (left), the collision-free plan in pink (middle), and the dense depthmap generated from the mesh (right) for each timestep along the approach.

the "every-pixel" methods that are currently in vogue for dense depth estimation massively oversample scenes relative to their true geometric complexity. Many environments of interest can be well approximated using a relatively small set of planes. Consider the geometry of a city composed of a series of buildings. Each building typically has 5 exposed sides. Each side is mostly flat. Each building usually rests on a flat groundplane. To first order, therefore, the geometry of a city can be well-described as a finite set of planes or *piecewise planar*. It is unlikely that all 8 million depth estimates in a 4K image – or even all 300 thousand depth estimates in a VGA image – are necessary to encode the geometry for this type of scene. Indeed, if each plane is composed of a set of triangles (as is common in computer graphics), and each triangle requires only 3 different depth values (one for each triangle vertex), it is likely that far fewer depth values would be needed to reconstruct the scene than the total number of pixels in each image. More complicated environments, like forests, parks, or indoor scenes, may require more planes and triangles than a city environment would need to adequately capture the geometry, but a piecewise planar representation is a very good approximation, especially for important tasks like obstacle avoidance and AR.

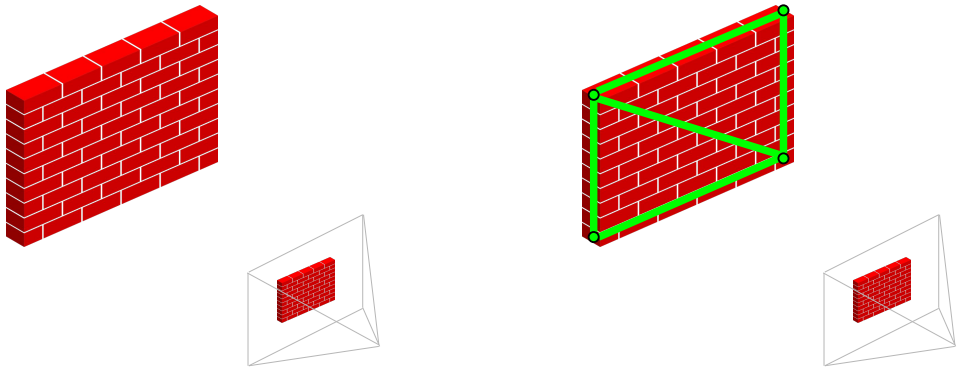


Figure 3-2: *Piecewise Planar Environments* – Many environments of interest can be well-approximated using a finite set of planes. Consider the environment on the left with a brick wall. A dense, every-pixel depth estimation method might devote thousands of depth estimates to encode the geometry. It can be described much more succinctly, however, using sets of planes built from triangular meshes. As can be seen on the right, only two triangles with a total of four depth estimates are needed to capture the geometry of the wall.

Furthermore, oversampling scenes causes other problems in depth estimation. Even if a scene is geometrically complex, the fine details may not be observable given the baseline of the available camera images. For instance, moving a camera 1 cm will not provide enough baseline to triangulate the depths for a complex scene 1 km away. Attempting to estimate depth for every pixel in these regions wastes valuable computational resources. In addition, these weakly observable pixels may be easily corrupted by noise or matched incorrectly, degrading the quality of the produced depths and requiring stronger or more sophisticated regularization schemes that are often computationally demanding.

One naive workaround to the above issues might be to run an "every-pixel" method on downsampled input images, but this degrades the observable depth information even more by reducing the resolution at which pixels may be associated across frames. Another workaround might be to sparsify the reconstruction by only estimating depth for certain pixels across the image, but this, of course, introduces holes and breaks most regularization approaches that require dense local information to filter out noise.

Instead, we exploit our observation that piecewise planar models can approxi-

mate many environments of interest well and propose a novel alternative that we call FLaME (*Fast Lightweight Mesh Estimation*) that directly estimates a triangular mesh of the scene (similar to the stereo work of Pillai et al. [165]). Our method is advantageous for several reasons. First, meshes are more compact, efficient representations of the geometry and therefore require fewer depth estimates to encode the scene for a given level of detail. Second, by interpreting the mesh as a graph we show that we can exploit its connectivity structure to apply (and accelerate) state-of-the-art second-order variational regularization techniques that otherwise require GPUs to run online. Third, by reformulating the regularization objective in terms of the vertices and edges of this graph, we allow the smoothing optimization to be both *incremental* (in that new terms can be trivially added and removed as the graph is modified over time) and *keyframeless* (in that the solution can be easily propagated in time without restarting the optimization).

We show significant improvements over existing approaches on benchmark data in terms of runtime, CPU load, density, and accuracy, and present results from flight experiments running FLaME in-the-loop onboard a small MAV flying at up to 3.5 meters-per-second (see Figure 3-1 and 3-3).

3.1 Related Work

Dense monocular 3D reconstruction has its roots in the multi-view stereo (MVS) [68, 91, 77] and structure-from-motion (SfM) [4, 45] literature, where typical approaches require offline batch processing. Real-time solutions have largely come from the monocular SLAM community, beginning with small-scale sparse methods [41] and gradually evolving to large-scale sparse [110] and semi-dense [58, 56, 65] approaches.

Real-time dense solutions have become recently feasible with the widescale proliferation of GPUs. The Dense Tracking and Mapping (DTAM) algorithm of Newcombe et al. [153] is one of the first such algorithms, which estimates dense keyframe meshes by minimizing the photometric error defined over a 3D cost volume using a variational approach. The method of Graber et al. [78] also uses variational optimization

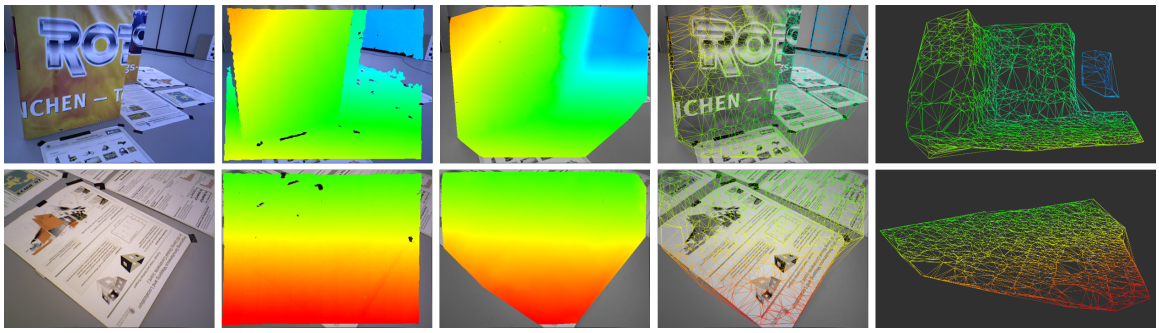


Figure 3-3: *Second Order Smoothing* – FLaME estimates dense inverse depth meshes by optimizing a non-local, second-order variational smoothness cost over a semi-sparsely sampled *Delaunay Graph*. Minimizing this cost function promotes piecewise planar structure as shown above on benchmark data collected from a handheld Kinect [207]. From left to right, each column shows the input RGB image, the Kinect depthmap, the generated FLaME depthmap, the FLaME mesh in the current view, and the FLaME mesh projected into 3D. Note the smooth planar reconstructions that are enabled by the approach and the accuracy of the depthmaps relative to those from the Kinect.

to extract surfaces, but represents the surface as the zero level-set of a signed-distance function (SDF). The MonoFusion algorithm of Pradeep et al. [172] also uses an SDF surface representation and fuses depth information defined over the SDF using the method of Curless and Levoy [37] before raycasting to extract the mesh.

While representing significant progress, these aforementioned methods are currently constrained to small scenes and require desktop-class GPUs with gentle, handheld camera motion that reduces their utility for fast flight through unknown environments. While the multi-level method of Greene et al. [80] is able to reconstruct larger scenes without the aid of GPU acceleration, it does not fuse its dense depthmaps into a coherent surface and suffers from keyframe alignment errors. The MobileFusion algorithm of Ondruska et al. [159] extends MonoFusion to run at 25 Hz on a commodity smartphone, but further sacrifices the volume under reconstruction.

3.2 Variational Smoothing

In this section, we will give a brief overview of variational smoothing methods, which are powerful strategies for removing noise and outliers from signals of interest. Their

name derives from their usage of mathematical techniques from the calculus of variations. FLaME will leverage concepts from variational smoothing to efficiently denoise depth estimates defined over a mesh.

In the continuous setting, variational methods pose the regularization problem as one of *energy minimization*. Suppose we observe a noisy scalar function $z : \mathcal{X} \to \mathbb{R}$ over some domain $\mathcal{X} \subset \mathbb{R}^n$. We will compute a denoised version of this function by minimizing an energy functional E of the following form:

$$E(f) = E_{smooth}(f) + \lambda E_{data}(f), \tag{3.1}$$

where $f : \mathcal{X} \to \mathbb{R}$ denotes our decision variable, E_{smooth} enforces some form of smoothness or regularity over the space of solutions, and E_{data} is a measure of how well f fits the measurements z. The scalar $\lambda > 0$ controls the balance of data-fitting versus smoothness. Our final solution will be the minimizer $f^* = \arg\min_f E(f)$ of this objective.

Common choices for the data-fitting term $E_{data}(f)$ include the standard squared-L_2 norm with the observed signal z:

$$E_{data}(f) = \int_{\mathcal{X}} |f(\mathbf{x}) - z(\mathbf{x})|^2 d\mathbf{x}, \tag{3.2}$$

or an outlier-robust L_1 norm:

$$E_{data}(f) = \int_{\mathcal{X}} |f(\mathbf{x}) - z(\mathbf{x})| d\mathbf{x}. \tag{3.3}$$

The choice of the smoothness term E_{smooth} is generally more involved. In many important applications, we wish to remove noise from z, but do not wish to unnecessarily smooth away edges or sharp discontinuities that may be part of the true signal. One possible option that attempts to capture this preference is *total variation* (TV), defined as the L_2 norm of the gradient of the function f:

$$E_{smooth}(f) = \int_{\mathcal{X}} ||\nabla f(\mathbf{x})||_2 \, d\mathbf{x}. \tag{3.4}$$

This form of TV assumes that f is differentiable, but generalizations exist for functions that belong to the broader class $L^1(\mathcal{X})$, the space of functions whose absolute value is Lebesgue integrable [30].

By penalizing the L_2 norm of the gradient (as opposed to the square of the L_2 norm, for example), the TV regularizer biases functions to be *piecewise constant*, and therefore solutions that have sharp discontinuities are not penalized as much as smoothly varying functions. 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 [183]. 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. (See Section 2.5.2 for more information.)

While TV enforces attractive structure on the space of acceptable solutions by allowing for sharp discontinuities, the true underlying signal we are searching for may not actually be piecewise constant. Natural images, for example, are not piecewise constant, but have small variations between sharp discontinuities. A different, more powerful choice of regularization is the second-order Total Generalized Variation (TGV^2) semi-norm of [22]:

$$\text{TGV}^2(f) = \min_{\mathbf{w}(\mathbf{x})\in\mathbb{R}^2} \alpha \int_{\mathcal{X}} |\nabla f(\mathbf{x}) - \mathbf{w}(\mathbf{x})| \, d\mathbf{x} + \beta \int_{\mathcal{X}} |\nabla \mathbf{w}(\mathbf{x})| \, d\mathbf{x}, \tag{3.5}$$

which introduces auxiliary function $\mathbf{w} : \mathcal{X} \to \mathbb{R}^2$ and weights $\alpha, \beta \geq 0$. This functional penalizes discontinuities in the first two derivatives of f and therefore promotes piecewise *affine* (or planar) solutions. The contributions of the first and second derivatives to the overall cost are controlled by α and β.

It is important to note that this functional only incorporates *local* information through the gradient operator. In the context of images, it is reasonable that some distant locations in the image may actually be correlated (e.g., points inside object boundaries). A *non-local* extension NLTGV^2 was therefore developed in Ranftl et al. [174] so that information beyond immediately neighboring pixels could influence

Algorithm 1 Method of Chambolle and Pock [31]

// Choose $\sigma, \tau > 0$, $\theta \in [0, 1]$.
while not converged **do**
  $\mathbf{q}^{k+1} = \text{prox}_{F^*}(\mathbf{q}^k + \sigma \mathbf{D}\bar{\mathbf{x}}^k)$
  $\mathbf{x}^{k+1} = \text{prox}_G(\mathbf{x}^k - \tau \mathbf{D}^* \mathbf{q}^{k+1})$
  $\bar{\mathbf{x}}^{k+1} = \mathbf{x}^{k+1} + \theta(\mathbf{x}^{k+1} - \mathbf{x}^k)$

the objective:

$$\begin{aligned} &\text{NLTGV}^2(f) = \\ &\min_{\mathbf{w}(\mathbf{x}) \in \mathbb{R}^2} \int_{\mathcal{X}} \int_{\mathcal{X}} \alpha(\mathbf{x}, \mathbf{y}) \left| f(\mathbf{x}) - f(\mathbf{y}) - \langle \mathbf{w}(\mathbf{x}), \mathbf{x} - \mathbf{y} \rangle \right| d\mathbf{x} d\mathbf{y} + \\ &\qquad \int_{\mathcal{X}} \int_{\mathcal{X}} \beta(\mathbf{x}, \mathbf{y}) \left| w_1(\mathbf{x}) - w_1(\mathbf{y}) \right| d\mathbf{x} d\mathbf{y} + \\ &\qquad \int_{\mathcal{X}} \int_{\mathcal{X}} \beta(\mathbf{x}, \mathbf{y}) \left| w_2(\mathbf{x}) - w_2(\mathbf{y}) \right| d\mathbf{x} d\mathbf{y}, \end{aligned} \tag{3.6}$$

for $\mathbf{w}(\mathbf{x}) = (w_1(\mathbf{x}), w_2(\mathbf{x}))$ and weight functions $\alpha(\mathbf{x}, \mathbf{y}) \geq 0$ and $\beta(\mathbf{x}, \mathbf{y}) \geq 0$, which encode the weighted, non-local gradients.

The work of Pinies et al. [166] showed that when f is interpreted as an inverse depthmap $\xi : \Omega \to \mathbb{R}_+$, smoothing with NLTGV^2 leads not only to piecewise affine solutions over the image domain $\mathcal{X} = \Omega$, but also over $\mathbb{R}^3$ when ξ is projected into 3D (a non-trivial result). This means that by smoothing an inverse depthmap with NLTGV^2, we can obtain fully piecewise planar geometry.

Although the choices of E_{data} and E_{smooth} outlined above are generally not differentiable, they are convex and can thus be efficiently minimized using convex optimization techniques. If we discretize E_{smooth} and E_{data}, one popular optimization scheme is the first-order, primal-dual method of Chambolle and Pock [31], which solves optimization problems of the following form:

$$\min_{\mathbf{x} \in \mathbb{R}^n} F(\mathbf{D}\mathbf{x}) + G(\mathbf{x}), \tag{3.7}$$

where $F : \mathbb{R}^m \to \mathbb{R}_+$ and $G : \mathbb{R}^n \to \mathbb{R}_+$ are convex and $\mathbf{D} : \mathbb{R}^n \to \mathbb{R}^m$ is a linear operator that usually encodes discrete gradients. Here the $F(\mathbf{D}\mathbf{x})$ term corresponds to E_{smooth} and the $G(\mathbf{x})$ term corresponds to E_{data}.

The essence of the Chambolle and Pock approach is to represent F in terms of its *convex conjugate* and dual variable $\mathbf{q} \in \mathbb{R}^m$, resulting in the following saddlepoint problem:

$$\min_{\mathbf{x}\in\mathbb{R}^n} \max_{\mathbf{q}\in\mathbb{R}^m} \langle \mathbf{D}\mathbf{x}, \mathbf{q} \rangle - F^*(\mathbf{q}) + G(\mathbf{x}). \tag{3.8}$$

Optimal values for primal variable $\mathbf{x}$ and dual variable $\mathbf{q}$ are then obtained by repeated application of the *proximal* operator that generalizes gradient descent to non-differentiable functions [161] (see Algorithm 1).

The method described in the rest of this chapter will apply NLTGV^2 regularization, optimized using the method of Chambolle and Pock, to fast mesh estimation. Generally, NLTGV^2 regularization and the method of Chambolle and Pock require GPU acceleration. Our method will reformulate the optimization onto a graph induced by a triangular mesh, allowable for significant computational savings.

3.3 Method

FLaME directly estimates an *inverse depth mesh* of the environment that efficiently encodes the scene geometry and allows for efficient, incremental, and keyframeless second-order variational regularization to recover smooth surfaces. In the following sections, we will represent images as scalar functions defined over the pixel domain $\Omega \subset \mathbb{R}^2$, such that $I_k : \Omega \to \mathbb{R}$ denotes the image taken at time index k. We will represent the pose of the camera at time k with respect to the world frame W by the transform $\mathbf{T}_k^W \in \mathbb{SE}(3)$.

Given an image sequence I_k from a moving camera with known pose $\mathbf{T}_k^W$ then, our task entails:

- Estimating the inverse depth for a set of sampled pixels (Section 3.3.1)
- Constructing the mesh using the sampled points (Section 3.3.2)
- Defining a suitable smoothness cost over the graph induced by the mesh (Section 3.3.3)

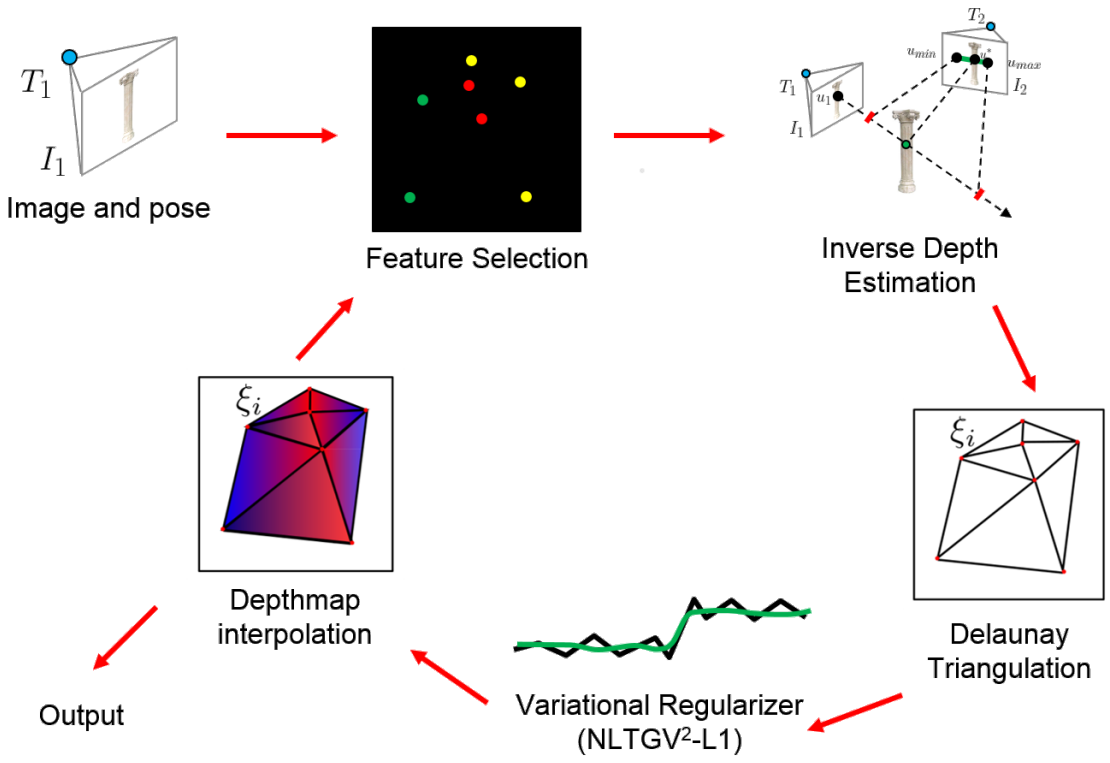


Figure 3-4: *FLaME Overview* – FLaME operates on image streams with known poses. Inverse depth is estimated for a set of *features* using the fast, filtering-based approach of [58]. When the inverse depth estimate for a given feature converges, it is inserted as a new vertex in a graph defined in the current frame and computed through Delaunay triangulations. This Delaunay graph is then used to efficiently smooth away noise in the inverse depth values and promote piecewise planar structure by minimizing a second-order variational cost defined over the graph.

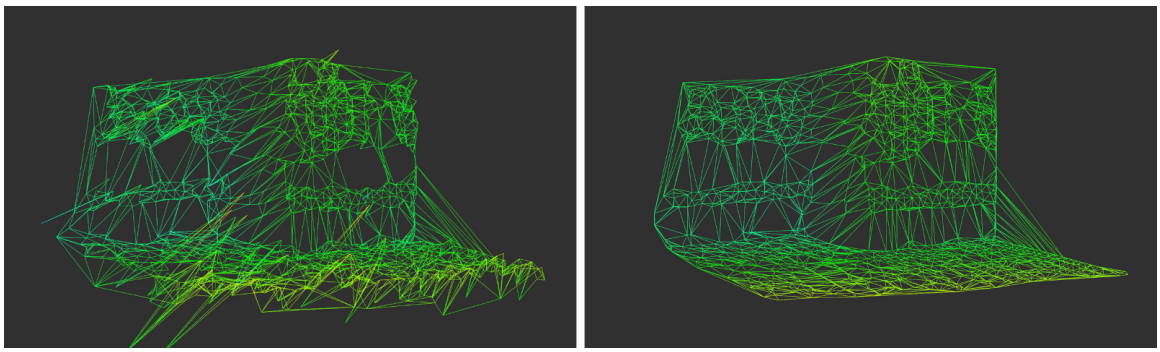


Figure 3-5: *Spatial Regularization* – FLaME minimizes a non-local, variational smoothness cost defined over a Delaunay graph, which efficiently generates piecewise planar mesh reconstructions from noisy inverse depth estimates. The above images show the meshes produced from raw, unsmoothed inverse depth values (left) and those smoothed with FLaME (right).

- Minimizing the smoothness cost (Section 3.3.4)
- Projecting the mesh from frame to frame (Section 3.3.5)

See Figure 3-4 for a block diagram of the data flow.

3.3.1 Feature Inverse Depth Estimation

We first estimate the inverse depth for a set "trackable" pixels (or *features*) sampled over the image domain that will serve as candidate vertices to insert into our mesh. Let $\mathcal{F}_k$ denote the current set of features in I_k . Each feature $f \in \mathcal{F}_k$ is detected at timestep f_t and defined at location $f_{\mathbf{u}} \in \Omega$ in the image I_{f_t} at pose $\mathbf{T}_{f_t}^W$.

We select features by dividing Ω into grid cells of size $2^L \times 2^L$ based on a user-set detail level L (see Figure 3-6) and selecting a pixel in each cell as a new feature if certain criteria are met. First, we do not select features in cells that contain the projection of another feature (this ensures we maintain a certain desired detail level). If no other feature in $\mathcal{F}_k$ falls into a given grid cell, then for each pixel $\mathbf{u}$ in the grid cell we compute a *trackability score* $s(\mathbf{u}) = \left|\nabla I_k(\mathbf{u})^T \mathbf{e}_{\mathbf{u}}\right|$ based on the image gradient $\nabla I_k(\mathbf{u})$ and epipolar direction $\mathbf{e}_{\mathbf{u}}$ induced by the previous frame. This score is a simple metric for determining pixels that will be easy to match in future frames given the camera motion. If the pixel in the window with the maximum score passes a threshold, we add it as a feature to $\mathcal{F}_k$.

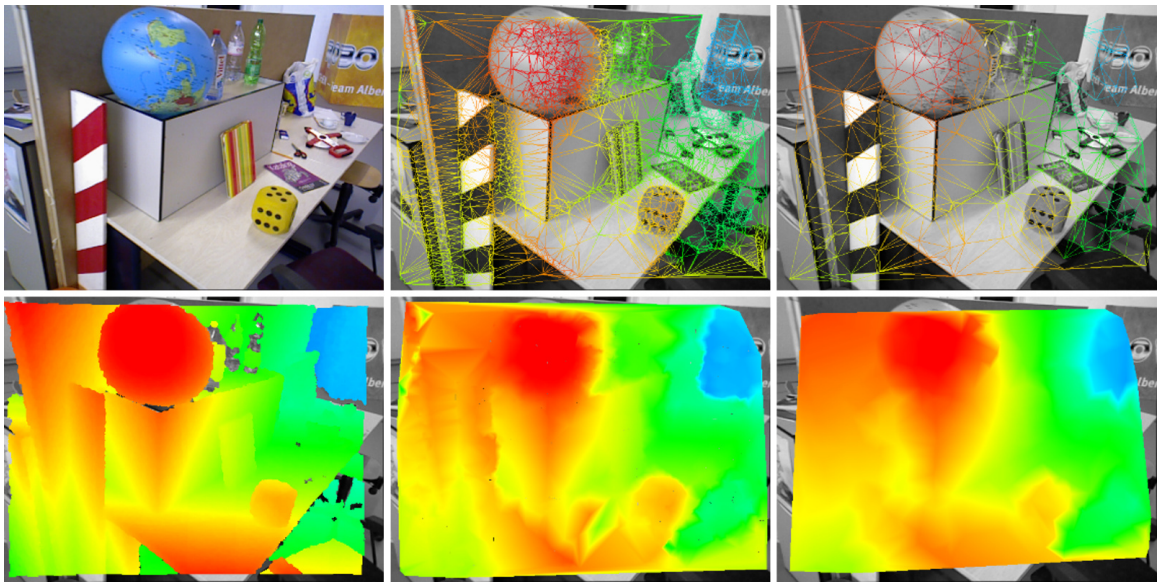


Figure 3-6: *Detail vs. Speed* – The density of tracked features can be tuned to favor geometric detail or speed. Here we compare the depthmaps (bottom row, columns 2 and 3) generated from the smoothed graph (top row, columns 2-3) to a Kinect depthmap (bottom left) using different settings of detail parameter L (see Section 3.3.1). The input RGB image is shown in the top left and the depthmaps are colored by depth. Changing L from 4 (column 2) to 5 (column 3) results in nearly a 50% speedup (see Table 3.1).

Next, we estimate an inverse depth mean ξ_f and variance σ_f^2 for each $f \in \mathcal{F}_k$ by matching a reference patch of pixels around $\mathbf{u}_f$ in future frames using a direct search along the epipolar line. For a given match, we compute an inverse depth *measurement* with mean ξ_z and variance σ_z^2 according to the noise model of Engel et al. [58] and fuse it with the feature's current estimate using standard Bayesian fusion:

$$\xi_f \leftarrow \frac{\xi_f \sigma_z^2 + \xi_z \sigma_f^2}{\sigma_f^2 + \sigma_z^2}, \quad \sigma_f^2 \leftarrow \frac{\sigma_f^2 \sigma_z^2}{\sigma_f^2 + \sigma_x^2}. \tag{3.9}$$

3.3.2 Mesh Construction

We construct our mesh using the set of features $\mathcal{F}_k^* \subseteq \mathcal{F}_k$ whose inverse depth variance is lower than a threshold σ_{max}^2: $\mathcal{F}_k^* = \{f \in \mathcal{F}_k : \sigma_f^2 < \sigma_{max}^2\}$. We project these features into the current camera frame $\mathbf{T}_k^w$ and then compute a 2D Delaunay triangulation of the projected pixel locations using the fast method of Shewchuk [196, 197]. We denote the Delaunay triangulation by $\mathcal{DT}(\mathcal{F}_k^*) = (\mathcal{V}_k, \mathcal{T}_k)$, where $\mathcal{V}_k$ is the set of mesh

vertices and $\mathcal{T}_k$ is the set of triangles. The Delaunay triangulation is optimal in the sense that it maximizes the minimum angle for each triangle in $\mathcal{T}_k$

We denote the feature corresponding to vertex $v \in \mathcal{V}_k$ with $v_f \in \mathcal{F}_k^*$ and let $v_{\mathbf{u}} \in \Omega$ denote the pixel location of v, which we initialize with the projection of f into the current frame k. Letting $\mathbf{K} \in \mathbb{R}^{3\times3}$ represent the intrinsic camera matrix, the functions $\pi(x, y, z) = (x/z, y/z)$ and $\pi^{-1}(\mathbf{u}, d) = \mathbf{K}^{-1}(d \cdot \bar{\mathbf{u}})$ denote the perspective projection function and its inverse for pixel $\mathbf{u} \in \Omega$ given depth d, where $\bar{\mathbf{u}} = \begin{bmatrix}\mathbf{u}^T & 1\end{bmatrix}^T$ denotes a pixel using homogeneous coordinates. The projection of f into the current frame k is then given by:

$$v_{\mathbf{u}} = \pi\left(\mathbf{K}\mathbf{T}_{f_t}^k \pi^{-1}\left(f_{\mathbf{u}}, \xi_f^{-1}\right)\right). \tag{3.10}$$

(Note that de-homogenization is implied for notational simplicity.)

Finally, we assign an inverse depth to each new vertex that we refer to as v_z and initialize it to the feature inverse depth ξ_f for corresponding feature f projected into the current frame. Note that although we perform our triangulation in 2D using the vertex pixel locations, we can project the mesh to 3D using this inverse depth value. We can also obtain a dense inverse depthmap $\xi : \Omega \to \mathbb{R}_+$ by linearly interpolating the inverse depth values of the mesh vertices.

3.3.3 Non-Local Second Order Variational Cost

Now equipped with an inverse depth mesh $\mathcal{DT}(\mathcal{F}_k^*)$, we formulate our non-local, graph-based variational regularizer that will efficiently smooth away noise in the mesh and promote planar structure.

We start with the continuous NLTGV^2-L_1 variational cost for a fully dense inverse depthmap $\xi : \Omega \to \mathbb{R}_+$, which sets the smoothing term $E_{smooth}(\xi)$ to NLTGV^2 defined in Equation (3.6) and the data fidelity term $E_{data}(\xi)$ to the robust L_1 norm in Equation (3.3):

$$E(\xi) = \text{NLTGV}^2(\xi) + \lambda \int_\Omega |\xi(\mathbf{u}) - z(\mathbf{u})| \, d\mathbf{u}. \tag{3.11}$$

Here $z : \Omega \to \mathbb{R}_+$ is our raw, unsmoothed inverse depthmap.

We will approximate this functional over the fully dense ξ using our inverse depth mesh $\mathcal{DT}(\mathcal{F}_k^*)$. We first reinterpret $\mathcal{DT}(\mathcal{F}_k^*)$ as a directed *Delaunay graph* $\mathcal{DG}(\mathcal{F}_k^*) = (\mathcal{V}_k, \mathcal{E}_k)$, with identical vertices $\mathcal{V}_k$ and directed, non-parallel edges $\mathcal{E}_k$ generated from the triangle set $\mathcal{T}_k$ (the direction of each edge is arbitrary). For each vertex $v \in \mathcal{V}_k$, we assign a smoothed inverse depth value that we denote v_ξ and an auxiliary variable $\mathbf{w} \in \mathbb{R}^2$ such that $v_\mathbf{w} = (v_{w_1}, v_{w_2})$. We let $v_\mathbf{x}$ denote $(v_\xi, v_\mathbf{w})$.

The graph version of our L_1 data fidelity term is straightforward to define in terms of $\mathcal{DG}(\mathcal{F}_k^*)$ by replacing the integral over the image domain Ω with a sum over the vertices of $\mathcal{DG}(\mathcal{F}_k^*)$:

$$\int_\Omega |\xi(\mathbf{u}) - z(\mathbf{u})| \, d\mathbf{u} \approx \sum_{v \in \mathcal{V}_k} |v_\xi - v_z| \,, \tag{3.12}$$

where v_z is the inverse depth of feature v_f projected into the current frame.

Discretizing the NLTGV^2 smoothing term over $\mathcal{DG}(\mathcal{F}_k^*)$ simply requires a special setting of the weight functions $\alpha,\ \beta : \Omega \times \Omega \to \mathbb{R}$. In the non-local, variational framework, these functions control the influence of inverse depth values over their spatial neighbors and thus should be defined in terms of the edge set $\mathcal{E}_k$ of $\mathcal{DG}(\mathcal{F}_k^*)$.

For each edge $e \in \mathcal{E}_k$, we denote the associated vertices as $v^i, v^j \in \mathcal{V}_k$ (note again that the edges are directed from i to j). We then assign weights $e_\alpha, e_\beta \geq 0$ to each edge and set the functions α and β to the following:

$$\alpha(\mathbf{u}, \mathbf{v}) = e_\alpha \delta(\mathbf{u} - v^i_\mathbf{u}, \mathbf{v} - v^j_\mathbf{u}) \quad \text{for } e \in \mathcal{E} \tag{3.13}$$

$$\beta(\mathbf{u}, \mathbf{v}) = e_\beta \delta(\mathbf{u} - v^i_\mathbf{u}, \mathbf{v} - v^j_\mathbf{u}) \quad \text{for } e \in \mathcal{E}. \tag{3.14}$$

Setting α, β in terms of delta functions that encode the connectivity in the graph (i.e., the edges $\mathcal{E}_k$) reduces the double integral over Ω in Equation 3.6 to a summation

over $\mathcal{E}_k$:

$$\begin{aligned}\mathrm{NLTGV}^2(\xi) \approx & \sum_{e\in\mathcal{E}_k} e_\alpha \left| v^i_\xi - v^j_\xi - \langle v^i_{\mathbf{w}}, v^i_{\mathbf{u}} - v^j_{\mathbf{u}} \rangle \right| \quad + \\ & \sum_{e\in\mathcal{E}_k} e_\beta \left| v^i_{w_1} - v^j_{w_1} \right| + \sum_{e\in\mathcal{E}_k} e_\beta \left| v^i_{w_2} - v^j_{w_2} \right| \\ = & \sum_{e\in\mathcal{E}_k} \left|\left| \mathbf{D}_e(v^i_{\mathbf{x}}, v^j_{\mathbf{x}}) \right|\right|_1 .\end{aligned} \tag{3.15}$$

Here $\mathbf{D}_e$ is a linear operator that acts on the vertices corresponding to edge e:

$$\mathbf{D}_e(v^i_{\mathbf{x}}, v^j_{\mathbf{x}}) = \begin{bmatrix} e_\alpha \left(v^i_\xi - v^j_\xi - \langle v^i_{\mathbf{w}}, v^i_{\mathbf{u}} - v^j_{\mathbf{u}} \rangle \right) \\ e_\beta \left(v^i_{w_1} - v^j_{w_1} \right) \\ e_\beta \left(v^i_{w_2} - v^j_{w_2} \right) \end{bmatrix}. \tag{3.16}$$

The final form of our graph-based $\mathrm{NLTGV}^2 - L_1$ cost functional is now

$$E(\mathcal{DG}(\mathcal{F}^*_k)) = \sum_{e\in\mathcal{E}_k} \left|\left| \mathbf{D}_e(v^i_{\mathbf{x}}, v^j_{\mathbf{x}}) \right|\right|_1 + \lambda \sum_{v\in\mathcal{V}_k} |v_\xi - v_z| . \tag{3.17}$$

Note that by defining the NLTGV^2-L_1 variational cost in terms of the $\mathcal{DG}(\mathcal{F}^*_k)$, we can trivially augment and refine the objective by simply adding new vertices and edges to the graph, just as the mesh $\mathcal{DT}(\mathcal{F}^*_k)$ is augmented and refined using incremental triangulations.

3.3.4 Graph Optimization

Having reformulated the NLTGV^2-L_1 cost in terms of graph $\mathcal{DG}(\mathcal{F}^*_k)$, we now apply the optimization method of Chambolle and Pock [31]. We see the summation over $\mathcal{E}_k$ and the summation over $\mathcal{V}_k$ in Equation 3.17 correspond to $F(\mathbf{D}(\mathbf{x}))$ and $G(\mathbf{x})$, respectively, in the Chambolle and Pock objective in Equation 3.7 for $\mathbf{x} = [v_{\mathbf{x}}]$ for $v \in \mathcal{V}_k$, and that we can follow the same optimization approach.

We first generate the saddlepoint problem induced by this new graph-based cost by re-expressing the L_1 norm corresponding to the sum over edges in terms of its convex conjugate, which can in turn be composed of the conjugates of each term in

the sum:

$$\left|\left|\mathbf{D}_e(v^i_{\mathbf{x}}, v^j_{\mathbf{x}})\right|\right|_1 = \max_{e_{\mathbf{q}}} \langle \mathbf{D}_e(v^i_{\mathbf{x}}, v^j_{\mathbf{x}}), e_{\mathbf{q}} \rangle - \delta_Q(e_{\mathbf{q}}). \tag{3.18}$$

Here we have assigned a dual variable $\mathbf{q} \in \mathbb{R}^3$ to each edge, denoted by $e_{\mathbf{q}}$. The indicator term δ_Q is the conjugate L_1^* and is defined as

$$\delta_Q(e_{\mathbf{q}}) = \sum_{i=1}^{3} \delta_{q_i}(e_{q_i}) \tag{3.19}$$

$$\delta_q(q) = \begin{cases} 0 & \text{if } |q| \leq 1 \\ \infty, & \text{otherwise} \end{cases}. \tag{3.20}$$

The NLTGV^2-L_1 saddlepoint problem can now be written in terms of $\mathcal{DG}(\mathcal{F}_k^*)$ as:

$$\min_{v_{\mathbf{x}}} \max_{e_{\mathbf{q}}} \sum_{e \in \mathcal{E}_k} \langle \mathbf{D}_e(v^i_{\mathbf{x}}, v^j_{\mathbf{x}}), e_{\mathbf{q}} \rangle - \delta_Q(e_{\mathbf{q}}) + \lambda \sum_{v \in \mathcal{V}_k} |v_\xi - v_z| \,. \tag{3.21}$$

To optimize Equation 3.21, we first perform semi-implicit, subgradient ascent over $e_{\mathbf{q}}$ for each $e \in \mathcal{E}_k$:

$$e^{n+1}_{\mathbf{q}} = e^n_{\mathbf{q}} + \sigma \mathbf{D}_e(v^i_{\mathbf{x}}, v^j_{\mathbf{x}}) - \sigma \partial \delta_Q(e^{n+1}_{\mathbf{q}}) \tag{3.22}$$

where $\sigma > 0$ is the dual step size and $\partial \delta_Q(\mathbf{q})$ is the subgradient of $\delta_Q(\mathbf{q})$. The ascent step is *semi-implicit* in that the post-step variable $e^{n+1}_{\mathbf{q}}$ appears on both sides of Equation 3.22 and is a *subgradient* step due to the presence of non-differentiable δ_Q.

Moving the $e^{n+1}_{\mathbf{q}}$ terms to the left side of the equation yields:

$$e^{n+1}_{\mathbf{q}} + \sigma \partial \delta_Q(e^{n+1}_{\mathbf{q}}) = e^n_{\mathbf{q}} + \sigma \mathbf{D}_e(v^i_{\mathbf{x}}, v^j_{\mathbf{x}}), \tag{3.23}$$

which we can express in terms of the proximal operator:

$$e^{n+1}_{\mathbf{q}} = \text{prox}_{F^*}\left(e^n_{\mathbf{q}} + \sigma \mathbf{D}_e(v^i_{\mathbf{x}}, v^j_{\mathbf{x}})\right). \tag{3.24}$$

Algorithm 2 NLTGV$^2 - L_1$ Graph Optimization

// Choose $\sigma, \tau > 0$, $\theta \in [0, 1]$.
while not converged **do**
  for each $e \in \mathcal{E}_k$ **do**
    $e_{\mathbf{q}}^{n+1} = \text{prox}_{F^*}\left(e_{\mathbf{q}}^n + \sigma \mathbf{D}_e(v_{\bar{\mathbf{x}}}^i, v_{\bar{\mathbf{x}}}^j)\right)$
  for each $v \in \mathcal{V}_k$ **do**
    $v_{\mathbf{x}}^{n+1} = \text{prox}_G\left(v_{\mathbf{x}}^n - \tau \sum_{e \in \mathcal{N}_{in}(v)} \mathbf{D}_{in}^*(e_{\mathbf{q}}^{n+1}) - \tau \sum_{e \in \mathcal{N}_{out}(v)} \mathbf{D}_{out}^*(e_{\mathbf{q}}^{n+1})\right)$
    $v_{\bar{\mathbf{x}}}^{n+1} = v_{\mathbf{x}}^{n+1} + \theta\left(v_{\mathbf{x}}^{n+1} - v_{\mathbf{x}}^n\right)$

The proximal operator can be interpreted as the equivalent of a gradient step for a non-differentiable function [161].

We next wish to perform semi-implicit subgradient *descent* over $v_{\mathbf{x}}$ for each $v \in \mathcal{V}_k$, but some care must be taken with forming the adjoint operator $\mathbf{D}_e^*$. We observe that the operator $\mathbf{D}_e(v_{\mathbf{x}}^i, v_{\mathbf{x}}^j)$ maps two primal vertex variables (corresponding to the source and destination vertex) to the dual space for each edge. The adjoint must therefore map a single dual edge variable to the space of of two primal variables (again corresponding to the source and destination vertex). Starting from the expression of $\mathbf{D}_e$ in Equation 3.16, we form the adjoint as

$$\mathbf{D}_e^*(e_{\mathbf{q}}) = \begin{bmatrix} e_\alpha e_{q_1} \\ -e_\alpha(e_u^i - e_u^j)e_{q_1} + e_\beta e_{q_2} \\ -e_\alpha(e_v^i - e_v^j)e_{q_1} + e_\beta e_{q_3} \\ -e_\alpha e_{q_1} \\ -e_\beta e_{q_2} \\ -e_\beta e_{q_3} \end{bmatrix}. \tag{3.25}$$

We can then partition the top three rows of $\mathbf{D}_e^*$ into $\mathbf{D}_{in}^*$, which maps $e_{\mathbf{q}}$ to the source primal vertex space, and the bottom three rows of $\mathbf{D}_e^*$ into $\mathbf{D}_{out}^*$, which maps $e_{\mathbf{q}}$ to the destination primal vertex space:

$$\mathbf{D}_e^*(e_{\mathbf{q}}) = \begin{bmatrix} \mathbf{D}_{in}^*(e_{\mathbf{q}}) \\ \mathbf{D}_{out}^*(e_{\mathbf{q}}) \end{bmatrix}. \tag{3.26}$$

The semi-implicit subgradient descent equations for each $v_{\mathbf{x}}$ is then given by

$$v_{\mathbf{x}}^{n+1} = v_{\mathbf{x}}^{n} - \tau \partial G(v_{\mathbf{x}}^{n+1}) - \tau \sum_{e \in \mathcal{N}_{in}(v)} \mathbf{D}_{in}^{*}(e_{\mathbf{q}}^{n+1}) - \tau \sum_{e \in \mathcal{N}_{out}(v)} \mathbf{D}_{out}^{*}(e_{\mathbf{q}}^{n+1}), \tag{3.27}$$

where $\tau > 0$ is the primal step size and the incoming and outgoing edges of v are denoted as $\mathcal{N}_{in}(v)$ and $\mathcal{N}_{out}(v)$, respectively. Solving for $v_{\mathbf{x}}^{n+1}$ in terms of the proximal operator of G then yields

$$v_{\mathbf{x}}^{n+1} = \text{prox}_{G}\left(v_{\mathbf{x}}^{n} - \tau \sum_{e \in \mathcal{N}_{in}(v)} \mathbf{D}_{in}^{*}(e_{\mathbf{q}}^{n+1}) - \tau \sum_{e \in \mathcal{N}_{out}(v)} \mathbf{D}_{out}^{*}(e_{\mathbf{q}}^{n+1}) \right). \tag{3.28}$$

The final step of the Chambolle and Pock method is a simple extragradient step applied to each vertex. The full optimization is summarized in Algorithm 2. By expressing the optimization in terms of the graph $\mathcal{DG}(\mathcal{F}_k^*)$, we can trivially add and remove vertices and edges to the objective as new features are added to $\mathcal{F}_k^*$ and triangulated. In addition, by matching the density of vertices to the geometric complexity of the observed environment, each optimization iteration is both fast to perform and quick to converge (see Figure 3-5 for a comparison between smoothed and unsmoothed meshes). The graph interpretation also provides additional intuition into the optimization, which alternates between operations on the vertices and edges of the graph: smoothing updates are passed from vertices to neighboring edges, and then from the edges back to the corresponding vertices.

3.3.5 Frame-to-Frame Propagation

We propagate the Delaunay graph $\mathcal{DG}(\mathcal{F}_k^*)$ so that the optimized surface is always available in the current frame. At each timestep, we set the vertex location $v_{\mathbf{u}}$ for $v \in \mathcal{V}_k$ to the projection based on the smoothed inverse depth value v_ξ, which we then update to be expressed in the current frame as well. We also set the unsmoothed v_z inverse depth value to the projection of the underlying feature inverse depth ξ_f.

With new vertex locations and potentially new features, we retriangulate to maintain the Delaunay optimality property and add and remove edges to reflect the new

connectivity in the triangulation. We also remove vertices and features that project outside the view of the current camera, although these could be saved and displayed separately. Since the relative transform from frame-to-frame is small for high framerate cameras, and because the optimization is typically able to converge before the next frame is available, the optimization is relatively unaffected by the projection step and we benefit from the smoothed surface always available at the current camera.

3.4 Evaluation

Our implementation of FLaME is written in C++ and makes use of the Boost Graph Library [199] and the Delaunay triangulation implementation from Shewchuk [196, 197]. The primary processing thread handles stereo matching and inverse depth filtering, performs Delaunay triangulations, updates the Delaunay graph, and publishes output. A second thread continuously performs the graph optimization steps outlined in Algorithm 2. The third thread samples new features every N frames ($N = 6$ in our experiments). For all experiments we set the edge weight $e_\alpha = 1/||v^i_{\mathbf{u}} - v^j_{\mathbf{u}}||_2$ (the reciprocal of the edge length in pixels) and $e_\beta = 1$. We set the parameter λ that controls the balance between unsmoothed data and the regularizer between $[0.1, 0.35]$.

3.4.1 Benchmark Datasets

We quantitatively compare the FLaME reconstructions to existing approaches and show how we are able to produce accurate, dense geometry extremely efficiently at the current frame. We interpolate the FLaME meshes to fully dense inverse depthmaps and measure their accuracy and completeness against two existing CPU-based approaches: LSD-SLAM [56] and Multi-Level Mapping (MLM) [80]. We use image and pose sequences from the TUM RGB-D SLAM Benchmark dataset (VGA at 30 Hz) [207] and the EuRoC MAV datasets (WVGA at 20 Hz) [24]. Pose ground truth for both datasets was generated using a motion capture system. Structure ground truth was approximated using an RGB-D sensor for the TUM sequences and a 3D laser scanner for the EuRoC sequences.

Performance on Benchmark Datasets

		DM	RE [%]	AD [%]	Cores	Time [ms]	FPS [Hz]
TUM	LSD	181	18	19	2.5	16	62
	MLM	103	12	32	2.1	17	57
	L=3	**4950**	8.5	**54**	2.0	16	61
	L=4	**4950**	6.8	**54**	1.7	7.3	136
	L=5	**4950**	**6.6**	51	**1.4**	**4.2**	**236**
EuRoC	LSD	874	18	17	1.6	16	61
	MLM	734	17	25	1.0	14	69
	L=3	**12595**	12	36	1.3	13	78
	L=4	**12595**	11	**37**	1.2	7.0	143
	L=5	**12595**	**10**	33	**0.8**	**4.3**	**230**

Table 3.1: We evaluate FLaME with various settings of detail parameter L (see Section 3.3.1) on two benchmark datasets [207, 24]. FLaME produces depthmaps (DM) with both lower relative inverse depth error (RE) and a higher density of accurate inverse depths (AD), while taking less processing time per frame (Time) and using less CPU load (Cores), than state-of-the-art approaches LSD-SLAM [56] and MLM [80]. Refer to Section 3.4.1 for a more detailed description of the metrics and experimental setup.

The pose tracking and SLAM backend modules of LSD-SLAM and MLM were disabled in the experiments so that all three algorithms used the motion capture poses and all performance differences can be attributed to the different depth estimation techniques.

All metrics were captured on a desktop computer with an Intel Core i7 4820K 3.7 GHz CPU. We use three sequences from the TUM dataset (long_office_household, structure_texture_far, and nostructure_texture_near_withloop) and the 6 sequences from the EuRoC dataset with structure ground truth (V1_01, ..., V2_03).

We report two main measures for depthmap accuracy and completeness: the *relative inverse depth error* and the *density of accurate depth estimates*. The relative inverse depth error is the error in inverse depth relative to ground truth, averaged over all pixels and depthmaps. The density of accurate depth estimates is the fraction of inverse depth estimates that are within 10% of ground truth for each depthmap, averaged over all depthmaps. We also report both runtime per frame and CPU load

Accurate Inverse Depth Density [%]

	Sequence	LSD	MLM	L=3	L=4	L=5
TUM	fr3_loh	18.9	30.4	**48.1**	47.9	44.1
	fr3_nstn	16.5	30.6	52.5	**53.7**	51.9
	fr3_stf	26.8	47.8	**72.6**	72.0	69.1
EuRoC	V1_01	18.2	26.4	36.6	**37.8**	34.4
	V1_02	18.3	27.9	39.2	**40.5**	37.8
	V1_03	11.0	17.0	26.4	**27.0**	24.7
	V2_01	25.9	39.3	**48.1**	46.9	41.6
	V2_02	20.5	28.9	**37.2**	**37.2**	32.2
	V2_03	11.5	19.0	27.8	**28.9**	26.7

Table 3.2: Here we present the fraction of inverse depths per depthmap that are within 10% of groundtruth for each benchmark video sequence [207, 24] for LSD-SLAM [56], MLM [80], and FLaME with different settings of parameter L. FLaME outperforms the competing approaches, with $L = 4$ providing a nice balance between the number of vertices per depthmap and the amount of smoothing performed.

over the datasets.

The results are summarized in Table 3.1 and Table 3.2 and as can be seen, FLaME produces dense geometry more accurately and efficiently than the competing approaches. On the EuRoC sequences FLaME with $L = 5$ produces reconstructions at up to 230 Hz using less than one Intel i7 4820K CPU core and achieves the lowest relative inverse depth error across the different systems. Although finer settings of $L = 3$ and $L = 4$ produce slightly better density metrics, as expected, they fair slightly worse in terms of relative inverse depth error compared to $L = 5$. We believe the primary reason for this unintuitive result is that the graph optimization takes longer to converge for these parameter settings given the greater number of vertices and edges. Since the mesh takes longer to converge, the camera moves before the mesh can settle, resulting in higher inverse depth error. Initial depthmap convergence is very fast, however, usually within the first second of operation. In addition, FLaME produces accurate meshes at the current frame, while the competing approaches (which are both keyframe based) produce reconstructions far more infrequently, which is particularly dangerous for mobile robot navigation.

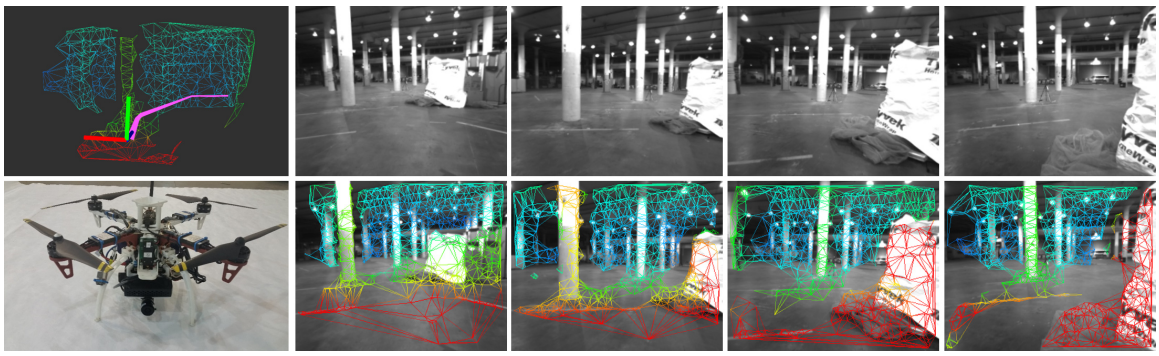


Figure 3-7: *Flight Experiments* – FLaME can be used to enable online perception for autonomous navigation. We conducted indoor and outdoor flight experiments running FLaME onboard a small micro-aerial vehicle (MAV) (bottom left) with a forward-facing camera flying at up to 3.5 meters-per-second. The image on the top left shows the collision-free trajectory (pink) that is generated to navigate around a pillar obstacle. The images to the right show the inverse depth meshes as the vehicle approaches the obstacle field.

We also experimented with corrupting the ground truth positions with additive Gaussian noise to characterize the effect of pose error. With no artificial noise, the density of accurate inverse depths for the TUM `fr3_stf` sequence is 71%. However, with translation noise with a standard deviation of 1 cm, this density drops to 30%, which demonstrates the importance of accurate pose information on the depth estimation process.

3.4.2 Flight Experiments

We also provide results from experiments with FLaME running completely onboard, in-the-loop on a small autonomous quadrotor flying at up to 3.5 meters-per-second.

The quadrotor (see Figure 3-7) weighed 3 kg and was equipped with a Point Grey Flea 3 camera running at 320×256 image resolution at 60 Hz, an inertial measurement unit (IMU), and a laser altimeter. The pose of the camera was provided by an external visual-inertial odometry pipeline [202]. Collision-free motion plans were generated using A* on a 2D occupancy grid updated using slices from the FLaME meshes. All computation was performed onboard an Intel Skull Canyon NUC flight computer, with no prior information provided to the robot.

We flew the vehicle through an indoor warehouse environment and an outdoor for-

Timing and Load on Autonomous MAV		
Metric	Indoor	Outdoor
Vehicle Speed [m/s]	2.5	3.5
Depthmaps	803	1046
CPU Load [cores]	1.6	1.7
Runtime [ms]	9.4	11
Peak FPS [Hz]	106	91

Table 3.3: FLaME is efficient enough to allow for real-time perception onboard small, computationally constrained micro-aerial vehicles (MAVs). We flew our quadrotor fully autonomously in both indoor and outdoor environments with no prior information and used geometry from FLaME to plan around obstacles online.

est with obstacles that the vehicle had to plan around using perception from FLaME. Runtime and load metrics on the flight computer are summarized in Table 3.3. Even on the flight computer, FLaME was still able to produce dense reconstructions at over 90 Hz (the detail parameter was set to $L = 3$ to account for the lower image resolution), with sufficient accuracy to plan around obstacles.

3.4.3 Improvements

While FLaME performs well on both the benchmark datasets and flight experiments described above, there are a few ways that performance might be improved. First, FLaME relies on accurate poses in order to reliably track features and estimate depths. As noted in the preceding sections, the density of accurate depth estimates drops drastically with the addition of noise on the poses. Accounting for these inevitable pose errors in some way would significantly improve the robustness of the full system. Second, the feature tracking paradigm as described in Section 3.3.1, while optimized for speed, does fail for a sizable fraction of detected features due to image noise and adversarial camera motion. Although in practice these failed feature tracks can simply be removed and new features sampled, more reliably tracking potential mesh vertices across frames would lead to more stable meshes. Lastly, these potential mesh vertices are chosen by dividing the image domain into a grid based on a user-defined parameter L and sampling easily trackable pixels within each grid cell. This parameter L needs

to be tuned for a particular image resolution and environment, which hampers the method's overall ability to generalize to new scenes. More intelligently selecting features, for example by learning which high-gradient pixels correspond to important geometric structure, could both obviate the need for tuning by the end user as well as reduce the number of required depth estimates even further.

3.5 Conclusion

In this chapter, we presented a novel dense monocular depth estimation algorithm capable of reconstructing geometric meshes on computationally constrained platforms. FLaME exploits the prior information that many environments of interest can be well-described using piecewise planar triangular meshes. Since the number of depth estimates needed for a given mesh is small compared to the number of pixels in an image, our method is significantly faster than every-pixel methods. In addition, by reformulating the reconstruction problem as a variational smoothing problem over a time-varying Delaunay graph, we can apply sophisticated spatial regularization techniques that would be intractable otherwise using only a standard CPU, allowing for both for efficient, incremental smoothing of noisy depth estimates and low-latency mesh estimation.

Chapter 4

Metric Monocular SLAM using Learned Scale Factors

This chapter proposes an approach for monocular SLAM that generates *metrically-scaled* solutions without requiring additional sensors or compute. Monocular SLAM, where both egomotion and environmental structure are estimated from a single moving camera, has undergone tremendous advances over the past twenty years. Early filter-based approaches [41] have quickly evolved to sophisticated, hierarchical, factor graph-based optimizations, such as the methods of Klein and Murray [110], Mur-Artal et al. [145], and Engel et al. [55].

As outlined in Chapter 1, however, monocular cameras are fundamentally *bearing* sensors that only observe the *angle* of incident light on the sensor plane, not the metric range to the structure from which the light was reflected. Objects that are different sizes may actually appear identical when projected onto the image plane. For example, a toy RC car and a real car may be indistinguishable when placed the appropriate distances away from the camera. The *geometric* content of the images — meaning the raw pixel intensities absent of any higher-level information regarding *what* is being observed — would be identical in each scenario. This means that conventional monocular SLAM systems, which rely solely on the geometric image content, can only estimate camera egomotion and environmental structure up to an arbitrary scale factor. Additional information must be exploited to resolve the metric

scale of the solution. This lack of metric scale observability not only means that monocular SLAM solutions can be arbitrarily scaled, but also means that the solutions can *drift* in scale over time, especially when new areas of the scene are uncovered. Each time the camera rotates quickly to observe new parts of the scene, the scale of the SLAM solution can change significantly since the old and new portions of the map may only have a small number of landmarks in common. While leveraging priors over the camera's altitude or the size of known objects in the scene [69] to uncover the metric scale is possible, the current most popular technique is to fuse the images with an inertial measurement unit (IMU), which measures linear accelerations and angular velocities at metric scale.

Though significant progress has been made on this front [143, 124, 64, 202, 173], these existing visual-inertial SLAM approaches have a number of drawbacks. Beyond the additional hardware that must be calibrated and time-synchronized, these algorithms are difficult to implement, often require expert parameter tuning, are extremely sensitive to errors in the accelerometer biases, and require high-acceleration motion to excite the IMU and make scale observable. (The last point is particularly troublesome for mobile robot navigation as it can significantly complicate the motion planning problem.)

We note, however, that real-world scenes exhibit a surprising amount of regularity and structure that is captured in monocular imagery beyond simply the geometric content. For instance, the 3D geometry, color, and texture of a scene are all highly correlated with each other: walls are usually perpendicular to the ground and white, roads are usually flat and grey, stop signs are usually red, trees are usually green, and so on. Furthermore, semantically meaningful objects like cars, people, and doors only appear in a specific range of sizes and orientations. This higher-level (or semantic) information content, which we can interpret as a prior over the distribution of environments and objects, contains information about the metric scale of the scene that we can exploit. For example, if a car is observed by a monocular camera, we can deduce its depth based on how large it appears in the image.

In recent years, methods that apply end-to-end deep learning techniques to the

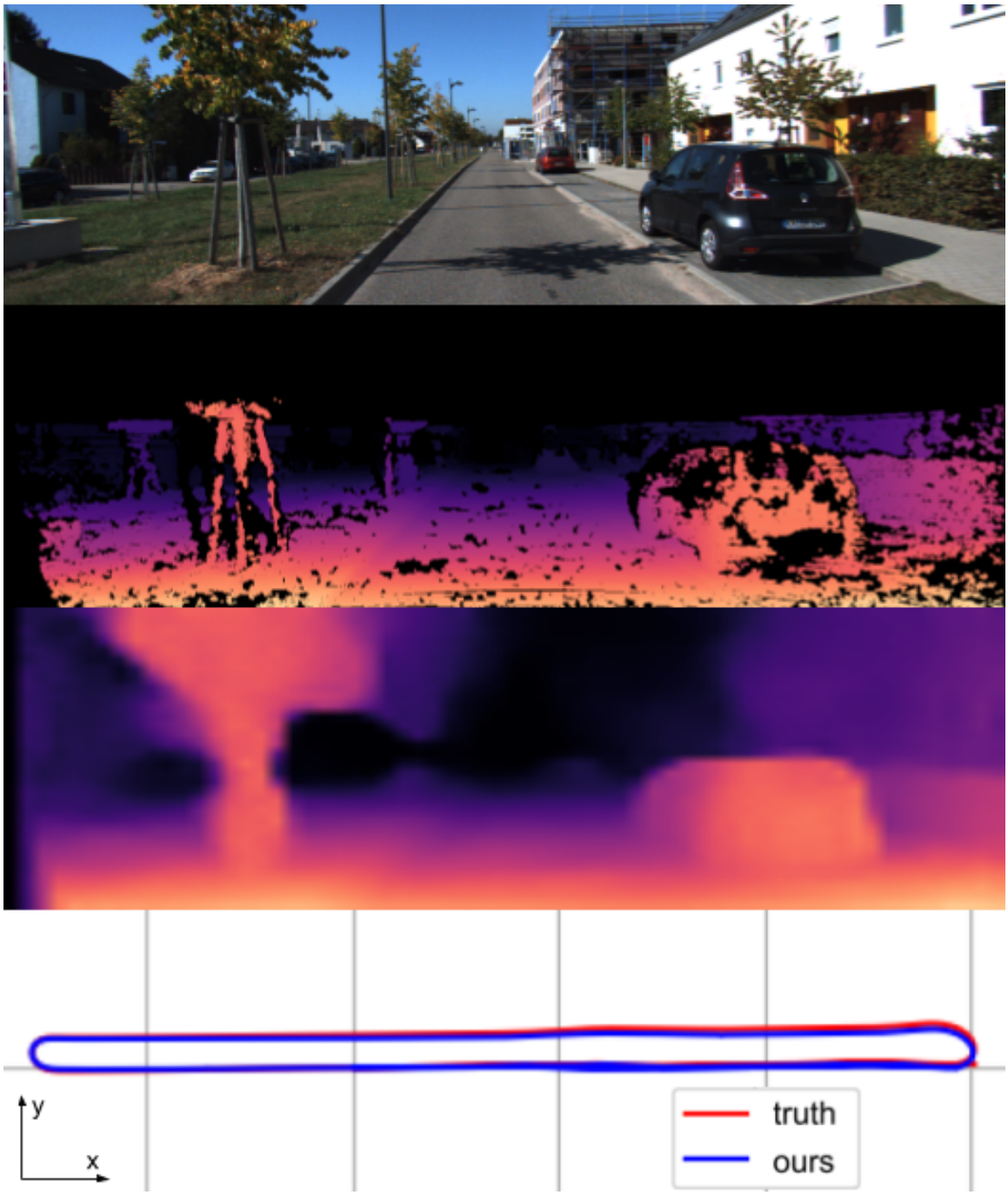


Figure 4-1: *Metric Monocular SLAM* – Our method is capable of estimating metric camera motion from monocular images without additional sensors or hardware acceleration by leveraging depth predictions from a small neural network. *Top row*: Input image from the KITTI dataset [72]. *Second row*: Groundtruth depths from LIDAR scans. *Third row*: Coarse depthmap predicted with our network. *Bottom row*: Resulting metrically scaled trajectory (blue) versus the groundtruth (red).

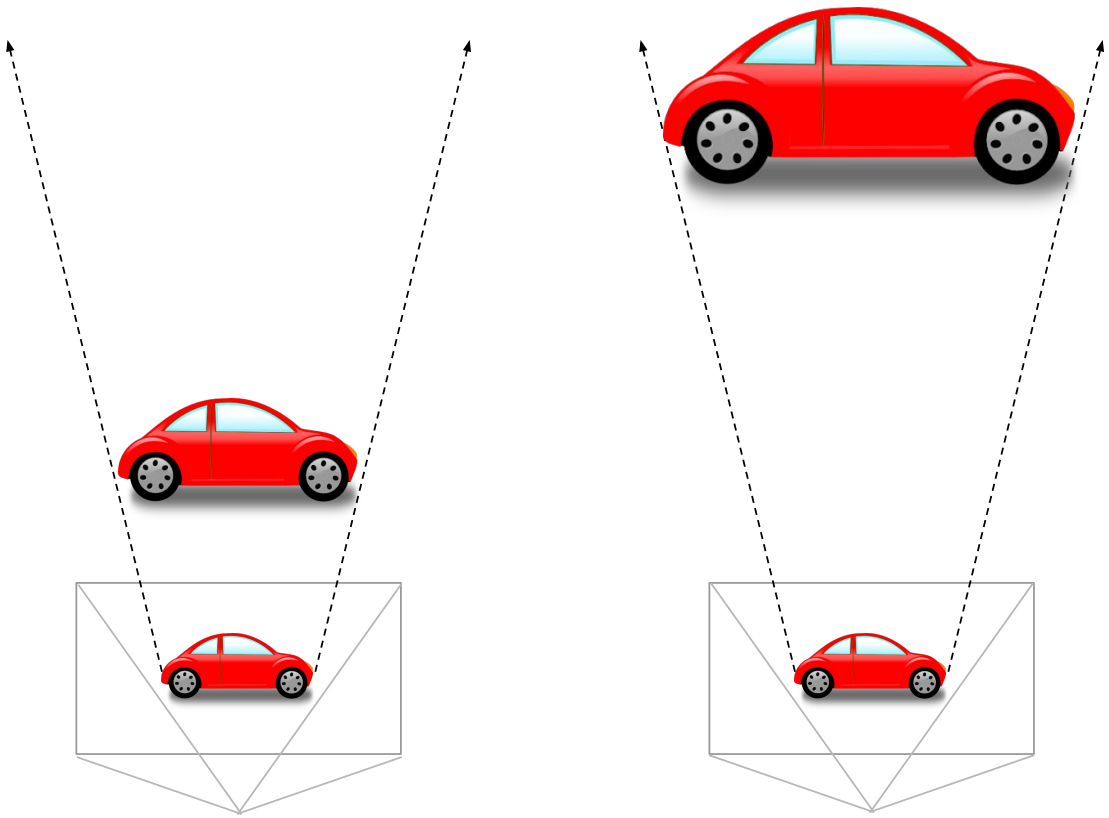


Figure 4-2: *Monocular Scale Ambiguity* – Monocular cameras are *bearing* sensors that detect the *direction* of incident light. As a consequence, they cannot observe the *metric scale* of the environment purely from the pixel intensities alone. Objects of different sizes may appear identical when projected onto the image plane. Consider the example above of a small toy car and a larger real car. Placing the toy car close to the camera and the real car far from the camera results in identical images. The metric scale of the scene is therefore ambiguous using solely image data.

monocular SLAM problem have been developed that take advantage of this additional structure. Spurred by the rapid adoption of deep, convolutional neural networks (CNNs) for a variety of computer vision tasks, these techniques extract higher-level information — such as metric scale — from large datasets of images, encode it in the weights of a CNN, and then use the CNN to drive localization and mapping [230, 126, 34, 99, 241] (see Section 2.4.5 for more information). While these end-to-end SLAM systems show promise and can output scaled solutions, however, their tracking performance still lags behind geometric approaches and they require GPU acceleration to perform inference.

Given that the *relative geometry* (i.e., the geometry up to scale) of the monocular SLAM problem is directly observable from the image data, we believe a more targeted application of machine learning (which does not ignore directly observable quantities in the sensor data) will ultimately lead to more robust systems.

To that end, we propose a monocular SLAM solution that combines metric information that can be inferred using a neural network with the state-of-the-art in factor graph-based geometric SLAM. We first train a small CNN to regress metric depth from monocular images given calibrated stereo frames as training data. Unlike existing learned depth estimation approaches [75, 76, 164, 84, 169], our technique leverages the insight that when used to estimate scale, these learned depth predictions need only be coarse in image space. This reduction in resolution allows us to shrink our network to the point that performing inference on a standard CPU becomes computationally tractable. Simply downsampling the input images and training the network to minimize photoconsistency between stereo training pairs yields inaccurate depths, however, as the disparity between the left and right images decreases with image resolution. We make several improvements to our network architecture and training procedure to address this lack of depth observability, while keeping the efficiency that comes with using coarse input images.

First, although coarse images are used as input to the network at test time, we train on full resolution images and compute additional photoconsistency loss terms in a fine-to-coarse manner. Incorporating these loss terms at training time means that photoconsistency errors that are only observable at fine image scales can still be used to learn the disparity at the coarser image scales that we care about.

Second, we provide an additional supervision signal to the network by estimating a full-resolution disparity map using conventional block-matching stereo. Although these directly-computed disparity maps can be sparse and noisy, they nonetheless provide a loss signal that can allow the network to learn the correct disparity values at coarse image scales where photoconsistency may be insufficient.

These improvements allow us to estimate spatially coarse, but depth-accurate predictions in only 30 ms per frame on a standard CPU. At runtime we divide the SLAM

problem into a local visual odometry (VO) module and a global pose graph module (see Figure 4-4). The local VO module performs conventional monocular SLAM over a small sliding window of keyframes, while the global pose graph incorporates metric depth measurements from the network to both constrain the solution scale and eliminate scale drift. After each iteration of solving for the camera poses and landmark positions, the current scale of the global pose graph can be used to warp the local VO so that metrically scaled geometry is available for the most recent image.

Our method has notable advantages over existing approaches. Unlike inertial-based systems, we do not require extra sensors or special motion to generated scaled outputs. Unlike end-to-end learning-based systems, we do not ignore the observable epipolar geometry present in the live images and can take advantage of factor graph optimization. Unlike learned monocular depth estimation methods, we target the network specifically for scale estimation and can thus shrink the network to allow for fast inference without hardware acceleration. We show compelling results on the KITTI benchmark dataset in addition to real-world experiments with a handheld camera.

4.1 Related Work

Applying machine learning to solve aspects of the monocular SLAM problem has seen a recent resurgence in the literature due to the increasing expressive power of deep neural networks. While some methods target the monocular visual odometry problem specifically [34, 99], single-view depth estimation has seen an explosion of progress in the last decade. Initial methods used explicit supervision from ground truth models or LIDAR scans to regress depth from images [186, 127], while more recent approaches use self-supervision in the form of calibrated stereo imagery in order to regress depth [75, 169]. These self-supervised networks are increasingly augmented with separate pose estimation modules so that they can be applied directly to monocular video instead of calibrated stereo imagery [241, 230, 126, 76].

Our approach is most similar to the hybrid learned/geometric methods of [213,

163, 236], which combine learned priors with geometric SLAM. CNN-SLAM [213] uses a CNN to predict a depthmap for each frame in a keyframe SLAM graph, which is then iteratively refined and fused into a global map. DPC-Net [163] trains a network to provide *corrections* to an existing visual odometry pipeline. DVSO [236] predicts a hypothetical stereo image from a monocular image and then uses the pair of images to drive a stereo visual odometry system [229]. We also take inspiration from the methods of Strasdat et al [203] and Engel et al. [56], which leverage pose graph optimization over the group of similarity transforms in order to reduce scale drift.

4.2 Method

In this section, we will briefly clarify notation before describing the algorithmic building blocks of our method in detail. As in previous chapters, we will represent the image taken at time k by the function $I_k : \Omega \to \mathbb{R}$ over the pixel domain $\Omega \subset \mathbb{R}^2$. $\mathbf{K} \in \mathbb{R}^{3\times3}$ once again will denote the intrinsic camera parameters. We represent the pose of the camera at time k relative to frame j by $\mathbf{T}_k^j \in \mathbb{SE}(3)$. An element of the group of 3D similarity transforms Sim(3) is denoted by $\mathbf{S}_k^j$ and can be represented as a homogeneous transform matrix in $\mathbb{R}^{4\times4}$ as follows:

$$\mathbf{S}_k^j = \begin{bmatrix} s\mathbf{R} & \mathbf{t} \\ 0 & 1 \end{bmatrix} \tag{4.1}$$

for rotation matrix $\mathbf{R} \in \mathbb{SO}(3)$ and translation vector $\mathbf{t} \in \mathbb{R}^3$. Once more, the perspective projection function is denoted by $\pi(x, y, z) = (x/z, y/z)$. Vectors represented in homogeneous coordinates are denoted by $\bar{\mathbf{x}} = (\mathbf{x}, 1) \in \mathbb{R}^{n+1}$ for $\mathbf{x} \in \mathbb{R}^n$.

Given rectified stereo images, we let $D_l : \Omega \to \mathbb{R}$ represent the disparity map that warps the right image I_r to the left image I_l such that $I_l(u) = I_r(u + D_l(u))$. Similarly we let D_r represent the disparity map that warps the left image to the right image. A disparity map D can be converted to inverse depthmap Z given the horizontal focal length f_x of the cameras and the horizontal baseline B as $Z(u) = D(u)/(Bf_x)$.

4.2.1 Single-view Depth Regression

Following the self-supervised approach of Godard et al. [75], we estimate the metric inverse depthmap Z for a given image by treating it as the left image I_l of a calibrated stereo pair and training a CNN to predict the corresponding right image I_r via the disparity maps D_l and D_r. We therefore wish to learn a function f with parameters θ that maps I_l to I_r (and vice versa) via D_l and D_r:

$$\begin{aligned} (D_l, D_r) &= f(I_l; \theta) \\ I_l(u) &= I_r(u + D_l(u)) \\ I_r(u) &= I_l(u + D_r(u)). \end{aligned} \tag{4.2}$$

With a dataset of stereo pairs $\mathcal{D} = \{I_l, I_r\}_j$ and a loss function l that measures the quality of the predictions, we can estimate the parameters θ by solving the following optimization problem:

$$\theta^* = \arg\min_{\theta} \sum_{I_l, I_r \in \mathcal{D}} l(f(I_l; \theta), I_l, I_r). \tag{4.3}$$

Network Architecture

We choose f to be a convolutional neural network for the power of these models to capture complex patterns in image data, while still being practical to train. Specifically, we base our network on the pyramidal model detailed by Poggi et al. [169] augmented with residual blocks [90]. This network significantly reduces the number of parameters required to regress disparity compared to the seminal approaches of [241, 75, 126]. Since we are interested in scale estimation, however, we can further simplify the network architecture. The model is built using three main building blocks: a feature extractor block, a disparity estimator block, and an upsampler block repeated at multiple image scales as shown in Figure 4-3.

The feature extractor block is built using four 3×3 convolutional layers with ReLU activations [149] arranged with skip connections into two residual blocks [90]. The first convolutional layer in the block also performs downsampling with a stride

of 2. The number of filters depends on the image scale. The disparity estimator block is composed of a series of four 3×3 residual layers, with the first three using ReLU activations and the final output layer using a sigmoid activation to ensure positive disparities. The number of filters per layer in this block is 96, 64, 32, and 8, respectively. The upsampler block is simply a transpose convolution with stride 2 with the same number of filters as the input.

Given an input image of a particular resolution, we define 7 pyramid levels of interest: L^0 (the base image) through L^6 (the coarsest resolution). To ensure computational efficiency on constrained hardware, we only extract features from L^3 to L^6 by stacking feature extractor blocks with 16, 32, 64, and 128 filters at each respective scale. At L^6, we attach a disparity estimator block directly to the feature extractor outputs to yield D^6 – the disparity map for the coarsest image scale. For L^3 to L^5 we take the features from each scale and concatenate them with those from the next coarsest scale after passing them through an upsampler block. These concatenated features are then fed into a disparity estimator block to generate $D^3 \dots D^5$. The finest disparity maps $D^0 \dots D^2$ are generated by simple bilinear interpolation for efficiency. This simple model is expressive enough to learn high-quality (but coarse) disparity maps despite having only 2.3 million trainable parameters.

Loss Function

We train our network to regress disparities by minimizing a loss function composed of four terms: a photoconsistency loss l_p, a left-right consistency loss l_{lr}, a disparity regularization term l_r, and a supervision term l_s, defined at each image scale:

$$\begin{aligned} l(f(I_l;\theta), I_l, I_r) = \sum_{i=0}^{6} \lambda_p \left(l_p(I_l^i, \hat{I}_l^i) + l_p(I_r^i, \hat{I}_r^i) \right) + \lambda_{lr} l_{lr}(D_l^i, D_r^i) + \\ \lambda_r \left(l_r(D_l^i) + l_r(D_r^i) \right) + \lambda_s \left(l_s(D_l^i) + l_s(D_r^i) \right). \end{aligned} \tag{4.4}$$

The photoconsistency term l_p measures the photometric error between the input image I and the image predicted using the estimated disparity maps $\hat{I}$. Following Godard et al. [75], we set l_p to be a combination of structural similarity SSIM and a

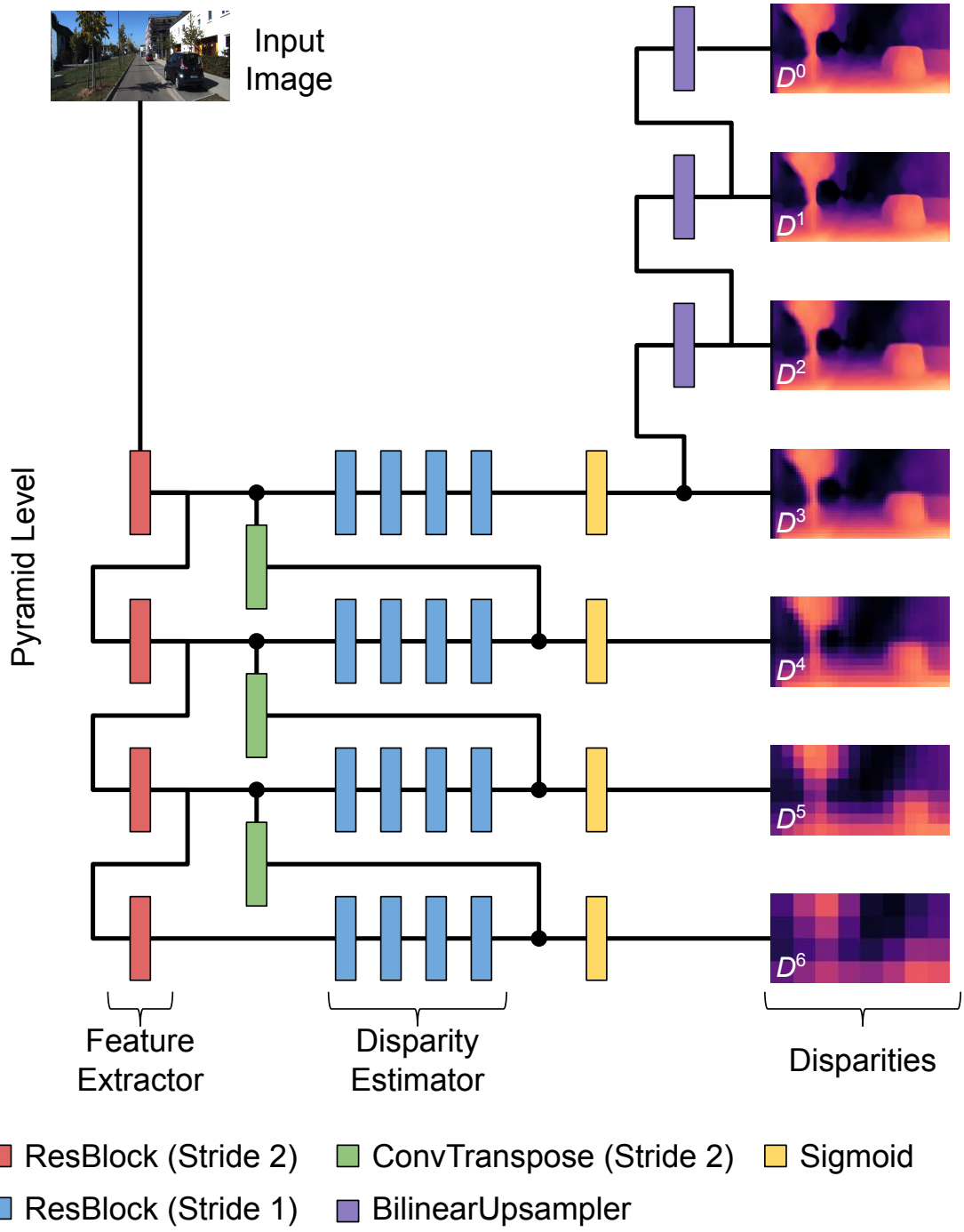


Figure 4-3: *Network Architecture* – Our network follows the pyramidal structure of [169] with a series of feature extractor blocks and disparity estimator blocks at several image scales. Each processing block is comprised of residual blocks [90] with the number of filters varying depending on the pyramid level.

simple L_1 error:

$$l_p(I, \hat{I}) = \frac{1}{N} \sum_{\mathbf{u} \in \Omega} \alpha \frac{1 - \text{SSIM}(I(\mathbf{u}), \hat{I}(\mathbf{u}))}{2} + (1 - \alpha)|I(\mathbf{u}) - \hat{I}(\mathbf{u})|, \tag{4.5}$$

where N is the number of pixels in the image at a given scale and $\alpha > 0$ controls the weighting between the SSIM and L_1 terms.

The left-right consistency loss l_{lr} measures the discrepancy between the left and right disparity maps after warping them into each other:

$$l_{lr}(D_l, D_r) = \frac{1}{N} \sum_{\mathbf{u} \in \Omega} |D_l(\mathbf{u}) - D_r(\mathbf{u} + D_l(\mathbf{u}))| + |D_r(\mathbf{u}) - D_l(\mathbf{u} + D_r(\mathbf{u}))|. \tag{4.6}$$

The disparity regularization term l_r penalizes non-smooth disparity maps where the image gradient is low:

$$l_r(D) = \frac{1}{N} \sum_{\mathbf{u} \in \Omega} e^{-||\nabla_x I(\mathbf{u})||} \nabla_x D(\mathbf{u}) + e^{-||\nabla_y I(\mathbf{u})||} \nabla_y D(\mathbf{u}). \tag{4.7}$$

This loss is applied to both the left and right disparity maps.

The final supervision loss l_s measures the Huber error between the estimated disparity maps D_l and D_r and disparity maps generated using traditional block-matching B_l and B_r:

$$l_s(D) = \frac{1}{N} \sum_{\mathbf{u} \in \Omega} ||D(\mathbf{u}) - B(\mathbf{u})||_\epsilon, \tag{4.8}$$

where $\epsilon > 0$ is the parameter that governs when the Huber norm switches between squared and linear error.

4.2.2 Local Visual Odometry

Our local monocular VO pipeline is divided into a frontend module that builds a factor graph $\mathcal{G}_L = (\mathcal{V}_L, \mathcal{F}_L)$ from the raw image stream and a backend module that optimizes variables $\mathcal{V}_L$ (see Figure 4-4). $\mathcal{V}_L$ contains keyframe poses $\mathcal{K}_L$ and landmark map $\mathcal{M}_L$. The factor set $\mathcal{F}_L$ is composed of reprojection factors r_p that link keyframes

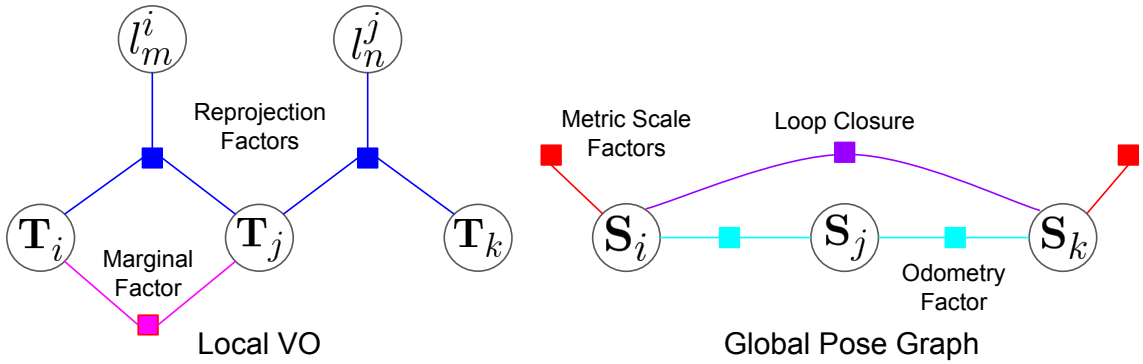


Figure 4-4: *Factor Graphs* – Our monocular SLAM backend is composed of two factor graphs: a local visual odometry (VO) graph (left) and a global pose graph (right) The local VO module estimates unscaled camera poses and landmarks, while the global pose graph fuses marginalized keyframes from the local VO module with metric scale factors generated by our neural network.

and landmarks.

Frontend

At each new frame I_k, we detect corners using the method of [198] and track them from frame to frame using Lucas-Kanade [133, 11]. When the average pixel motion of the features between the last keyframe and current image exceeds a threshold, we create a new keyframe with pose $\mathbf{T}_k^W \in \mathbb{SE}(3)$, initialized by running motion-only Bundle Adjustment with respect to the existing map $\mathcal{M}_L = \{l_j^i\}$ comprised of landmarks l_j^i. Each landmark l_j^i is parameterized by its pixel location $\mathbf{u}_j \in \Omega$, its inverse depth $\xi_j \in \mathbb{R}^+$, and the frame it was detected in i. Features that were detected in the current frame k are initialized as new landmarks. Observations of pre-existing landmarks in k are added to $\mathcal{F}_L$ as reprojection factors.

Reprojection Factors

Each time a landmark l_j^i is observed in a new keyframe k, we insert a reprojection factor into the graph that constrains the landmark's inverse depth and the poses of the keyframes in which it was observed. Suppose that landmark l_j^i is observed in keyframe k at pixel location $\mathbf{p}_j \in \Omega$. The reprojection error r_p from this observation

is given by

$$r_p(\mathbf{T}_i^W, \mathbf{T}_k^W, l_j^i) = \pi\left(\mathbf{K}\mathbf{T}_W^k \mathbf{T}_i^W \mathbf{K}^{-1}\bar{\mathbf{u}}_j/\xi_j\right) - \mathbf{p}_j. \tag{4.9}$$

Backend

After a new keyframe is initialized and all new factors are added to the graph, we enqueue a solve operation that will take place in a background thread. The total cost represented by $\mathcal{G}_L$ can be written as

$$E_L(\mathcal{K}_L, \mathcal{M}_L) = \sum_{i,j,k\in\mathcal{F}_L} ||r_p^{ijk}\left(\mathbf{T}_i^W, \mathbf{T}_j^W, l_k^i\right)||_\epsilon \tag{4.10}$$

where $||\cdot||_\epsilon$ represents the Huber norm with parameter $\epsilon > 0$.

This objective function is a (robust) sum of squared residuals, which we can optimize using the Levenberg-Maquardt algorithm [135, 156].

Marginalization

We marginalize out old keyframes to ensure real-time processing. Suppose we wish to marginalize out keyframe k and its child landmarks. We will denote this set of variables by $v_y = \{\mathbf{T}_k^W, \{l_j^k\}\}$. We then find the factors $\mathcal{F}_{sep}$ that connect v_y to $\mathcal{G}_L$. Let v_x denote the variables in $\mathcal{V}_L$ that are connected to $\mathcal{F}_{sep}$, but are not in v_y. The variables v_x and v_y and the factors $\mathcal{F}_{sep}$ form a subgraph $\mathcal{G}_{sep} \subset \mathcal{G}_L$. The cost associated with this subgraph is given by

$$E_{sep}(v_x, v_y) = \sum_{i,j,k\in\mathcal{F}_{sep}} ||r_p\left(\mathbf{T}_k^W, \mathbf{T}_j^W, l_k^k\right)||_\epsilon. \tag{4.11}$$

Linearizing r_p around the current estimates of v_x and v_y yields a quadratic cost in the tangent space of v_x and v_y. We can then eliminate the v_y component of the cost via the Schur complement, leaving a quadratic factor $\mathcal{F}_\delta$ on v_x.

With v_y eliminated, we remove the variables v_y and the factors $\mathcal{F}_{sep}$ from $\mathcal{G}_L$ and add the marginal factor $\mathcal{F}_\delta$ onto the remaining variables v_x. The marginalized

keyframe $\mathbf{T}_k^W$ and landmarks l_j^k are then passed to the global pose graph (see Section 4.2.3).

4.2.3 Global Pose Graph

Once a local keyframe k and its child landmarks $\{l_j^k\}$ are marginalized out of the local VO module, we freeze the landmark inverse depth values ξ_j and insert a new pose $\mathbf{S}_k^W \in \text{Sim}(3)$ into a global pose graph $\mathcal{G}_G = (\mathcal{V}_G, \mathcal{F}_G)$. Here $\mathcal{V}_G$ contains only Sim(3) pose variables. The factor set $\mathcal{F}_G$ contains relative odometry factors, loop closure factors, and scale factors.

Relative Odometry Factors

We link the newly inserted pose variable k to the rest of $\mathcal{V}_G$ using a relative odometry factor r_{odom} between k and the most recent global pose variable j. Let $\hat{\mathbf{S}}_k^j$ denote the relative transform between k and j when k is marginalized out of the local window. r_{odom} is then given by

$$r_{odom}(\mathbf{S}_k^W, \mathbf{S}_j^W) = \log\left(\mathbf{S}_W^j \mathbf{S}_k^W \hat{\mathbf{S}}_j^k\right), \tag{4.12}$$

where $\log : \text{Sim}(3) \to \mathfrak{sim}(3)$ denotes the logarithmic map between Sim(3) and its Lie algebra $\mathfrak{sim}(3)$. We let $\mathcal{F}_{odom}$ denote the set of all odometry factors.

Loop Closure Factors

When local keyframe k is marginalized out, we compare I_k to the images corresponding to the poses in $\mathcal{V}_G$ using a bag-of-words (BoW)-based descriptor vector [148] generated with ORB features [182]. If a match is detected, we then match the features across the two frames and use the matches to estimate the relative Sim(3) transform between the two poses. A loop factor r_{loop} is then added to $\mathcal{F}_G$ with the same form as r_{odom}. We let $\mathcal{F}_{loop}$ denote the set of all loop factors.

Scale Factors

We employ the inverse depth estimation network described in Section 4.2.1 to generate scale measurements for a pose in the global graph $\mathcal{G}_G$. The child landmarks l_j^i of a pose $\mathbf{S}_i^W$ have arbitrarily scaled inverse depths ξ_j. Using our inverse depth estimation network, we can estimate the metric inverse depth z_j for each landmark. The ratio of the unscaled inverse depth ξ_j to the metric inverse depth z_j is an estimate of the scale s_i of the pose $\mathbf{S}_i^W$. We can add these measurements as unary factors r_s on the scale variable s_i:

$$r_s(\mathbf{S}_i^W) = s_i - \xi_j / z_j. \tag{4.13}$$

We let $\mathcal{F}_s$ denote the set of all scale factors.

Backend

The total cost represented by $\mathcal{G}_G$ can then be written as:

$$\begin{aligned} E_G(\mathcal{V}_G) = & \sum_{j,k \in \mathcal{F}_{odom}} \left|\left| r_{odom}(\mathbf{S}_k^W, \mathbf{S}_j^W) \right|\right|^2_{\Sigma_{odom}} + \\ & \sum_{j,k \in \mathcal{F}_{loop}} \left|\left| r_{loop}(\mathbf{S}_k^W, \mathbf{S}_j^W) \right|\right|^2_{\Sigma_{loop}} + \\ & \sum_{i \in \mathcal{F}_s} ||r_s(\mathbf{S}_i^W)||^2_{\epsilon_s} \end{aligned} \tag{4.14}$$

and can be optimized using Levenberg-Marquardt [135]. Here $\Sigma_{odom}, \Sigma_{loop} \in \mathbb{R}^{7\times7}$ denote the odometry and loop noise covariances, respectively and ϵ_s denotes the Huber noise parameter for the scale factors.

4.3 Evaluation

We demonstrate the performance of our approach quantitatively using the KITTI Odometry Benchmark [72] (Section 4.3.2) and qualitatively using handheld imagery collected from an indoor environment (Section 4.3.3).

4.3.1 Implementation Details

We designed our depth prediction network using Tensorflow [1] and set the base image size L^0 to 256×512 pixels. For all our experiments, the network is trained for 100 epochs using the Adam optimizer [109] on an NVIDIA 1080Ti GPU with a batch size of 8 and a learning rate of 0.0001, which is halved after 30 epochs and again after 40 epochs. We follow standard data augmentation practices by randomly flipping the training images left to right and perturbing the image color, including gamma and brightness shifting. We set the weights governing the terms in the loss function as $\lambda_p = 1.0$, $\lambda_{lr} = 1.0$, $\lambda_r = 0.1$, and $\lambda_s = 10.0$. Network inference is triggered at runtime using the REST API of the `tensorflow_serving` package.

Our geometric SLAM pipeline is implemented in C++ using the Ceres solver library [3]. Disparity maps from L^3 (32×64 pixels) are used to generate the metric scale factors for each keyframe pose. Both network inference and SLAM optimization are performed at runtime entirely on an Intel i7 4820K CPU.

4.3.2 KITTI Odometry Evaluation

We evaluate the odometry performance of our approach quantitatively using ten video sequences from the KITTI Odometry Benchmark [72]. We train our depth prediction network using the common training split of the raw KITTI stereo data from Eigen et al. [54], which consists of 22,600 training stereo pairs, 888 validation pairs, and 697 testing pairs.

Of the ten odometry sequences, images from runs 00, 06, 08, 09, and 10 are included in the training data of the depth prediction network. Runs 00, 03, 04, 05, and 06 have no overlap with the depth network training data. (Run 01 exhibits very little texture for feature detection and was not used to evaluate odometry.) Example trajectories for training and test runs are shown in Figure 4-5.

Quantitative performance is measured using relative pose error (RPE) [207] over a set of predefined path lengths (100 m to 800 m). Table 4.1 shows the relative translation error t_{rel} (expressed as a percent of distance traveled) and relative rotation

KITTI Odometry Benchmark

		SfMLearner [241]		Monodepth2 [76]		DVSO [236]		Ours	
	Run	t_{rel}	r_{rel}	t_{rel}	r_{rel}	t_{rel}	r_{rel}	t_{rel}	r_{rel}
Train	02	11.0	4.18	13.1	5.27	0.84	0.22	1.15	0.27
	06	10.7	6.31	17.0	12.9	0.73	0.35	2.61	1.22
	08	8.93	3.75	14.2	5.98	1.03	0.25	1.71	0.35
	09	10.6	4.07	17.7	6.18	0.83	0.21	1.70	0.48
	10	11.1	4.06	13.1	6.74	0.74	0.21	1.01	0.37
	Avg	10.4	4.11	13.8	5.56	0.89	0.23	1.39	0.33
Test	00	15.9	6.19	15.5	6.47	0.71	0.24	4.55	0.93
	03	11.1	4.52	10.2	2.93	0.77	0.18	4.72	0.21
	04	3.69	3.28	10.6	1.46	0.35	0.06	18.8	0.27
	05	10.8	4.66	12.6	6.51	0.58	0.22	2.36	0.33
	07	12.7	5.58	10.1	3.25	0.72	0.20	1.09	0.30
	Avg	13.7	5.63	14.4	6.69	0.67	0.24	3.85	0.73

Table 4.1: *KITTI Odometry Benchmark* – Here we show our pipeline's performance on the KITTI Odometry Benchmark [72]. t_{rel} denotes the relative translation error averaged over 100m to 800m path segments (expressed as a percent of distance traveled). r_{rel} denotes the relative rotation error averaged over the same path segments (expressed as degrees per 100m). The runs labeled "Train" are included in the training data for both our network and DVSO [236], while the runs labeled "Test" are not. Note that our method performs competitively on the benchmark despite only requiring a CPU.

error r_{rel} (expressed in degrees per 100m) for each run averaged over all path lengths, while Figure 4-6 shows these metrics for each path length averaged over all runs.

We compare our method against two end-to-end SLAM packages (SfMLearner [241] and Monodepth2 [76]) that are scaled to metric scale and a hybrid learning/geometric approach DVSO [236]. Note that SfMLearner and Monodepth2 are trained on runs 00-08. DVSO is trained using the same split as our method, but uses additional supervision from a sparse reconstruction method [55]. We are unable to do a full comparison to DVSO as the authors have not provided a public implementation of their technique, and so we compare to their published results.

Our method performs competitively on the benchmark, achieving a relative translation error of 1.39% and a relative rotation error of 0.33° / 100m on the training

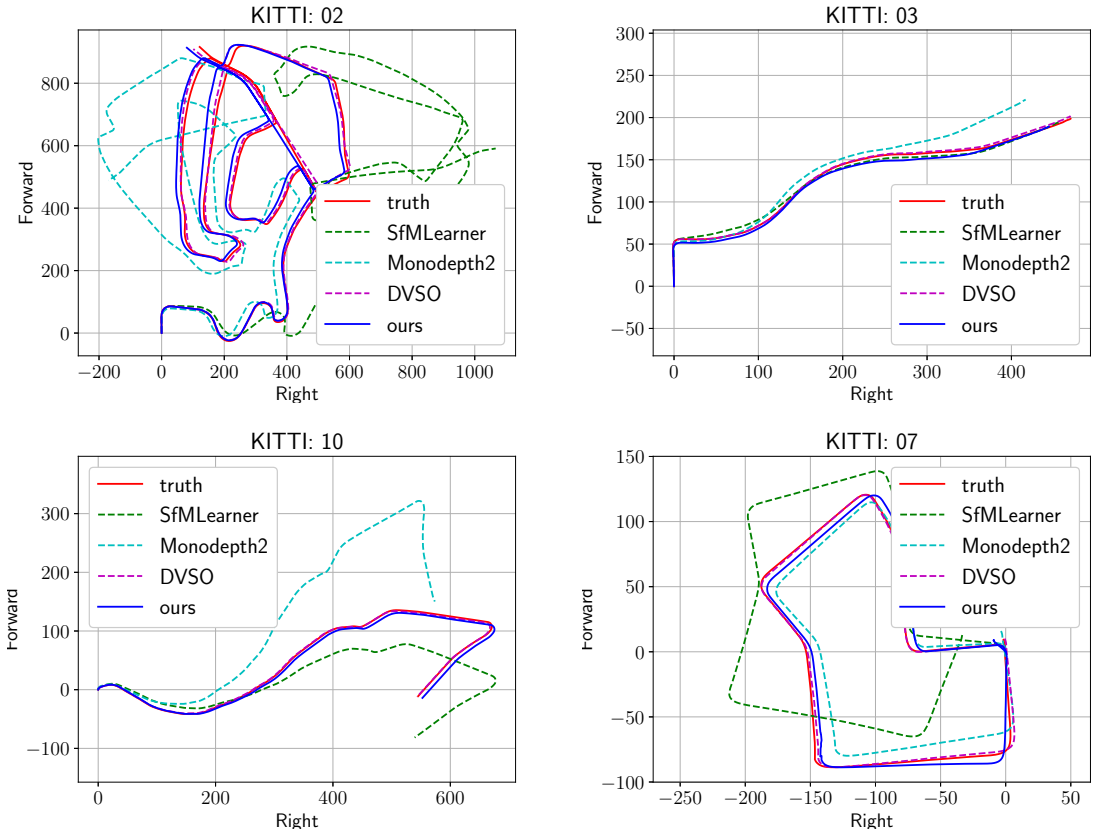


Figure 4-5: *KITTI Odometry Performance* – Our method gives compelling performance on the KITTI odometry benchmark. The left column shows sequences that were included in the training data of our metric depth prediction network. The right column shows sequences that were not used to train the network.

runs and 3.85% and 0.73° / 100m on the test runs. Scale drift is largely non-existent in the resulting trajectories. Furthermore, all computation is performed entirely on the CPU, while other methods require GPU acceleration. Network inference takes approximately 30 ms per frame. For comparison, the authors of DVSO report that evaluations of their network take 40 ms per frame on an NVIDIA Titan X Pascal GPU.

4.3.3 Handheld Odometry Evaluation

In addition to the quantitative results described in the previous section, we also qualitatively validate our system with imagery collected using a handheld stereo camera that captures time synchronized images at 16 Hz. The baseline between the left and

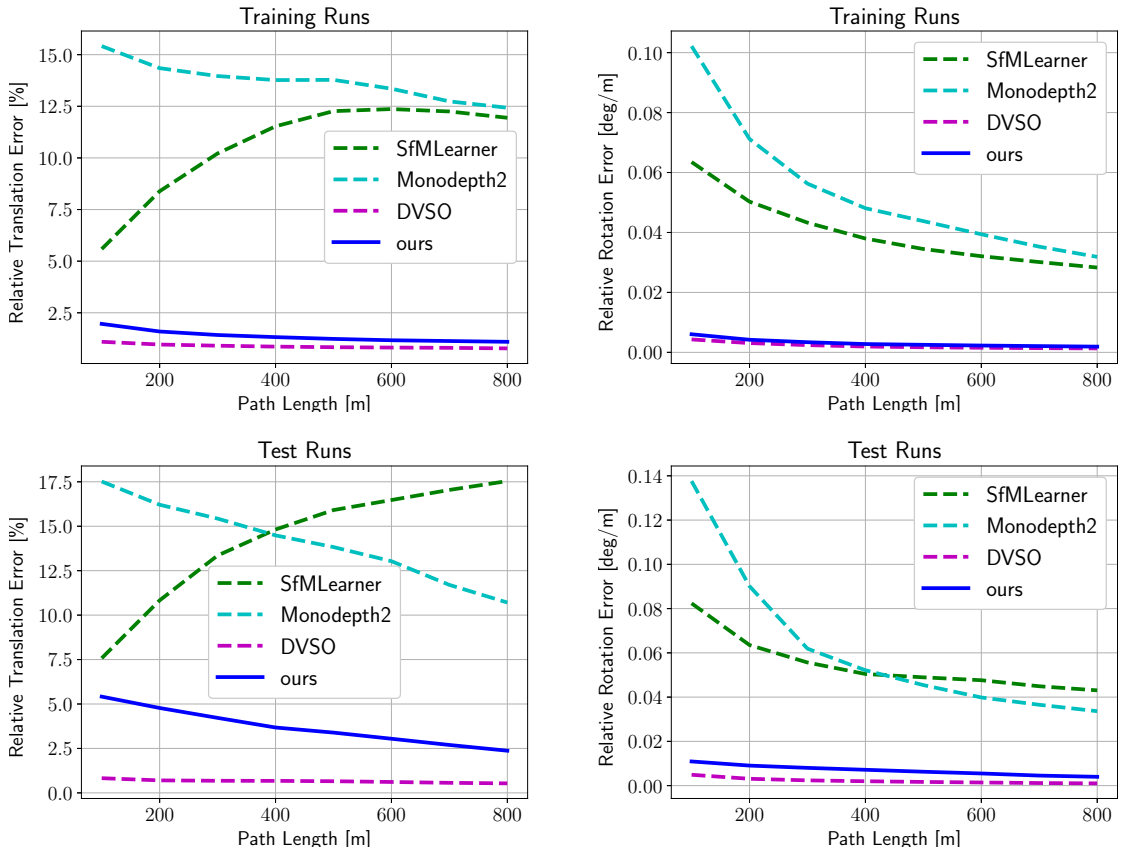


Figure 4-6: *Relative Pose Error vs. Distance Traveled* – The plots above show the relative translation (left) and rotation error (right) on the training (top) and test (bottom) runs from the KITTI Odometry Benchmark [72]. Our method achieves competitive performance on the benchmark despite not relying on GPU acceleration.

right cameras is 5 cm. The environment used for the experiment is a large, indoor laboratory common area and student thoroughfare between classrooms.

We collected a total of 16,980 stereo images, 11,548 of which were used for training our depth prediction network with 1,510 pairs used for validation. Two complete runs comprising 3,922 pairs were withheld to test our odometry performance. At runtime, the images from the left camera were used to compute our metrically scaled poses. The trajectories for the two test runs are shown in Figure 4-7. In the absence of groundtruth poses, we compare our monocular odometry estimates against that of Stereo ORB-SLAM2 [147], a state-of-the-art geometric stereo odometry pipeline. (Note that as a stereo method, its poses are metrically scaled since the baseline between the left and right cameras is known.) As evident in Figure 4-7, our method

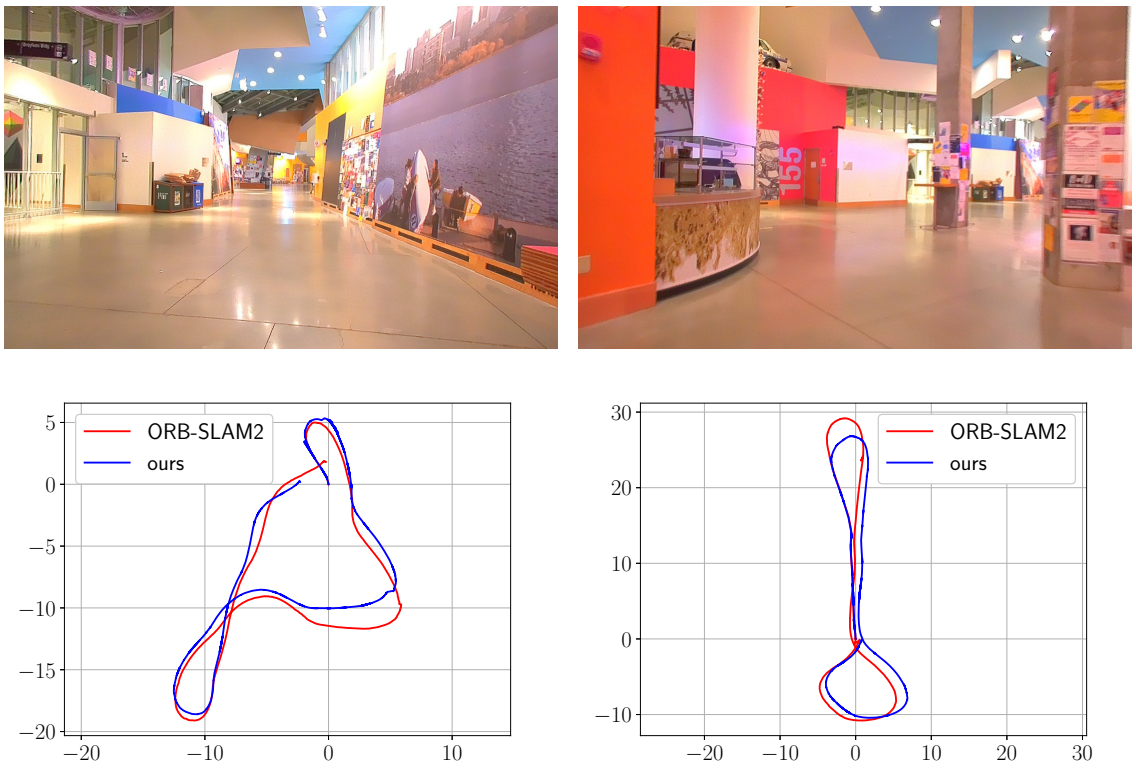


Figure 4-7: *Handheld Trajectories* – We demonstrate our method's performance using handheld camera data from an indoor environment. *Top row*: Sample training images from the environment. *Bottom row*: Comparison of poses from our approach (blue) against those of Stereo ORB-SLAM2 [147] on two test trajectories. Note that Stereo ORB-SLAM2's poses are correctly scaled as the stereo baseline is known *a priori*. Our technique is able to generate correctly scaled poses using only a single monocular camera.

is able to produce accurate poses at the correct metric scale despite only using a single monocular camera.

4.3.4 Improvements

In the preceding sections we demonstrated our method's ability to accurately track camera motion at the correct metric scale without additional sensors. There are, however, several aspects of the system that could be improved upon. First we note that although the true metric scale of the SLAM solution is correctly inferred over time, the poses remain unscaled until the first successful solve of the global pose graph.

Furthermore, the scale may jump while metric information is being accumulated into the graph via our unary scale factors. Developing a better initialization scheme and a more graceful way to handle these scale jumps would significantly improve the overall robustness of the approach. Finally, the ability of our method to correctly infer metric scale is limited by the data used to train our coarse depth prediction network. If the environment used when testing the system is significantly different from that used to train the network, performance will likely degrade. Improving the ability of the network to generalize beyond the initial training data (e.g., using some form of online training or domain adaption) would make the approach far more powerful.

4.4 Conclusion

In this chapter, we proposed an efficient method for monocular SLAM that is capable of estimating metrically-scaled, scale drift-free motion without additional sensors or compute by integrating metric depth predictions from a neural network into a geometric SLAM pipeline. This network leverages prior information about a scene distilled from large datasets of images to render metric scale observable. Since it is designed specifically for metric scale estimation, it can be much smaller and faster than competing systems. We make several improvements to our network architecture and training procedure to address the lack of depth observability when using coarse image input that allows us to estimate spatially coarse, but depth-accurate predictions in only 30 ms per frame. We show compelling results on the KITTI benchmark dataset in addition to real-world experiments with a handheld camera.

Chapter 5

View-Compensated Multi-View Stereo Depth Estimation

In this chapter we present an efficient method for dense monocular depth estimation that exploits known camera viewpoint changes to properly compensate features for robust pixel matching. We formulate the approach in the multi-view stereo (MVS) framework. MVS is a fundamental problem in computer vision where the geometry of a scene is estimated from a set of images taken from known, but otherwise unconstrained, viewpoints. While the scene geometry may be represented in a variety of ways, a common design choice is to designate one of the images as a privileged reference and estimate a depthmap with respect to that image. Classical methods [193, 77] generally start by defining a volume in the reference image's coordinate frame by sampling a set of depths for each reference pixel. Matching costs that record how consistent a depth hypothesis is with the neighboring (or *comparison*) images are then computed by projecting each pixel at each candidate depth into the comparison views and comparing intensities. After filtering the volume to reduce noise, the reference depthmap that minimizes the matching costs can be extracted.

Plane Sweep stereo techniques [35, 185, 211] compute the matching cost volume more efficiently by interpreting the volume as a set of planes, one for each depth hypothesis. The comparison images can then be projected (or *warped*) onto each plane, creating a set of transformed images (one for each depth hypothesis) that

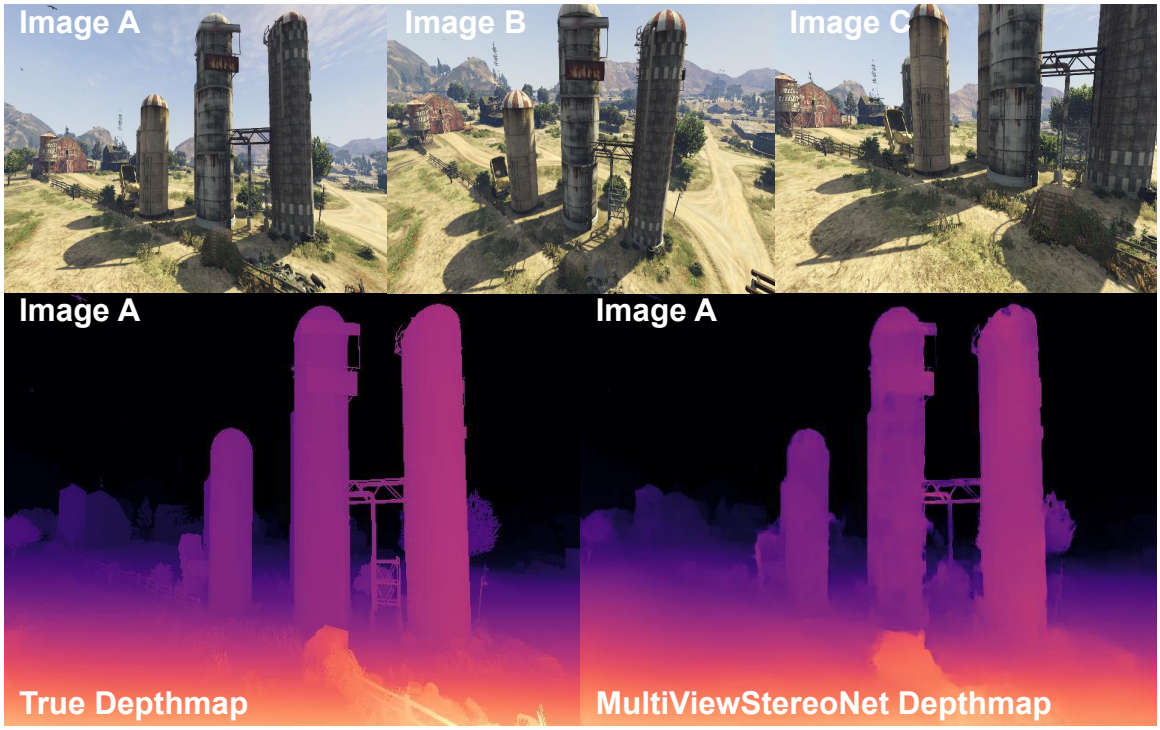


Figure 5-1: *MultiViewStereoNet* – We propose a novel, learning-based method for multi-view stereo (MVS) depth estimation that we call MultiViewStereoNet. By combining coarse stereo matching costs, guided refinement, and incrementally computed features that compensate for known viewpoint changes, our method is able to achieve reconstruction accuracy comparable to the state-of-the-art, while being significantly faster at runtime. The top row of the figure above shows input images that are used to generate the depthmap in the bottom right. The groundtruth depthmap is shown in the bottom left for comparison.

are then compared to the reference image directly to compute matching costs. The relative simplicity of the Plane Sweep architecture has made it the preferred way MVS solutions are formulated (which we will continue in this chapter).

In recent years, deep learning approaches have shown great promise at solving the MVS problem by exploiting prior information learned from large training datasets [96, 95, 237]. Instead of using raw pixel intensities or hand-crafted feature extractors and filtering schemes, these systems learn the MVS components from data by training stacked layers of convolutional neural networks (CNNs). The weights of these convolutional layers can encode additional global context and semantic information that can improve estimation performance in the presence of lighting changes, low texture, and other imperfections common in natural scenes.

Despite the rapid progress enabled by learned feature descriptors, MVS depth estimation is still a challenging problem in the wild, primarily due to the difficulty in robustly matching dense image features across the viewpoint changes common with freely moving cameras. Objects in a scene can appear radically different, or be occluded entirely, when viewed from disparate viewing angles or lighting. Any learning-based system must also generalize beyond the data used to train the network. MVS is a particularly difficult problem in this sense, as supporting wholly unconstrained camera motion at test time requires extensive training samples to ensure adequate coverage of the operating regimes.

Designing networks that can learn distinctive feature representations from limited data is therefore of primary importance to solving MVS. Current learning-based methods, however, do not leverage all information available to aid this process. In particular, modern networks extract learned features from each input image independently before projecting them onto the planes that comprise the cost volume. By applying the projection after feature extraction, the learned features must implicitly compensate for this projection and exhibit scale and rotation invariance despite never being exposed to the projection parameters. We assume we know the projection parameters (the camera intrinsics and extrinsics), however, which suggests more structure can be imposed on the feature extraction layers.

Our key insight is that by compensating for the known viewpoint changes during the feature extraction process itself, the network can learn features that are *specific* to the desired reference frame and *projected by construction*. This technique lessens the burden on the network to achieve scale and rotation invariance and therefore increases robustness to viewpoint changes during matching.

Compensating for viewpoint changes in this way can be computationally expensive, however, if care is not taken. In principle, we must extract features not from a single comparison image, but from the *set* of warped comparison images produced by projecting the image data onto the planes that comprise the cost volume. One can naively apply a conventional feature extractor CNN to each warped image, but this approach quickly grows unmanageable as the number of planes (i.e., depth samples)

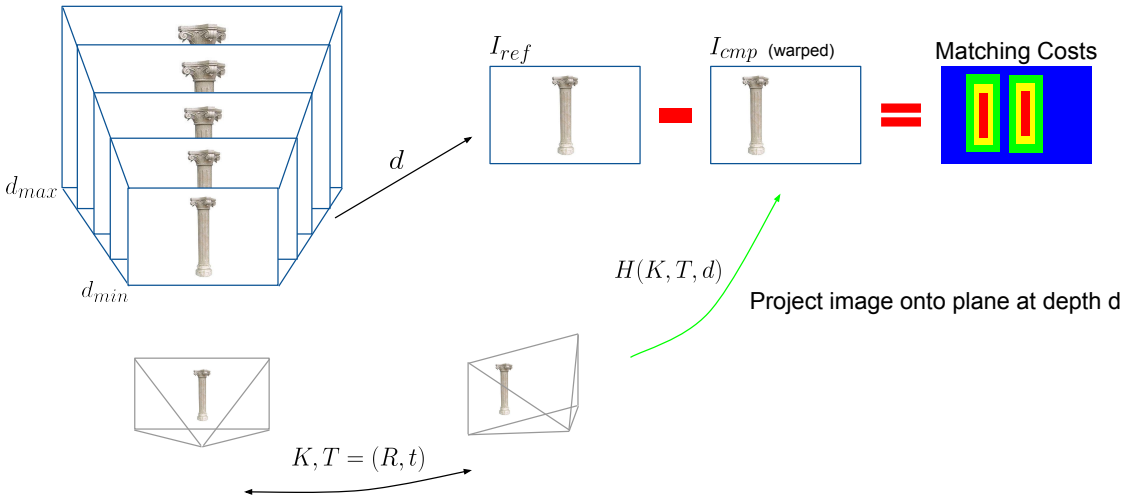


Figure 5-2: *Plane Sweep* – Plane Sweep [35, 185, 211] is a method for MVS that uses homography transforms to efficiently compute matching cost volumes. After designating one of the input images as a reference frame, a series of planes in that frame are generated by sampling depth hypotheses for each pixel. Comparison images are then projected onto these planes using homographies and compared to the reference image to generating matching costs. The depth hypothesis for each pixel that minimizes that costs can then be extracted, producing a dense depthmap.

increases and the feature extractor must be run repeatedly. Alternatively, layers of 3D convolutions could be used to extract features from the volume generated by concatenating the warped images, but these more complex layers are similarly expensive and prevent the use of commonly accepted network architectures built on stacks of 2D convolutions.

In this chapter, we overcome these limitations in two key ways. First, we generalize the approach of Khamis et al. (StereoNet) [108] from the two-view, rectified stereo domain to the multi-view, unrectified setting using differentiable Spatial Transformer Networks (STNs) [100]. Like StereoNet, we compute features and matching costs at a reduced image scale to produce coarse depthmaps that are then iteratively upsampled and refined with the image data as guidance. This type of architecture retains the benefits of learned stereopsis, but drastically reduces the amount of costly high-resolution feature matching for improved speed.

Second, we incrementally compute our projected features such that the bulk of the feature extraction layers need only be executed a single time across all depth

hypotheses. We initially generate a single feature map corresponding to the furthest depth plane in the volume with a conventional feature network. Then we apply a series of inexpensive homographies, coupled with simple refinement layers, that incrementally warp this feature map to other depth planes in the reference volume.

The combination of these two techniques allows our method, which we call Multi-ViewStereoNet, to achieve reconstruction accuracy comparable to the state-of-the-art, while being significantly more efficient.

5.1 Related Work

5.1.1 Two-View Stereo

Two-view stereo generally refers to the scenario where two cameras are rigidly mounted along a narrow baseline such that the corresponding images can be rectified onto a common image plane to estimate disparities [92, 188]. Early attempts to apply machine learning to this problem replaced one or more of classical building blocks with learned components before completely end-to-end were proposed. Zbontar and LeCun, for example, proposed a network to compute matching costs from small image patches, before using the costs in a classical pipeline [239]. Mayer et al. developed a network that directly regresses disparity using stacks of convolutions and deconvolutions [138] Kendall et al. aggregate global context in the stereo cost volume using 3D convolutions [107]. Khamis et al. similarly use 3D convolutions to aggregate information in the cost volume, but significantly reduce the spatial resolution of the volume for speed before applying image-guided refiners to upsample the resulting disparities [108]. Our solution takes inspiration from this network structure and generalizes it to the multi-view setting.

5.1.2 Multi-View Stereo

Learning-based approaches to MVS often follow the Plane Sweep paradigm [35], where matching costs from multiple images are aggregated in a single reference volume

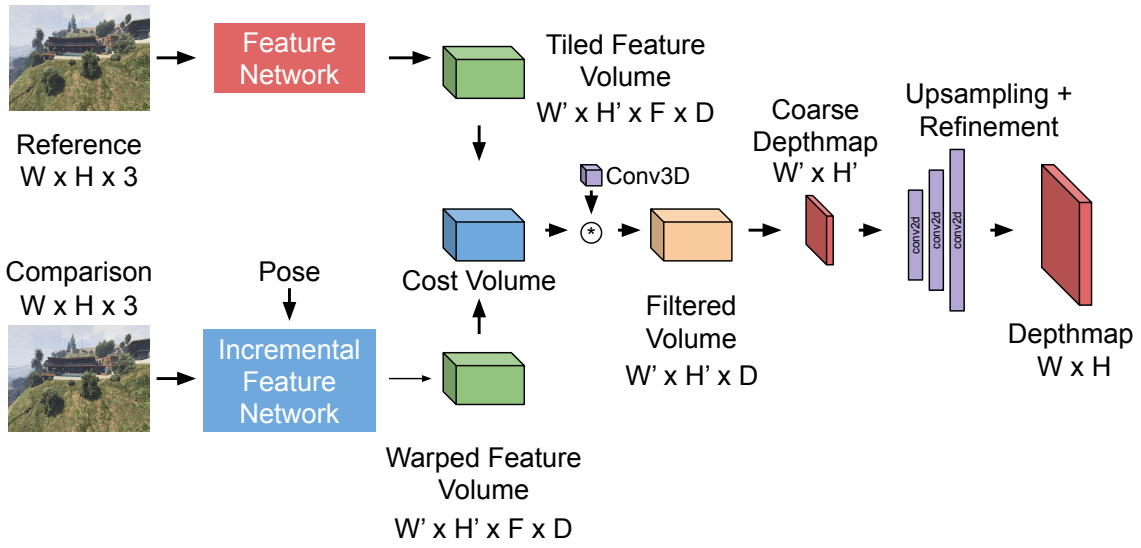


Figure 5-3: *MultiViewStereoNet Block Diagram* – In our network, coarse-resolution features are extracted from the input images using two subnetworks: a traditional CNN for the reference image and a novel CNN that compensates for the known viewpoint change for the comparison images. The two sets of feature volumes are then combined to form a cost volume, from which a coarse depthmap is extracted. A series of image-guided refiners is then used to upsample the depthmap to the input image size.

after geometric warping [35, 185, 211]. Yao et al., for example, extract features per image, transform them into the reference volume using a differentiable warp operation, then regularize the costs using multi-scale 3D convolutions [237]. Im et al. compute matching costs similarly, but refine the costs for each depth hypothesis using the reference image features [96]. Wang and Shen compute a multi-view cost volume using classical techniques, but then regress the depths using an encoder-decoder network [227]. Huang et al., on the other hand, estimate depthmaps on 64×64 pixel patches before tiling the results to the input resolution [95].

5.2 Method

The MultiViewStereoNet architecture is divided into four primary components as shown in Figure 5-3 and Figure 5-4. The reference image is first passed through a conventional feature extraction network composed of strided 2D convolutional layers

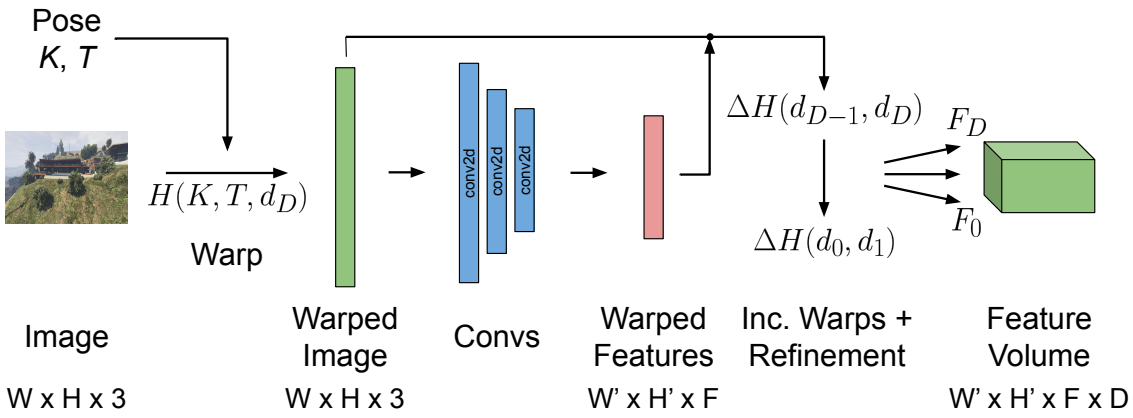


Figure 5-4: *Incremental Viewpoint-Compensated Feature Network* – Our novel feature network compensates for known viewpoint changes by projecting the comparison image before (rather than after) extraction. We incrementally compute each plane F_i of the comparison feature volume (corresponding to depth hypothesis d_i) from the previous plane using the relative homography ΔH between the planes. This allows for the feature maps to be computed for each depth hypothesis, while only requiring the bulk of the convolutional layers to be executed once, increasing the network's speed.

with residual connections [90] to generate a set of reference features. Each comparison image, on the other hand, is passed through our novel, viewpoint-compensated feature network that incrementally computes projected features for each candidate depth. These feature maps are then concatenated to form a warped feature *volume*. After tiling the reference feature map to create an identically sized feature volume, the absolute difference of the two volumes forms our coarse cost volume. We then apply a series of 3D convolutions and normalization steps to filter the costs, before extracting depths using a `softargmin` operator. The coarse depthmap is then passed through a series of upsampling and image-guided refinement layers to produce the final depthmap. The estimated depthmap at each image scale is then compared to the groundtruth depthmap to drive training.

5.2.1 Reference Feature Network

Our reference feature network is derived from that detailed by Khamis et al. [108]. The reference image is first passed through 4 2D convolutional layers with kernel size

5, stride 2, and 32 output channels. The resulting feature map is then passed through 6 residual blocks with kernel size 3, stride 1, and the same 32 output channels. We make two modifications to the residual connections described in [108]. First, we use only a single convolution in the skip connection, instead of the normal two as described by He et al. [90], which achieves similar performance with fewer parameters. Second, we replace the batch normalization [97] layer with a group normalization layer [233] to better support small batch sizes during training. For an image of size $H \times W \times 3$, this network produces a feature map of size $H' \times W' \times C$ for $W' = W/16$, $H' = H/16$, and $C = 32$. Given a set of D candidate depth samples $\{d_i\}_{i=1}^{D}$, we then tile the feature map to produce our reference feature volume of size $H' \times W' \times D \times C$.

5.2.2 Incremental Viewpoint-Compensated Feature Network

Before describing our incremental, viewpoint-compensated feature network, we first review some key concepts from multi-view geometry [89, 210]. Assume we have two cameras: a reference camera r and a comparison camera c, with the same intrinsic parameters $K \in \mathbb{R}^{3\times3}$. Let $\Omega_r, \Omega_c \subset \mathbb{R}^2$ denote the image domain of each camera, respectively. Let $I_r : \Omega_r \to \mathbb{R}^3$ and $I_c : \Omega_r \to \mathbb{R}^3$ designate the images from the two cameras with 3 channels (e.g., representing RGB values). Finally, let $R_c^r \in \mathbb{SO}(3)$ and $t_c^r \in \mathbb{R}^3$ represent rotation and translation of the comparison camera with respect to the reference camera, which we assume are known.

If the scene geometry can be represented by a single plane with normal vector $n \in \mathbb{R}^3$ and depth $d > 0$ with respect to the reference coordinate system, the transform that projects (homogeneous) pixels from Ω_r to Ω_c is given by the function $H(d) : \Omega_r \to \Omega_c$, which can be represented by a 3×3 homography matrix:

$$H(d) = K(R_r^c - t_r^c n^T/d)K^{-1}. \tag{5.1}$$

The image $\tilde{I}_d : \Omega_r \to \mathbb{R}^3$ represents the projection of the comparison image onto the

reference plane at depth d and is given by

$$\tilde{I}_d(\mathbf{u}) = I_c(\pi(H(d)\bar{\mathbf{u}})), \tag{5.2}$$

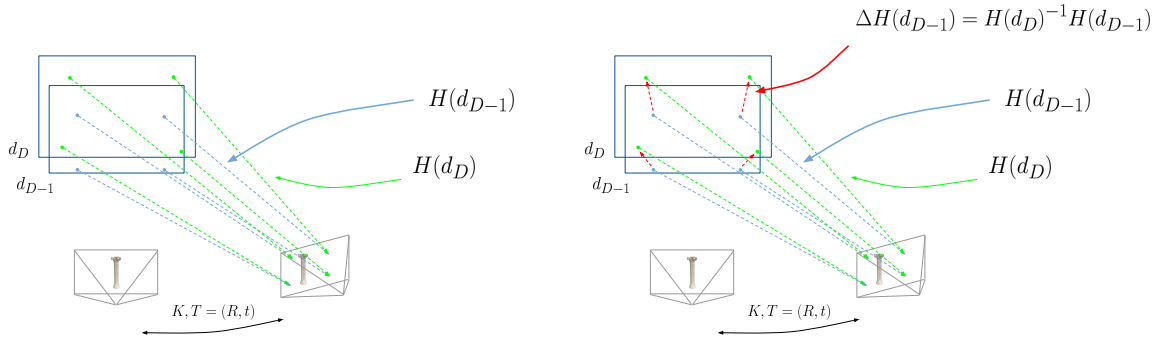
where $\bar{\mathbf{x}} = (\mathbf{x}, 1)$ signifies a homogeneous pixel coordinate and $\pi(x, y, z) = (x/z, y/z)$ denotes the perspective projection function. When implemented, the pixel domain Ω_r is uniformly sampled to generate discrete pixels before applying $H(d)$ and the indexing into I_c is accomplished using bilinear interpolation, which describes a type of Spatial Transformer Network (STN) [100].

In Plane Sweep stereo [35, 185, 211], one computes a series of such transformed images (one for each candidate depth d_i), which are then compared to the reference image to compute matching costs. In existing learned MVS systems, the original image channels are simply replaced by learned features $F : \Omega \to \mathbb{R}^C$, where C denotes the number of feature channels. Note, however, that the feature extraction occurs *before* the projection, which means the features must implicitly compensate for any scale, rotation, or perspective changes between the cameras. One could extract features for each warped image $\tilde{I}_d$ independently or concatenate the $\tilde{I}_d$ into a volume and apply 3D convolutions, but both options are computationally expensive. Instead, we will take an *incremental* approach to feature extraction, building the feature map F_i for candidate depth d_i from the neighboring feature map F_{i+1} for depth d_{i+1}.

We compute the initial feature map F_D corresponding to the maximum candidate depth d_D by transforming the comparison image I_c by $H(d_D)$ to form $\tilde{I}_{d_D}$ and applying the feature extraction network described in Section 5.2.1. Computing F_{D-1} from F_D is accomplished by applying the *relative* homography $\Delta H(d_{D-1}, d_D)$ from depth plane d_{D-1} to d_D given by

$$\Delta H(d_{D-1}, d_D) = H(d_D)^{-1} H(d_{D-1}). \tag{5.3}$$

Note that these homographies use a scaled intrinsic matrix to reflect the downsam-



(a) Homography for two planes.

(b) Relative homography between planes.

Figure 5-5: *Relative Homography* – Our proposed feature network relies on the *relative homography* between two planes in order to incrementally compute feature maps. On the left we have the homographies $H(d_D)$ and $H(d_{D-1})$, which each transform pixels from the reference image domain to the comparison image domain assuming the points lie on planes at depth d_D and d_{D-1}, respectively. The relative homography transforms pixels between these two planes directly, show in the figure on the right. The relative homography can be computed from $H(d_D)$ and $H(d_{D-1})$ as $\Delta H(d_{D-1}, d_D) = H(d_D)^{-1} H(d_{D-1})$.

pling of the image domain. The features F_{D-1} can then be computed as

$$F_{D-1}(\mathbf{u}) = F_D(\pi(\Delta H(d_{D-1}, d_D)\mathbf{u}). \tag{5.4}$$

We once again use an STN to implement this projection: the domain of the feature maps is discretized to yield features at individual pixels before applying the relative homography and indexing into F_D based on the projected pixel locations is accomplished using bilinear interpolation.

This incremental technique allows for the feature maps for all candidate depths to be computed using only a single invocation of the feature extraction network, while still appropriately compensating for the known viewpoint changes. It is possible, however, that the relative homographies $\Delta H(d_{i-1}, d_i)$ generate pixel locations that lie outside (or on the boundary) of the valid domain in the parent feature map F_i. To account for these edge cases, we apply another instance of refinement using the warped image as guidance. We concatenate F_i and $\tilde{I}_i$ along the channel dimension and apply 3 convolutional layers with kernel size 3, stride 1, with a single skip connection. The outputs of these layers are then added to the original feature map.

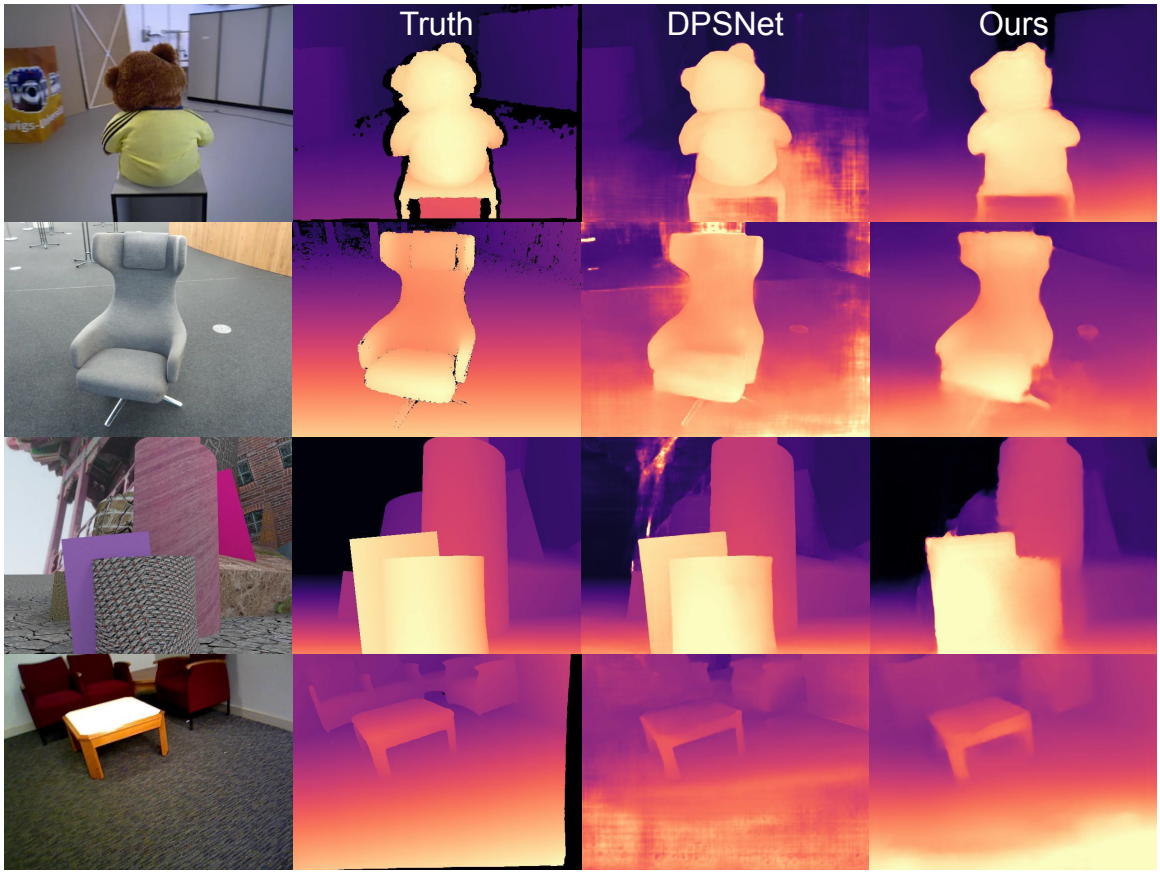


Figure 5-6: *DeMoN Depthmaps* – Here we show the qualitative reconstruction performance of MultiViewStereoNet on the DeMoN benchmark test set [223]. MultiViewStereoNet is capable of producing depthmaps comparable to DPSNet, while being significantly faster.

After feature refinement, we concatenate the feature maps across all the depth samples into a warped feature volume of size $H' \times W' \times D \times C$.

5.2.3 Cost Volume Formulation and Filtering

To compute matching costs, we take the absolute difference of the reference and warped comparison feature volumes described in Section 5.2.1 and 5.2.2. Although there is some evidence that asymmetric distance measures [108] or concatenating feature channels [107] improves matching quality, we found simple absolute differences to work well.

The cost volume is passed through a series of 3D convolutional layers designed to pool global context information and reduce noise. We follow the architecture

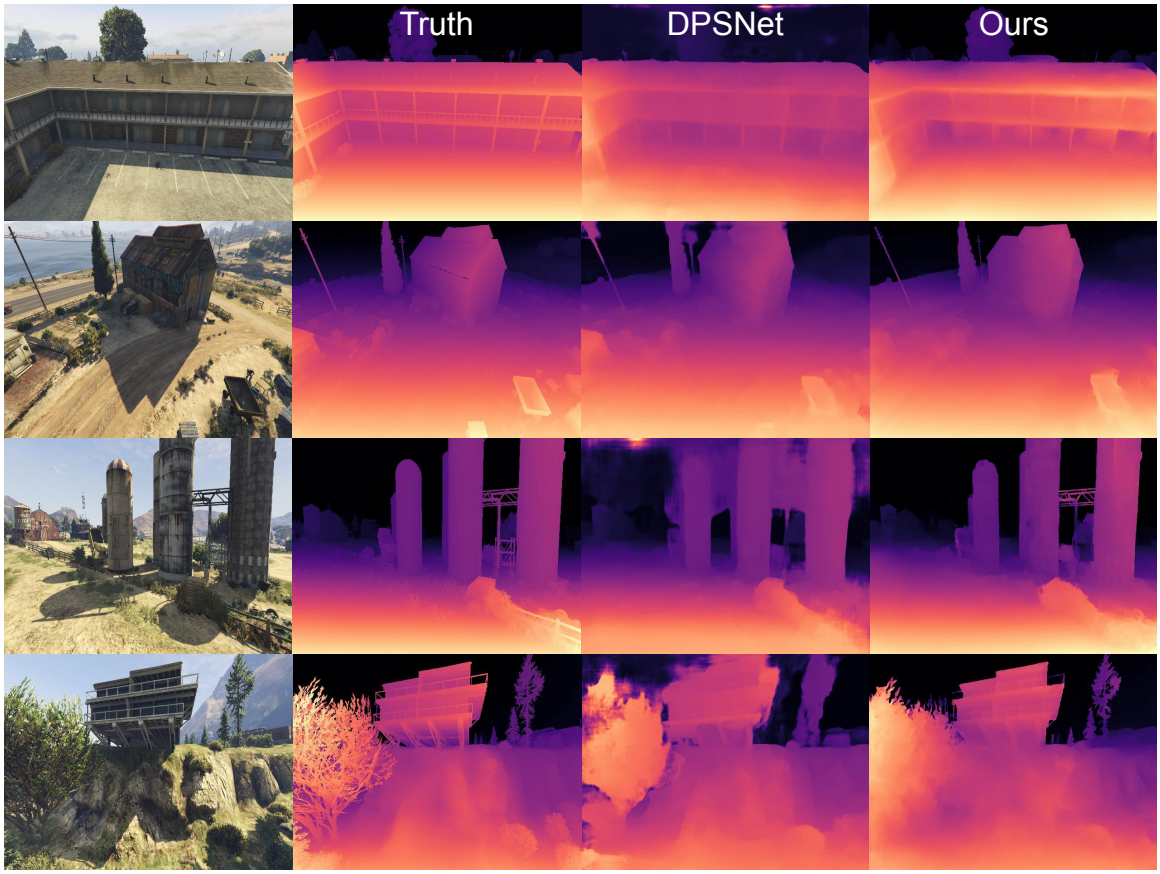


Figure 5-7: *GTA-SfM Depthmaps* – MultiViewStereoNet is capable of producing compelling depth estimates on challenging imagery, such as that from the GTA-SfM dataset. Here, example depthmaps from the test set are shown.

proposed by Khamis et al. [108] and use 4 3D convolutional layers with each followed by a groupnorm [233] operation and LeakyReLU activation. The input and output channels for these layers are of equal size. A final 3D convolutional layer is then applied which reduces the feature dimension to a scalar, resulting in a volume of size $H' \times W' \times D$. The kernel sizes for all layers is set to $3 \times 3 \times 3$.

5.2.4 Depth Regression and Guided Refinement

We extract a coarse depthmap of size $H' \times W'$ from the cost volume before upsampling and refining the outputs to the input resolution. Relying on upsampling and refinement, rather than high-resolution feature matching, has significant advantages in terms of efficiency, as described by Khamis et al. in StereoNet [108]. StereoNet,

however, assumes the input images are rectified, meaning the feature projection described in Equation 5.2 can be accomplished by a simple shift of indices. Here, we employ a similar upsampling and guided refinement scheme, but apply it to our multi-view features and cost volume.

For a given pixel $\mathbf{u}$, let c_i denote the matching costs for depth sample i. We form a probability distribution σ over the depths by applying the `softmin` operator:

$$\sigma(c_i) = \frac{\exp(-c_i)}{\sum_j \exp(-c_j)}. \tag{5.5}$$

The depth $D(\mathbf{u})$ for pixel $\mathbf{u}$ is then given by the mean of this distribution, $D(\mathbf{u}) = \sum_i d_i \sigma(c_i)$. The combination of these two operations (`softmin` following by an average) is a differentiable approximation of the arg min function.

Next, we iteratively upsample the coarse depthmap using bilinear interpolation and then pass it through an image-guided refinement network to resolve fine structures. The refinement network concatenates the coarse depthmap and appropriately-sized reference image before passing them through an initial convolutional layer with 32 output channels. Group normalization is applied along with a LeakyReLU activation. After this, 6 dilated residual block layers are applied, where each block consists of a single convolution with group normalization and LeakyReLU activation, followed by a skip connection. The dilation strides are set to $(1, 2, 4, 8, 1, 1)$. The input and output channels for these layers is kept at 32. A final convolution is then applied to reduce the output to a single channel representing a depth residual. This residual is then added to the input depthmap. We apply an initial refinement round to the coarse depthmap extracted from the cost volume followed by 4 rounds of upsampling by a factor of 2 and refinement to yield the final output depthmap of size $H \times W$.

5.2.5 Multi-View Fusion

In conventional Plane Sweep Stereo, fusing depth information from multiple comparison views is achieved by simply averaging matching costs for each comparison image. We found this approach to be brittle in a learned MVS context since certain locations

in the reference volume will only be observable from a subset of the cameras. Careful bookkeeping is then required to keep track of these locations and perform the averaging correctly. We found performing the fusion at the *depthmap* level, rather than the cost volume level, to be more robust. We pass each comparison image through a subset of the network to produce a set of coarse depthmaps of size $H' \times W'$. We then average these depthmaps and then proceed with the final upsampling and refinement layers described in Section 5.2.4.

5.3 Evaluation

5.3.1 Implementation Details

We implemented our network using PyTorch [162]. Training was performed on 8 NVIDIA V100 GPUs with a batch size of 8 per GPU, while testing was performed on a single NVIDIA GTX 1080Ti with a batch size of 1. We used the Adam optimizer [109] for training, with a learning rate of 0.001. We use the pseudo-Huber loss described by Barron [12] against groundtruth depth labels applied to the depthmaps at each image scale. The depth samples $\{d_i\}$ for our cost volumes are generated by sampling uniformly in inverse depth space between 0 (infinite depth) and a maximum inverse depth value computed for each training example based on a maximum disparity of 192 pixels. We set $D = 12$ for all experiments. To remove any dependence on the metric scale of the geometry, we normalize the camera poses to have unit baseline before computation. All training was performed using two input images per sample (i.e., one depthmap is estimated using two images), although multiple images may be used during test time.

5.3.2 DeMoN Benchmark

We evaluate our approach against the DeMoN dataset [223] commonly used to benchmark MVS systems. This dataset includes 51k training scenes assembled from both real and simulated imagery. We use the same training split as [96], which yields 168k

DeMoN Benchmark

Dataset	Method	Abs Rel ↓	Sq Rel ↓	RMSE ↓	$\text{RMSE}_{\log}$ ↓	α_1 ↑	α_2 ↑	α_3 ↑	Runtime [sec] ↓
MVS	COLMAP	0.38	1.26	1.48	0.50	0.48	0.66	0.84	-
	DeepMVS	0.23	0.62	1.15	0.30	0.67	0.89	0.94	80.9
	MVDepthNet	0.20 ± 0.02	0.47 ± 0.10	0.71 ± 0.06	0.25 ± 0.02	**0.80 ± 0.02**	0.90 ± 0.01	0.94 ± 0.01	**0.121 ± 0.005**
	DPSNet	**0.08 ± 0.01**	**0.07 ± 0.01**	**0.40 ± 0.03**	**0.15 ± 0.01**	**0.90 ± 0.01**	**0.96 ± 0.01**	**0.98 ± 0.01**	0.630 ± 0.001
	Ours	**0.18 ± 0.03**	**0.36 ± 0.18**	**0.59 ± 0.06**	**0.22 ± 0.01**	0.79 ± 0.02	**0.92 ± 0.01**	**0.96 ± 0.01**	**0.065 ± 0.001**
SUN3D	COLMAP	0.62	3.24	2.32	0.66	0.33	0.55	0.72	-
	DeepMVS	0.28	0.44	0.94	0.36	0.56	0.74	0.90	80.9
	MVDepthNet	**0.18 ± 0.01**	**0.19 ± 0.04**	**0.55 ± 0.03**	0.24 ± 0.01	0.74 ± 0.02	0.91 ± 0.01	0.96 ± 0.01	**0.121 ± 0.005**
	DPSNet	**0.16 ± 0.01**	**0.13 ± 0.01**	**0.45 ± 0.02**	**0.20 ± 0.01**	**0.79 ± 0.02**	**0.93 ± 0.01**	**0.98 ± 0.01**	0.630 ± 0.001
	Ours	0.19 ± 0.02	0.24 ± 0.06	**0.55 ± 0.04**	**0.21 ± 0.01**	**0.76 ± 0.02**	**0.92 ± 0.01**	**0.97 ± 0.01**	**0.065 ± 0.001**
Scenes11	COLMAP	0.62	3.71	3.66	0.87	0.39	0.57	0.67	-
	DeepMVS	0.21	0.37	0.89	0.27	0.69	0.89	0.97	80.9
	MVDepthNet	**0.08 ± 0.01**	**0.13 ± 0.01**	**0.63 ± 0.02**	**0.16 ± 0.01**	**0.93 ± 0.01**	**0.97 ± 0.01**	**0.98 ± 0.01**	**0.121 ± 0.005**
	DPSNet	**0.09 ± 0.01**	**0.20 ± 0.02**	**0.76 ± 0.03**	**0.15 ± 0.01**	**0.93 ± 0.01**	**0.97 ± 0.01**	**0.98 ± 0.01**	0.630 ± 0.001
	Ours	0.13 ± 0.01	0.27 ± 0.02	0.92 ± 0.02	0.22 ± 0.01	0.87 ± 0.01	0.95 ± 0.01	0.97 ± 0.01	**0.065 ± 0.001**
RGBD	COLMAP	0.54	1.76	1.51	0.72	0.27	0.50	0.72	-
	DeepMVS	0.29	0.43	0.87	0.35	0.55	0.81	0.92	80.9
	MVDepthNet	0.21 ± 0.01	0.36 ± 0.04	1.07 ± 0.06	0.34 ± 0.02	0.66 ± 0.02	0.82 ± 0.02	0.89 ± 0.01	**0.121 ± 0.005**
	DPSNet	**0.16 ± 0.01**	**0.23 ± 0.04**	**0.73 ± 0.06**	**0.24 ± 0.02**	**0.79 ± 0.02**	**0.90 ± 0.01**	**0.95 ± 0.01**	0.630 ± 0.001
	Ours	**0.17 ± 0.01**	**0.25 ± 0.04**	**0.80 ± 0.05**	**0.22 ± 0.01**	**0.76 ± 0.02**	**0.92 ± 0.01**	**0.97 ± 0.01**	**0.065 ± 0.001**

Table 5.1: *DeMoN Benchmark* – Our network achieves reconstruction accuracy comparable to the state-of-the-art, while being significantly faster. Here we compare MultiViewStereoNet to existing methods COLMAP [190], DeepMVS [95], MVDepthNet [227], and DPSNet [96] on the two-view DeMoN Benchmark. The rows of the table correspond to the different splits of the datasets (MVS, Sun3D, Scenes11, RGBD), while the columns show commonly used depth accuracy metrics such as Abs Rel (the mean absolute relative depth error) and depth completion metrics such as α_1 (the fraction of pixels with less than 25% depth error). Each metric is computed per depthmap and then averaged across the test set. Standard errors are shown beside each mean (standard errors for COLMAP and DeepMVS are not available). The top two performing methods according to each metric are bolded. Runtime metrics were computed using VGA image resolution on an NVIDIA GTX 1080Ti GPU for all algorithms.

training samples and 708 test samples at VGA resolution. Groundtruth depths are provided via RGBD sensors or simulation. For this dataset, we train for 45 epochs and compute standard depth metrics against groundtruth that are summarized in Table 5.1. We compare our proposed network against DPSNet [96], MVDepthNet [227], DeepMVS [95], and a traditional reconstruction pipeline based on COLMAP [190]. As shown in Table 5.1, our approach achieves depth reconstruction accuracy and completeness comparable to the state-of-the-art, while being significantly more efficient. Figure 5-6 shows qualitative performance on the DeMoN test set, where we compare favorably with DPSNet.

GTA-SfM Dataset					
Images	Method	Abs Rel ↓	RMSE ↓	α_1 ↑	Runtime [ms] ↓
2	DPSNet	0.103	26.97	0.94	662
2	Ours	0.084	19.69	0.923	65.1
3		0.077	19.47	0.932	79.7
4		0.075	19.41	0.934	92.7
5		0.075	19.40	0.935	106.5

Table 5.2: *Multi-View Evaluation* – Our network can fuse information from multiple images to produce depth estimates. Here we show depth estimation performance as the number of comparison images increases (one image is designated as the reference).

5.3.3 Multi-View Evaluation

We also evaluate our network on the GTA-SfM dataset presented by Wang and Shen [228]. This dataset contains 17k training images and 2k testing images (VGA resolution) from trajectories produced inside the Grand Theft Auto V video game. For each training image, we randomly sample a single comparison view from the camera sequence to form training samples. For each test image, we randomly sample N comparison views. We train on this dataset for 150 epochs and compare reconstruction performance against DPSNet [96] as the number of comparison views varies. Table 5.2 summarizes the results, which shows our method achieves better depth accuracy and completeness than DPSNet. Furthermore, performance increases as more comparison images are used, although the effect quickly plateaus. Each additional comparison image takes an additional 15 milliseconds to process. Figure 5-1 and Figure 5-7 shows qualitative performance on this dataset, where again, we compare favorably with DPSNet.

5.3.4 Viewpoint Compensation Evaluation

We also investigate the effect of viewpoint compensation on reconstruction performance using a simple modification of the GTA-SfM test set. For each test sample, we rotate the comparison image by 180 degrees about the optical axis, simulating extreme viewpoint changes. (This rotation can be accomplished easily using two

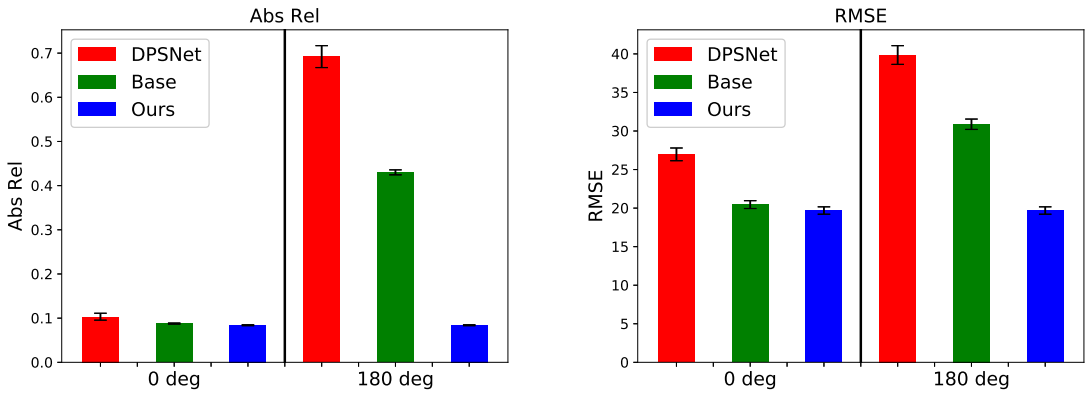


Figure 5-8: *Viewpoint Compensation* – Our viewpoint-compensated features take the known camera poses into account during extraction. Here we compare reconstruction performance of MultiViewStereoNet, a base network without viewpoint compensation, and DPSNet on the GTA-SfM dataset using two test sets: a standard test set and a set where one of the images for each sample is rolled about the optical axis by 180 degrees. Despite being given the parameters of the roll, the base network and DPSNet see a significant increase in absolute relative depth error (left) and RMSE depth error (right). MultiViewStereoNet sees no significant difference in performance across the two test sets. Standard errors are shown above each bar.

"flips" of the image data in succession: a left-right flip, followed by an up-down flip.) We similarly update the camera poses and compare depth estimation performance against the unmodified test set. We compare performance on these two test sets against DPSNet [96] and an additional "base" network that has the same structure as MultiViewStereoNet, but uses a traditional feature extraction module that does not compensate for viewpoint changes. In this baseline network, features are extracted for each right image independently before projecting them onto the planes in the reference view. The network is otherwise identical to MultiViewStereoNet.

As shown in Figure 5-8, not performing viewpoint compensation can have severe effects on depth quality. Both the base network and DPSNet, which do no compensation, see a significant increase in both absolute relative depth error and root-mean-square depth error between the two test sets, despite the same information being presented to the networks. Our solution, which does utilize viewpoint compensation, sees no drop in performance. Figure 5-9 shows the qualitative effect of compensation, where the DPSNet depthmap suffers a significant drop in quality

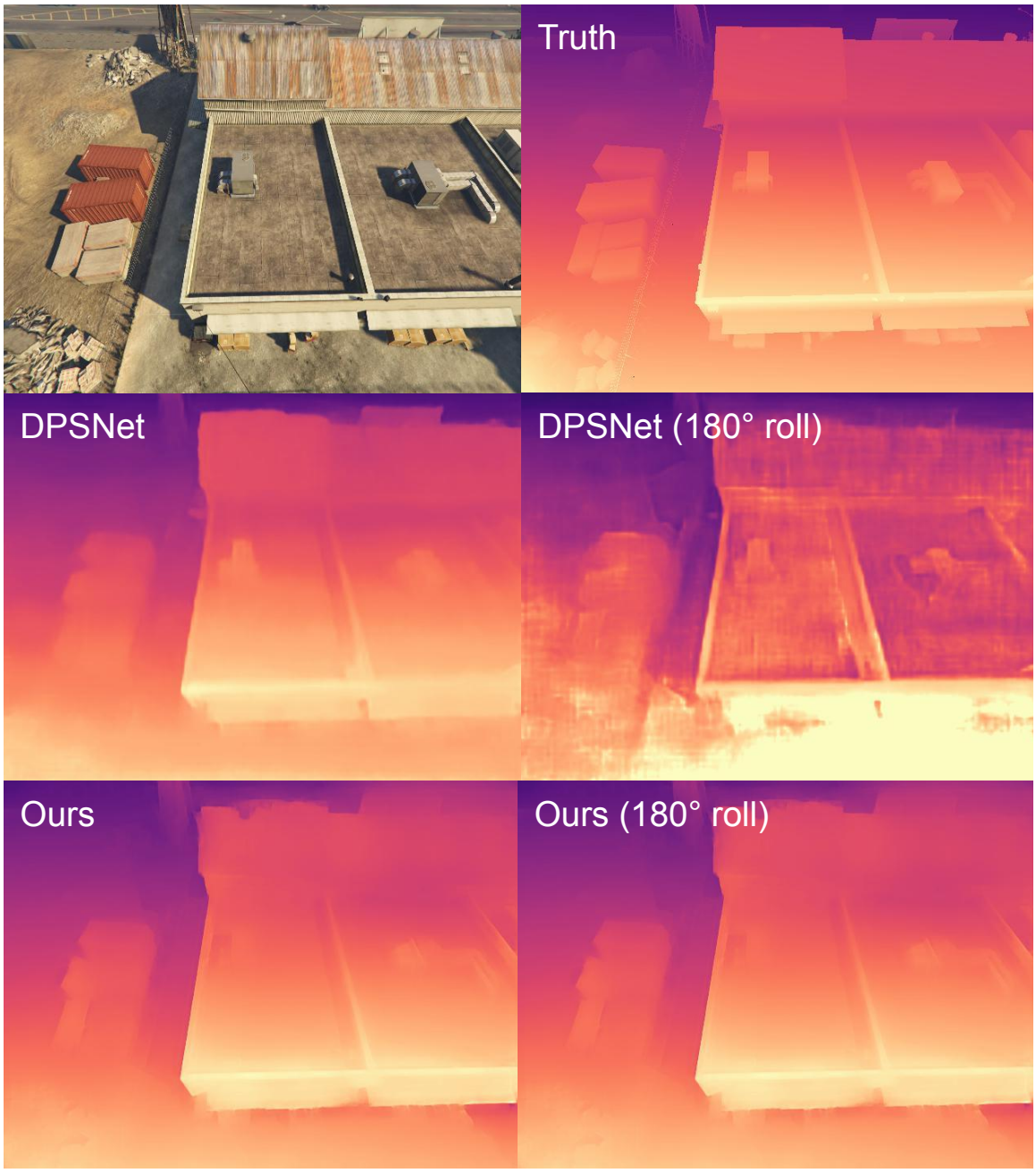


Figure 5-9: *Viewpoint Compensation* – Here we show the qualitative effect of viewpoint-compensated features on the generated depthmaps. DPSNet sees a significant drop in depth quality when one of the input images is rotated about the optical axis by 180 degrees. MultiViewStereoNet sees no change in quality.

when the comparison image is rotated, while that of our solution is unaffected.

5.3.5 Refinement Ablation Experiments

In this section we analyze the effect of the guided refinement layers discussed in Section 5.2.4 in an ablation study. Our full method uses 5 total refinement layers: two for the coarsest depthmap resolution (corresponding to a 16× downsampling of the input) and one for each successive level in a power-of-two image pyramid. Crucially, the refinement layers use the input images as *guidance*, filling in missing details in the depthmap by referring to the image data. We compute depth estimation metrics on the GTA-SfM test set after removing each refinement layer.

The results are presented in Table 5.3. Each row in the table corresponds to removing the finest resolution refinement layer remaining from the previous row. As can be seen, additional refinement layers improve depth estimation performance at the cost of runtime. In particular, the refinement layer corresponding to the finest resolution accounts for almost half of the runtime, validating the general strategy presented in Khamis et al. [108] that suggests that operations at the full resolution should be avoided when possible, especially expensive feature matching. Resource-constrained platforms may benefit from forgoing these final refinement layers in order to bring load and latency to more manageable levels. Future work may focus on further improving the speed of the fine-resolution refinement layers.

5.3.6 Sensitivity to Pose Errors

Our method assumes knowledge of the input camera poses and is therefore sensitive to any errors in the pose estimates. To quantify this dependence, we add zero-mean Gaussian noise onto the true poses of the GTA-SfM test set and compute the absolute relative depth error and RMSE depth error as the standard deviation of the noise increases. As can be seen in Figure 5-10, MultiViewStereoNet can tolerate moderate amounts of noise in translation (up to a standard deviation of 5 cm or so relative to the normalized 1m baseline) without a severe drop in performance. Depth error begins

Refinement Ablation Experiments

Method	Abs Rel ↓	RMSE ↓	α_1 ↑	Runtime [ms] ↓
5 Refiners (Full)	0.084	19.69	0.923	65.1
4 Refiners	0.087	22.37	0.921	31.7
3 Refiners	0.092	24.51	0.915	20.2
2 Refiners	0.098	26.35	0.912	18.4
1 Refiners	0.109	28.73	0.897	17.5
0 Refiners	0.116	30.71	0.891	18.2

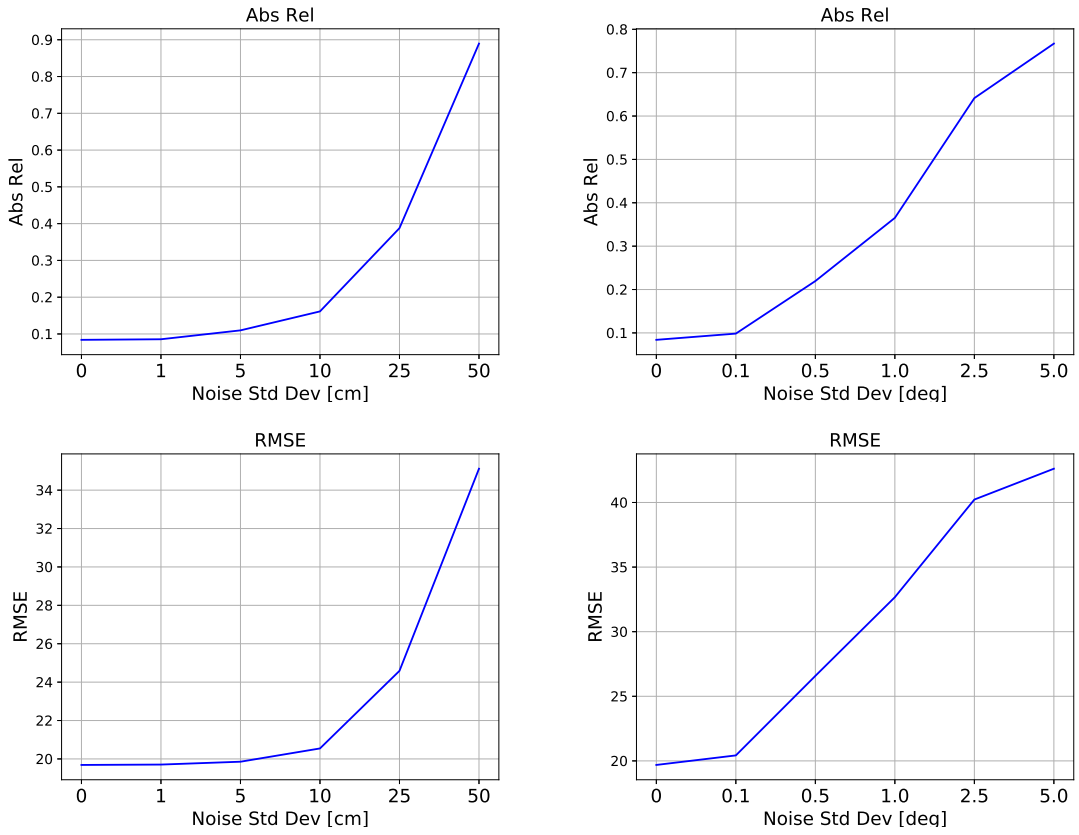
Table 5.3: *Refinement Ablation Experiments* – The table above shows the effect of our guided refinement layers. Our full method uses 5 total refinement layers: two for the coarsest depthmap resolution and one for each successive pyramid level. Each row in the table corresponds to removing the finest resolution refinement layer from the previous row. As can be seen, additional refinement layers improve depth estimation performance at the cost of runtime.

to increase dramatically with translation noise above a standard deviation of 10 cm, however. Unsurprisingly, our method is more sensitive to orientation errors. While rotation noise with a standard deviation of 0.1 degrees can be handled gracefully, noise with a standard deviation above 0.5 degrees leads to significant increases in depth error.

One approach to add robustness to pose errors would be to refine the incoming pose estimates internally such that the poses and depths are jointly optimized to reduce the training loss. A separate refiner network could be envisioned that adds a small residual to the camera poses to compensate for any errors. We leave the investigation of this pose refinement network as future work.

5.3.7 Improvements

In addition to the impact of pose accuracy on depth estimation performance detailed in the previous section, there are a number of ways in which our method could be improved upon. First, the addition of a photometric consistency loss (similar to the one used to train the coarse depth prediction network described in Chapter 4) could be applied to augment the supervised training loss. Relying on fully labeled depthmaps to infer the weights of the network limits the amount of data available for training.



(a) Translation Noise (b) Rotation Noise

Figure 5-10: *Pose Error Sensitivity* – Our method assumes knowledge of the input camera poses and is therefore sensitive to any errors in the pose estimates. To quantify the effect, here we add Gaussian noise onto the true poses and plot the absolute relative depth error and RMSE depth error as the standard deviation of the noise increases. In the left column, we show the effect of translation noise, while in the right column we show the effect of rotation noise.

However, photometric consistency could be used to provide a training signal when groundtruth depths are incomplete (e.g., when LIDAR is used to label the training data) or missing entirely (e.g., when off-the-shelf video is available but lacks depth annotations). Second, we have observed that the choice of depth samples that define our cost volume planes has a large impact on our final depthmap quality. Using a poor set of depth samples relative to the true scene depths reduces the quality of depth information at the coarsest image scales and therefore increases the burden on the subsequent refinement layers to provide the correct depth information. It may be possible, however, to learn a good set of depth samples directly from the reference

image (similar to the depth prediction techniques outlined in Section 2.5.3).

5.4 Conclusion

In this chapter we presented a learning-based method for MVS depth estimation capable of recovering depth from images taken from known, but otherwise arbitrary, viewpoints. Our key insight is that by compensating for the known viewpoint changes during feature extraction, our network can learn features that are projected by construction. This technique lessens the burden on the network to learn invariant features, thereby increasing robustness to viewpoint changes during matching. We employ low-resolution techniques from Khamis et al. [108] and present a novel incremental extraction network to perform this viewpoint compensation efficiently. We show reconstruction performance on benchmark datasets comparable to the state-of-the-art, while being significantly more efficient.

Chapter 6

Conclusion

In this thesis, we have shown three innovations to the problems of monocular SLAM and monocular depth estimation that bring full 3D understanding of the environment viewed by a moving monocular camera one step closer to reality. Monocular SLAM and monocular depth estimation are critical components in many emerging fields, such as self-driving cars, autonomous drones, and augmented reality. The ability to accurately and efficiently infer both the pose of a moving camera and a dense model of the surrounding environment from the images alone is of fundamental importance to realizing these new technologies. We have shown that the current state of the art in monocular SLAM and depth estimation fall short of the requirements for these applications, however, especially in terms of accuracy and computational efficiency on low-SWaP platforms. We have argued in this thesis, however, that untapped sources of prior information, along with targeted applications of machine learning, may be used to address these shortcomings. Many approaches to monocular SLAM and monocular depth estimation do not use all the information available to them to produce solutions. By identifying and exploiting these facets of the two problems, we are able to show improvements in accuracy and speed above the state of the art.

In Chapter 3 we showed how dense monocular depth estimation methods *over-sample* scenes with points relative to the actual geometric complexity of surroundings. Even for moderate image resolutions, the number of samples in a given dense depthmap number in the hundreds of thousands or even millions. This sampling

density is fixed, no matter how simple the observed environment actually is, which is a significant computational burden that simultaneously increases the likelihood of noise and outliers corrupting the depths. Many environments of interest can be well-described as *piecewise planar*, however, and each of these planes can be represented with only a small number of parameters when encoded using *triangular meshes*. We exploited insight and developed and fast meshing algorithm that efficiently estimates triangular meshes of the environment. By intelligently adapting the number of mesh vertices to the observed scene, we are able to generate accurate reconstructions with far fewer depths, drastically accelerating the depth estimation process. We also showed that interpreting the triangular mesh as a *graph* allows us to apply sophisticated spatial regularization techniques that would be intractable otherwise using only a standard CPU.

In Chapter 4, we showed how metric scale is fundamentally unobservable in monocular SLAM when only the geometric content of the images is considered. Monocular cameras are *bearing* sensors that only measure the direction of incoming light, not the metric distance that the light traveled to the sensor. However, for monocular SLAM solutions to be of any utility in robotics or augmented reality, the true metric scale of the solution is required. Typically, approaches to sidestepping this issue usually involve additional sensors and hardware, such as stereo cameras or inertial methods, but these techniques can be brittle and difficult to implement and tune. We instead reasoned that the *semantic* content of the incoming images could be used infer the metric scale. To that end, we trained a small neural network that regresses depth from monocular images. The metric scale of the depths is encoded in the network weights after being trained on a large dataset of calibrated stereo imagery. We then fuse the metric depths from this network with the unscaled poses produced from a sparse monocular SLAM frontend to generate fully scaled solutions without additional hardware, sensors, or compute.

Finally, in Chapter 5, we showed how dense monocular depth estimation relies on robustly associating pixels across multiple views. The appearance of objects can be altered significantly when viewed from different vantage points, however, under-

going scale, rotation, and perspective transformations depending on the arrangement of the input cameras. A popular approach to addressing this challenge is to learn alternative representations of the imagery that can be more reliably associated. However, the current solutions for learned depth estimation do not fully take advantage of the knowledge of the camera poses. Feature extraction in these networks is performed on each image independently, without considering the relative poses of the cameras. This means that the features learned must be invariant to scale, rotation, and perspective changes, which is a challenging machine learning problem. Instead, we reasoned that by compensating for the change in viewpoint between the reference and the comparison images inside the feature extraction process itself, we would be able to more reliably associate features and improve depth estimation quality across different views. Performing this compensation naively, however, can be computationally expensive. We therefore developed a novel feature extraction network that *incrementally* computes features that are viewpoint compensated, obviating the need for many additional convolutional layers. We then paired this new feature network with the fast refinement techniques proposed in StereoNet [108], but generalized to the multi-view setting, for additional computational savings. The combination of these two improvements allow for improved depth estimation accuracy and completeness, while being significantly faster that the state of the art.

We believe the techniques presented in this thesis move monocular SLAM and monocular depth estimation one step closer to being fully usable in the wild. Identifying and appropriately leveraging prior information about these perception problems can realize meaningful advances in the field. Before concluding, we will briefly discuss some additional areas of future work that could build on top of the ideas presented in the preceding chapters.

6.1 Future Work: Learning Mesh Vertices

The vertices in the meshing approach described in Chapter 3 are selected based on the gradient magnitude and direction at a particular pixel. The image is first divided

into grid cells and the pixel in each grid cell with the strongest *trackability score* is selected for depth estimation. When the inverse depth variance for that pixel drops below a threshold, the pixel is added as a vertex into the mesh. The size of the grid cells is controlled by a user-defined parameter. In our experiments, we found reasonable settings for this parameter that balance reconstruction detail with speed. However, more can be done to intelligently select the mesh vertices.

For instance, note that the pixels considered for potential mesh vertices are those that lie in regions of the image with high gradient. These pixels may or may not correspond to regions of the observed surface that require an additional vertex to improve reconstruction accuracy. Rather than considering pixels that only have high image gradient, we would ideally want to consider pixels with high image gradient that also have a high likelihood of corresponding to points on the 3D surface with high curvature (or *surface gradient*) such as corners and depth discontinuities. If we were able to select only these pixels for mesh vertices, we would be able to reduce the number of depths that need to be estimated even further without affecting reconstruction accuracy. Predicting which pixels in the image may correspond to points with high surface gradient may be a promising application of deep learning. Training data with labels for high surface gradient pixels would be required, which may limit the number of applicable datasets, but simulated imagery may be viable alternative, especially if access to the underlying meshes of the simulated environment could be obtained.

6.2 Future Work: Learning Scale without Depth

The metric monocular SLAM method outlined in Chapter 4 relies on a small neural network to regress metric depth from low-resolution images. These metric depth measurements are then fused with unscaled poses from a sparse monocular SLAM pipeline to generic a fully metric solution. One of the benefits of this approach is that its use of machine learning is *targeted* – it does not ignore the epipolar constraints observable in the input data and only exploits prior information for the unobservable

scale. An even more targeted strategy would be to regress a metric scale value directly from pairs of images without going through an intermediate depth layer.

Learning the metric scale of the *motion* between two images – as opposed to the metric *depth* from a single image – should be possible. The network would need to take as input two images and perhaps their relative pose estimated from the unscaled monocular SLAM pipeline and output a single number representing the metric scale of the translation between the images. The network could then be queried with pairs of keyframes from the monocular SLAM graph in the same way that loop closure candidates are verified. Once a scale estimate is produced, a factor linking these keyframes could be inserted into the graph, constraining the scale of the solution to the true metric scale. How to construct this network is unclear, however. Given the presence of the images, some form of CNN would probably be most appropriate, but the question of how many layers, how many feature channels, how many residual connections, whether any fully connected layers should be used, etc. would need to be investigated.

6.3 Future Work: Learning Depth Online

The learned depth estimation approach presented in Chapter 5 is trained using input images with known poses and groundtruth depth labels. While datasets do exist that satisfy these requirements, generating pose and depth labels can be an expensive and time consuming process, usually requiring additional sensors such as GPS, active motion capture systems, or LIDAR. Reducing the amount of labeled data required would open up a far larger and more varied set of training data to improve depth estimation quality.

Recent advances in self-supervised depth estimation, however, could allow for our MultiViewStereoNet approach to be trained without depth labels. Rather than minimizing a supervised training loss, the depthmap estimated by the network could be used to warp the reference image into the comparison image frame. The photometric error between warped reference image and the comparison image would provide a

self-supervised training signal to infer the network weights. Furthermore, if a separate sparse monocular SLAM process were used to generate poses, this self-supervised training procedure could be run entirely online, allowing the network to adapt gracefully to new environments. This type of "online adaption" of neural networks is an emerging field that may allow deep neural networks to mitigate the issue of generalizing beyond their training sets. Determining how best to optimize the network online, however, is an open problem that would require serious investigation. Should a standard stochastic gradient descent step be applied for every new image or should certain criteria be met to trigger the adaption? Should all the weights in the network be updated or only a subset? How do we prevent the network from "forgetting" the training data it has seen previously? These questions — and many more — would need to be explored for a compelling solution.

This potential research direction, along with the others listed in this chapter, may hopefully prove to be fertile ground for future work that build upon the ideas discussed in this thesis. As more and more thought is invested in the monocular SLAM and monocular depth estimation problems, our hope is that more sources of untapped prior information are identified to further develop and advance the field.

Bibliography

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, 2015. Software available from tensorflow.org.

[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] Sameer Agarwal and Keir Mierle. Ceres Solver: Tutorial & Reference. *Google Inc*, 2012.

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

[5] H Alvarez, Lina María Paz, J Sturm, and D Cremers. Collision Avoidance for Quadrotors with a Monocular Camera. In *Experimental Robotics*. Springer, 2016.

[6] Padmanabhan Anandan. A Computational Framework and an Algorithm for the Measurement of Visual Motion. *IJCV*, 1989.

[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] Alireza Bab-Hadiashar and David Suter. Robust Optic Flow Computation. *IJCV*, 1998.

[9] Tim Bailey and Hugh Durrant-Whyte. Simultaneous Localization and Mapping: Part II. *IEEE Robotics & Automation Magazine*, 2006.

[10] Henry Harlyn Baker. Depth from Edge and Intensity Based Stereo. Technical report, DTIC Document, 1982.

[11] Simon Baker and Iain Matthews. Lucas-Kanade 20 Years On: A Unifying Framework. *IJCV*, 2004.

[12] Jonathan T Barron. A General and Adaptive Robust Loss Function. In *Proc. CVPR*, 2019.

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

[14] Peter N Belhumeur. A Bayesian Approach to Binocular Steropsis. *IJCV*, 1996.

[15] Peter N Belhumeur and David Mumford. A Bayesian Treatment of the Stereo Correspondence Problem using Half-Occluded Regions. In *Proc. CVPR*, 1992.

[16] James R Bergen, Patrick Anandan, Keith J Hanna, and Rajesh Hingorani. Hierarchical Model-based Motion Estimation. In *Proc. ECCV*, 1992.

[17] Michael J Black and Paul Anandan. The Robust Estimation of Multiple Motions: Parametric and Piecewise-Smooth Flow Fields. *Computer Vision and Image Understanding*, 1996.

[18] Michael J Black and Anand Rangarajan. On the Unification of Line Processes, Outlier Rejection, and Robust Statistics with Applications in Early Vision. *IJCV*, 1996.

[19] Michael Bleyer, Christoph Rhemann, and Carsten Rother. PatchMatch Stereo-Stereo Matching with Slanted Support Windows. In *Proc. BMVC*, 2011.

[20] Michael Bloesch, Jan Czarnowski, Ronald Clark, Stefan Leutenegger, and Andrew J Davison. CodeSLAM – learning a compact, optimisable representation for dense visual SLAM. In *Proc. CVPR*, 2018.

[21] Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast Approximate Energy Minimization via Graph Cuts. *Trans. PAMI*, 2001.

[22] Kristian Bredies, Karl Kunisch, and Thomas Pock. Total Generalized Variation. *Journal on Imaging Sciences*, 2010.

[23] Adrian Broadhurst, Tom W Drummond, and Roberto Cipolla. A Probabilistic Framework for Space Carving. In *Proc. ICCV*, 2001.

[24] Michael Burri, Janosch Nikolic, Pascal Gohl, Thomas Schneider, Joern Rehder, Sammy Omari, Markus W Achtelik, and Roland Siegwart. The EuRoC Micro Aerial Vehicle Datasets. *IJRR*, 2016.

[25] Cesar Cadena, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, José Neira, Ian Reid, and John J Leonard. Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age. *Trans. on Robotics*, 2016.

[26] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. BRIEF: Binary Robust Independent Elementary Features. In *Proc. ECCV*, 2010.

[27] Luca Carlone. A Convergence Analysis for Pose Graph Optimization via Gauss-Newton Methods. In *Proc. ICRA*, 2013.

[28] Luca Carlone and Frank Dellaert. Duality-based Verification Techniques for 2D SLAM. In *Proc. ICRA*, 2015.

[29] Luca Carlone, David M Rosen, Giuseppe Calafiore, John J Leonard, and Frank Dellaert. Lagrangian Duality in 3D SLAM: Verification Techniques and Optimal Solutions. In *Proc. IROS*, 2015.

[30] Antonin Chambolle, Vicent Caselles, Daniel Cremers, Matteo Novaga, and Thomas Pock. An Introduction to Total Variation for Image Analysis. *Theoretical foundations and numerical methods for sparse recovery*, 2010.

[31] 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.

[32] Alessandro Chiuso, Paolo Favaro, Hailin Jin, and Stefano Soatto. MFm: 3-D Motion From 2-D Motion Causally Integrated Over Time-Part II: Implementation. In *Proc. ECCV*, 2000.

[33] Javier Civera, Andrew J Davison, and JM Martinez Montiel. Inverse Depth Parametrization for Monocular SLAM. *Trans. on Robotics*, 2008.

[34] Ronald Clark, Sen Wang, Hongkai Wen, Andrew Markham, and Niki Trigoni. VINet: Visual-Inertial Odometry as a Sequence-to-Sequence Learning Problem. In *AAAI*, 2017.

[35] Robert T Collins. A Space-Sweep Approach to True Multi-Image Matching. In *Proc. CVPR*, 1996.

[36] W Bruce Culbertson, Thomas Malzbender, and Greg Slabaugh. Generalized Voxel Coloring. In *International Workshop on Vision Algorithms*, 1999.

[37] Brian Curless and Marc Levoy. A Volumetric Method for Building Complex Models from Range Images. In *Proc. on Computer Graphics and Interactive Techniques*, 1996.

[38] Jan Czarnowski, Tristan Laidlow, Ronald Clark, and Andrew J Davison. DeepFactors: Real-time probabilistic dense monocular SLAM. *IEEE Robotics and Automation Letters*, 2020.

[39] Shreyansh Daftry, Sam Zeng, Arbaaz Khan, Debadeepta Dey, Narek Melik-Barkhudarov, J Andrew Bagnell, and Martial Hebert. Robust Monocular Flight in Cluttered Outdoor Environments. *arXiv:1604.04779*, 2016.

[40] Navneet Dalal and Bill Triggs. Histograms of Oriented Gradients for Human Detection. In *Proc. CVPR*. IEEE, 2005.

[41] Andrew J Davison. Real-Time Simultaneous Localisation and Mapping with a Single Camera. In *Proc. ICCV*, 2003.

[42] Andrew J Davison. FutureMapping: The Computational Structure of Spatial AI Systems. *arXiv preprint arXiv:1803.11288*, 2018.

[43] Frank Dellaert, Dieter Fox, Wolfram Burgard, and Sebastian Thrun. Monte Carlo Localization for Mobile Robots. In *Proc. ICRA*, 1999.

[44] Frank Dellaert and Michael Kaess. Square Root SAM: Simultaneous Localization and Mapping via Square Root Information Smoothing. *IJRR*, 2006.

[45] Frank Dellaert, Steven M Seitz, Charles E. Thorpe, and Sebastian Thrun. Structure from Motion without Correspondence. In *Proc. CVPR*, 2000.

[46] 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.

[47] M. W. M. 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.

[48] Jeffrey Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell. Long-Term Recurrent Convolutional Networks for Visual Recognition and Description. In *Proc. CVPR*, 2015.

[49] Arnaud Doucet, Nando De Freitas, and Neil Gordon. An Introduction to Sequential Monte Carlo Methods. In *Sequential Monte Carlo Methods in Practice*. IEEE, 2001.

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

[51] Ethan Eade and Tom Drummond. Edge Landmarks in Monocular SLAM. In *Proc. BMVC*, 2006.

[52] Ethan Eade and Tom Drummond. Scalable Monocular SLAM. In *Proc. CVPR*, 2006.

[53] Ethan Eade and Tom Drummond. Monocular SLAM as a Graph of Coalesced Observations. In *Proc. ICCV*, 2007.

[54] David Eigen, Christian Puhrsch, and Rob Fergus. Depth Map Prediction from a Single Image using a Multi-Scale Deep Network. In *NeurIPS*, 2014.

[55] Jakob Engel, Vladlen Koltun, and Daniel Cremers. Direct Sparse Odometry. *Trans. PAMI*, 2017.

[56] Jakob Engel, Thomas Schöps, and Daniel Cremers. LSD-SLAM: Large-Scale Direct Monocular SLAM. *Proc. ECCV*, 2014.

[57] Jakob Engel, Jürgen Sturm, and Daniel Cremers. Camera-Based Navigation of a Low-Cost Quadrocopter. In *Proc. IROS*, 2012.

[58] Jakob Engel, Jurgen Sturm, and Daniel Cremers. Semi-Dense Visual Odometry for a Monocular Camera. In *Proc. ICCV*, 2013.

[59] 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.

[60] Ryan M Eustice, Hanumant Singh, and John J Leonard. Exactly Sparse Delayed-State Filters. In *Proc. ICRA*, 2005.

[61] Ryan M Eustice, Hanumant Singh, and John J Leonard. Exactly Sparse Delayed-State Filters for View-Based SLAM. *Trans. on Robotics*, 2006.

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

[63] 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.

[64] Christian Forster, Luca Carlone, Frank Dellaert, and Davide Scaramuzza. IMU Preintegration on Manifold for Efficient Visual-Inertial Maximum-a-Posteriori Estimation. In *Proc. RSS*, 2015.

[65] Christian Forster, Matia Pizzoli, and Davide Scaramuzza. SVO: Fast Semi-Direct Monocular Visual Odometry. In *Proc. ICRA*, 2014.

[66] Dieter Fox, Wolfram Burgard, Frank Dellaert, and Sebastian Thrun. Monte Carlo Localization: Efficient Position Estimation for Mobile Robots. In *Proc. AAAI*, 1999.

[67] Pascal Fua and Yvan G Leclerc. Object-Centered Surface Reconstruction: Combining Multi-Image Stereo and Shading. *IJCV*, 1995.

[68] Yasutaka Furukawa and Jean Ponce. Accurate, Dense, and Robust Multiview Stereopsis. *Trans. PAMI*, 2010.

[69] Dorian Gálvez-López, Marta Salas, Juan D Tardós, and JMM Montiel. Real-Time Monocular Object SLAM. *Robotics and Autonomous Systems*, 2016.

[70] Ravi Garg, Vijay Kumar Bg, Gustavo Carneiro, and Ian Reid. Unsupervised CNN for Single View Depth Estimation: Geometry to the Rescue. In *Proc. ECCV*, 2016.

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

[72] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite. In *Proc. CVPR*, 2012.

[73] Davi Geiger, Bruce Ladendorf, and Alan Yuille. Occlusions and Binocular Stereo. *IJCV*, 1995.

[74] Stuart Geman and Donald Geman. Stochastic Relaxation, Gibbs distributions, and the Bayesian Restoration of Images. *Trans. PAMI*, 1984.

[75] Clément Godard, Oisin Mac Aodha, and Gabriel J. Brostow. Unsupervised Monocular Depth Estimation with Left-Right Consistency. In *Proc. CVPR*, 2017.

[76] Clément Godard, Oisin Mac Aodha, Michael Firman, and Gabriel J Brostow. Digging into Self-Supervised Monocular Depth Estimation. In *Proc. ICCV*, 2019.

[77] Michael Goesele, Brian Curless, and Steven M Seitz. Multi-View Stereo Revisited. In *Proc. CVPR*, 2006.

[78] Gottfried Graber, Thomas Pock, and Horst Bischof. Online 3D Reconstruction using Convex Optimization. In *Proc. ICCV Workshops*, 2011.

[79] W Nicholas Greene. Real-time Dense Simultaneous Localization and Mapping using Monocular Cameras. Master's thesis, Massachusetts Institute of Technology, 2016.

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

[81] W. Nicholas Greene and Nicholas Roy. FLaME: Fast Lightweight Mesh Estimation using Variational Smoothing on Delaunay Graphs. In *Proc. ICCV*, 2017.

[82] W Nicholas Greene and Nicholas Roy. Metrically-Scaled Monocular SLAM using Learned Scale Factors. In *Proc. ICRA*, 2020.

[83] W Nicholas Greene and Nicholas Roy. MultiViewStereoNet: Fast Multi-View Stereo Depth Estimation using Incremental Viewpoint-Compensated Feature Extraction. In *Proc. ICRA*, 2021.

[84] Vitor Guizilini, Rares Ambrus, Sudeep Pillai, Allan Raventos, and Adrien Gaidon. 3D Packing for Self-Supervised Monocular Depth Estimation. In *Proc. CVPR*, 2020.

[85] Anthony Ha. Daily Crunch: Facebook unveils the Oculus Quest 2. `https://techcrunch.com/2020/09/16/daily-crunch-oculus-quest`.

[86] Frank R Hampel, Elvezio M Ronchetti, Peter J Rousseeuw, and Werner A Stahel. *Robust Statistics: The Approach Based on Influence Functions*. John Wiley & Sons, 2011.

[87] Chris Harris and Mike Stephens. A Combined Corner and Edge Detector. In *Alvey Vision Conference*, 1988.

[88] Christopher G Harris and JM Pike. 3D Positional Integration from Image Sequences. *Image and Vision Computing*, 1988.

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

[90] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *Proc. CVPR*, 2016.

[91] Vu Hoang Hiep, Renaud Keriven, Patrick Labatut, and Jean-Philippe Pons. Towards High-Resolution Large-Scale Multi-View Stereo. In *Proc. CVPR*, 2009.

[92] Heiko Hirschmuller. Stereo Processing by Semi-Global Matching and Mutual Information. *Trans. PAMI*, 2007.

[93] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural computation*, 1997.

[94] Russell Hotten. Carmakers Face Challenge from Google and Apple. `https://www.bbc.com/news/business-31720645`.

[95] Po-Han Huang, Kevin Matzen, Johannes Kopf, Narendra Ahuja, and Jia-Bin Huang. DeepMVS: Learning Multi-View Stereopsis. In *Proc. CVPR*, 2018.

[96] Sunghoon Im, Hae-Gon Jeon, Stephen Lin, and In So Kweon. DPSNet: End-to-end Deep Plane Sweep Stereo. In *Proc. ICLR*, 2019.

[97] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv preprint arXiv:1502.03167*, 2015.

[98] Hiroshi Ishikawa and Davi Geiger. Occlusions, Discontinuities, and Epipolar Lines in Stereo. In *Proc. ECCV*, 1998.

[99] Ganesh Iyer, J Krishna Murthy, Gunshi Gupta, Madhava Krishna, and Liam Paull. Geometric Consistency for Self-Supervised End-to-End Visual Odometry. In *Proc. CVPR Workshops*, 2018.

[100] Max Jaderberg, Karen Simonyan, Andrew Zisserman, et al. Spatial Transformer Networks. In *NeurIPS*, 2015.

[101] Simon J Julier and Jeffrey K Uhlmann. A Counter Example to the Theory of Simultaneous Localization and Map Building. In *Proc. ICRA*. IEEE, 2001.

[102] 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.

[103] Michael Kaess, Ananth Ranganathan, and Frank Dellaert. iSAM: Incremental Smoothing and Mapping. *Trans. on Robotics*, 2008.

[104] Rudolph E Kalman and Richard S Bucy. New Results in Linear Filtering and Prediction Theory. 1961.

[105] Rudolph Emil Kalman. A New Approach to Linear Filtering and Prediction Problems. 1960.

[106] Sing Bing Kang, Richard Szeliski, and Jinxiang Chai. Handling Occlusions in Dense Multi-View Stereo. In *Proc. CVPR*, 2001.

[107] Alex Kendall, Hayk Martirosyan, Saumitro Dasgupta, Peter Henry, Ryan Kennedy, Abraham Bachrach, and Adam Bry. End-to-End Learning of Geometry and Context for Deep Stereo Regression. In *Proc. ICCV*, 2017.

[108] Sameh Khamis, Sean Fanello, Christoph Rhemann, Adarsh Kowdle, Julien Valentin, and Shahram Izadi. StereoNet: Guided Hierarchical Refinement for Real-Time Edge-Aware Depth Prediction. In *Proc. ECCV*, 2018.

[109] Diederik P Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980*, 2014.

[110] Georg Klein and David Murray. Parallel Tracking and Mapping for Small AR Workspaces. In *Proc. ISMAR*, 2007.

[111] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques*. MIT Press, 2009.

[112] Vladimir Kolmogorov and Ramin Zabih. Computing Visual Correspondence with Occlusions using Graph Cuts. In *Proc. ICCV*, 2001.

[113] Kurt Konolige and Willow Garage. Sparse Sparse Bundle Adjustment. In *Proc. BMVC*, 2010.

[114] Kurt Konolige, Giorgio Grisetti, Rainer Kümmerle, Wolfram Burgard, Benson Limketkai, and Regis Vincent. Efficient Sparse Pose Adjustment for 2D Mapping. In *Proc. IROS*, 2010.

[115] Kirsten Korosec. Uber's Self-Driving Unit Starts Mapping Washington, D.C. Ahead of Testing. `https://techcrunch.com/2020/01/23/ubers-self-driving-unit-starts-mapping-washington-d-c-ahead-of-testing`.

[116] Frank R. Kschischang, Brendan J. Frey, and H.-A. Loeliger. Factor Graphs and the Sum-Product Algorithm. *IEEE Transactions on Information Theory*, 2001.

[117] Rainer Kümmerle, Giorgio Grisetti, Hauke Strasdat, Kurt Konolige, and Wolfram Burgard. g^{2}o: A General Framework for Graph Optimization. In *Proc. ICRA*, 2011.

[118] Kiriakos N Kutulakos and Steven M Seitz. A Theory of Shape by Space Carving. *IJCV*, 2000.

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

[120] Jennifer Langston. To the moon and beyond: How HoloLens 2 is helping build NASAâĂŹs Orion spacecraft. `https://news.microsoft.com/innovation-stories/hololens-2-nasa-orion-artemis`.

[121] Frederic Lardinois. Google Maps gets improved Live View AR directions. `https://techcrunch.com/2020/10/01/google-maps-gets-improved-live-view-ar-directions`.

[122] John J Leonard and Hugh F Durrant-Whyte. Mobile Robot Localization by Tracking Geometric Beacons. *IEEE Transactions on Robotics and Automation*, 1991.

[123] John J Leonard and Hugh F Durrant-Whyte. Simultaneous Map Building and Localization for an Autonomous Mobile Robot. In *Proc. IROS*, 1991.

[124] Stefan Leutenegger, Simon Lynen, Michael Bosse, Roland Siegwart, and Paul Furgale. Keyframe-Based Visual-Inertial Odometry using Nonlinear Optimization. *IJRR*, 2015.

[125] Guoying Li. Robust Regression. *Exploring data tables, trends, and shapes*, 1985.

[126] Ruihao Li, Sen Wang, Zhiqiang Long, and Dongbing Gu. UnDeepVO: Monocular Visual Odometry through Unsupervised Deep Learning. In *Proc. ICRA*, 2018.

[127] Fayao Liu, Chunhua Shen, Guosheng Lin, and Ian Reid. Learning Depth from Single Monocular Images using Deep Convolutional Neural Fields. *Trans. PAMI*, 2015.

[128] H Christopher Longuet-Higgins. A Computer Algorithm for Reconstructing a Scene from Two Projections. *Nature*, 1981.

[129] Manolis I. A. Lourakis and Antonis A. Argyros. SBA: A Software Package for Generic Sparse Bundle Adjustment. *Transactions on Mathematical Software (TOMS)*, 2009.

[130] David G Lowe. Object Recognition from Local Scale-Invariant Features. In *Proc. ICCV*, 1999.

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

[132] Feng Lu and Evangelos Milios. Globally Consistent Range Scan Alignment for Environment Mapping. *Autonomous Robots*, 1997.

[133] Bruce D Lucas, Takeo Kanade, et al. An Iterative Image Registration Technique with an Application to Stereo Vision. In *IJCAI*, 1981.

[134] 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.

[135] Donald W Marquardt. An Algorithm for Least-Squares Estimation of Nonlinear Parameters. *Journal of the Society for Industrial and Applied Mathematics*, 1963.

[136] Larry Matthies and STEVENA Shafer. Error Modeling in Stereo Navigation. *Journal on Robotics and Automation*, 1987.

[137] Larry Henry Matthies. Dynamic Stereo Vision. Technical report, Carnegie Mellon University, 1989.

[138] Nikolaus Mayer, Eddy Ilg, Philip Hausser, Philipp Fischer, Daniel Cremers, Alexey Dosovitskiy, and Thomas Brox. A Large Dataset to Train Convolutional Networks for Disparity, Optical Flow, and Scene Flow Estimation. In *Proc. CVPR*, 2016.

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

[140] Michael Montemerlo, Sebastian Thrun, Daphne Koller, and Ben Wegbreit. FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem. In *Proc. AAAI*, 2002.

[141] 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.

[142] Hans P Moravec. Obstacle Avoidance and Navigation in the Real World by a Seeing Robot Rover. Technical report, Stanford University, 1980.

[143] Anastasios I. Mourikis and Stergios I. Roumeliotis. A Multi-State Constraint Kalman Filter for Vision-aided Inertial Navigation. In *Proc. ICRA*, 2007.

[144] David Mumford and Jayant Shah. Optimal Approximations by Piecewise Smooth Functions and Associated Variational Problems. *Communications on Pure and Applied Mathematics*, 1989.

[145] Raul Mur-Artal, J. M. M. Montiel, and Juan D. Tardos. ORB-SLAM: a Versatile and Accurate Monocular SLAM System. *Trans. on Robotics*, 2015.

[146] Raúl Mur-Artal and Juan D Tardós. Probabilistic Semi-Dense Mapping from Highly Accurate Feature-Based Monocular SLAM. In *Proc. RSS*, 2015.

[147] Raúl Mur-Artal and Juan D. Tardós. ORB-SLAM2: an Open-Source SLAM System for Monocular, Stereo and RGB-D Cameras. *Trans. on Robotics*, 2017.

[148] Rafael MuÃśoz-Salinas and Rafael Medina-Carnicer. UcoSLAM: Simultaneous Localization and Mapping by Fusion of KeyPoints and Squared Planar Markers. *Pattern Recognition*, 2020.

[149] Vinod Nair and Geoffrey E Hinton. Rectified Linear Units Improve Restricted Boltzmann Machines. In *Proc. ICML*, 2010.

[150] 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.

[151] Yu Nesterov. Smooth Minimization of Non-smooth Functions. *Mathematical Programming*, 2005.

[152] Richard A Newcombe and Andrew J Davison. Live Dense Reconstruction with a Single Moving Camera. In *Proc. CVPR*, 2010.

[153] Richard A. Newcombe, Steven J. Lovegrove, and Andrew J. Davison. DTAM: Dense Tracking and Mapping in Real-Time. In *Proc. ICCV*, 2011.

[154] David Nistér. An Efficient Solution to the Five-Point Relative Pose Problem. *Trans. PAMI*, 2004.

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

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

[157] Yuichi Ohta and Takeo Kanade. Stereo by Intra-and Inter-Scanline Search using Dynamic Programming. *Trans. PAMI*, 1985.

[158] Kyel Ok, Dinesh Gamage, Tom Drummond, Frank Dellaert, and Nicholas Roy. Monocular Image Space Tracking on a Computationally Limited MAV. In *Proc. ICRA*, 2015.

[159] 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.

[160] Ee Ping Ong and Michael Spann. Robust Optical Flow Computation based on Least-Median-of-Squares Regression. *IJCV*, 1999.

[161] Neal Parikh and Stephen Boyd. Proximal Algorithms. *Foundations and Trends in Optimization*, 2014.

[162] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *NeurIPS*, 2019.

[163] Valentin Peretroukhin and Jonathan Kelly. DPC-Net: Deep Pose Correction for Visual Localization. *IEEE Robotics and Automation Letters*, 2018.

[164] Sudeep Pillai, Rareş Ambruş, and Adrien Gaidon. SuperDepth: Self-Supervised, Super-Resolved Monocular Depth Estimation. In *Proc. ICRA*, 2019.

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

[166] Pedro Piniés, Lina Maria Paz, and Paul Newman. Dense Mono Reconstruction: Living with the Pain of the Plain Plane. In *Proc. ICRA*, 2015.

[167] Matia Pizzoli, Christian Forster, and Davide Scaramuzza. REMODE: Probabilistic, Monocular Dense Reconstruction in Real Time. In *Proc. ICRA*, 2014.

[168] Thomas Pock, Daniel Cremers, Horst Bischof, and Antonin Chambolle. An Algorithm for Minimizing the Mumford-Shah functional. In *Proc. ICCV*, 2009.

[169] Matteo Poggi, Filippo Aleotti, Fabio Tosi, and Stefano Mattoccia. Towards Real-Time Unsupervised Monocular Depth Estimation on CPU. In *Proc. IROS*, 2018.

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

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

[172] 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.

[173] Tong Qin, Peiliang Li, and Shaojie Shen. VINS-Mono: A Robust and Versatile Monocular Visual-Inertial State Estimator. *Trans. On Robotics*, 2018.

[174] René Ranftl, Kristian Bredies, and Thomas Pock. Non-local Total Generalized Variation for Optical Flow Estimation. In *Proc. ECCV*, 2014.

[175] David M Rosen, Luca Carlone, Afonso S Bandeira, and John J Leonard. SE-Sync: A Certifiably Correct Algorithm for Synchronization over the Special Euclidean Group. *IJRR*, 2019.

[176] David M Rosen, Charles DuHadway, and John J Leonard. A Convex Relaxation for Approximate Global Optimization in Simultaneous Localization and Mapping. In *Proc. ICRA*, 2015.

[177] David M Rosen, Michael Kaess, and John J Leonard. RISE: An Incremental Trust-Region Method for Robust Online Sparse Least-Squares Estimation. *Trans. on Robotics*, 2014.

[178] Antoni Rosinol, Marcus Abate, Yun Chang, and Luca Carlone. Kimera: An Open-Source Library for Real-Time Metric-Semantic Localization and Mapping. In *Proc. ICRA*, 2020.

[179] Antoni Rosinol, Torsten Sattler, Marc Pollefeys, and Luca Carlone. Incremental Visual-Inertial 3D Mesh Generation with Structural Regularities. In *Proc. ICRA*, 2019.

[180] Edward Rosten and Tom Drummond. Machine Learning for High-Speed Corner Detection. In *Proc. ECCV*, 2006.

[181] Sébastien Roy and Ingemar J Cox. A Maximum-Flow Formulation of the N-Camera Stereo Correspondence Problem. In *Proc. ICCV*, 1998.

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

[183] Leonid I Rudin, Stanley Osher, and Emad Fatemi. Nonlinear Total Variation Based Noise Removal Algorithms. *Physica D: Nonlinear Phenomena*, 1992.

[184] Anshel Sag. How VR And AR Could Be A Solution To Coronavirus Cancellations For Conferences. https://www.forbes.com/sites/moorinsights/2020/03/02/how-vr-and-ar-could-be-a-solution-to-corona-virus-cancellations-for-conferences

[185] Hideo Saito and Takeo Kanade. Shape Reconstruction in Projective Grid Space from Large Number of Images. In *Proc. CVPR*, 1999.

[186] Ashutosh Saxena, Min Sun, and Andrew Y Ng. Make3D: Learning 3D Scene Structure from a Single Still Image. *Trans. PAMI*, 2008.

[187] Daniel Scharstein and Richard Szeliski. Stereo Matching with Nonlinear Diffusion. *IJCV*, 1998.

[188] Daniel Scharstein and Richard Szeliski. A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms. *IJCV*, 2002.

[189] Grant Schindler, Frank Dellaert, and Sing Bing Kang. Inferring Temporal Order of Images from 3D Structure. In *Proc CVPR*, 2007.

[190] Johannes Lutz Schönberger, Enliang Zheng, Marc Pollefeys, and Jan-Michael Frahm. Pixelwise View Selection for Unstructured Multi-View Stereo. In *Proc. ECCV*, 2016.

[191] Thomas Schöps, Jakob Engel, and Daniel Cremers. Semi-Dense Visual Odometry for AR on a Smartphone. In *Proc. ISMAR*. IEEE, 2014.

[192] Thomas Schöps, Torsten Sattler, Christian Häne, and Marc Pollefeys. 3D Modeling on the Go: Interactive 3D Reconstruction of Large-Scale Scenes on Mobile Devices. In *3D Vision (3DV)*, 2015.

[193] 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.

[194] Steven M Seitz and Charles R Dyer. Photorealistic Scene Reconstruction by Voxel Coloring. *IJCV*, 1999.

[195] 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.

[196] Jonathan Richard Shewchuk. Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator. In *Applied Computational Geometry: Towards Geometric Engineering*, 1996.

[197] Jonathan Richard Shewchuk. Delaunay Refinement Algorithms for Triangular Mesh Generation. *Computational Geometry*, 2002.

[198] Jianbo Shi and Carlo Tomasi. Good Features to Track. In *Proc. CVPR*, 1994.

[199] Jeremy G Siek, Lie-Quan Lee, and Andrew Lumsdaine. *The Boost Graph Library: User Guide and Reference Manual, Portable Documents*. Pearson Education, 2001.

[200] Randall Smith, Matthew Self, and Peter Cheeseman. A Stochastic Map for Uncertain Spatial Relationships. In *Proc. ISRR*, 1987.

[201] Noah Snavely, Steven M Seitz, and Richard Szeliski. Photo Tourism: Exploring Photo Collections in 3D. In *Transactions on Graphics (TOG)*, 2006.

[202] Ted J. Steiner, Robert D. Truax, and Kristoffer Frey. A Vision-aided Inertial Navigation System for Agile High-speed Flight in Unmapped Environments. In *Proc. IEEE Aerospace Conference*, 2017.

[203] Hauke Strasdat, J Montiel, and Andrew J Davison. Scale Drift-Aware Large Scale Monocular SLAM. *Proc. RSS*, 2010.

[204] Hauke Strasdat, JMM Montiel, and Andrew J Davison. Real-time monocular SLAM: Why filter? In *Proc. ICRA*, 2010.

[205] Hauke Strasdat, José MM Montiel, and Andrew J Davison. Visual SLAM: Why filter? *Image and Vision Computing*, 2012.

[206] 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.

[207] 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.

[208] Peter Swerling. *A Proposed Stagewise Differential Correction Procedure for Satellite Tracking and Prediction*. Rand Corporation, 1958.

[209] Richard Szeliski. A Multi-View Approach to Motion and Stereo. In *Proc. CVPR*, 1999.

[210] Richard Szeliski. *Computer Vision: Algorithms and Applications*. Springer Science & Business Media, 2010.

[211] Richard Szeliski and Polina Golland. Stereo Matching with Transparency and Matting. *IJCV*, 1999.

[212] Chengzhou Tang and Ping Tan. BA-Net: Dense bundle adjustment network. *arXiv preprint arXiv:1806.04807*, 2018.

[213] Keisuke Tateno, Federico Tombari, Iro Laina, and Nassir Navab. CNN-SLAM: Real-Time Dense Monocular SLAM with Learned Depth Prediction. In *Proc. CVPR*, 2017.

[214] Brad Templeton. TeslaâĂŹs âĂŸFull Self-DrivingâĂŹ Is 99.9% There, Just 1,000 Times Further To Go. `https://www.forbes.com/sites/bradtempleton/2020/10/23/teslas-full-self-driving-is-999-there-just-1000-times-further-to-go`.

[215] Demetri Terzopoulos. Regularization of Inverse Visual Problems Involving Discontinuities. *Trans. PAMI*, 1986.

[216] Demetri Terzopoulos and Dimitri Metaxas. Dynamic 3D Models with Local and Global Deformations: Deformable Superquadrics. In *Proc. ICCV*, 1990.

[217] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. MIT Press, 2005.

[218] Sebastian Thrun, Dieter Fox, Wolfram Burgard, and Frank Dellaert. Robust Monte Carlo Localization for Mobile Robots. *Artificial intelligence*, 2001.

[219] Sebastian Thrun, Yufeng Liu, Daphne Koller, Andrew Y Ng, Zoubin Ghahramani, and Hugh Durrant-Whyte. Simultaneous Localization and Mapping with Sparse Extended Information Filters. *IJRR*, 2004.

[220] Sebastian Thrun and Michael Montemerlo. The GraphSLAM Algorithm with Applications to Large-Scale Mapping of Urban Structures. *IJRR*, 2006.

[221] Carlo Tomasi and Takeo Kanade. Detection and Tracking of Point Features. Technical report, Carnegie Mellon University, 1991.

[222] Bill Triggs, Philip F McLauchlan, Richard I. Hartley, and Andrew W Fitzgibbon. Bundle Adjustment – A Modern Synthesis. In *International Workshop on Vision Algorithms*, 1999.

[223] B. Ummenhofer, H. Zhou, J. Uhrig, N. Mayer, E. Ilg, A. Dosovitskiy, and T. Brox. DeMoN: Depth and Motion Network for Learning Monocular Stereo. In *Proc. CVPR*, 2017.

[224] Olga Veksler. *Efficient Graph-Based Energy Minimization Methods in Computer Vision*. PhD thesis, Cornell University, 1999.

[225] Matthew R Walter, Ryan M Eustice, and John J Leonard. Exactly Sparse Extended Information Filters for Feature-Based SLAM. *IJRR*, 2007.

[226] 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.

[227] Kaixuan Wang and Shaojie Shen. MVDepthNet: Real-Time Multiview Depth Estimation Neural Network. In *Proc. 3DV*, 2018.

[228] Kaixuan Wang and Shaojie Shen. Flow-Motion and Depth Network for Monocular Stereo and Beyond. In *Proc. ICRA*, 2020.

[229] Rui Wang, Martin Schworer, and Daniel Cremers. Stereo DSO: Large-Scale Direct Sparse Visual Odometry with Stereo Cameras. In *Proc. ICCV*, 2017.

[230] Sen Wang, Ronald Clark, Hongkai Wen, and Niki Trigoni. DeepVO: Towards End-to-End Visual Odometry with Deep Recurrent Convolutional Neural Networks. In *Proc. ICRA*, 2017.

[231] Andreas Wendel, Michael Maurer, Gottfried Graber, Thomas Pock, and Horst Bischof. Dense Reconstruction On-the-Fly. In *Proc. CVPR*, 2012.

[232] Stephen Williams, Vadim Indelman, Michael Kaess, Richard Roberts, John J Leonard, and Frank Dellaert. Concurrent Filtering and Smoothing: A Parallel Architecture for Real-Time Navigation and Full Smoothing. *IJRR*, 2014.

[233] Yuxin Wu and Kaiming He. Group Normalization. In *Proc. ECCV*, 2018.

[234] H. Yang, J. Shi, and L. Carlone. TEASER: Fast and Certifiable Point Cloud Registration. *Trans. on Robotics*, 2020.

[235] Heng Yang, Pasquale Antonante, Vasileios Tzoumas, and Luca Carlone. Graduated Non-Convexity for Robust Spatial Perception: From Non-Minimal Solvers to Global Outlier Rejection. *IEEE Robotics and Automation Letters*, 2020.

[236] N. Yang, R. Wang, J. Stueckler, and D. Cremers. Deep Virtual Stereo Odometry: Leveraging Deep Depth Prediction for Monocular Direct Sparse Odometry. In *Proc. ECCV*, 2018.

[237] Yao Yao, Zixin Luo, Shiwei Li, Tian Fang, and Long Quan. MVSNet: Depth Inference for Unstructured Multi-View Stereo. In *Proc. ECCV*, 2018.

[238] Ramin Zabih and John Woodfill. Non-Parametric Local Transforms for Computing Visual Correspondence. In *Proc. ECCV*, 1994.

[239] Jure Zbontar and Yann LeCun. Computing the Stereo Matching Cost with a Convolutional Neural Network. In *Proc. CVPR*, 2015.

[240] Huangying Zhan, Chamara Saroj Weerasekera, Jia-Wang Bian, and Ian Reid. Visual Odometry Revisited: What Should be Learnt? In *Proc. ICRA*, 2020.

[241] Tinghui Zhou, Matthew Brown, Noah Snavely, and David G. Lowe. Unsupervised Learning of Depth and Ego-Motion from Video. In *Proc. CVPR*, 2017.