BOR^{2}G: Building Optimal Regularised Reconstructions with GPUs (in cubes)

Michael A. Tanner

New College

University of Oxford

A thesis submitted for the degree of

Doctor of Philosophy

Hilary 2017

To Katie

Acknowledgements

I will be forever indebted to Paul Newman, Pedro Piniés, and Lina Maria Paz. Over the past few years, each played the role of teacher, mentor, and tutor—guiding me through my fumbling exploration of the robotics research world. Thank you for the late nights, which often turned into early mornings. Thank you for patiently explaining and re-explaining "easy" concepts to me until I finally grasped them. Thank you for encouraging me to press ahead even though I was convinced my research had reached a dead end. This thesis would not have been possible without your support.

Thanks to my fellow members of the Oxford Robotics Institute: the engineers who ensure our platforms actually move, the postdocs and researchers who provide salient nudges that save months of research time, and my student cohort who are always willing to lend an ear or hand. In particular, Alex Bewley's TensorFlow work was a godsend, enabling me to complete the neural network research presented in Chapter 6.

Finally, the all-too-often unsung heroes are my wife (Katie) and children (Eliza, Piper, and Gideon) who provided me continual encouragement. Katie, I love that you were always waiting at the door, excitedly asking to hear about my latest and greatest research breakthroughs. I continue to fall deeper in love with you every day. Eliza, Piper and Gideon were never-ending fountains of distraction, entertainment, and joy to cheer me up even on the toughest of days.

Michael Tanner
10 March 2017

Abstract

Robots require high-quality *maps*—internal representations of their operating workspace—to localise, path plan, and perceive their environment. Until recently, these maps were restricted to sparse, 2D representations due to computational, memory, and sensor limitations. With the widespread adoption of high-quality sensors and graphics processors for parallel processing, these restrictions no longer apply: dense 3D maps are feasible to compute in real time (i.e., at the input sensor's frame rate).

This thesis presents the theory and system to create large-scale dense 3D maps (i.e., reconstruct continuous surface models) using only sensors found on modern autonomous automobiles: 2D laser, 3D laser, and cameras. In contrast to active RGB-D cameras, passive cameras produce noisy surface observations and must be regularised in both 2D and 3D to create accurate reconstructions. Unfortunately, straight-forward application of 3D regularisation causes undesired surface interpolation and extrapolation in regions unexplored by the robot. We propose a method to overcome this challenge by informing the regulariser of the specific subsets of 3D surfaces upon which to operate. When combined with a compressed voxel grid data structure, we demonstrate our system fusing data from both laser and camera sensors to reconstruct 7.3 km of urban environments. We evaluate the quantitative performance of our proposed method through the use of synthetic and real-world datasets—including datasets from Stanford's Burghers of Calais, University of Oxford's RobotCar, University of Oxford's Dense Reconstruction, and Karlsruhe Institute of Technology's KITTI—compared to ground-truth laser data. With only stereo camera inputs, our regulariser reduces the 3D reconstruction metric error between 27% to 36% with a final median accuracy ranging between 4 cm to 8 cm.

Furthermore, by augmenting our system with object detection, we remove ephemeral objects (e.g., automobiles, bicycles, and pedestrians) from the input sensor data and target our regulariser to interpolate the occluded urban surfaces. Augmented with Kernel Conditional Density Estimation, our regulariser creates reconstructions with median errors between 5.64 cm and 9.24 cm.

Finally, we present a machine-learning pipeline that learns, in an automatic fashion, to recognise the errors in dense reconstructions. Our system trains on image and laser data from a 3.8 km urban sequence. Using a separate 2.2 km urban sequence, our pipeline consistently identifies error-prone regions in the image-based dense reconstruction.

Contents

List of Figures

List of Tables

List of Abbreviations

BOR^{2}G Building Optimal Regularised Reconstructions with GPUs

TV Total Variation

TGV Total Generalised Variation

HVG Hashing Voxel Grid

KCDE Kernel Conditional Density Estimation

1D one dimensional

2D two dimensional

3D three dimensional

SDF Signed Distance Function

TSDF Truncated Signed Distance Function

MAP *maximum a posteriori*

VO Visual Odometry

IMU Inertial Measurement Unit

SLAM Simultaneous Localisation and Mapping

DTAM Dense Tracking and Mapping

CUDA Compute Unified Device Architecture

PDF Probability Density Function

GPGPU General Purpose Graphics Processing Unit

GPU Graphics Processing Unit

CPU Central Processing Unit

SfM Structure from Motion

ORI Oxford Robotics Institute

NASA National Aeronautics and Space Administration

LIBELAS Library for Efficient Large-scale Stereo Matching

LUT lookup table

LF Legendre-Fenchel

GLSL OpenGL Shading Language

BDA Building Damage Assessment

A map does not just chart, it unlocks and formulates meaning; it forms bridges between here and there, between disparate ideas that we did not know were previously connected.

— Reif Larsen, *The Selected Works of T.S. Spivet*

1 Introduction

Contents

Dense reconstruction—continuous three dimensional (3D) surface models created from sensor data—is a vibrant research area in which the computer vision and robotics communities have shown a surge of interest over the past decade. These maps aid a robot in visualising, localising, and navigating within an operating environment. The 3D dense reconstructions targeted in this thesis stand in contrast to traditional two dimensional (2D) point-based (i.e., sparse) maps. 3D maps allow the robot to operate in more complex environments (e.g., hills, multi-floor buildings, warehouses, etc.) while providing the potential to improve the robustness of algorithms in previously-ambiguous scenarios. For example, depending on the sensor mounting height, a map created from a horizontal 2D lidar sensor might struggle to differentiate between a building support column and a cardboard box; 3D maps with height information help resolve this ambiguity. A dense (i.e., continuous) map also reduces uncertainty for algorithms using the map—e.g., a path planner can now differentiate between a physical gap in a wall rather than low sensor coverage.

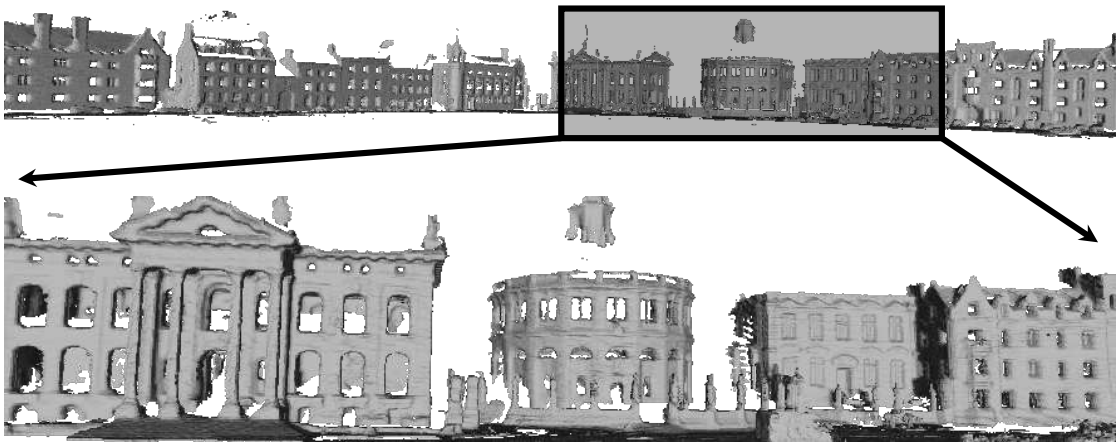


Figure 1.1: Example Dense 3D Reconstructions (Mesh). The software pipeline presented in this thesis created the above urban reconstruction by fusing data from the Oxford Broad Street dataset.

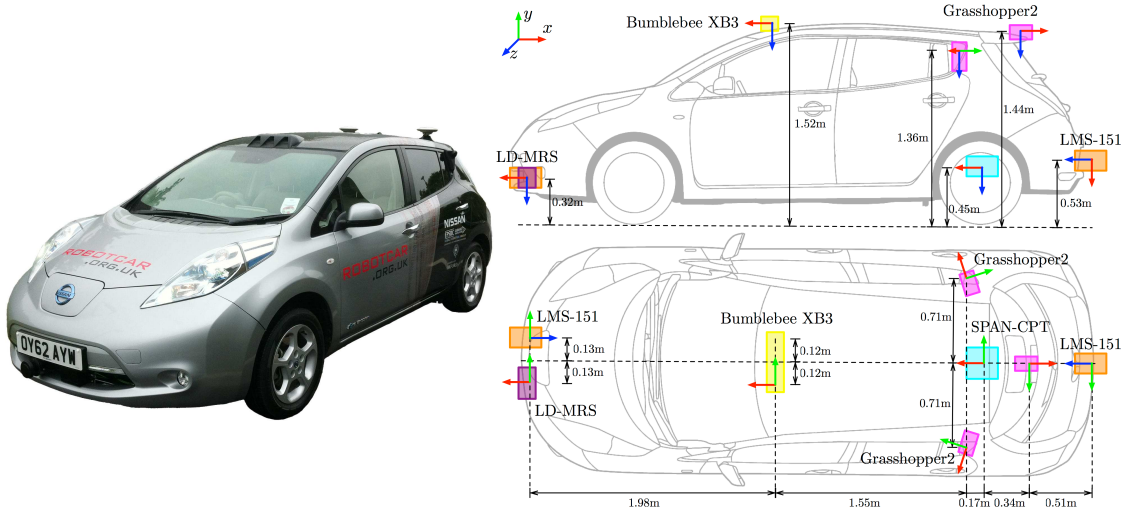
Due to computational and memory constraints, early robotics work focused on sparse 2D feature maps created from lidar scans. With the pressure of the multi-billion dollar gaming industry, the capability of commodity Graphics Processing Units (GPUs) rapidly expanded until they rivalled the parallel processing power previously available only with supercomputers. For example, the NVIDIA TITAN X contains 3,584 Compute Unified Device Architecture (CUDA) cores, 12 GiB of memory, and draws only 250 W of power. Each CUDA core processes up to 32 threads. Therefore, at its peak performance, the TITAN X can handle 114,688 threads. This computing power opens up entirely new realms of feasible real-time problems for robotics applications—most importantly for this thesis: dense 3D reconstructions.

Why is such computing power required to create dense 3D maps? *Voxel grids*—the generally accepted data structure used in dense 3D reconstructions—subdivide an environment into a regular grid of *voxels* (cubes), each of which stores the distance to the nearest surface. Voxel grids provide a guaranteed level of reconstruction quality while requiring only a fixed amount of memory, no matter the number of sensor observations fused. If one desires to reconstruct a workspace with a precision of 4 millimetres, this requires a $512 \times 512 \times 512$ voxel grid with 1 GiB of memory (assuming 2 floating point values in each voxel). In addition, each depth map fused

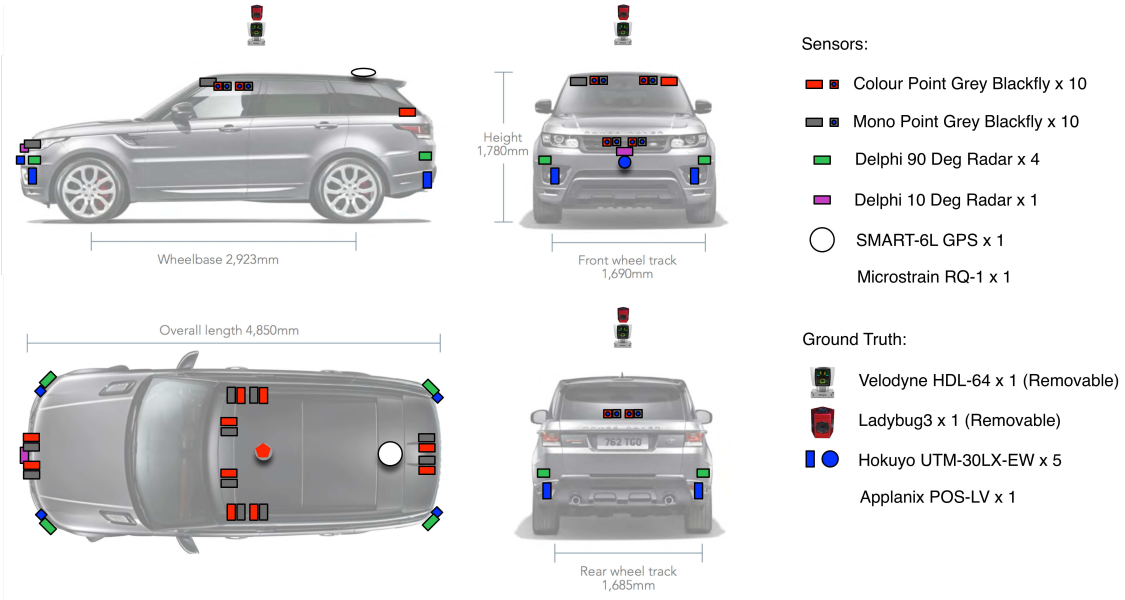
Figure 1.2: Example Dense 3D Reconstructions (Coloured). Our software pipeline creates dense 3D models of robot workspaces, with a focus on autonomous-automobile sensors and environments. The above coloured reconstructions were created by applying our reconstruction software to the KITTI dataset [1].

requires computation in each of the 134,217,728 voxels. This is all presupposed upon a sensor that provides accurate range measurements.

In late 2010, Microsoft released the Kinect camera that generated RGB and per-pixel depth-to-surface measurements (so-called RGB-D) at 30 Hz with 640×480 resolution. Newcombe et al., leveraging the Kinect in concert with the emerging capabilities of General Purpose Graphics Processing Unit (GPGPU), published the KinectFusion system that generated dense 3D models in real time (i.e., at the input sensor's frame rate) [2]. This caused an explosion of research in the dense reconstruction community, with researchers proposing methods to improve the storage efficiency of voxel grids to increase the size of feasible reconstructions.



(a) Nissan LEAF



(b) Jaguar Land Rover

Figure 1.3: Example Dense Reconstruction Data Collection Platforms. This thesis presents methods to create large-scale dense 3D models from autonomous-automobile platforms. Above are two such examples in use by ORI. Note the combination of camera and lidar sensors provide a continuous 360° coverage around each vehicle. See Figure 2.9 on page 22 for a detailed depiction of (a) with images of the available sensors.

These 3D maps provide great utility to a wide variety of robotics domains: scene segmentation, material prediction, localisation, augmented reality, visualisation, and remote inspection (nuclear, chemical, etc.). Unfortunately, because of the high-quality RGB-D sensors, researchers focused on hand-held scenarios where an operator manually manipulated the sensor to reconstruct objects of interest. But RGB-D

sensors cannot be used on many autonomous vehicle applications due to range limitations (approximately 0.5 m to 6 m) and its restriction to indoor environments.

In this thesis, we present the theory required to create a dense reconstruction pipeline that fuses data from commodity (passive) camera or (active) lidar sensors to produce high-quality, large-scale reconstructions (see Figure 1.1 on page 2 and Figure 1.2 on page 3) from an automobile platform. Two example platforms are shown shown in Figure 1.3 on the facing page. This system is substantial: including custom time synchronisation, hardware drivers, and data structures, the software encompasses over 480K lines of source code. To create high-quality models from noisy sensor inputs, we must regularise both the 2D input depth maps *and* the volumetric 3D surface representation stored in the voxel grid. We build upon prior 3D regularisation research [3][4], but by means of a technique we call **Ω**-labelling, we extend the object-centric focus to work during vehicle *exploration*. We rely upon quantitative analysis to rigorously evaluate the quality of our final reconstruction when compared to 3D lidar ground-truth data. We hope the datasets and evaluations presented in this thesis will become valuable tools of comparison for the research community.

1.1 Contributions

This thesis presents six contributions to the state-of-the-art dense reconstruction:

1. We propose a method to regularise 3D data stored in a conventional (Chapter 3) and compressed (Chapter 4) volumetric data structures, thereby enabling regularisation of significantly larger scenes created from robot *exploration*.

2. We present and develop in Chapters 3, 4, and 5 the theory required to construct a state-of-the-art dense-reconstruction system pipeline for large-scale, mobile-robotics applications and quantitatively analyse the quality of the resulting reconstructions.

3. We provide, by way of our quantitative results in Chapter 4 and the models released in the supplementary materials, a new KITTI benchmark for the community to compare dense-map reconstructions.

4. We release (Chapter 4) the 1.6 km Oxford Broad Street dataset as a real-world reconstruction scenario for a mobile-robotics platform with camera and lidar sensors.

5. We formulate in Chapter 5 a new Kernel Conditional Density Estimation (KCDE) data term for the energy functional to interpolate occluded surfaces in dense reconstructions.

6. In Chapter 6, we present a framework that, through the use of deep neural networks, learns to evaluate the quality of our 3D reconstructions.

1.2 Publications

The preponderance of work in this thesis has been published, though one is still in peer review,

- G. Pascoe, W. Maddern, **M. Tanner**, P. Piniés, and P. Newman. "NID-SLAM: Robust Monocular SLAM Using Normalised Information Distance". In: Computer Vision (ICCV), IEEE International Conference On. Honolulu, Hawaii, July 2017. [5] (background for material in Chapter 2 and Chapter 4)

- **M. Tanner**, P. Piniés, L. M. Paz, and P. Newman. "BOR2G: Building Optimal Regularised Reconstructions with GPUs (in Cubes)". In: International Conference on Field and Service Robotics (FSR). Toronto, ON, Canada, June 2015. [6] (Chapter 3)

- **M. Tanner**, P. Piniés, L. M. Paz, and P. Newman. "DENSER Cities: A System for Dense Efficient Reconstructions of Cities". In: ArXiv e-prints (Apr. 2016). [7] (Chapter 4)

- **M. Tanner**, P. Piniés, L. M. Paz, and P. Newman. "Keep Geometry in Context: Using Contextual Priors for Very-Large-Scale 3D Dense Reconstructions". In: Robotics: Science and Systems, Workshop on Geometry and Beyond: Representations, Physics, and Scene Understanding for Robotics. June 2016. [8] (Chapter 4)

- **M. Tanner**, P. Piniés, L. M. Paz, and P. Newman. "BOR2G: Building Optimal Regularized Reconstructions with GPUs". In: IEEE Transactions on Robotics In Review (TBD). [9][1] (Chapter 3 and Chapter 4)

- **M. Tanner**, P. Piniés, Lina Maria Paz, and Paul Newman. "What Lies Behind: Recovering Hidden Shape in Dense Mapping". In: Proceedings of the IEEE International Conference on Robotics and Automation (ICRA). Stockholm, Sweden, May 2016. [10][2] (Chapter 5)

1.3 Thesis Structure

In Chapter 2, we present the preliminary research and tools required to understand our contributions to the dense reconstitution community. In Chapter 3, we explain the theory of regularising dense 3D reconstructions—avoiding the inadvertent spurious surface interpolation and extrapolation to which gradient-based regularisers are subject—created from monocular cameras on a mobile-robotics platform. We augment our software pipeline in Chapter 4 by levering a compressed data structure while fusing multiple sensor modalities to create large-scale reconstructions. With an eye on single-pass automobile-based surveying applications, we present a method in Chapter 5 to remove dynamic and ephemeral objects (e.g., pedestrians, bicyclists, automobiles) from sensor data and target a regulariser to interpolate the occluded urban structure. Finally, in Chapter 6, we propose a machine learning pipeline that learns, through extensive training on laser and image-depth-map reconstructions, to find areas in our 3D reconstructions that contain errors. Overall, this thesis

[1] Under revision for second round of peer reviews
[2] 1 of 3 finalists for **Best Student Paper Award**

proposes an end-to-end dense reconstruction framework suitable for modern mobile-robotics applications.

Everything is related to everything else, but near things are more related than distant things.

— Tobler's First Law of Geography

2 Preliminaries and Tools

Contents

THIS thesis presents the theory and systems required to create and critique large-scale, dense, 3D reconstructions from an autonomous vehicle platform. In this chapter, we introduce the research and tools currently available to the robotics community to create and evaluate dense reconstructions.

We begin in Section 2.1 on the next page with reference frames and coordinate transformations, providing the mathematical framework to denote the position of

sensors and objects in the operating environment. We introduce camera and lidar models in Section 2.2 on page 15, emphasising the importance of accurate intrinsic and extrinsic calibration. In Section 2.3 on page 24, we provide an overview of algorithms to infer 3D structure from passive monocular and stereo cameras—the resulting *depth maps* are used as inputs to our dense reconstruction pipelines in Chapter 3 and Chapter 4. Section 2.4 on page 30 provides a overview of the voxel grid data structure and the algorithm to convert a series of depth maps into a continuous 3D model. Finally, in Section 2.5 on page 38, we present the CloudCompare software tool we will use to quantitatively evaluate the metric error of our dense reconstructions.

2.1 Reference Frames

Reference frames are fundamental to robotics and our dense-reconstruction applications. Mathematically described by the special Euclidean group $\mathbb{SE}(n)$, reference frames enable a system designer to describe the position of any sensor or platform at any point of time. This section introduces the mathematical notation (Section 2.1.1) and convention (Section 2.1.2 on page 12) used throughout this thesis.

2.1.1 Special Euclidean Group

The $\mathbb{SE}(n)$ special Euclidean group represents the set of all possible rigid-body rotations and translations available in n-dimensional space (typically $n = \{2, 3\}$) [11][12]. Many of the examples in this chapter use $\mathbb{SE}(2)$ transformations for clarity, however the same concepts extend to the $\mathbb{SE}(3)$ transformations that are used throughout this thesis.

$\mathbb{SE}(n)$ consists of an $n \times n$ rotation matrix, R, and an n-dimension translation vector, $\mathbf{t}$. The n dimensional vector, $\mathbf{p}$, may be transformed via,

$$\hat{\mathbf{p}} = R\mathbf{p} + \mathbf{t} \tag{2.1}$$

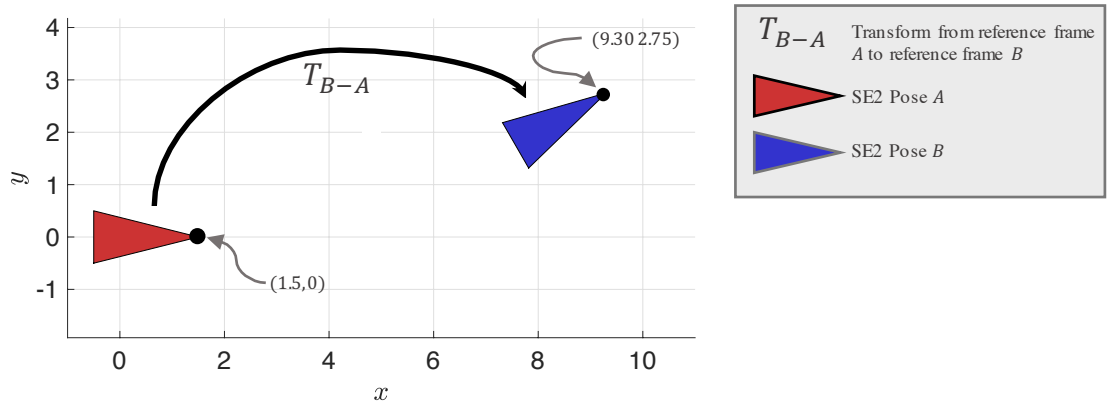


Figure 2.1: $\mathbb{SE}(2)$ Rotation and Translation Example. The $\mathbb{SE}(2)$ special Euclidean group represents any rigid-body rotation or translation for 2D points. In the above example, the Object A (red triangle) is rotated 30° and then translated $(8, 2)^T$ units to form Object B (blue triangle).

where the point is first rotated about reference frames' the origin and *then* translated. For simplicity, R and $\mathbf{t}$ are commonly combined into a single matrix, T,

$$T = \left[\begin{array}{c|c} R & t \\ \hline \mathbf{0} & 1 \end{array}\right] \tag{2.2}$$

where T may be applied to a n-dimensional homogeneous vector,

$$\hat{\mathbf{p}} = T \begin{bmatrix} \mathbf{p} \\ 1 \end{bmatrix} \tag{2.3}$$

R is orthonormal, so the inverse transformation may be efficiently computed [11],

$$T^{-1} = \left[\begin{array}{c|c} R^T & -R^T t \\ \hline \mathbf{0} & 1 \end{array}\right] \tag{2.4}$$

A specific example clarifies the notation. In $\mathbb{SE}(2)$ the general transform matrix T takes the form,

$$T = \begin{bmatrix} \cos\theta & -\sin\theta & t_x \\ \sin\theta & \cos\theta & t_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.5}$$

where θ is the rotation angle about the origin and $\mathbf{t} = (\mathbf{t_x}, \mathbf{t_y})^\mathbf{T}$ is the translation vector. Due to the rigid-body constraint (i.e., orthonormal R), the transformed triangle is both the same shape and size as the original. In other words, $\mathbb{SE}(n)$ transformations can neither warp nor scale objects—see Figure 2.1 for an example.

2.1.2 Convention

In robotics and dense reconstruction contexts, $\mathbb{SE}(3)$ transformations enable one to mathematically describe the position of a platform or sensor relative to one another at any point in time. This section describes the notation and mathematical conventions used to describe these transformations.

All points in space must be described relative to a given reference frame. For example, the `robot` and `camera` reference frames are commonly used throughout this thesis. If the camera observes a point, relative to itself, of $\mathbf{p}_{\text{camera}} = (\mathbf{x_c}, \mathbf{y_c}, \mathbf{z_c}, \mathbf{1})^{\mathbf{T}}$, this point could be converted to the `robot` reference frame via,

$$\mathbf{p}_{\text{robot}} = \mathbf{T}_{\text{robot—camera}}\, \mathbf{p}_{\text{camera}} \tag{2.6}$$

and vice versa,

$$p_{\text{camera}} = T^{-1}_{\text{robot—camera}}\, \mathbf{p}_{\text{robot}} = \mathbf{T}_{\text{camera—robot}}\, \mathbf{p}_{\text{robot}} \tag{2.7}$$

Therefore each reference frame describes the same point, but in terms relative to their respective origins and orientation, by using $T_{\text{robot—camera}}$ (and its inverse) to "translate" between the frames.

As shown in the above example, transforms are written in the standard form,

$$T_{\text{DestinationFrame—SourceFrame}} \tag{2.8}$$

Using the above notation, in a fully-connected graph of n reference frames, $n(n-1)/2$ transforms and their inverse must be computed to handle the transformation between any two reference frames [13]. This quickly becomes intractable in large-scale mobile-robotics applications. For example, we may define a unique reference frame for each sensor, robot, and stereo pair of images. A data collection lasting just one hour would require over 9.7 million transforms.

We mitigate this problem—thus requiring just n transforms—by introducing the transform composition operator $\oplus$,

$$T_{\text{a—d}} = T_{\text{a—b}} \oplus T_{\text{b—c}} \oplus T_{\text{c—d}} \tag{2.9}$$

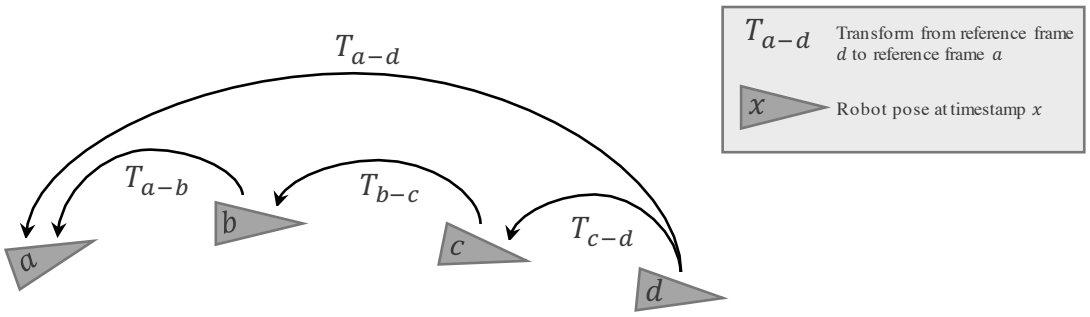


Figure 2.2: $\mathbb{SE}(2)$ Pose Composition Example. Poses may be combined together to create a composite pose, as shown in the above example. Mathematically, this is denoted as $T_{\text{a—d}} = T_{\text{a—b}} \oplus T_{\text{b—c}} \oplus T_{\text{c—d}}$. This is particularly useful when a relative pose chain is provided by one sensor (e.g., VO operating on a stream of stereo images) and one seeks to compute the current location of a mobile-robotics platform relative to its original position.

where this composition is graphically depicted in Figure 2.2. The inverse of this transform is,

$$T_{\text{d—a}} = (T_{\text{a—b}} \oplus T_{\text{b—c}} \oplus T_{\text{c—d}})^{-1} = (T_{\text{a—d}})^{-1} \tag{2.10}$$

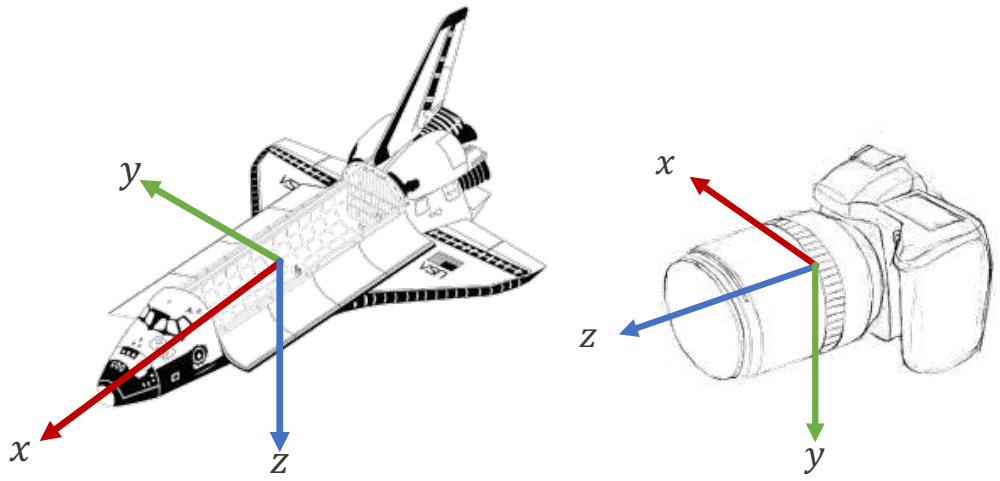
When $\mathbb{SE}(3)$ transformations are in their homogeneous 4×4 matrix form (in contrast to Euler angles or quaternions [14]), composition ($\oplus$) is equivalent to matrix multiplication.

In describing the reference frame for a given object, one must define "up/down", "left/right", and "forward/backward" (i.e., direction of x, y, and z). We use two conventions in this thesis, as shown in Figure 2.3 on the following page. The NASA Body Axis convention is the standard for all ORI sensors and platforms where x is forward, y is to the right, and z is down [15]. However, when projecting points in or out of a camera (see Section 2.2.1 on page 15) we use the Computer Vision convention of z forward, x to the right, and y down. We convert between the frames via,

$$T_{\text{CV—NASA}} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.11}$$

In practice, a typical pose-chain in our dense reconstructions might look something like this,

$$T_{\text{CvLocal—NasaWorld}} = T_{\text{CV—NASA}} \oplus T_{\text{CameraNasa—RobotNasa}} \oplus T_{\text{RobotNasaTk—NasaWorld}} \tag{2.12}$$



(a) NASA Body Axis (b) Computer Vision

Figure 2.3: NASA Body Axis vs. Computer Vision Reference Frame. The above figures outline the two primary reference frames used throughout this thesis. The NASA Body Axis frame is used as the standard reference frame for all sensors and platforms [15]. However, as the literature for vision applications utilise the "Computer Vision" reference frame, we transform coordinates to this frame of reference whenever performing image-specific tasks. For example, to compute a global-frame point p_g (in NASA Body Axis) to a local camera reference frame, one might compute: $T_{\text{CvLocal—NasaWorld}} = T_{\text{CV—NASA}} \oplus T_{\text{CameraNasa—RobotNasa}} \oplus T_{\text{RobotNasaTk—NasaWorld}}$ where $T_{\text{RobotNasaTk—NasaWorld}}$ is the transform from the robot position at time t_k to the global-frame origin. Images from [16][17].

where $T_{\text{RobotNasaTk—NasaWorld}}$ converts from the robot position at time t_k to the global-frame origin. To compute the global-frame point $\mathbf{p_g}$ (in NASA Body Axis) to a local camera reference frame via,

$$\mathbf{p}_c = T_{\text{CvLocal—NasaWorld}}\, \mathbf{p}_g \tag{2.13}$$

We store our transforms in the following data structure,

Listing 2.1: Transform Data Structure

```
struct Transform {
  String sourceFrame;
  String destinationFrame;
  DateTime sourceTimestamp;
  DateTime destinationTimestamp;
  Matrix4 homogeneousTransform; // 4 x 4 matrix
};
```

with methods that allow one to compose multiple transforms into a composite transform if the source and destination timestamps match. We store timestamps

Figure 2.4: Assortment of Sensors. Above are six sensors used in this thesis for dense reconstructions. Clockwise from top-left: (1) Point Grey Bumblebee XB3 colour stereo camera, (2) Point Grey Bumblebee2 colour stereo camera, (3) Point Grey Grasshopper2 colour monocular camera, (4) Asus XtionPro RGB-D camera, (5) Velodyne HDL-32E 3D lidar, and (6) SICK LMS-151 2D lidar. The top row are passive sensors, whereas the bottom row's sensors actively project light into the environment to measure distances to surfaces.

in this structure since some reference frames are only valid at one point in time (e.g., the VO relative pose between two sequential stereo images [18]).

2.2 Sensor Overview

Without sensors, robots are blind—operating in an open-loop capacity. To remedy this, and produce the large-scale dense reconstructions that are the primary focus of this thesis, we mount a variety of camera and lidar sensors on our mobile-robotics platform, see Figure 2.4. In the following two subsections, we provide an overview of the models and intrinsic calibration used for camera (Section 2.2.1) and lidar (Section 2.2.2 on page 19) sensors. In Section 2.2.3 on page 21, we emphasise the importance of accurate extrinsic calibration.

2.2.1 Cameras

The pinhole camera model, depicted in Figure 2.5 on the following page, is a simple geometric description of how a camera works [20][22]. It provides a means to project 3D points into the 2D image plane, and vice versa.

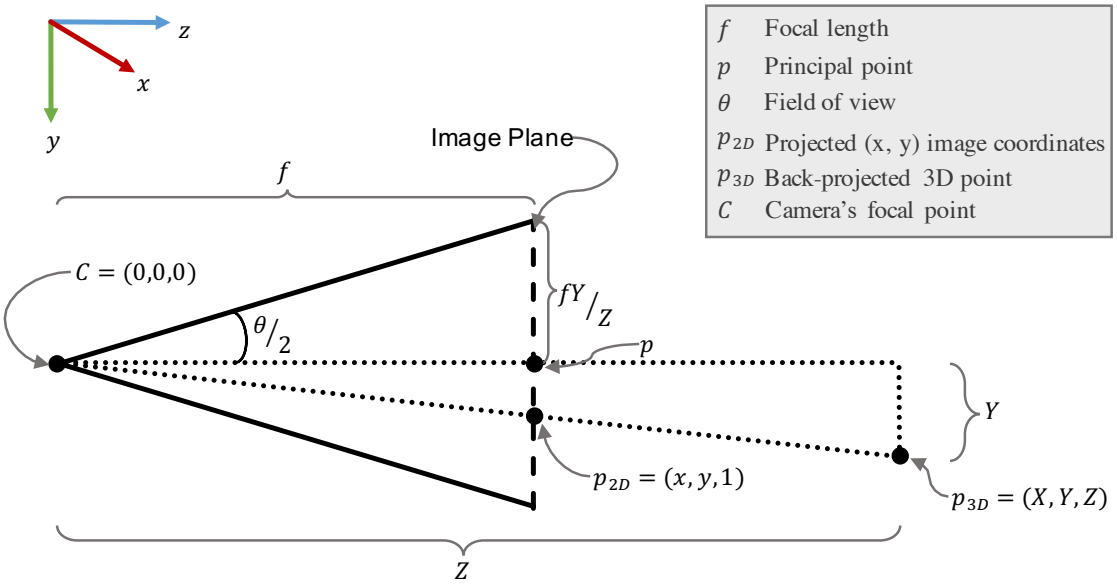


Figure 2.5: Pinhole Camera Model. The Pinhole Camera Model, summarised by the camera model matrix K, is a simple method to map 3D points in space to a 2D coordinate on the image plane. The camera lens focus all light rays to the camera centre, c, while the image's individual pixels (image plane) may be thought of as f metres (the focal length) in front of c. The principal point, p, is located at the centre of the image plane. Figure inspired by [19][20][21].

The camera focal point, c, is the point at which the camera lens focuses all light. The image plane, located f metres in front of c, is the virtual surface upon which the image rests. If the pixels are rectangular, then we must track two separate focal lengths, f_x and f_y. A light source emits a photon that reflects off a surface and then travels to the camera centre. When the photon passes through the image plane, the pixel it passes through becomes (in this model) the "colour" of that photon.

With this model, the 3D point $\mathbf{p_{3D}} = \mathbf{(X, Y, Z)}$ is mapped to the image plane via [20]

$$(X, Y, Z)_{3D} \rightarrow \left(\frac{f_x X}{Z}, \frac{f_y Y}{Z}\right)_{2D} \tag{2.14}$$

or in its matrix form [20],

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \rightarrow \begin{bmatrix} f_x X \\ f_y Y \\ Z \end{bmatrix} = \begin{bmatrix} f_x & 0 & 0 \\ 0 & f_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{2.15}$$

However, the above equation assumes that the $(0, 0)^T$ coordinate is at the centre of the image. To account for the reality that there is an offset from the image

origin (usually located in the top-left of the image) we introduce the principal point, $\mathbf{p}$, offset,

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \rightarrow \begin{bmatrix} f_x X + Z p_x \\ f_y Y + Z p_y \\ Z \end{bmatrix} = \begin{bmatrix} f_x & 0 & p_x \\ 0 & f_y & p_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{2.16}$$

The key parameters of the pinhole camera model can therefore be represented by the following *camera calibration matrix*,

$$K = \begin{bmatrix} f_x & 0 & p_x \\ 0 & f_y & p_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.17}$$

This "world to pixels" matrix provides the means to convert world-space 3D coordinates to the equivalent image-space 3D pixel coordinates with the computation,

$$\mathbf{p_{2d}} = \pi\left(\mathbf{p_{3d}}\right) = \mathbf{K p_{3d}} \tag{2.18}$$

or back-project pixels (assuming the pixel has a known depth value, d) via,

$$\mathbf{p_{3d}} = \pi^{-1}\left(\mathbf{p_{2d}}, \mathbf{d}\right) = \mathbf{d K^{-1}} \begin{bmatrix} \mathbf{p_{2d}} \\ 1 \end{bmatrix} \tag{2.19}$$

See Figure 2.11 on page 25 for an example of an RGB-D (i.e., an RGB image where each pixel has a depth value) back-projected into a 3D point cloud using Equation 2.19.

The above camera model is only valid when there is no lens distortion in the image. However, as all cameras have lens distortion to one degree or another, we must correct the distortion before applying Equation 2.18 or Equation 2.19. An example of this process is shown in Figure 2.6 on the next page. The input image is significantly distorted as it was captured by a Point Grey Grashopper2 camera with a fish-eye lens. Distortion can be found in images by verifying if straight lines are represented correctly, especially on the border regions of the image where the distortion is usually the greatest. In Figure 2.6 on the following page, the parking lines on the road clearly curved in the raw image.

A common method to remove this effect is to use a radial distortion model $L(\cdot)$ to account for the warping from the camera lens [20],

$$\begin{bmatrix} u_d \\ v_d \end{bmatrix} = L(r) \begin{bmatrix} u \\ v \end{bmatrix} \tag{2.20}$$

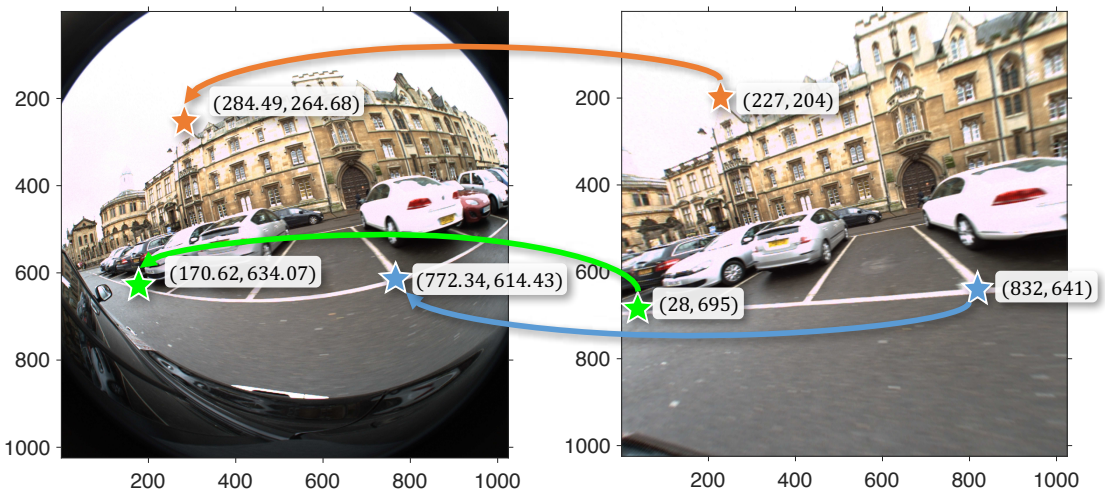


Figure 2.6: Distorted vs. Undistorted Images. All camera lenses cause distortion. In the above example, note how straight lines do not appear straight in (left) (e.g., white parking road markings), but they are corrected in (right). By building a custom LUT for the camera lens, this effect is removed. This is accomplished by mapping (via $\ell(\cdot)$) each pixel of the undistorted image (u, v) to interpolated (u_v, v_d) values in the distorted image—three such mappings are shown in the above images. The camera's "intrinsic" calibration, of which the LUT is part, needs only be computed once per camera as long as the lens is not replaced.

where $(u_d, v_d)^T \in \mathbb{R}^2$ (interpolated pixel coordinates in the distorted image), $(u, v)^T \in \mathbb{Z}^2$ (pixels coordinates in the undistorted image), r is the radial distance $\sqrt{(u - u_c)^2 + (v - v_c)^2}$ a sample point (u, v) is from the centre of distortion (u_c, v_c), and $L(r)$ is the distortion function. To make it computational tractable, $L(\cdot)$ may be approximated via a Taylor series expansion,

$$L(r) = 1 + k_1 r + k_2 r^2 + k_3 r^3 + \ldots + k_n r^n \tag{2.21}$$

An image can be undistorted in real time by either limiting the number of Taylor series terms (e.g., $n = 4$) or use a high-order approximation of $L(\cdot)$ to pre-compute a LUT $\ell(\cdot)$. The distortion is then removed by applying $\ell(\cdot)$ to map each pixel (u, v) in the undistorted image I_u to an interpolated location (u_d, v_d) in the distorted image I_d,

$$\begin{aligned} \ell &: (u, v) \rightarrow (u_d, v_d) \\ I_u(u, v) &= I_d(\ell(u, v)) = I_d(u_d, v_d) \end{aligned} \tag{2.22}$$

Figure 2.6 provides an example of the mapping of three pixels between the distorted and undistorted images.

The combination of K and the undistortion LUT (or $\{k_1, k_2, \ldots\}$ for $L(\cdot)$) is the camera's *intrinsic calibration*. This may be computed, via standard software [23], by the system developer upon purchase of the camera. Occasionally vendors who specialise in cameras for computer vision or robotics platforms will compute the intrinsic calibration as part of the manufacturing process.

As a summary of the key variables required to undistort and project points from the image plane to 3D space, here is an example camera model data structure,

Listing 2.2: Camera Model

```
struct CameraModel {
  double focalLengthX;
  double focalLengthY;
  double principalPointX;
  double principalPointY;

  // LUT contains IMAGE_WIDTH * IMAGE_HEIGHT elements
  Vector2[] undistortionLookupTable;
};
```

2.2.2 Lidar

Lidar is a time-of-flight sensor that actively projects light into the environment and calculates the distance (d) to the nearest surface via,

$$d = \frac{ct}{2} \tag{2.23}$$

where c is speed of light and t is the time from projecting the light pulse to when the return was detected.

The SICK LMS-151 is a 2D lidar that sweeps a single laser beam at 0.5° increments along a plane with a viewing angle of 270° at 50 Hz (i.e., 540 depth observations per scan). When mounted in a push-broom orientation (see the rear LMS-151 in Figure 2.9 on page 22) it produces detailed *sparse* reconstructions of urban environments, as shown in Figure 2.7 on the next page.

The system to create sparse reconstructions of this quality is the topic of another thesis [25], the basic concepts are used throughout this thesis to generate ground-truth data to evaluate image-only or interpolated dense 3D reconstructions

Figure 2.7: Coloured LIDAR Point Cloud Example. Lidar is one of the few sensors on mobile-robotics platforms that provides robust, accurate, and far-ranged depth measurements. The above image plots the lidar point cloud captured by a push-broom LMS-151 (see Figure 2.9 on page 22) in an urban environment. Note how the push broom orientation of the lidar sensor enables it to produce an accurate sparse reconstruction of the 3D structure. Laser points are coloured via cameras that are oriented to provide continuous 360° coverage around the vehicle. Creating sparse reconstructions like this require accurate intrinsic, extrinsic, and temporal calibration [24].

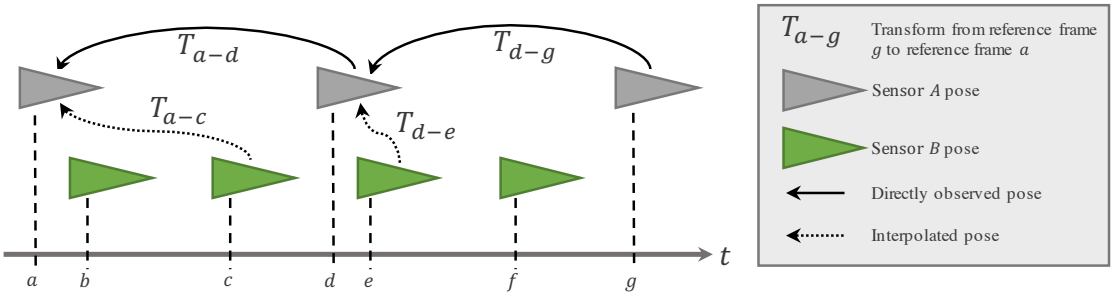


Figure 2.8: Pose Interpolation Example. When a platform contains multiple, unsynchronised sensors, the pose for each sensor must be interpolated. This commonly occurs in this thesis' experiments when capturing data with cameras and lidar sensors. The camera (represented by "Sensor A") in the above figure provides the $\mathbb{SE}(3)$ pose chain. Since the lidar sensor does not directly observe or compute the pose, it must interpolate the pose between two subsequent camera poses. The extrinsic calibration must be precise for this system to work properly—see Figure 2.9 on page 22 and Figure 2.10 on page 23 for more details. In addition, the *temporal* calibration—accounting for clock differences between sensors—must also be corrected [24].

(Chapters 3, 4, and 5) or to fuse into a voxel grid to improve the dense reconstruction quality (Chapter 4). These sparse reconstructions are created by,

1. Compute the vehicle's trajectory via VO [26].

2. Interpolate the $\mathbb{SE}(3)$ pose of each laser point from the LMS-151 (see Figure 2.8 on the facing page).

3. Project all laser points into the most relevant camera sensor (from a suite of six cameras with continual 360° coverage around the vehicle) to colour the point cloud [25].

Unlike cameras, lidar sensors require neither intrinsic calibration nor a sophisticated model. Rather, a simple ray-based model (origin at lidar sensor and termination at reflectance point as described in Section 4.2.1.1 on page 83) is sufficient to provide the results in Figure 2.7 on the facing page. However, pose interpolation and colouring the point cloud requires accurate intrinsic, extrinsic, and temporal (cross-sensor clock biases) calibration [24].

2.2.3 Extrinsic Calibration

Any system with more than one sensor must accurately characterise their *extrinsic calibration*, the relative $\mathbb{SE}(3)$ pose between sensors. The ORI RobotCar platform is used to capture data for a variety of experiments throughout this thesis [27]. As shown in Figure 2.9 on the next page, it contains a variety of sensors, representative of the suite used on modern autonomous automobiles. These sensors include: 1x GPS, 1x INS, 1x 3D lidar, 2x 2D lidar, 3x monocular cameras, and 1x stereo camera.

As this type of setup is typical (and conservative compared to our more recent platforms), we seek to simplify the extrinsic calibration process by defining a single reference frame against which all sensors are compared: `robot`. This is usually located at the centre of the vehicle's rear axle.

The sensors may first be coarsely calibrated via physical measurements before more sophisticated methods are used to refine the calibration [28][29]. For example, if two cameras have any overlap, then multiple objects are placed in the overlapping field of view and their known locations are used to solve for the unknown relative transformation between the sensors. Lidar sensors are calibrated by seeking an

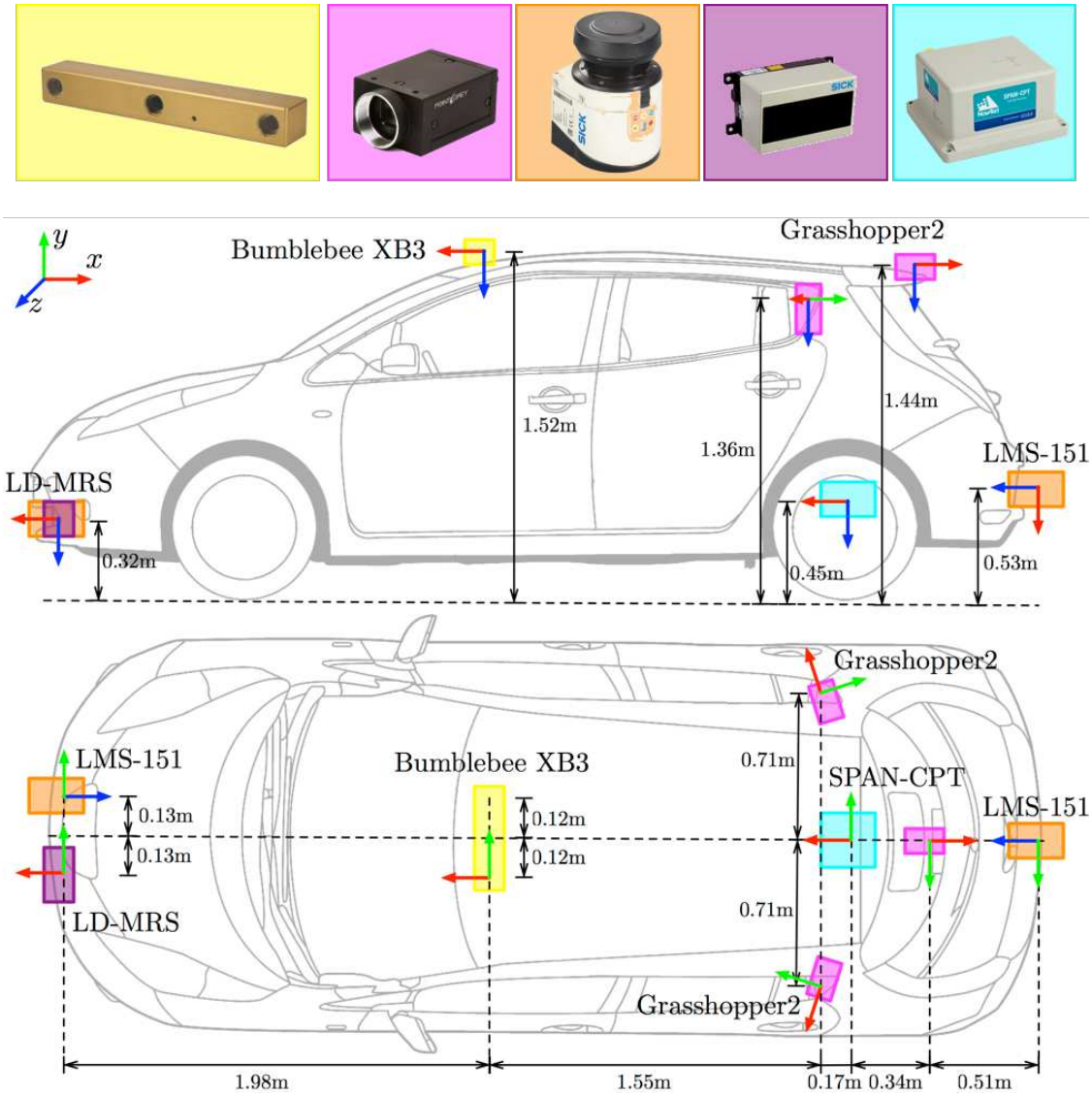


Figure 2.9: University of Oxford RobotCar Sensor Layout. One of the primary data-collection platforms for this thesis is the University of Oxford's RobotCar [27]. This platform's range of sensors are representative of a typical modern autonomous vehicle. The 360° coverage with multiple sensor modalities is ideal for high-quality dense reconstructions. A forward-facing stereo camera provides poses (via SLAM or VO) and depth maps while the laser sensor is directly fused into the 3D dense reconstruction or used as ground-truth data for metric evaluations. It is critical that the extrinsic calibration (i.e., the relative $\mathbb{SE}(3)$ pose) between the sensors be computed accurately. See Figure 2.10 on the facing page for an example of the effects of poor extrinsic calibration. Figure from [27].

overlap between the reflectance values in the 3D point cloud (similar to Figure 2.7 on page 20) and the camera images.

Accurate calibration is vital in many robotics applications, but particularly when fusing data from multiple sensors (and sensor modalities) for dense reconstructions,

(a) Correct Calibration

(b) 5° RPY Calibration Error

Figure 2.10: Camera-lidar Calibration. The relative $\mathbb{SE}(3)$ pose of the sensors on a mobile-robotics platform must be very accurate. In (a) the extrinsic calibration between the camera and lidar sensor is correct, resulting in the laser points accurately overlaying the 3D structure when projected in the image. However, in (b), when the calibration angles are perturbed by 5°, the 3D structure does not align correctly. In dense reconstruction applications, this results in each sensor fusing conflicting data into the voxel grid, producing poor quality reconstructions. See Figure 2.9 on the facing page for the layout of sensors on the ORI RobotCar platform used to collect the above data.

as is required in Chapter 4. Figure 2.10 shows the effects of poor calibration. In the well-calibrated image, laser points projected into the image overlap properly with the scene's underlying 3D urban structure. However, in the poorly-calibrated

image, the projected laser points do not align correctly. If one attempted to fuse the lidar and image data into a voxel grid to create a dense reconstruction, the conflicting range information would create destructive interference, resulting in inaccurate surface models.

With proper extrinsic calibration and pose interpolation (see Figure 2.8 on page 20), the precise $\mathbb{SE}(3)$ pose of each sensor is always known. However, this calibration should be regularly updated to account for changes due to temperature variance or physical perturbations.

2.3 Depth Maps

Depth maps are one of the primary input used throughout this thesis to create dense 3D reconstructions. A depth map is an image where the "depth" of each pixel—i.e., the z component of the distance from the camera's optical centre through the pixel to the first surface observed—is known.

An example of the utility of depth maps is depicted in Figure 2.11. Given an RGB image and an associated depth map, the pixels may be back-projected into space (Equation 2.19 on page 17) to create a *sparse* reconstruction of the environment. If a pose source is available (e.g., VO), sequential point clouds may be accumulated to create moderate-scale sparse reconstructions. Unfortunately, large-scale reconstructions are not feasible with this approach as the memory requirements grow linearly with subsequent depth maps. We address this issue and explain how to create *dense* 3D reconstructions in Section 2.4 on page 30.

The most straight-forward method to create depth maps is with an *active* sensor (e.g., Microsoft Kinect or Asus Xtion Pro). These "RGB-D" (RGB and Depth) sensors actively project infrared light patterns into the environment and compute the depth based on perturbations to the light patterns [30]. This results in a high quality (640×480) depth map, accurate within millimetres for near objects, at a rate of 30 Hz. However, the range limitations (approximately 5 m) and restricted daytime outdoor use limit the utility of RGB-D sensors in autonomous-driving applications [31].

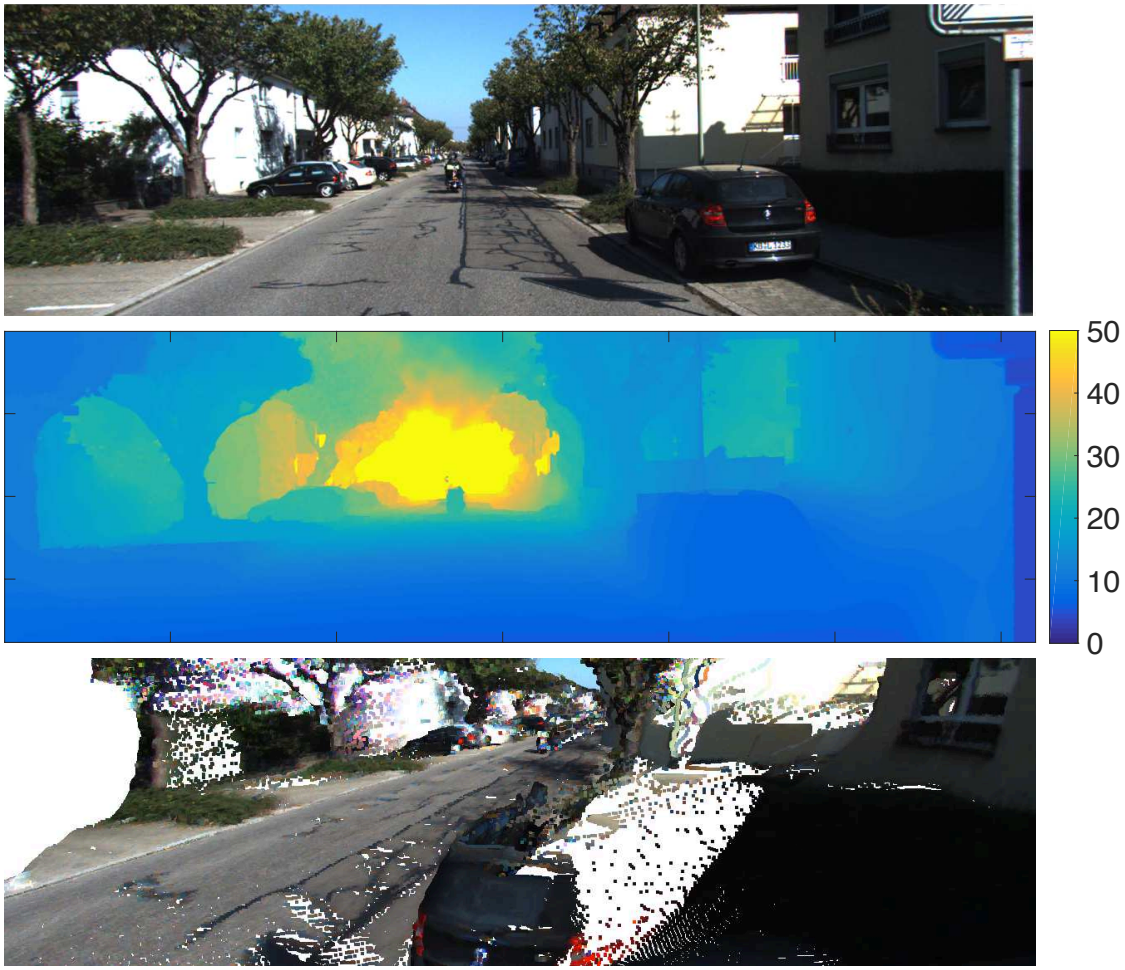


Figure 2.11: RGB-D Point Cloud Example. The RGB image (top) and its associated depth map image (middle) can be projected out into 3D space by using the camera's calibration matrix K. Each pixel in the depth map represents the depth (in metres) of the surface observed by the camera. In this case, the depth map is metrically accurate since the known stereo-baseline avoids the scale-invariance of monocular camera depth maps. The RGB image is from the KITTI dataset [1], while the depth map and projection are computed by ORI software.

There are a wide variety of techniques are presented in the literature to generate depth maps from *passive* monocular or stereo cameras. Traditional methods for stereo cameras rely upon *stereo matching* to find corresponding pixels in a set of images [32][33][34]. Once these correspondences are identified, depth of these pixels is directly computed via geometric reasoning—the depths for the remaining pixels, where there were no (or incorrect) correspondences, may be inferred by regularising with a *prior* [35].

Monocular approaches traditionally used similar geometric approaches, but more

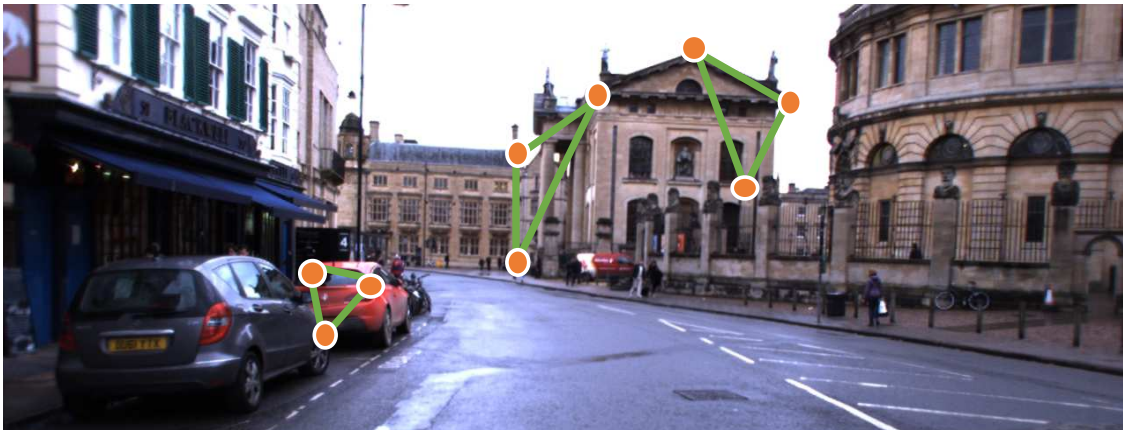


Figure 2.12: Feature Matching and Depth Interpolation. LIBELAS relies upon support points to compute the initial seed estimates for each pixel's depth. This is achieved by identifying a set of point features (orange circles in the above figure) in the image, computing their depth via data association in the stereo image pair, and then interpolating dense depth estimates throughout the image by interpolating within the Delaunay triangulation (green lines in the above figure). [42]

than two images are usually required to produce high-quality depth maps because subsequent images are not rectified (i.e., each row in one image aligns with the same row in another image) [36][37][38]. However, recent machine-learning techniques (in particular: convolutional neural networks) demonstrate the ability to produce accurate, dense depth estimates from a single image [39][40][41].

In the subsequent two sections, we present two widely-used methods for depth-map estimation—Library for Efficient Large-scale Stereo Matching (LIBELAS) for stereo (Section 2.3.1) and Dense Tracking and Mapping (DTAM) for monocular (Section 2.3.2 on page 28). As discussed in the preceding paragraphs, understand that these are merely two examples to provide intuition into the process, but other methods may be used based on system computation versus performance trade-offs.

2.3.1 Stereo

Stereo depth map generation is generally more straight-forward than monocular approaches because there is a known baseline (the distance between the left and right sensors) and the image are rectified, where each row of pixels in one image aligns with the same row in the other image.

Geiger et al. present the robust (and open source) LIBELAS approach to generate stereo depth maps [42] that we outline in the following steps:

1. Identify *support points*, S, within the stereo pair (Figure 2.12 on the preceding page). While data association of features in different images is known to be challenging, these support points are selected because they are the most unique and identifiable features in the images (e.g., high texture, high gradient). Only robust features are selected—i.e., features similarly identified in left-to-right and right-to-left image correspondences.

2. Connect all support points with Delaunay triangulation [43]. Delaunay triangles balance the edge lengths so no one edge dominates, which would inappropriately skew the interpolation towards one vertex.

3. Create a generative probabilistic model that serves as a *prior* for disparity estimation,

$$p(\delta_n, \mathbf{o}_n^{(l)}, \mathbf{o}_n^{(r)}, S) \propto p(\delta_n | S, \mathbf{o}_n^{(l)}) p(\mathbf{o}_n^{(r)} | \mathbf{o}_n^{(l)}, \delta_n) \tag{2.24}$$

   This formulation assumes the support points (S) and observations ($\mathbf{o}$) are conditionally independent given the disparity (d_n). δ_n is the disparity of pixel n, and $\mathbf{o}_n$ is the pixel's observation vector ($\cdot^{(l)}$, $\cdot^{(r)}$ denotes left or right image). $p(\delta_n | S, \mathbf{o}_n^{(l)})$ is the image prior built from support points and observations. $p(\mathbf{o}_n^{(r)} | \mathbf{o}_n^{(l)}, \delta_n)$ is the image likelihood (i.e., consistency between observations in the left and right image). These are modelled as Gaussian and Laplace distributions, respectively,

$$\begin{aligned} p(\delta_n | S, \mathbf{o}_n^{(l)}) &\propto \gamma + \exp\left(-\frac{(\delta_n - \mu(S, \mathbf{o}_n^{(l)})^2}{2\sigma^2} \right) \\ p(\mathbf{o}_n^{(r)} | \mathbf{o}_n^{(l)}, \delta_n) &\propto \exp\left(-\beta || \mathbf{f}_n^{(l)} - \mathbf{f}_n^{(r)} ||_1 \right) \end{aligned} \tag{2.25}$$

   where $\mu(\cdot, \cdot)$ is a mean function and $\mathbf{f}_n^{(\cdot)}$ is the pixel's feature vector for the left or right image. Both distributions are constrained to operate in the immediate region around the current pixel and neighbouring support points.

4. Solve the energy-maximisation optimisation problem to converge upon a solution that maximises the likelihood disparity (δ) given the support-point prior and the feature vector for pixel correspondences between the left and right images,

$$\underset{\delta}{\operatorname{argmin}} \int_{\Psi} -\beta||\mathbf{f}_n^{(l)} - \mathbf{f}_n^{(r)}(\delta)||_1 - \log\left(\gamma + \exp\left(-\frac{(\delta_n - \mu(S, \mathbf{o}_n^{(l)})^2}{2\sigma^2}\right)\right) d\Psi \quad (2.26)$$

where u and v are the x and y coordinates for pixel n and $\mathbf{f}_n^{(r)}(\delta)$ is the feature vector at $(u^{(l)} - \delta, v^{(l)})$. Note, we discuss methods to minimise energy equations such as this in Chapter 3.

This, as with many dense stereo depth map algorithms, produces a disparity (δ) per pixel rather than metric depth (d). The disparity is the *pixel* offset between features in the two images (i.e., given feature a in the left image appears in column u, find the feature in the right image at column $u - \delta$). The equation to convert between depth and disparity is, by similar triangles,

$$d = \frac{bf_x}{\delta} \quad (2.27)$$

where f_x is the focal length of the camera (Section 2.2.1 on page 15) and δ is the pixel disparity.

2.3.2 Monocular

Estimating a dense depth map from a monocular camera proves challenging because the images are not rectified and there is no known baseline. Without image rectification, rows in subsequent images do not align, thus increasing the difficulty of feature matching. Without a known baseline or world measurement, monocular depth maps are scale invariant and (over time) suffer from scale drift. The latter limitation may be removed or reduced by using a metric pose source—e.g., lidar [45][46][47][47], VO [48], or Inertial Measurement Unit (IMU) VO [49][18][50][51][52]—and interpolate the $\mathbb{SE}(3)$ pose of each monocular image. We assume a metric pose source is available to simplify the discussion in this section. Without it, one would need to bootstrap the system with an initial scale estimate [53].

The following outlines the DTAM algorithm proposed by Newcombe et al. [44],

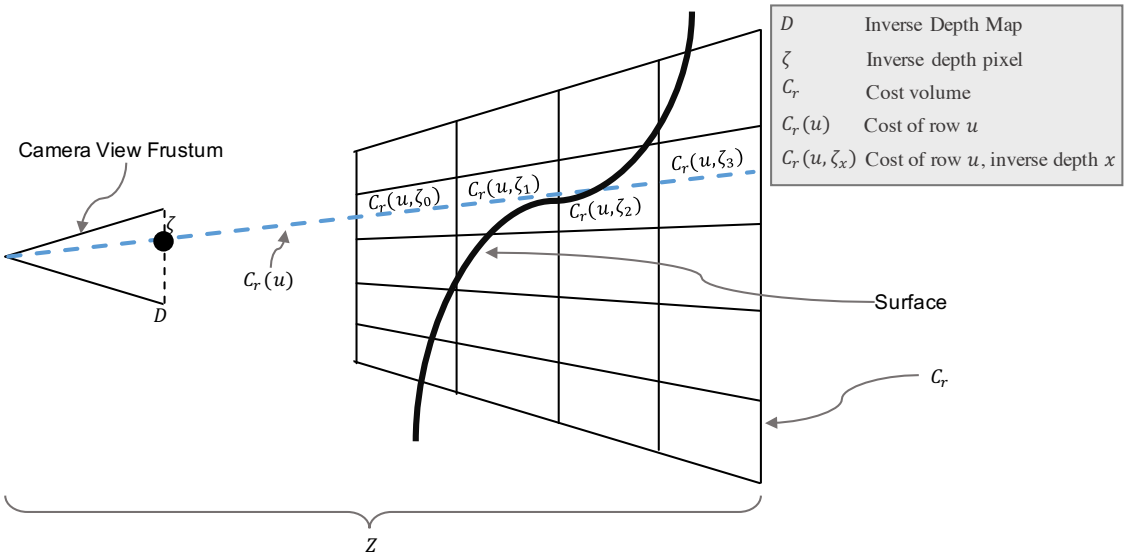


Figure 2.13: DTAM Cost Volume Overview. DTAM creates one inverse depth map (and an associated cost volume) that is continually updated with sequential monocular camera frame. Each subsequent frame is first warped back to the original frame before being fused into the cost volume, C_r. In the above example, the inverse depth of a single pixel ζ contains a row of costs associated with a set of discrete possible inverse depths, each represented by a cell, $C_r(\mathbf{u}, \zeta_x)$, in the cost volume. The inverse depth map is computed via $\underset{d}{\text{argmin}}\, C(\mathbf{u}, \zeta)$. [44]

1. On the first frame, construct a frustum cost volume (C_r) in front of the camera (Figure 2.13). Each cost volume row ($C_r(\mathbf{u})$) contains the photometric error (cost) of a given pixel ($\mathbf{u}$) for a discrete set of inverse depths (ζ) in the cell $C_r(\mathbf{u}, \zeta)$. Due to the structure of the cost volume, this approach limits inverse depth estimates to the range between the first and last cells, $[\zeta_{min}, \zeta_{max}]$, in C_r.

2. For each subsequent image I_m, each cell of C_r is updated with the photometric error (ρ_r) from projecting the first keyframe (I_r) image's pixels into the new image at various depths,

$$\rho_r(I_m, \mathbf{u}, \zeta) = I_r(\mathbf{u}) - I_m(\pi(KT_{mr}\pi^{-1}(\mathbf{u}, \zeta))) \tag{2.28}$$

3. Compute the inverse depth (ζ) for each pixel $\mathbf{u}$,

$$\underset{\zeta}{\text{argmin}} \int_{\Omega} \left(g(\mathbf{u})||\nabla\zeta||_{\epsilon} + \lambda C\left(\mathbf{u}, \zeta(\mathbf{u})\right)\right) d\mathbf{u} \tag{2.29}$$

where $g(\mathbf{u})$ is a function to reduce the stair-step tendencies of the $||\nabla\zeta||_\epsilon$ (Huber norm) term,

$$g(\mathbf{u}) = \exp\left(-\alpha||\nabla I_r(\mathbf{u})||_2^\beta\right) \tag{2.30}$$

The first half of the energy minimisation is a regularisation term that seeks to ensure the smoothness of the output image while allowing for sharp discontinuities. The latter term ($\lambda C(\cdot)$) penalises solutions that vary too far from the cost volume's observed photometric error. In concert, these two terms ensure the final depth map is reasonably smooth (i.e., low gradient) while minimising photometric error.

Unlike the stereo camera approach, monocular cameras require a sequence of tens to hundreds of images to create a single high-quality depth map. These depth maps are of "reasonable" quality, but a sole depth map is rarely of sufficient quality to create a dense 3D reconstruction. In the next section, we outline the voxel grid data structure that allows us to fuse a sequence of depth maps and extract a continuous 3D model.

2.4 Voxel Grids

Our aim in this thesis is to efficiently reconstruct urban outdoor or large-scale environments while continually improving the accuracy of the representation with subsequent observations. Our data fusion system is built to process depth-map estimates (camera sensor) and range observations (lidar sensor) within the same data structure. The representation of the surfaces plays an important role in our system. An *explicit* representation of each of the depth/range observations (e.g., as a point cloud) is a poor choice as storage grows without bound and the surface reconstruction does not improve when *revisiting* a location.

Instead, we prefer an *implicit* representation of the surface [54]. A common approach is to select a subset of space in which one will reconstruct surfaces and divide that subspace into a uniform voxel grid. Each voxel stores depth/range observations represented by their corresponding Truncated Signed Distance Function

(TSDF), u_{TSDF}. The TSDF value of each voxel is computed such that the zero-crossing level set (isosurface) is the continuous 3D surface model. The voxel grid is a discrete field; however, since the TSDF value stores the precise distance to the nearest surface, the surface reconstruction is even more precise than the voxel size.

We will first introduce two dominant data structures used in the dense reconstruction community (Section 2.4.1) before reviewing how range data is fused into the voxel grid (Section 2.4.2 on page 34) and the surface is extracted (Section 2.4.3 on page 36).

2.4.1 Data Structures

There are a plethora of 3D reconstruction data structures in the literature, ranging from fixed conventional voxel grids [2], moving voxel grids [55], hierarchical [56][57][58], hash tables [59], or a combination thereof [60].

For small-scale reconstructions in this thesis (Chapter 3 and Chapter 5) we use the simple conventional voxel grid. When reconstructing large outdoor scenes (Chapter 4) we instead select the hash-table method. The following two subsections discuss these choices in more detail.

2.4.1.1 Conventional

In 2011, Newcombe published a paper that presented the KinectFusion method of creating dense 3D reconstructions in real time (i.e., at the input sensor's frame rate) by fusing data from an RGB-D camera into a conventional voxel grid using a GPU [2]. The conventional voxel grid divides space into a regular grid of voxels, as shown in Figure 2.14 on the next page, in which the sensor data is fused as TSDF values (see Section 2.4.2 on page 34).

Due to GPU memory constraints, only a small subset of space (7 m^3 in the Newcombe's paper) can be reconstructed using this conventional approach where the voxel grid is fixed in space [44][2]. This presents a particular problem in mobile-robotics applications since the exploration region of the robot would be restricted to a prohibitively small region. In addition, long-range depth/range sensors (e.g.,

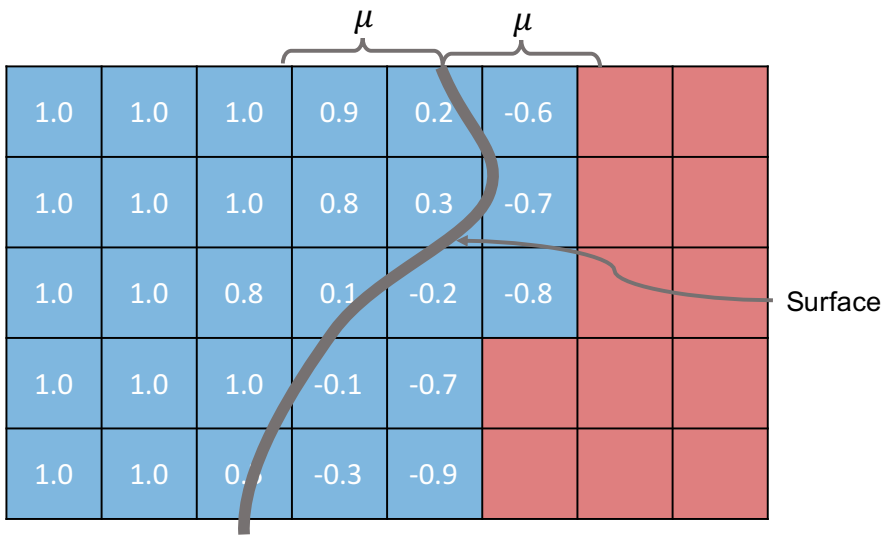


Figure 2.14: Conventional Voxel Grid (2D example). The conventional voxel grid subdivides a fixed area in space into a uniform grid of voxels. The TSDF values ($u_{TSDF} \in (-1, 1]$) in each voxel are proportional to the average distance of that voxel to the nearest surface (see Figure 2.16 on page 34). Positive TSDF values indicate the voxel's centroid is in front of the surface; negative values are behind the surface. μ represents the threshold in front and behind the surface around which a linear u_{TSDF} value is stored—this prevents negative TSDF values from overwriting surfaces behind the observed surface (e.g., in the red region of the above figure). The surface is extracted by solving for the zero-crossing level set (isovalue = 0).

laser, stereo-camera-based depth maps) cannot be fully utilised since their range exceeds the size of the voxel grid (or local voxel grid if a local-mapping approach is used) [55]. These limitations are acceptable for small-scale, local reconstructions, but a more sophisticated data structure is required for large-scale reconstructions with longer-range sensors (Chapter 4, specifically Table 4.1 on page 79).

2.4.1.2 Hashing

In recent years, researchers proposed a variety of techniques to remove spacial limitations [55][56][58][60]. They leverage the fact that the overwhelming majority of voxels do not contain any valid TSDF data since they are never directly observed by the range sensor or they constitute free space ($u_{TSDF} = 1$). A compressed data structure only allocates and stores data in voxels that are near a surface.

One of the more prolific compression approaches is Nießner's HVG [59]. HVG creates an "infinite" (within numerical limitation) virtual grid that subdivides the

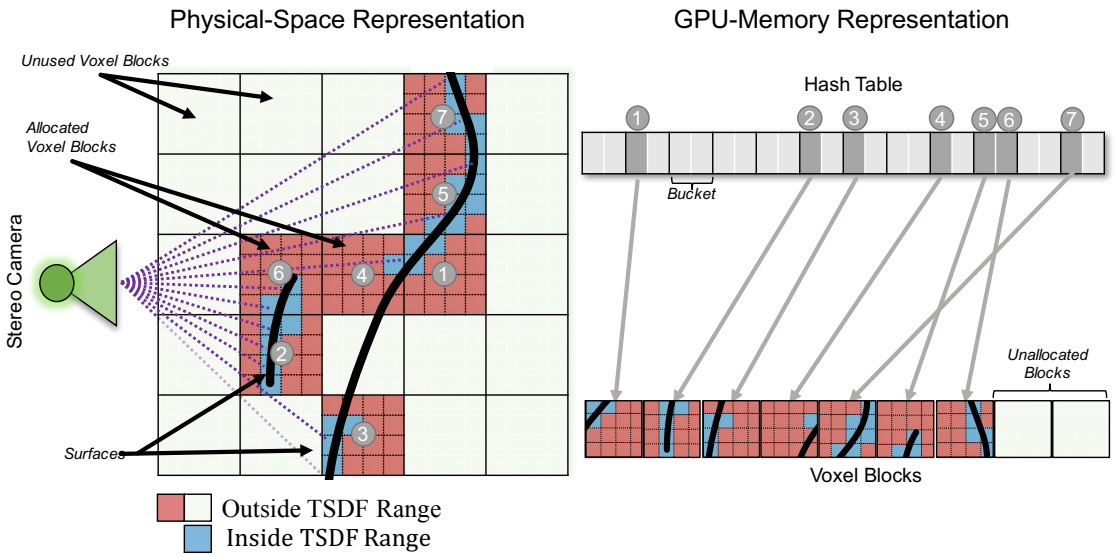


Figure 2.15: HVG Data Structure Overview. At the lowest level, HVG voxel blocks are similar to conventional voxel grids (Figure 2.14 on the preceding page)—albeit much smaller. Conventional voxel grids are typically contain approximately 512^3 voxels, while each HVG voxel block contains only 8^3 voxels. However, HVG can reconstruct much larger environments by only reserving memory in the regions immediately surrounding the reconstructed surface, whereas conventional voxel grids implicitly reserve memory even in known free space. In the above figure, only seven voxel blocks are reserved in memory to reconstruct the surface. On the right is the logical layout of GPU memory. The hash table serves as a LUT (indexed by the hash of the voxel block's world position) to find the appropriate voxel block in global memory. For this example, two voxel blocks remain available for surfaces observed in subsequent sensor scans.

environment with a course and fine level of detail (see Figure 2.15). The course level contains *voxel blocks* (small conventional voxel grids composed of $8 \times 8 \times 8$ voxels) while the fine level contains *voxels*. No memory is used on the GPU until a surface is observed within that voxel block ($\pm\mu$).

As part of initialisation, the HVG algorithm reserves a portion of the GPU memory to store n_{blocks} of voxel blocks. In addition, and as part of the overhead, a *hash table* with n_{hash} elements (where $n_{hash} \gg n_{blocks}$) enables efficient lookup and reservation of voxel blocks during run time. The hash function,

$$i_h = H(x, y, z) = (x \cdot p_1 + y \cdot p_2 + z \cdot p_3) \mod n_h \tag{2.31}$$

deterministically converts to coordinates in world space (x, y, z) to an index i_h

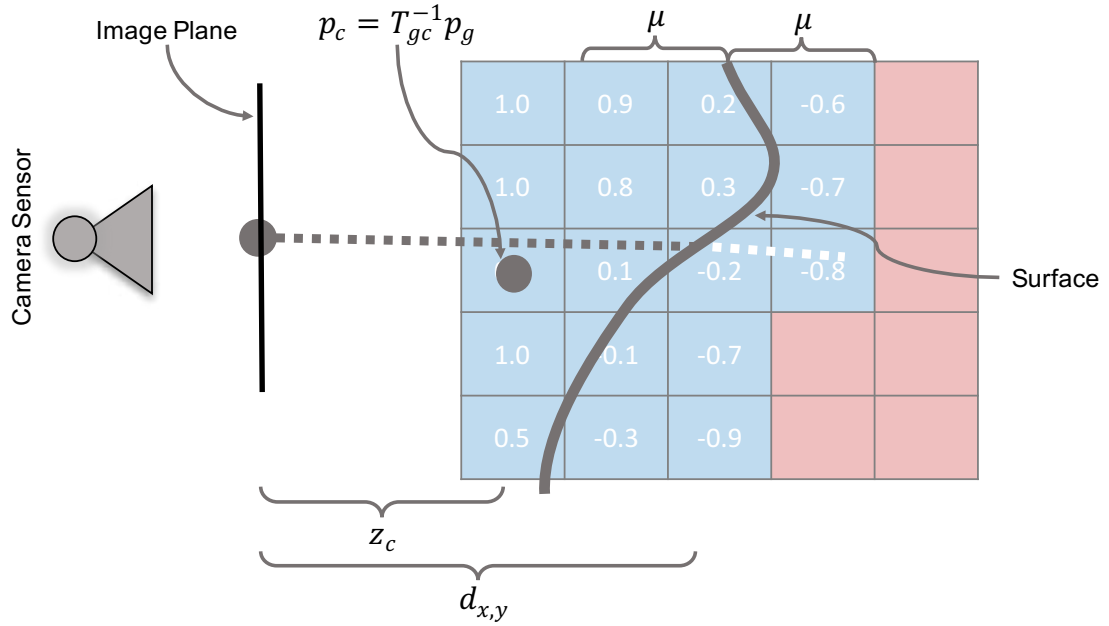


Figure 2.16: Depth Map Voxel Grid Fusion. A depth map, D, is fused into voxel grids by projecting each voxel into the nearest-neighbour depth pixel, $d_{x,y}$. This is achieved by converting the global-frame voxel centroid, p_g, to a camera-frame point via $p_c = T_{gc}^{-1} p_g$ and then applying the camera calibration matrix, K (see Section 2.2.1 on page 15). This algorithm is well suited for GPGPU implementation. A modern GPU can fuse data into a 512^3 voxel grid at 30 Hz.

within a hash table with n_h elements by using three large prime numbers,

$$\begin{aligned} p_1 &= 73856093 \\ p_2 &= 19349669 \\ p_3 &= 83492791 \end{aligned} \tag{2.32}$$

selected to minimise hash collisions. The hash table entry at index i_h contains a pointer to the raw voxel block data. A depiction of the relationship between the reconstruction, hash table, and raw voxel data is provided in Figure 2.15 on the previous page). We refer the reader to the original HVG paper [59] for further implementation details.

2.4.2 Data Fusion

Unlike conventional voxel grids, HVG requires a two-step data-fusion process because voxel blocks in which data is fused may not yet be reserved in the hash table. The

voxel blocks are reserved first by efficiently ray casting (see Section 4.2.1.2 on page 85) the depth map and reserving memory from the range $d \pm \mu$.

Once memory is reserved, the data fusion (Figure 2.16 on the preceding page) is similar between conventional voxel grids and HVG. If one considers each HVG voxel block to be a conventional voxel grid, then the update equations are nearly identical to those presented by [2]. For each voxel, perform the following operations for every new depth map, D:

1. Calculate the voxel's global-frame centre $\mathbf{p}_g = [x_g, y_g, z_g]^T$ with respect to the camera coordinate frame as $\mathbf{p}_c = \mathbf{T}_{gc}^{-1}\mathbf{p}_g$, where $\mathbf{T}_{gc} \in \mathbb{SE}(3)$ is the camera-to-global coordinate frame transformation.

2. Project $\mathbf{p}_c$ into D to determine the nearest pixel $d_{x,y}$.

3. If the pixel (x, y) lies within the depth map, evaluate $u_{SDF} = d_{x,y} - z_c$. $u_{SDF} > 0$ indicates the voxel is between the surface and the camera, $u_{SDF} < 0$ for voxels occluded from the camera by the surface, and $u_{SDF} = 0$ for voxel centroids that exactly intersect the surface.

4. Update the voxel's current f (Signed Distance Function (SDF) value) and w (weight or confidence in f) at time k,

$$
\begin{aligned}
w_k &= \begin{cases} w_{k-1} + w_s & u_{SDF} \geq -\mu \\ w_{k-1} & u_{SDF} < -\mu \end{cases} \\
f_k &= \begin{cases} \frac{u_{SDF} + w_{k-1} f_{k-1}}{w_k} & u_{SDF} \geq -\mu \\ f_{k-1} & u_{SDF} < -\mu \end{cases}
\end{aligned}
\tag{2.33}
$$

   where w_{k-1} and f_{k-1} are the previous values of f and w for that voxel, and w_s is the weight (confidence) assigned to the sensor's depth measurement.

This is well suited for real-time (i.e., at the input sensor's frame rate) processing on a GPU (via CUDA or OpenCL) as, once the memory is properly allocated, no atomic operations are required to fuse a depth map into the voxel grid. Each voxel is allocated an individual GPU thread that computes the updated f and w values.

While the SDF is still truncated at $-\mu$ behind surfaces, we do not clamp distance values in front of the surface—each voxel stores the metric distance to the surface. Therefore, we refer to this as SDF rather than the truncate-and-clamp TSDF presented in Curless [54]. We elect to use this SDF for f as this allows us to simultaneously fuse data from different sensors, each with distinct μ values based on the sensor's precision, into the same voxel grid. Our SDF μ should be proportional to the accuracy of the range sensor. For example, μ might be large for points far away from the stereo camera but be smaller for laser scans or close stereo observations. Note that μ must always be greater than the voxel width or else the isosurface cannot be extracted.

2.4.3 Surface Extraction

The reconstruction's surfaces are implicitly stored in the voxel grid and may be extracted either via ray casting or by directly computing the isosurface function. In the ray casting approach, one positions a virtual camera near the voxel grid and steps along the ray for each pixel until the ray moves from a positive region of the voxel grid to a negative region [61]. There are two major limitations of this approach. First, the ray-casted image must be updated with each movement of the camera. This is very computational inefficient as GPU hardware is optimised for rasterising images, not ray casting. Second, the "surfaces" are only represented in image form: the 3D shape is not explicitly represented.

An alternative is to extract the surface model as a series of triangles with the Marching Cubes algorithm [62][63][64]. This algorithm operates independently on a per-voxel basis rather than per-pixel. A given voxel is compared to the values of seven neighbouring voxels to determine if there is an isosurface that intersects between the vertices. For example, using the notation from Figure 2.17 on the facing page, if v_0 (the voxel of interest) has a positive TSDF value (u_0) but its neighbour's (v_1) TSDF value (u_1) is negative, that indicates the surface intersects the edge e_0. The surface intersection point (p_i) is computed,

$$s_n = p_0 + (i - u_0)\frac{p_1 - p_0}{u_1 - u_0} \tag{2.34}$$

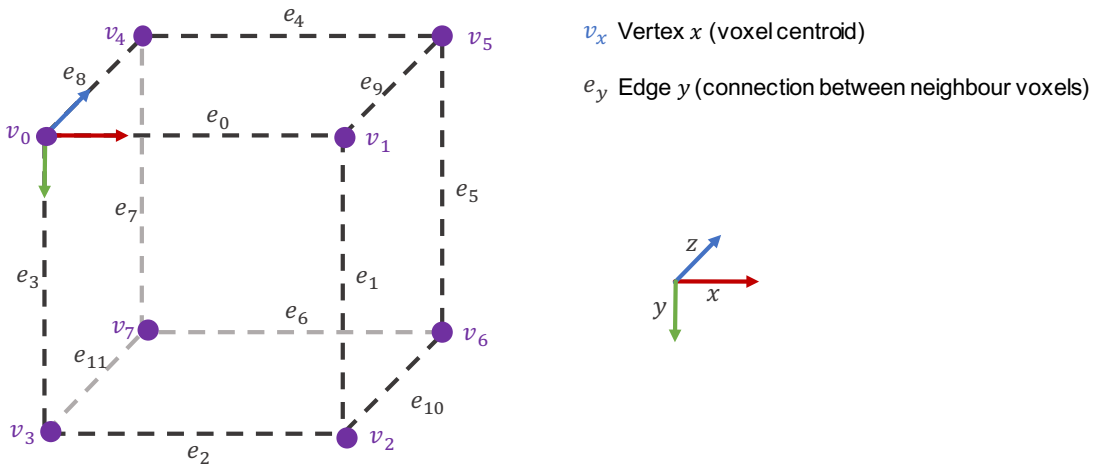


Figure 2.17: Marching Cubes Notation Overview. The marching cubes algorithm extracts the surface from a voxel grid by independently processing each voxel, v_0 in the above example. One byte, b_{mc}, is computed where each bit is 1 if the voxel is less than the isovalue and 0 otherwise. This value is used in a series of LUTs (see Figure 2.18 on the next page) to compute how many triangles are contained within the cube (formed by 8 neighbouring voxel centroids) and interpolate their precise location along the edges. Anywhere between 0 to 15 triangles may be found in each cube.

where n is the edge number, i is the isovalue (always 0 in this thesis), and p_0 and p_1 are the world-space coordinates of v_0 and v_1. This interpolation is performed for each edge of the cube (Figure 2.17) that is intersected by the isosurface.

This is where the utility of Marching Cubes' LUTs (Figure 2.18 on the following page) becomes apparent.

1. A byte of memory, b_{mc}, is reserved where each bit stores $v_n < i$.

2. `EDGE_CASES`$[b_{mc}]$ returns the 12-bit number where each bit indicates which edge (from Figure 2.17) is intersected. This simplifies the process of identifying edges on which to apply Equation 2.34 on the facing page and thus producing a list of n known intersection points.

3. `NUM_TRIANGLES`$[b_{mc}]$ returns the number of triangles contained within the cube so the appropriate amount of memory may be reserved on the GPU.

4. Finally, `TRIANGLE_CASES`$[b_{mc}]$ returns the vertex order for the triangles. For example, if $b_{mc} = 3$, from Figure 2.18 on the next page we see that there are two triangles (`NUM_TRIANGLES`) with vertices (s_1, s_8, s_3) and (s_9, s_8, s_1).

```
uint16_t EDGE_CASES[256] = {
  0x000, 0x109, 0x203, 0x30A, 0x406, 0x50F, 0x605, 0x70C,
  // ...
};

uint8_t NUM_TRIANGLES[256] = {
  0,     1,     1,     2,     1,     2,     2,     3,
  // ...
};

#define NUM_TRIANGLES_PER_VOXEL 5
#define X -1
int8_t TRIANGLE_CASES[256][3*NUM_TRIANGLES_PER_VOXEL] = {
  {X,  X,  X,  X,  X,  X,  X,  X,  X,  X,  X,  X,  X,  X,  X },
  {0,  8,  3,  X,  X,  X,  X,  X,  X,  X,  X,  X,  X,  X,  X },
  {0,  1,  9,  X,  X,  X,  X,  X,  X,  X,  X,  X,  X,  X,  X },
  {1,  8,  3,  9,  8,  1,  X,  X,  X,  X,  X,  X,  X,  X,  X },
  {1,  2,  10, X,  X,  X,  X,  X,  X,  X,  X,  X,  X,  X,  X },
  {0,  8,  3,  1,  2,  10, X,  X,  X,  X,  X,  X,  X,  X,  X },
  {9,  2,  10, 0,  2,  9,  X,  X,  X,  X,  X,  X,  X,  X,  X },
  {2,  8,  3,  2,  10, 8,  10, 9,  8,  X,  X,  X,  X,  X,  X },
  // ...
};
```

Figure 2.18: Marching Cubes LUTs. Once the isovalue byte, b_{mc} is computed (see Figure 2.17 on the previous page), this value is used in a series of three LUTs. First, e_{mc} = `EDGE_CASES`$[b_{mc}]$ returns a `uint16_t` with 12 bits indicating which edges in the cube are intersected by a triangle and therefore the triangle vertex must be interpolated. Second, n_{mc} = `NUM_TRIANGLES`$[b_{mc}]$ returns the number of triangles contained within that cube. Finally, r_{mc} = `TRIANGLE_CASES`$[b_{mc}]$ returns the order of cube-edge vertices (computed in the previous step) for each triangle. As the computation of the triangles in each cube is independent, this is well suited for GPGPU.

The surface normal can be computed using a process similar to the above, but linearly interpolate the TSDF *gradient* (∇u_{TSDF}) rather than the TSDF value (u_{TSDF}). As each voxel's computation is independent of the other voxels, this algorithm may be efficiently implemented on a GPU and the resulting mesh streamed on demand to other robotics modules.

Once the surface is extracted, we can evaluate the reconstructions accuracy against ground-truth data.

2.5 Mesh and Point Cloud Metric Evaluation

In subsequent chapters, we provide metric error analysis to evaluate the quality of reconstructions created by our algorithms. CloudCompare is an open-source tool designed for this purpose [65]. In particular, we found three specific features of CloudCompare indispensable in our analysis: downsampling, registration,

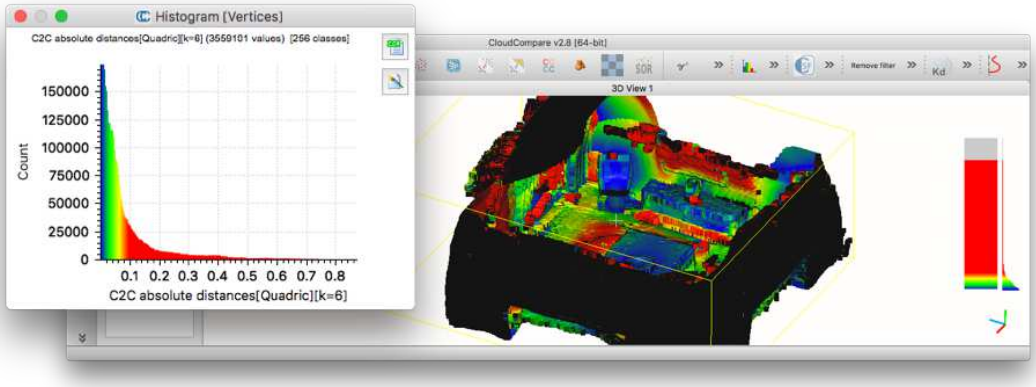


Figure 2.19: CloudCompare Example: Error Analysis. Given two aligned point clouds or meshes, CloudCompare can compute the metric error. Each vertex in the evaluation point cloud is compared to the ground-truth model via a user-selected method (e.g., nearest point, linear model of k-nearest point, etc.). CloudCompare outputs summary histograms, statistics, and visualisations (see above) of the errors.

and error analysis.

Downsampling is required when the point clouds are too large to fit into the systems memory. The level of detail in these ground-truth files is often unnecessary. For example, in Chapter 4, a single ground-truth model—created by consolidating all laser scans from a 3.8 km vehicle trajectory—required 13.79 GiB. The reconstruction used 10 cm voxels, but many lidar points were less than 1 cm apart. Even further-apart points could be modelled with a local model function. CloudCompare provides the functionality to downsample the point clouds into a representative subset.

The `reference` (ground truth) and `compared` (evaluation) surface models must be registered before performing an error analysis. If both models were created with the same global reference frame, this is not required. However, sometimes a common reference frame is not feasible because of an unknown relative transformation during data collection (e.g., Chapter 5). The user aligns point clouds in CloudCompare by selecting four (or more) matching points in the two reconstructions.

Finally, once segmented, downsampled, and registered, CloudCompare performs metric point cloud error analysis, the results of which are shown in Figure 2.19. CloudCompare compares the `compared` point cloud to a `reference` point cloud. The fidelity of this analysis is improved by utilising a *local model* (e.g., least-squares

plane) of the surface for each point-to-surface comparison.

The real-world ground truth used in this thesis is exclusively 3D laser point clouds. Fusing the laser data into a voxel grid and extracting the surface model would produce a more visually appealing (and dense) ground truth, but the fusion process implicitly introduces an averaging prior that would bias the error metrics. Instead, we process raw laser point clouds and downsample them when operating under memory constraints.

2.6 Conclusions

This chapter introduced the fundamental concepts required to understand the body of work presented in this thesis. Reference frames and coordinate transformations (Section 2.1 on page 10) enable us to describe the precise $\mathbb{SE}(3)$ position of each sensor and platform at a given point in time. Sensor models and intrinsic/extrinsic calibration (Section 2.2 on page 15) ensure we correctly process the raw data received from cameras and lidar sensors. Depth maps (Section 2.3 on page 24) generated from monocular or stereo cameras provide real-time data about the 3D structure of the environment without requiring an expensive (lidar) and limited-range (RGB-D) active sensor. Voxel grids in concert with depth map data fusion (Section 2.4 on page 30) enable *dense* 3D reconstruction for large-scale environments. Finally, CloudCompare (Section 2.5 on page 38) enables us to evaluate the metric quality of these reconstructions against ground-truth data.

Building upon the concepts introduced in this chapter, in Chapter 3, we will present a method to create high-quality dense reconstructions by regularising the 3D surfaces to remove noise from low-quality monocular depth maps. We extend this in Chapter 4 to large-scale environments with inputs from multiple sensor modalities. Finally, in Chapter 5, we apply the regularisation techniques from Chapter 3 to remove ephemeral objects (e.g., automobiles) and interpolate occluded urban structure—thus further improving the quality of our dense reconstructions.

All models are wrong, but some are useful.

— George E. P. Box, *Box & Draper (1988)*

3

Dense 3D Reconstruction and Optimisation

Abstract

Camera-generated depth maps are prone to noise that make it difficult to create quality reconstructions. This noise is especially prevalent with geometric-based monocular camera algorithms, to which we restrict our attention in this chapter. We present the theory to regularise noisy reconstructions for real-world applications. We consider this from the perspective of a vehicle moving *through* the environment, in contrast to the object-centric reconstructions common in the literature.

Our approach limits the regulariser to only operate in regions of the voxel grid where surface data was directly observed by the camera. Failing to do so results in the generation of spurious surfaces, a problem not previously addressed in the dense reconstruction literature.

We successfully demonstrate a Total Variation (TV) regulariser improves reconstructions in synthetic and real-world datasets. When compared to 3D lidar point clouds, our regulariser reduces the metric dense reconstruction error by 61% in outdoor environments and 51% in indoor environments, with final respective median errors of 14.41 cm and 15.08 cm.

Contents

PASSIVE cameras, in mobile-robotics applications, struggle to create high-quality depth maps in real-world environments. When fusing a sequence of depth maps into a voxel grid, the resulting reconstructions are noisy. In this chapter, we present the theory—using fused monocular-camera-generated depth maps—to regularise the noisy surfaces and create high-quality dense 3D models

Figure 3.1: An example 3D reconstruction created using the techniques described in this chapter from an automobile's forward-facing monocular camera.

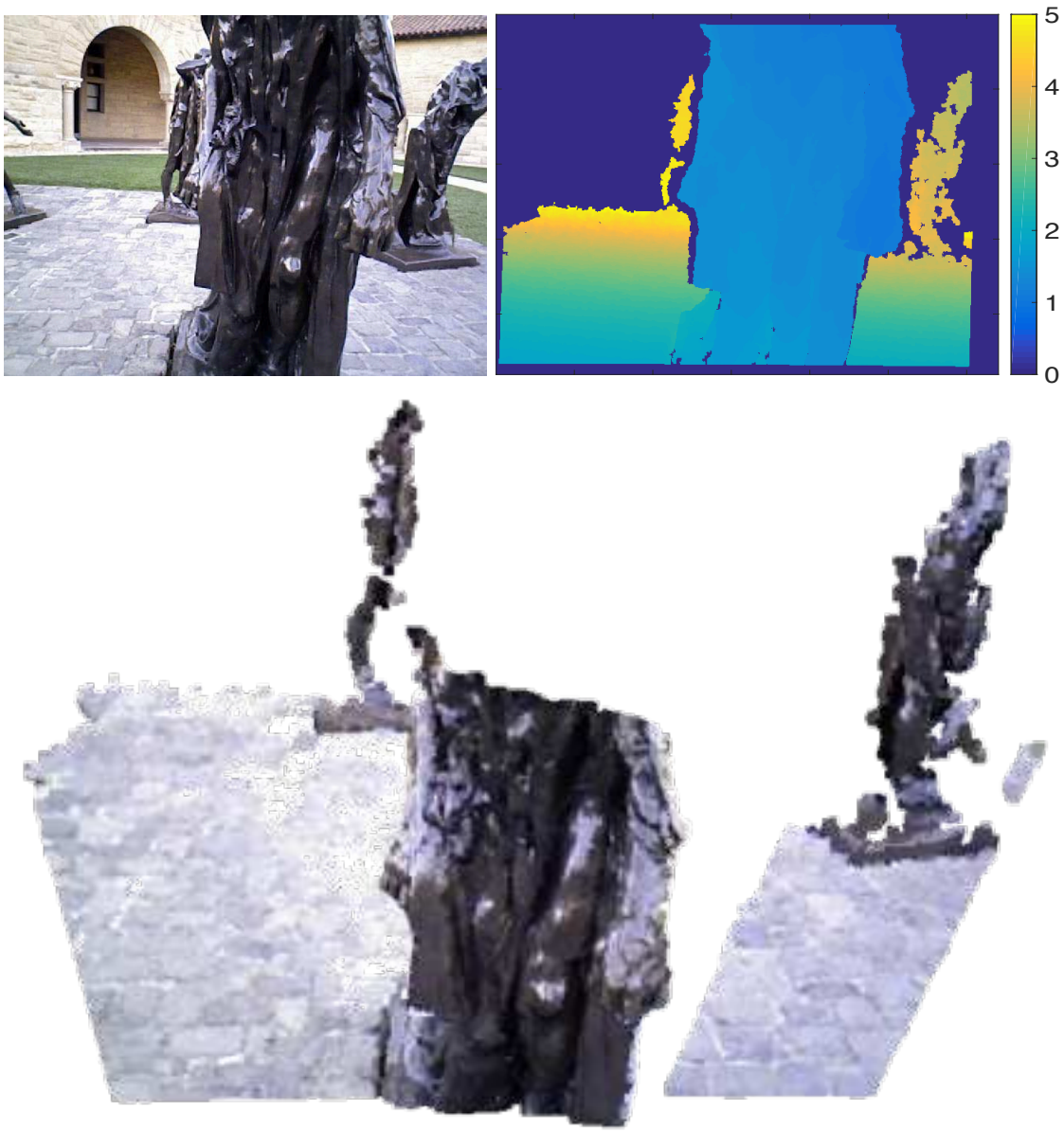


Figure 3.2: Sample RGB-D Camera Depth Map and Point Cloud. Clockwise from top-left: RGB image, depth map (metres), and projected 3D point cloud. Captured with an Asus Xtion Pro as part of the Stanford Burghers of Calais dataset [66], this frame is representative of the high quality depth data captured by RGB-D cameras. Contrast the quality of this sparse reconstruction with Figure 3.3 on the next page.

(e.g., Figure 3.1) that accurately reproduce the observed environment.

Dense reconstruction research in recent years has primarily focused on processing a stream of high-quality RGB-D data that, for close objects, can produce stunningly accurate surface models [2][67][59][68][69]. While each pixel of the RGB-D sensor

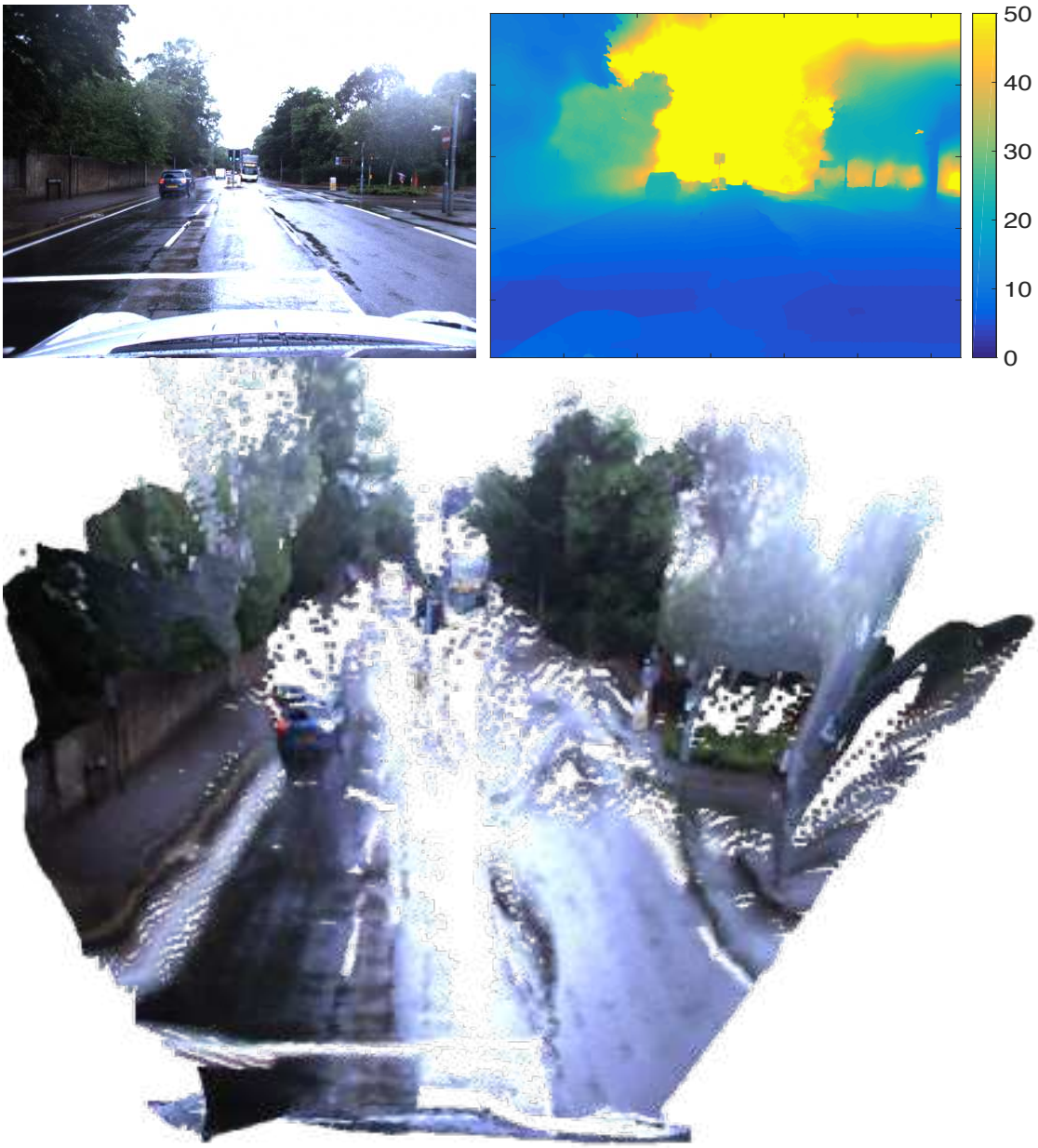


Figure 3.3: Sample Stereo Camera Depth Map and Point Cloud. Clockwise from top-left: RGB image, depth map (metres), and projected 3D point cloud. Captured with an Point Grey Bumblebee XB3 stereo camera as part of the Oxford RobotCar dataset [27], this frame is representative of how passive cameras struggle to reconstruct outdoor environments. Even when provided a known baseline with registered images, the wet, reflective road creates significant noise in the 3D point cloud. This is why our system must regularise in both 2D (as done in the above depth map) and 3D (as documented in this chapter). Contrast the quality of this sparse reconstruction with Figure 3.2 on the previous page.

contains errors, the *average* of thousands of depth maps taken of a single object from a variety of angles creates millimetre-level accurate reconstructions.

An example of one such depth map from a Asus Xtion Pro is shown in Figure 3.2 on page 43. Notice that the depth map, when projected into 3D space as a point cloud, captures the fine details of the statue and ground. When the sensor is moved slowly and intentionally by the operator, such that all surfaces are fully observed from multiple directions (and depth maps received at 30 Hz), the reconstructions from this type of dataset are of very high quality (Figure 4.1 on page 77).

In contrast to RGB-D sensors, depth maps created from *passive* cameras suffer from significant noise. Figure 3.3 on the facing page presents a typical depth map generated by an automobile's forward-facing stereo camera—the common use case we seek to optimise throughout this thesis. Notice how the depth map struggles with the planer surface of the road because of the wet/reflective surface, and the points on the road closest to the automobile are seen as curving into the ground. These types of irregularities are even more pronounced in depth maps created from monocular camera image sequences. However, the depth *range* of passive sensors is an order of magnitude larger than RGB-D cameras (50 m vs. 5 m). Range is the salient feature in mobile-robotics applications since the passive camera's depth map provides more time to react to objects in front of the vehicle.

However, an automobile-mounted sensor also is limited in its observation of the environment. In contrast to typical hand-held RGB-D datasets, the mounted forward-facing camera may only capture a few frames of any given object from non-ideal angles, so we must fuse all captured data into a voxel grid and utilise a 3D regulariser in an attempt to best reconstruct the observed surfaces. The regulariser serves as a geometric prior on the shape we expect of the environment, thus penalising noisy data that does not conform.

The critical challenge in regularising reconstructions made from forward-facing automobile cameras is that only a subset of voxels are directly observed by the camera. Applying regularisation to the entire voxel grid produces spurious surfaces (Section 3.4 on page 63). *We present a new method—the primary contribution of*

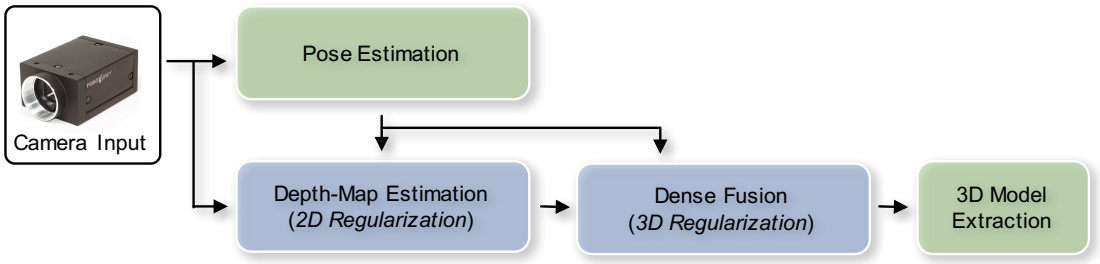


Figure 3.4: An overview of our software pipeline. Our Dense Fusion module accepts data from monocular camera depth maps. We regularise the 3D model, extract the surface, and then provide the final 3D model.

this chapter—in Section 3.5 on page 65 (that we call "Ω labelling") that targets regularisation to only directly-observed voxels.

Figure 3.4 provides a basic overview of the system presented in this chapter. First, we process a stream of camera images. The sparse Pose Estimation module (implemented via VO in this chapter, whereas Chapter 4 uses a full SLAM implementation) computes a $\mathbb{SE}(3)$ pose for each image while the Depth-Map Estimation module uses a sequence of images and poses to compute key-frame, 2D-regularised depth maps. Our Dense Fusion module, the primary focus of this chapter, fuses the sequence of depth maps and performs 3D regularisation directly on the noisy voxel data. The resulting model is extracted via Marching Cubes, as described in Section 2.4.3 on page 36 [62].

In Section 3.1 on the next page we present an overview of the fundamentals of optimisation to provide context for the application-specific selections we make in Section 3.2 on page 54. Next, in Section 3.3 on page 59, we review the specific algorithms required to implement our 3D regulariser. However, naïvely applying this regulariser to a partially-observed voxel grid produces unexpected results (Section 3.4 on page 63), that we address in Section 3.5 on page 65. Finally, in Section 3.6 on page 68, we quantitatively analyse the performance of our system in both indoor and outdoor environments.

3.1 Optimisation Overview

In this section, we provide a general introduction to optimisation before discussing our application-specific optimisation selections in Section 3.2 on page 54.

3.1.1 Energy

The goal of optimisation is to minimise a given function subject to certain constraints [70],

$$\begin{aligned} \text{minimise} \quad & E(u) \\ \text{subject to} \quad & g_i(u) \leq b_i, \quad i = \{1, \ldots, m\} \end{aligned} \tag{3.1}$$

where $E(u)$ is the equation that we wish to minimise and $g_i(u)$ is set of functions on which there are constraints $b_{1,2,\ldots,m}$.

Most energy-minimisation problems in this thesis are unconstrained (i.e., $m = 0$) and will take the form,

$$E(u) = E_{regularisation}(u) + E_{data}(u, f) \tag{3.2}$$

where $E(u)$ is the function we seek to minimise, u is the proposed solution, and f is the set of noisy input data. The two terms, *regularisation* and *data*, seek a denoised solution that meets a desired prior state (*regularisation*) but does not vary too far from the original observed data (*data*).

For example, if one expects the input to be a constant value, then the regulariser might take the form,

$$E_{regularisation}(u) = \int_{\Omega} ||\nabla u||_1 d\Omega \tag{3.3}$$

where Ω is the domain of integration (e.g., a set of observed data points). This specific term penalises sequential large local changes in u. This is essentially a prior that prevents the final solution from over-fitting the original data.

However, if left unchecked, the regulariser will force the optimiser to find a trivial solution (e.g., $u = 0$ for all data). The *data* term penalises these solutions,

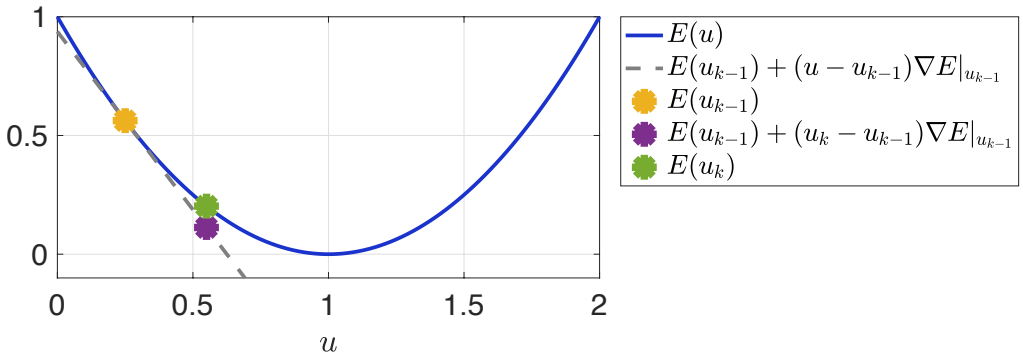


Figure 3.5: Gradient Descent Example. Gradient descent finds the minimum of a convex function $E(u)$ by taking small steps (the size of which is dictated by α) along the cost function. Given the current estimate u_k, the energy function is linearised around that point, an α-sized step is taken towards the minimum, and the process is continually repeated until the energy reduction falls below a set threshold.

thus forcing the optimiser to select a solution that appropriately conforms to both the prior (regulariser) and data terms. For example,

$$E_{data}(u, f) = \frac{\lambda}{2} \int_{\Omega} ||f - u||_2^2 d\Omega \tag{3.4}$$

penalises any solutions that vary from the input data. λ allows a trade off between solutions that closely match the regulariser's prior (small λ) vs. solutions that conform to the original data (large λ).

3.1.2 Gradient Descent

Except for the most trivial of problems, there is no closed-form solution to solve the energy-minimisation in a single step. However, if $E(u)$ is convex, a properly configured gradient descent (or ascent, if solving a maximisation problem) approach will converge on the single maximum or minimum solution. With the exception of a special data term we discuss in Chapter 5, all energy functions in this thesis are convex and are solved with a modified version of gradient descent or ascent. However, non-smooth convex functions (such as the L_1 norm used throughout this thesis) require reformulation before using gradient descent/ascent—see Section 3.1.3 on the facing page for more details.

As shown in Figure 3.5 on the preceding page, gradient descent works by taking small "steps" along the energy function, where each step moves closer to the global minimum. Since we constrain ourselves to convex problems, there are no local minima (just a single global minimum), and the system will converge upon the same final solution, no matter the initial value of u_k.

Gradient descent is a continual loop executing [70],

$$u_k = u_{k-1} - \alpha \nabla E|_{u_k} \tag{3.5}$$

where $\nabla E|_{u_k}$ is the Jacobian of the energy function and α, a small positive number, is the step size. If α is sufficiently small, the system is guaranteed to converge upon the optimal solution for convex energy functions. One may check for convergence by evaluating [70],

$$||\nabla E(u)||_2 \leq \eta \tag{3.6}$$

where η is a small positive number. If not converged, continue with gradient descent.

Gradient descent provides a simple method to find the optimal solution for smooth, convex energy minimisation problems.

3.1.3 The Legendre-Fenchel Transformation

The above formulation of gradient descent requires a smooth $E(u)$. However, as we rely upon the non-smooth L_1 norm term in our regularisers (see Section 3.2.1 on page 54), the energy function must be converted into an alternate form to solve the optimisation problem—we achieve this via the Legendre-Fenchel (LF) Transformation [70][71][72]. This section presents an introduction to the LF Transform with the relevant details required to implement an L_1 optimiser. Section 3.2.1 on page 54 provides the application-specific version of our regulariser while Section 3.3 on page 59 outlines the specific equations required to implement the optimiser within the context of a scene-reconstruction voxel grids.

Given a continuous but optionally smooth function $E(u)$, the LF Transform is defined in its scalar and vector form [72],

$$\begin{aligned} E^*(p) &= \underset{u\in\mathbb{R}}{\text{argsup}}\{up - E(u)\} \\ E^*(\mathbf{p}) &= \underset{\mathbf{u}\in\mathbb{R}^n}{\text{argsup}}\{< \mathbf{u}, \mathbf{p} > - E(\mathbf{u})\} \end{aligned} \tag{3.7}$$

where $E^*(p)$ is the conjugate, and $< \cdot, \cdot >$ is the inner-product operator. $E^*(p)$ is the "dual" of $E(u)$ and may be thought of as a parameterised representation of $E(u)$. It finds the point u on $E(u)$ that maximises the y-intercept of a line that passes through the point $(u, E(u))$—geometrically equivalent to computing the tangent line at u [72],

$$p = E'(u) \tag{3.8}$$

With respect to the energy functions used in this thesis, the LF Transform has four salient properties [72]:

1. Points on the function $E(u)$ are transformed into slopes on the function $E^*(p)$, and vice versa.
2. $E^*(u)$ is always convex.
3. The process is reversible: $(u, E(u)) \Longleftrightarrow (p, E^*(p))$.
4. For convex problems, the minimising the primal is equivalent to maximising its dual; they both converge to the same solution, as shown in Figure 3.6 on the next page.

This formulation allows us to approach the optimisation problem simultaneously from two viewpoints: primal (u) and dual (p). The convex, smooth dual formulation is critical to minimising non-smooth primal energy functions.

In Section 3.2.1 on page 54 we introduce the TV energy term that contains an L_1 norm.

$$E_{||\cdot||_1}(\mathbf{u}) = ||\mathbf{u}||_1 \tag{3.9}$$

Note, for clarity, the above is presented in standard vector form, but the TV primal variable is the gradient of a 3D scalar field (∇u). Since this energy equation is not

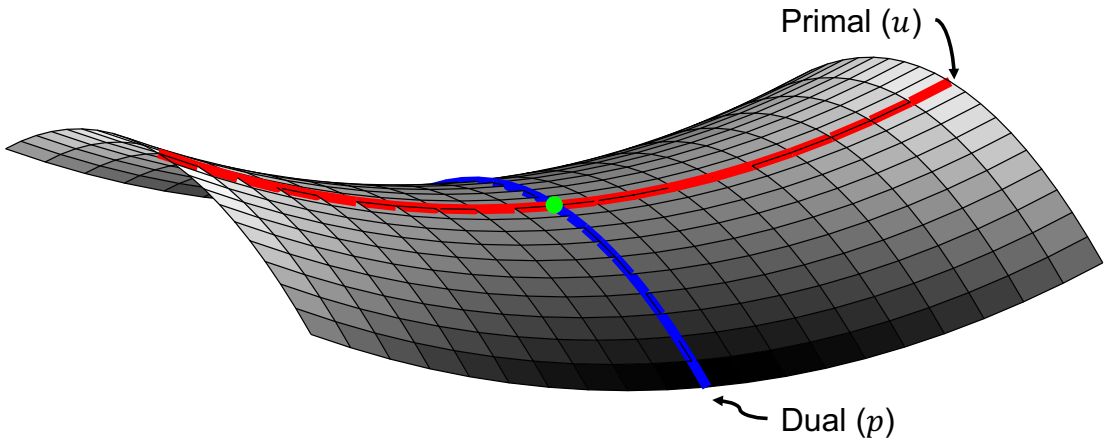


Figure 3.6: Primal-Dual Saddle Example. A non-differentiable energy function, $E(u)$, may be minimised by leveraging a LF Transform combined with a Primal-Dual formulation. Using these techniques, $\underset{u \in \mathbb{R}}{\text{argmin}}\{E(u)\}$ is converted into two separate, smooth optimisation problems: minimise the primal u and maximise the dual p. This is solved by alternating between gradient descent (u) and ascent (p) until converging upon the solution. Figure from [9].

smooth, one option is to *approximate* a smooth function via the Huber norm ($||\cdot||_\epsilon$) by constructing a smooth function composed of two convex functions [44],

$$||\mathbf{u}||_\epsilon = \begin{cases} \frac{||\mathbf{u}||_2^2}{2\epsilon} & \text{if } ||\mathbf{u}||_2 \leq \epsilon \\ ||\mathbf{u}||_1 & \text{otherwise} \end{cases} \tag{3.10}$$

where ϵ is a small number. This approximates the L_1 norm by replacing the non-differentiable portion of $E(\mathbf{u})$ at the $\mathbf{u} = \mathbf{0}$ "kink" with a quadratic function. Rather than use the Huber norm approximation, in this thesis we elect to use a more precise optimisation implementation—based on the LF Transform—directly on the L_1 norm. This enables the system to converge in fewer iterations by alternating between gradient descent ($\mathbf{u}$) and gradient ascent ($\mathbf{p}$)—see Figure 3.6 for a graphical depiction of this process.

After substituting the L_1 norm into Equation 3.7 on the facing page, it becomes,

$$E^*_{||\cdot||_1}(\mathbf{p}) = \underset{\mathbf{u} \in \mathbb{R}^n}{\text{argsup}}\{< \mathbf{u}, \mathbf{p} > - ||\mathbf{u}||_1\} \tag{3.11}$$

or on a per-element basis,

$$E^*_{||\cdot||_1}(\mathbf{p}) = \underset{u_i \in \mathbb{R}}{\text{argsup}} \left\{ \sum_i \left((u_i p_i - ||u_i||_1 \right) \right\} \tag{3.12}$$

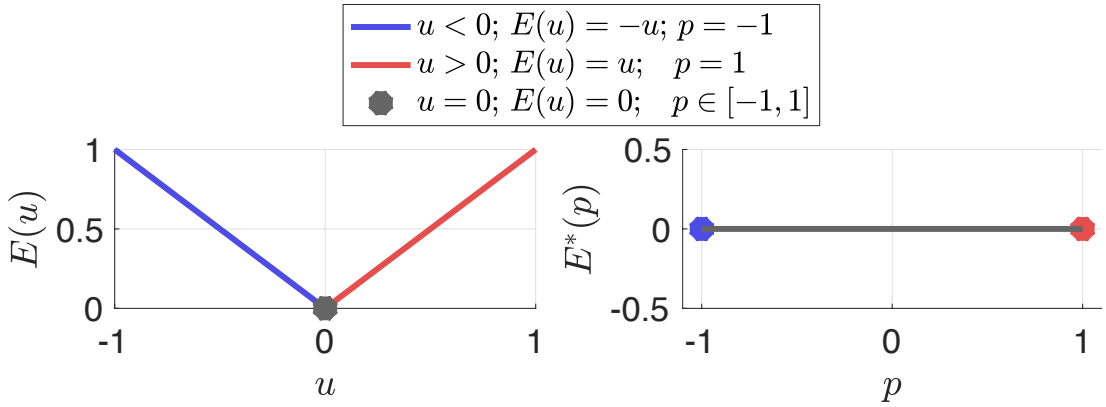


Figure 3.7: Total Variation Legendre-Fenchel Transform Overview. As the L_1 norm (left) is not smooth and therefore not differentiable, however a solution is possible if one divides the equation into three regions: $(u < 0)$, $(u > 0)$, and $(u = 0)$. The first two $(u < 0$ and $u > 0)$ regions are smooth and differentiable with respective slopes of $p = -1$ and $p = 1$. However, in the border region $(u = 0)$ there are a range of acceptable slope values, $p \in [-1, 1]$. This dual space, p, is depicted in (right). The LF Transform therefore transforms $E(u) = ||u||_1 \rightarrow E(u) = \underset{||p||_\infty \leq 1}{\text{argsup}} < x, p >$ that can be solved via projected gradient ascent. Figure inspired by [9].

this can be broken into two components,

$$\begin{aligned} E^*_{||\cdot||_1}(\mathbf{p}) &= \underset{u_i \in \mathbb{R}}{\text{argsup}} \left(\sum_i \begin{cases} u_i p_i - u_i & \text{if } u_i \geq 0 \\ u_i p_i + u_i & \text{if } u_i < 0 \end{cases} \right) \\ &= \underset{u_i \in \mathbb{R}}{\text{argsup}} \left(\sum_i \begin{cases} u_i (p_i - 1) & \text{if } u_i \geq 0 \\ u_i (p_i + 1) & \text{if } u_i < 0 \end{cases} \right) \end{aligned} \tag{3.13}$$

Since p, as previously discussed, is equivalent to the slope of u, we provide a simple one dimensional (1D) example of the L_1 norm in Figure 3.7 [73]. The slope (p) is constant for $u < 0$ $(p = -1)$ and $u > 0$ $(p = 1)$, but the L_1 norm is non-differentiable at $u = 0$. For gradient descent to work properly, we must approximate the slope at that point. The key insight is that there is not a single valid slope at $u = 0$, rather there is a valid *range* of slopes: $p_i \in [-1, 1]$. Any slope outside that range is invalid.

Given the restriction $p_i \in [-1, 1]$, we can state $(p_i - 1 \leq 0)$ and $(p_i + 1 \geq 0)$ which means both conditional terms in Equation 3.13 $((u_i(p_i - 1))$ if $u_i \geq 0$ and $(u_i(p_i + 1))$ if $u_i < 0)$ are always less than or equal to zero. Therefore, the supremum is *always* at $u_i = 0$ with the restriction $p_i \in [-1, 1]$. The original supremum problem

(Equation 3.11 on page 51) may now be rewritten [72],

$$\begin{aligned} E^*_{||\cdot||_1}(\mathbf{p}) &= \underset{||p_i||_1 \leq 1 \; \forall i}{\operatorname{argmax}} \{ < \mathbf{u}, \mathbf{p} > - \; ||\mathbf{u}||_1 \} \\ &= 0 \qquad \text{if } ||p_i||_1 \leq 1 \; \forall i \end{aligned} \tag{3.14}$$

The utility of the this formulation becomes apparent when used in an energy minimisation problem,

$$\underset{\mathbf{u} \in \mathbb{R}^n}{\operatorname{argmin}} \; ||\mathbf{u}||_1 \tag{3.15}$$

While one may solve this particular problem by inspection ($\mathbf{u} = \mathbf{0}$), when additional terms are added (e.g., Section 3.3.1 on page 59), the LF Transform process is required to apply gradient ascent and descent to find the optimal $\mathbf{u}$. The dual form of Equation 3.7 on page 50 is,

$$E(\mathbf{u}) = \underset{\mathbf{p} \in \mathbb{R}^n}{\operatorname{argsup}} \{ < \mathbf{u}, \mathbf{p} > - \; E^*(\mathbf{p}) \} \tag{3.16}$$

We therefore substitute Equation 3.9 on page 50 into the minimisation problem and use this dual form with $E^*_{||\cdot||_1}(\mathbf{p})$ to simplify,

$$\begin{aligned} \underset{\mathbf{u} \in \mathbb{R}^n}{\operatorname{argmin}} \; ||\mathbf{u}||_1 &= \underset{\mathbf{u} \in \mathbb{R}^n}{\operatorname{argmin}} \; \left(E_{||\cdot||_1}(\mathbf{u}) \right) \\ &= \underset{\mathbf{u} \in \mathbb{R}^n}{\operatorname{argmin}} \; \underset{\mathbf{p} \in \mathbb{R}^n}{\operatorname{argsup}} \{ < \mathbf{u}, \mathbf{p} > - \; E^*_{||\mathbf{u}||_1}(\mathbf{p}) \} \\ &= \underset{\mathbf{u} \in \mathbb{R}^n}{\operatorname{argmin}} \; \underset{||p_i||_1 \leq 1 \; \forall i}{\operatorname{argmax}} \{ < \mathbf{u}, \mathbf{p} > - \; 0 \} \\ &= \underset{\mathbf{u} \in \mathbb{R}^n}{\operatorname{argmin}} \; \underset{||p_i||_1 \leq 1 \; \forall i}{\operatorname{argmax}} \{ < \mathbf{u}, \mathbf{p} > \} \end{aligned} \tag{3.17}$$

This formulation is now solvable by alternating between projected gradient ascent ($\mathbf{p}$),

$$\begin{aligned} \mathbf{p}_k &= \xi(\mathbf{p}_{k-1} + \sigma \nabla E|_{\mathbf{p}_k}) \\ \xi(\mathbf{p}_{k-1}) &= \frac{\mathbf{p}_{k-1}}{\max(1, ||\mathbf{p}||_2)} \end{aligned} \tag{3.18}$$

and conventional gradient descent ($\mathbf{u}$),

$$\mathbf{u}_k = \mathbf{u}_{k-1} - \tau \nabla E|_{\mathbf{u}_k} \tag{3.19}$$

where σ and τ are the respective gradient ascent and descent step sizes. "Projected" gradient ascent is identical to gradient ascent, except that the projection operator $\xi(\cdot)$ enforces the constraint on p.

The primary take-away is this: by utilising the LF Transform's Primal-Dual formulation, we convert the non-smooth minimisation problem into two separate problems: a minimisation (u) and maximisation ($\mathbf{p}$). A visual depiction of this process is provided in Figure 3.6 on page 51 and Figure 3.7 on page 52.

3.2 Application-Specific Optimiser

As discussed in Section 3.1.1 on page 47, the optimisation energy functions usually take the form,

$$E(u) = E_{regularisation}(u) + E_{data}(u, f) \tag{3.20}$$

where the *regularisation* term is a *prior*—the assumed underlying structure of the data. The *data* term seeks to keep the proposed solution (u) similar to the originally observed values (f). A systems designer must select *regularisation* and *data* terms that meet their requirements. Based on our scene-reconstruction goal, we choose a TV regulariser (Section 3.2.1) and two data terms (Section 3.2.2 on page 57 and Section 3.2.3 on page 58).

3.2.1 Regulariser Term

The TV regulariser takes the form [74][75],

$$E(u) = \int_{\Omega} ||\nabla u||_1 d\Omega + E_{data}(u, f) \tag{3.21}$$

One immediate concern with the L_1 norm is that, though it is convex, it is not smooth so the conventional gradient descent methods cannot be directly used. As discussed in Section 3.1.3 on page 49, we elect to directly implement this energy function, rather than attempt to approximate it with the Huber norm. This does not affect the core contributions of this thesis, however the more precise implementation eliminates one additional parameter in the system (ϵ) while presenting a modern variational method implementation.

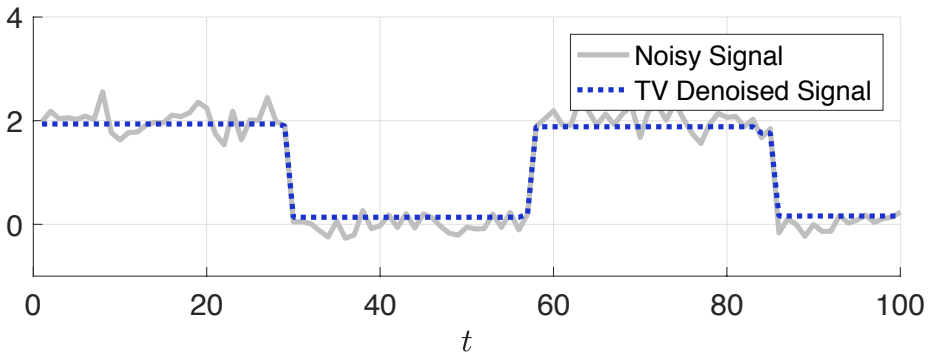


Figure 3.8: Simple Total Variation 1D Example. A Total Variation regulariser seeks to reduce noise by finding a piecewise-constant solution consistent with the noisy input data. In the above example, the input square wave was subjected to $\mathcal{N}(0, 0.2)$. The TV denoised signal is the solution from minimising the energy in Equation 3.21 on the facing page with and $E_{data}(u, f) = ||f - u||_2^2$. Note that TV gracefully handles the sharp discontinuities in the input signal, a desirable feature as this type of input appears often in urban environments (e.g., the intersection of pavement and a building's façade).

The LF Transform [76][77] enables us to transform the TV cost function into a saddle-point optimisation problem, similar to the problem discussed in Figure 3.6 on page 51. In a similar form to the L_1 vector LF Transform solution found in Equation 3.17 on page 53, the minimisation problem is transformed via,

$$\begin{aligned} \underset{u\in\mathbb{R}}{\operatorname{argmin}} ||\nabla u||_1 &= \underset{u\in\mathbb{R}}{\operatorname{argmin}} \underset{u\in\mathbb{R}}{\operatorname{argsup}}\{< \nabla u, \mathbf{p} > - E^*(\mathbf{p})\} \\ &= \underset{u\in\mathbb{R}}{\operatorname{argmin}} \underset{||\mathbf{p}||_\infty \leq 1}{\operatorname{argmax}}\{< \nabla u, \mathbf{p} >\} \end{aligned} \tag{3.22}$$

As discussed in Section 3.1.3 on page 49, this formulation can be solved by alternating between projected gradient ascent ($\mathbf{p}$) and conventional gradient descent (u). The full implementation is presented in Section 3.3 on page 59.

Why select TV given this difficulty in implementing the energy minimisation? TV is robust to outliers and, due to the nature of its piecewise-constant assumption, gracefully handles sharp discontinuities. These benefits are best seen via the 1D example in Figure 3.8. Given a noisy square wave as the input, the TV regulariser successfully captures the underlying signal while preserving the sharp square-wave transitions. Other regularisers (e.g., L_2 norm) have a tendency to produce a "smearing" effect and thus smooth out rather than preserve discontinuities.

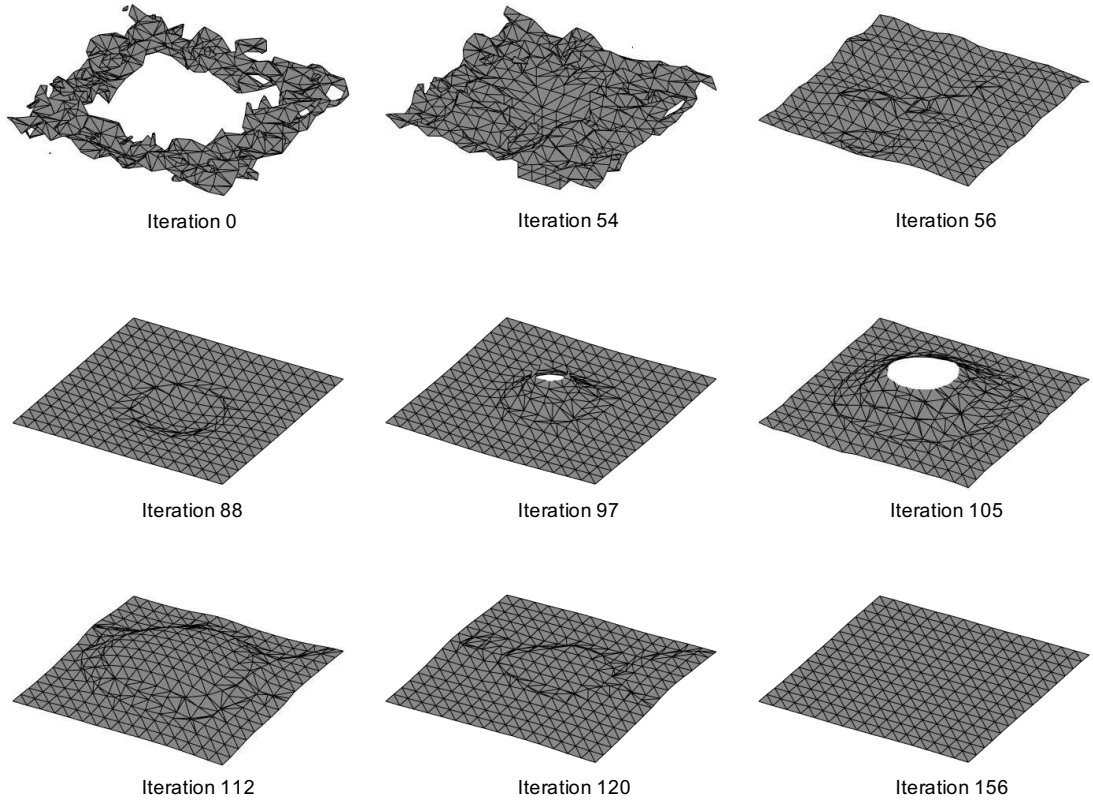


Figure 3.9: Regulariser's Gradient Ripple Effect. The gradient operator in the gradient creates a "ripple" effect throughout the energy minimisation process. In the above example, the input to the optimiser is a noisy plane with the centre removed. As it moves through each iteration, the one can visually see the energy minimisation solutions rippling through the reconstruction until it converges upon the most efficient (and in this case, correct) solution. Note that the above example also demonstrates the interpolation effect of naïve 3D regularisation, but Figure 3.11 on page 63 shows the undesirable extrapolation of the regulariser. This chapter introduces a method to avoid this spurious interpolation/extrapolation, whereas Chapter 5 leverages the interpolation to reconstruction hidden objects in urban scenes.

When regularising a noisy 3D reconstruction, as shown in Figure 3.9, the gradient in the regulariser causes data from neighbouring voxels to propagate. Over the course of many iterations, one can visually see this wave of energy propagate throughout the voxel grid before the optimiser converges upon the most likely solution.

Two additional critical concepts are illustrated in Figure 3.9. First, the TV regulariser nearly perfectly reconstructs the plane even when given partially-occluded, noisy input. Second, due to the nature of the gradient operator propagating data, the regulariser interpolated structure where none was originally observed (i.e.,

in the centre of the plane). We discuss the undesirable implications of this in Section 3.4 on page 63.

Having selected the regulariser, the last part of the energy function in Equation 3.20 on page 54 is the data term. We present two options in the following two subsections, each of which is tailored for different sensors. The SDF data term Section 3.2.2 is ideal for low-noise sensors (e.g., lidar, RGB-D cameras), while the histogram data term (Section 3.2.3 on the next page) is more robust in the presence of noise. We analyse the relative performance of these data terms in Section 4.3 on page 92.

3.2.2 SDF Data Term

In the method originally proposed by Curless [54] and popularised by Newcombe [2], data is fused into the voxel grid by storing in each voxel a weighted average of all depth observations. To produce good results, this method usually requires the noise to either be small or Gaussian.

The end result of fusion (Equation 2.33 on page 35) are two numbers stored in each voxel:

1. f — the signed metric distance to the nearest surface
2. w — the number of observations used to compute f

One may also think of f and w as storing the mean (f) and information (w, inverse of variance) to characterise the sampled distribution. As the energy minimisation seeks to solve for a denoised-version of f, a simple L_2 norm data term ensures the final u is consistent with the observed depths:

$$E(u) = \int_\Omega ||\nabla u||_1 d\Omega + \frac{\lambda}{2} \int_\Omega ||f - u||_2^2 d\Omega \tag{3.23}$$

However, as only f and w are retained during fusion, this data term may fail to properly capture the observed probability distribution—thus causing the optimiser to converge upon the wrong solution.

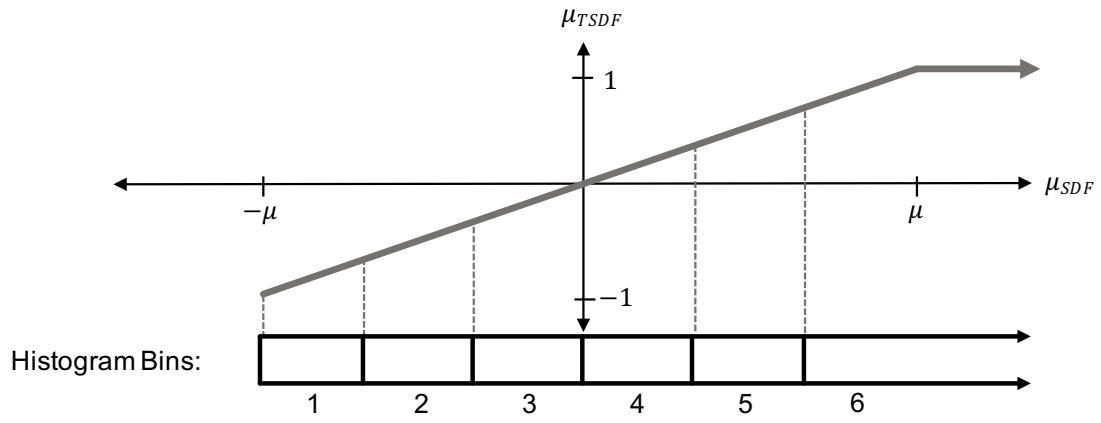


Figure 3.10: Visual depiction of the relationship between the TSDF, SDF, and histogram bins storage methods for voxel distance values. TSDF has traditionally been the favoured approach, but SDF is useful when fusing multiple sensor modalities so that each sensor has a different μ based on its precision. The histogram approach is useful to aid the regulariser by directly storing a PDF (vs. the weighted average stored by SDF/TSDF) of noisy surface measurements.

3.2.3 Histogram Data Term

In general, the more meaningful information provided to an optimiser, the better the final results. The SDF data-term method summarised all depth measurements for a voxel as two numbers: f and w. At the other extreme, one could store the entire set of N observed SDFs within each voxel [4],

$$E(u) = \int_{\Omega} ||\nabla u||_1 d\Omega + \frac{\lambda}{2} \int_{\Omega} \frac{1}{N} \sum_{t=1}^{N} ||f_t - u||_1 d\Omega \tag{3.24}$$

The main drawback with this approach is that we cannot just sequentially update a single f and w when a new depth map arrives. Instead, all previous u_{SDF} values must be stored in each voxel. This greatly limits the number of depth maps that can be fused due to memory constraints.

Rather than explicitly storing the entire measurement history or vital statistics, a more practical approach is to store depth measurements as a histogram representing the Probability Density Function (PDF) for that voxel [3]. This histogram approach is desirable since it uses a fixed amount of memory.

With a histogram representation, each voxel contains an array with n_{bins} elements where each element, h_b, stores the number of depth observations within the b^{th} histogram bin. The fusion process described in the previous section is thus augmented by linearly scale-and-clamp f in the interval $[-1, 1)$. The relationship between SDF, TSDF, and histogram bins are graphically depicted in Figure 3.10 on the facing page.

The energy minimisation becomes,

$$E(u) = \int_{\Omega} ||\nabla u||_1 d\Omega + \frac{\lambda}{2} \int_{\Omega} \sum_{b=1}^{n_{bins}} h_b ||c_b - u||_1 d\Omega \tag{3.25}$$

where the centre of the bins are computed as,

$$c_b = \frac{2b}{n_{bins}} - 1 \tag{3.26}$$

This method has been demonstrated in [3] on small-scale, object-centred environments where the final reconstruction was within a millimetre of the ground-truth laser scan with 99% completeness.

3.3 Implementation

In this section, we describe the specific algorithm to solve Equations 3.23 on page 57 and Equation 3.25. With the most relevant fundamentals of gradient descent and the LF Transformation discussed in Section 3.1.2 on page 48 and Section 3.1.3 on page 49, we point the reader to [78][77][76][72] for a more formal derivation of these steps. We primarily vary from their methods in our new definition for the gradient and divergence operators, as described in Section 3.5 on page 65.

3.3.1 SDF Optimisation Implementation

Equation 3.23 on page 57 is not smooth so it cannot be minimized with traditional techniques. As discussed in Section 3.2 on page 54, we convert the TV term to a differentiable form via the LF Transform [76][77] to transform our TV cost-function term into:

$$\begin{aligned}\underset{u\in\mathbb{R}}{\operatorname{argmin}}\,||\nabla u||_1 &= \underset{u\in\mathbb{R}}{\operatorname{argmin}}\,\underset{u\in\mathbb{R}}{\operatorname{argsup}}\left\{\int_\Omega <\nabla u, \mathbf{p}> - E^*(\mathbf{p})d\Omega\right\} \\ &= \underset{u\in\mathbb{R}}{\operatorname{argmin}}\,\underset{||\mathbf{p}||_\infty\leq 1}{\operatorname{argmax}}\left\{\int_\Omega <\nabla u, \mathbf{p}> d\Omega\right\} \\ &= \underset{u\in\mathbb{R}}{\operatorname{argmin}}\,\underset{||\mathbf{p}||_\infty\leq 1}{\operatorname{argmax}}\left\{\int_\Omega u\nabla\cdot\mathbf{p}\, d\Omega\right\}\end{aligned} \tag{3.27}$$

where the primal scalar u is the current denoised and interpolated SDF solution and $\nabla\cdot\mathbf{p}$ is the divergence of the dual vector field, $\nabla\cdot\mathbf{p} = \nabla p_x + \nabla p_y + \nabla p_z$. Substituted into Equation 3.23 on page 57, it becomes a saddle-point problem to maximise the new dual variable $\mathbf{p}$ while minimizing the original primal variable u,

$$\underset{u}{\operatorname{argmin}}\,\underset{||\mathbf{p}||_\infty\leq 1}{\operatorname{argmax}}\int_\Omega u\nabla\cdot\mathbf{p} + \lambda\int_\Omega ||f-u||_2^2\, d\Omega \tag{3.28}$$

Equation 3.28 can be efficiently solved via a Primal-Dual (Section 3.1.3 on page 49) optimization algorithm [77], as discussed in Section 3.1.3 on page 49. A more formal derivation of the below TV implementation may be found in [72][79], however we summarise the steps required as follows:

1. $\mathbf{p}$, u, and $\hat{u}$ are initialised to 0. $\hat{u}$ is a temporary variable that reduces the number of optimization iterations required to converge.

2. To solve the maximisation, update the dual variable $\mathbf{p}$,

$$\begin{aligned}\mathbf{p}_k &= \frac{\tilde{\mathbf{p}}}{\max(1, ||\tilde{\mathbf{p}}||_2)} \\ \tilde{\mathbf{p}} &= \mathbf{p}_{k-1} + \sigma_p\nabla\hat{u}\end{aligned} \tag{3.29}$$

   where σ_p is the dual variable's projected gradient ascent step size and the denominator of $\mathbf{p}_k$ enforces $||\mathbf{p}||_\infty \in [-1, 1]$ (see Figure 3.7 on page 52).

3. Then update u to minimize the primal variable,

$$\begin{aligned}u_k &= \frac{\tilde{u} + \tau\lambda w f}{1 + \tau\lambda w} \\ \tilde{u} &= u_{k-1} - \tau\nabla\cdot\mathbf{p}\end{aligned} \tag{3.30}$$

where τ is the gradient descent step size and w is the weight of the f SDF value.

4. Finally, the energy converges in fewer iterations with a "relaxation" step,

$$\hat{u} = u + \theta(u - \hat{u}) \tag{3.31}$$

where θ is a parameter to adjust the relaxation step size.

Repeat steps 2 - 4 until the energy of the system converges.

3.3.2 Histogram Optimisation Implementation

The implementation to minimise the histogram-based approach closely mirrors the implementation described Section 3.3.1 on page 59. The only difference is the primal update (Equation 3.30 on the facing page) becomes [80],

$$\begin{aligned} u_k &= \text{median}(c_1, ..., c_{n_{bins}}, b_0, ..., b_{n_{bins}}) \\ \tilde{u} &= u_k - \tau \nabla \cdot \mathbf{p} \\ b_i &= \tilde{u} + \tau \lambda W_i \\ W_i &= -\sum_{j=1}^{i} h_j + \sum_{j=i+1}^{n_{bins}} h_j \\ i &\in [0, n_{bins}] \end{aligned} \tag{3.32}$$

where, u_k is the primal solution at the k^{th} optimisation iteration, and W_i and b_i are the optimal weight and gradient-ascent solution for the i^{th} histogram bin.

For both the SDF and histogram optimisation implementations, the operations in each voxel are independent. Therefore, our implementations leverage massively-parallel GPU processing with careful synchronisation between subsequent primal and dual variable updates.

3.3.3 Data Structures

As a summary of the key data tracked during the data fusion and 3D regularisation stages our pipeline, we define the follow three voxel datatypes:

Listing 3.1: Voxel Data Structures

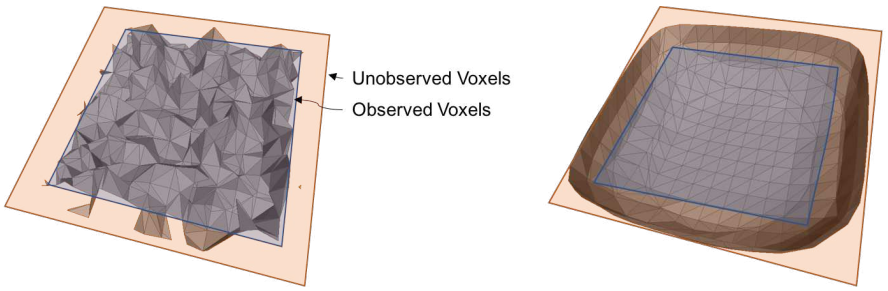
```
struct VoxelSdf {
  float sdf;
  uint8_t weight;
  uint8_t colour[3];
  bool is_omega;
};

struct VoxelHistogram {
  uint8_t histogram[N_BINS];
  uint8_t colour[3];
  bool is_omega;
};

struct VoxelTotalVariation {
  float u; // primal
  float p[3]; // dual
  float u_relax; // relaxation step
};
```

There are a few observations and forward pointers about these data structures that are valuable here, but they will be addressed in more detail in later sections.

First, sensor data (stereo-camera-based depth maps or laser) is continually fused into either `VoxelSdf` or `VoxelHistogram` voxel grid. In practice, we found the laser data works best with `VoxelSdf` and the camera-based depth maps work well (in the tested scenarios) with either `VoxelSdf` or `VoxelHistogram`. Section 4.3 on page 92 provides a more detailed analysis on why this is the case.

Second, depth-map fusion `colour` is read directly from the RGB reference image. When fusing lidar scans, the `colour` is the grayscale reflectance intensity value. If *both* lidar and depth images are fused into the same voxel grid, colour data always takes priority over grayscale data as to make the final model visually appealing (i.e., surfaces viewed by the camera are coloured correctly while surfaces viewed only by the laser are grayscale). Again, Section 4.3 on page 92 provides a variety of examples showing reconstructions using these techniques.

Third, the `is_omega` indicator variable in the above data structures is discussed in Section 3.4 on the next page. This variable is used to restrict our regulariser to regions in which surfaces were directly observed by the camera.



(a) Noisy Input Plane (b) Naïve Regularised Plane

Figure 3.11: Regulariser Surface Extrapolation Problem. The input reconstruction consists solely of a noisy plane around which there is a border of unobserved voxels in the voxel grid. The naïve regulariser (poorly) attempts to extrapolate surface data in these unobserved regions, resulting in a bowl-shaped solution. In real-world experiments, we found the extrapolated surfaces permeated the final solution where eventually non-existent ceilings and walls were added around roads in outdoor scenarios.

Finally, once all data is fused, the energy minimisation data is stored in a voxel grid of type `VoxelTotalVariation`. The final surface may be extracted directly from the `VoxelTotalVariation` voxel grid via Marching Cubes [62] or Ray Casting [81] to solve for the zero-crossing level set of $\boldsymbol{u}$. See Section 2.4.3 on page 36 for more details.

3.4 A Problem: Surface Extrapolation

Unfortunately, naïvely implementing the TV regularisation steps outlined in the previous section does not work as expected. We illustrate this via the simple synthetic example in Figure 3.11. A single, noisy depth map is fused into a voxel grid, however the depth map utilises only a subset of the available voxels (i.e., the plane is smaller than the voxel grid's dimensions). Instead of creating a denoised plane as expected, the optimiser converges upon a bowl-shaped surface. Why does this occur?

To understand the mechanics of this phenomenon, let us examine the even simpler 2D regularisation problem in Figure 3.12 on the following page. Given an input image, we occlude it in two different ways: a small interior section and then larger exterior sections. We then apply a regulariser to both of these occluded images. The regulariser sensibly *interpolates* the interior occluded region, however it struggles to *extrapolate* pixels in the exterior regions. It is the gradient operator of the regulariser that propagates data into these unobserved regions.

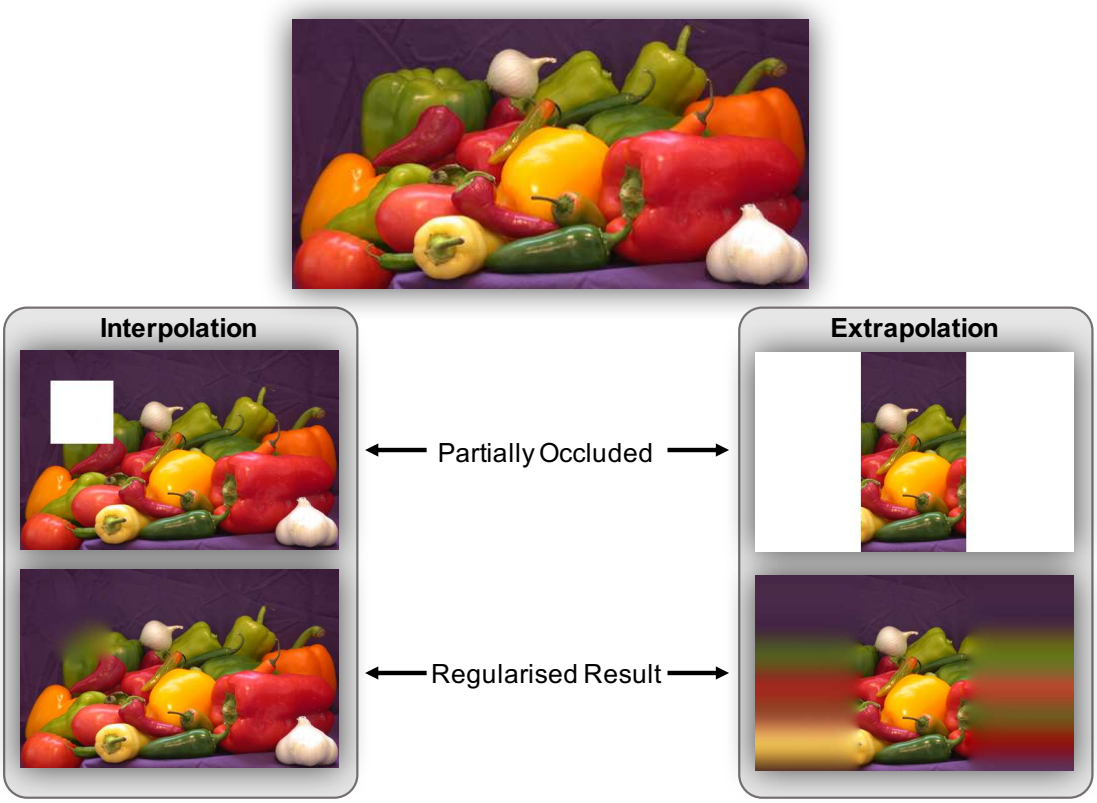
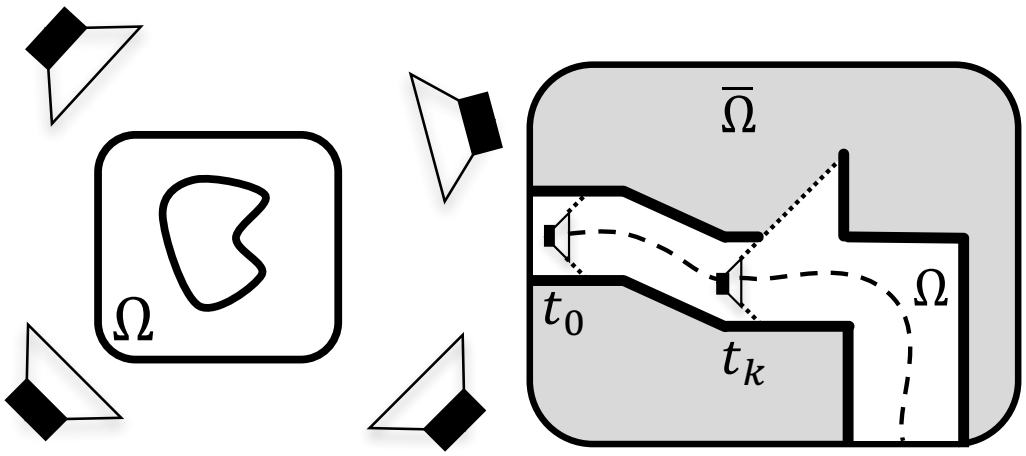


Figure 3.12: 2D Interpolation vs. Extrapolation. The above illustrates why a 3D regulariser produces unexpected results when processing noisy voxel grid data. The original image above represents the ground truth that is only partially observed in two instances. When the interior portion of the image is unobserved, the regulariser *interpolates* reasonable results based on surrounding neighbourhood data. However, when larger regions are unobserved and not wholly surrounded by valid image data, the regulariser performs poorly since it must *extrapolate*. A similar effect is found in our 3D voxel grid when we attempt to regularise after the depth sensor only directly observes a subset of the voxels.

This *extrapolation* is exactly what was happening in the 3D example in Figure 3.11 on the previous page. Since the plane occupied only a subset of the interior voxels, the regulariser—that operates over the entire voxel grid—effectively seeks to extrapolate surface data in the border regions. Therefore, spurious surfaces are generated in areas of the voxel grid in which no depth data is fused (i.e., regions that were not observed by the camera).

A more representative example is illustrated in Figure 3.13. 3D regularisation of dense reconstructions traditionally operates under the assumption that a voxel grid is fully observed by the sensor. This usually takes the form of the researcher



(a) Object-Centreed Fusion (b) Mobile-Robot-Centreed Fusion

Figure 3.13: Conventional voxel-grid-based reconstructions focus on object-centred applications where the objects are fully observed multiple times from various angles. Even though the internal portion of the object has not been observed, previous regularisation techniques do not make a distinction between Ω (observed regions) and $\bar{\Omega}$ (unobserved regions). However, in mobile-robotics applications the world environment is traversed and observed during exploration, inevitably resulting in never-observed voxels. For example, at camera capture t_x, it is unknown what exists in the camera's upper field of view. Not accounting for $\bar{\Omega}$ in regularisation results in incorrect surface generation.

manually selecting the position of the voxel grid such that it perfectly encloses the object they seek to reconstruct. However, in mobile-robotics applications, we neither know our path nor the locations of observed surfaces prior to data collection. Voxel grids therefore will only contain a subset of voxels that are directly observed. We must restrict the regulariser to operate solely upon these observed voxels to prevent the regulariser from creating spurious surfaces.

3.5 The Solution: Boundary Conditions (Ω Labels)

Since we are moving at an *a priori*-unknown trajectory through the world, we only observe surfaces in a subset of all allocated voxels. In this section we present a new technique to prevent the unobserved voxels from negatively affecting the regularisation results of the observed voxels — i.e., to ensure the regulariser does not extrapolate into unobserved regions. To achieve this, as illustrated in Figure 3.13, we define the complete voxel grid domain as Λ and use Ω to represent the subset

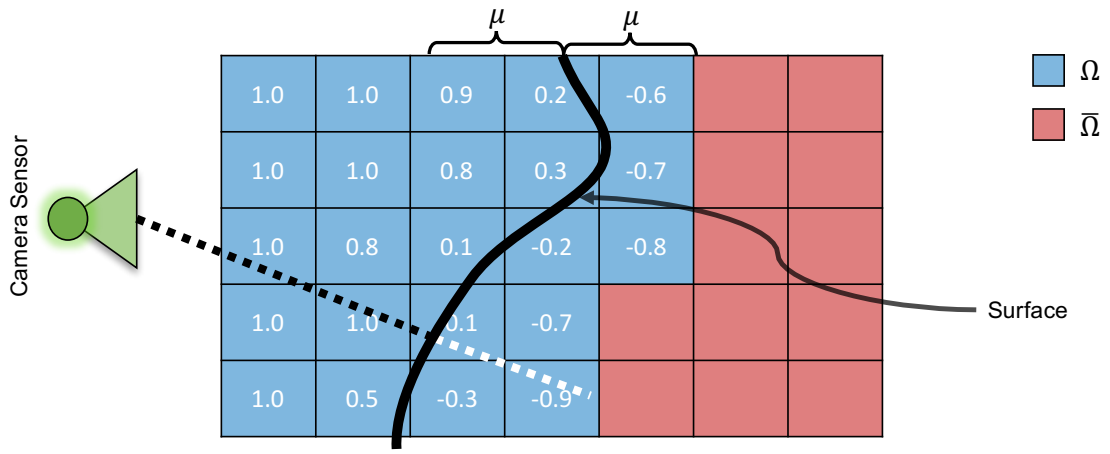


Figure 3.14: A graphical depiction of how the Ω label is utilised in a two-dimensional 'voxel' grid. The TSDF values represent the zero-crossing surface (black line). All voxels with valid TSDF values are part in Ω (blue voxels), and all remaining voxels are in $\bar{\Omega}$ (red voxels). These TSDF values $f \in [-1, 1]$ are a linear mapping from the signed distance a surface is from a given voxel centroid. There is no f value when the signed distance is less than $-\mu$, however when the signed distance is greater than μ (i.e., observed free space in front of a surface) all those voxels are updated with $f = 1$.

of voxels that have been directly observed and therefore will be regularised. The remaining subset, $\bar{\Omega}$, represents voxels that have never been observed. By definition, Ω and $\bar{\Omega}$ form a partition of Λ and therefore $\Lambda = \Omega \cup \bar{\Omega}$ and $\Omega \cap \bar{\Omega} = \varnothing$.

To the authors' knowledge, all prior works rely on a fully-observed conventional voxel grid before regularisation and they implicitly assume that $\Lambda = \Omega$ — i.e., every voxel in the voxel grid is directly observed by the sensor. This assumption is not valid in non-object-centric reconstruction applications. The collection platform's motion results in unobserved regions caused by object occlusion, field-of-view limitations, and trajectory decisions. Therefore, $\Omega \subset \Lambda$ as Figure 3.13 on the previous page illustrates. In practice, as discussed in Section 3.4 on page 63, we found failure to account for the Ω-$\bar{\Omega}$ boundaries causes the regulariser to spuriously extrapolate surfaces into undesired regions of the reconstruction.

Our approach introduces a new state variable, $\mathbf{1}_\Omega : \Lambda \to \{0, 1\}$, in each voxel indicating whether or not it was directly observed by a range sensor. Ω is therefore the set solely upon which the regulariser is constrained to operate, thus avoiding spurious surface generation in non-valid voxels ($\bar{\Omega}$). A graphical depiction of

how we augmented the voxel grid data structure with Ω and $\bar{\Omega}$ is provided in Figure 3.14 on the preceding page.

With the Ω label annotation in the voxel grid, we define the gradient term of the regulariser to enforce the observed-unobserved boundary conditions,

$$\nabla_x u_{i,j,k} = \begin{cases} u_{i+1,j,k} - u_{i,j,k} & \text{if } 1 \leq i < V_x \\ 0 & \text{if } i = V_x \\ 0 & \text{if } u_{i,j,k} \in \bar{\Omega} \\ 0 & \text{if } u_{i+1,j,k} \in \bar{\Omega} \end{cases} \tag{3.33}$$

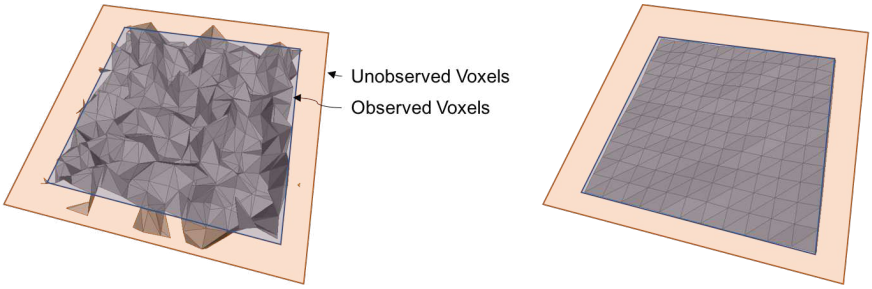
where $u_{i,j,k}$ is a voxel's SDF value at the 3D world integer coordinates (i, j, k), and V_x is the number of voxels in the x dimension. The gradient term in the regulariser encourages smoothness across neighbouring voxels. This explains why this new gradient definition excludes $\bar{\Omega}$ voxels — they have not been observed.

To solve the Primal-Dual optimisation, we must also define the corresponding divergence operator:

$$(\nabla \cdot \mathbf{p}_{i,j,k})^x = \begin{cases} \mathbf{p}^x_{i,j,k} - \mathbf{p}^x_{i-1,j,k} & \text{if } 1 < i < V_x \\ \mathbf{p}^x_{i,j,k} & \text{if } i = 1 \\ -\mathbf{p}^x_{i-1,j,k} & \text{if } i = V_x \\ 0 & \text{if } u_{i,j,k} \in \bar{\Omega} \\ \mathbf{p}^x_{i,j,k} & \text{if } u_{i-1,j,k} \in \bar{\Omega} \\ -\mathbf{p}^x_{i-1,j,k} & \text{if } u_{i+1,j,k} \in \bar{\Omega} \end{cases} \tag{3.34}$$

The regulariser operates only on the voxels within Ω, the domain of integration, and thus it neither spreads spurious surfaces into unobserved regions nor updates valid voxels with invalid SDF data.

Both equations are presented for the x-dimension, but the y and z-dimension equations can be obtained by variable substitution between i, j, and k. In this section we assumed the Ω label is a binary indicator with w (Section 2.4.2 on page 34) used to weight the confidence in the data term, however future work may expand upon this to vary the regulariser with a more sophisticated function—for example, a linear scale-and-threshold based on the number of observations.



(a) Noisy Input Plane

(b) Omega-Domain Regularised Plane

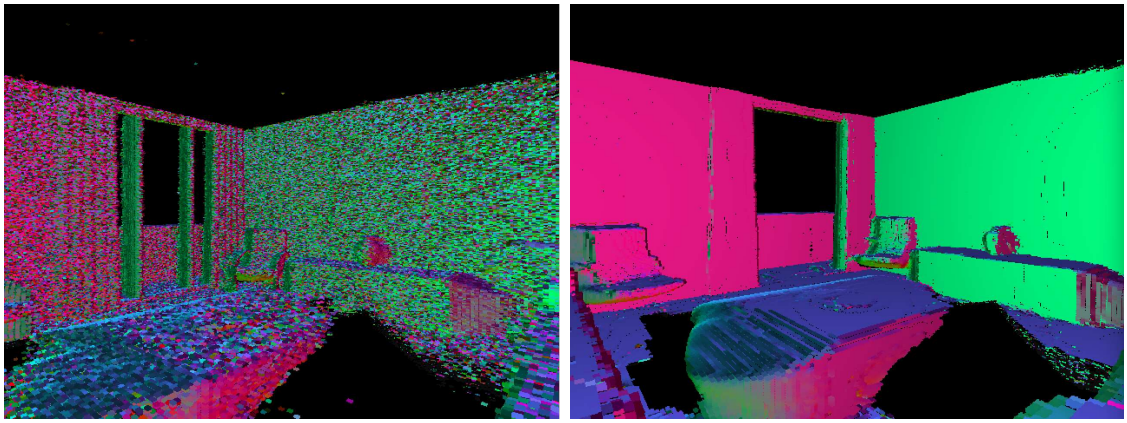
Figure 3.15: Surface Extrapolation Fixed Via Ω Labels. After applying Ω labels to account for the observed-unobserved boundaries in the voxel grid, the TV optimiser converges upon a more reasonable solution. Contrast this solution with the naïve regulariser's bowl-shaped solution in Figure 3.11 on page 63.

After using these new gradient and divergence definitions for the implementation from Section 3.3 on page 59, we again applied the regulariser to the original noisy-plane example. As shown in Figure 3.15, the Ω-labelling regulariser correctly smooths out the noisy surface and produces a flat plane. Having passed this simple test, we next evaluate Ω labelling on more rigorous datasets to ensure it improves large-scale, real-world reconstructions.

3.6 Results

In the previous section the Ω labelling concept was introduced and validated on a simple synthetic reconstruction (Figure 3.15). This section further validates the performance Ω labelling via a *qualitative* analysis on the public ICL-NUIM RGBD benchmark dataset [82] and a *quantitative* analysis with a real-world data collection from the University of Oxford's RobotCar platform [27]. All experiments were completed utilising an NVIDIA GeForce GTX TITAN (6 GB) GPU.

Our analysis compares two approaches to generating dense reconstructions. First, we fuse (Section 2.4.2 on page 34) the sequence of depth maps into a conventional voxel grid. Since depth maps from monocular cameras are subject to significant noise, we use the histogram ($n_{bins} = 5$) data term to store the PDF of TSDF values observed in each voxel. We refer to the resulting noisy model as the



(a) Curless Fused Reconstruction (b) Regularised Result

Figure 3.16: Comparison of the effect regularisation has on surface normals between a 'raw' Curless (a) and regularised (b) reconstruction of a synthetic environment [82]. The Phong-shading illustrates the Omega-domain-enabled TV regulariser smooths out the surface normals without inadvertently, spurious surface creation. The input depth maps are corrupted with σ_n = 10 cm Guassian noise.

'Curless' reconstruction, after the author who first proposed the TSDF data-fusion function [54]. Second, for the Building Optimal Regularised Reconstructions with GPUs (BOR^{2}G) reconstruction model, we apply our TV regulariser to the voxel grid to smooth out the noisy surfaces by enforcing our piecewise-constant assumption. We find that $\boldsymbol{\Omega}$ labelling enables the TV regulariser to reduce surface-normal noise while improving the metric quality of the reconstruction.

3.6.1 Imperial College ICL-NUIM Dataset

The ICL-NUIM dataset was released in 2014 to provide a benchmark to evaluate the performance of RGB-D systems with high-quality indoor ray-traced, synthetic sequences. Ground-truth camera trajectories, depth maps, and environment models are provided to enable evaluation. We use the "Living Room 'lr ktw"' sequence that consists of 882 images, captured at 30 Hz with a resolution of 640 × 480. Large sections of the floor and ceiling are unobserved throughout the camera trajectory, so this is an ideal dataset to evaluate whether or not the $\boldsymbol{\Omega}$ labelling concept works correctly—i.e., no unobserved surfaces are interpolated or extrapolated.

To perform a more robust qualitative evaluation than that provided in Figure 3.15, we add noise ($\mathcal{N}(0, 0.1)$) to the depth maps before fusing them into the voxel grid and then regularising.

As shown via the Phong shading of the resulting reconstructions in Figure 3.16 on the previous page, the 'raw' Curless (i.e., averaging observed TSDF values in each voxel) produces noisy surface normals in the reconstruction. After applying the TV regulariser, the surface normals align to the proper underlying observed 3D structure. Also note, by comparing figures (a) to (b), the $\mathbf{\Omega}$ labelling worked as intended since no surfaces are extrapolated or interpolated.

3.6.2 Oxford Datasets

The University of Oxford RobotCar dataset consists of approximately 1,000 km of GPS/INS, monocular camera, stereo camera, and laser data captured over the course of a year in an urban environment [27]. We utilised the vehicle to capture data in Woodstock, UK. A similar set of sensors (excluding the GPS/INS sensor) were mounted to an smaller mobile-robotics platform to capture an indoor dataset in the University of Oxford's Acland building.

We captured a sequences of images at 20 Hz with 1280 × 960 (Woodstock) and 1032 × 776 (Acland) resolution via a forward-facing monocular camera. To prevent scale drift, improve metric accuracy, and help isolate this analysis to the performance of the fusion/regularisation modules, camera poses were computed by VO on a stereo camera.

Table 3.1: Timing Results of BOR^{2}G *regularisation* on an NVIDIA GeForce GTX TITAN graphics card with 512^3 voxel grids. For the configuration parameters, only the volume's dimension changed, but the number of voxels (and hence memory requirements) remained consistent between experiments.

Experiment	Volume Dim (m)	Iterations	Reg. time (s)	Memory size (MB)
Woodstock	$6 \times 25 \times 10$	100	11.09	640 MB
Acland	$4 \times 6 \times 30$	100	11.24	640 MB

Table 3.2: Error analysis comparing Curless and BOR^{2}G methods. The BOR^{2}G error is less than half that of Curless.

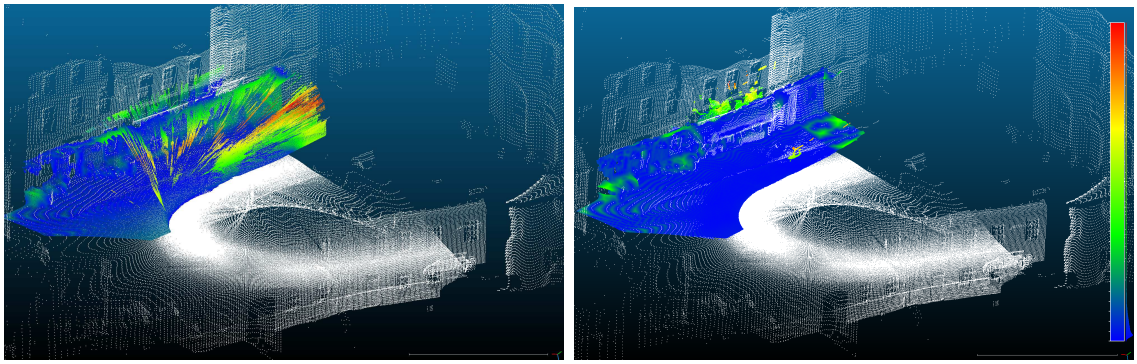
Experiment	Median Error (cm)	Standard Deviation (cm)
Woodstock (Curless)	37.30	57.08
Woodstock (BOR^{2}G)	**14.41**	**36.36**
Acland (Curless)	31.02	57.08
Acland (BOR^{2}G)	**15.08**	**35.37**

Ground-truth data was computed by composing the push-broom SICK-LMS 151 laser scans into a coherent 3D point cloud. Example ground-truth point clouds are shown in Figures 3.17 on the following page and 3.18 on page 73.

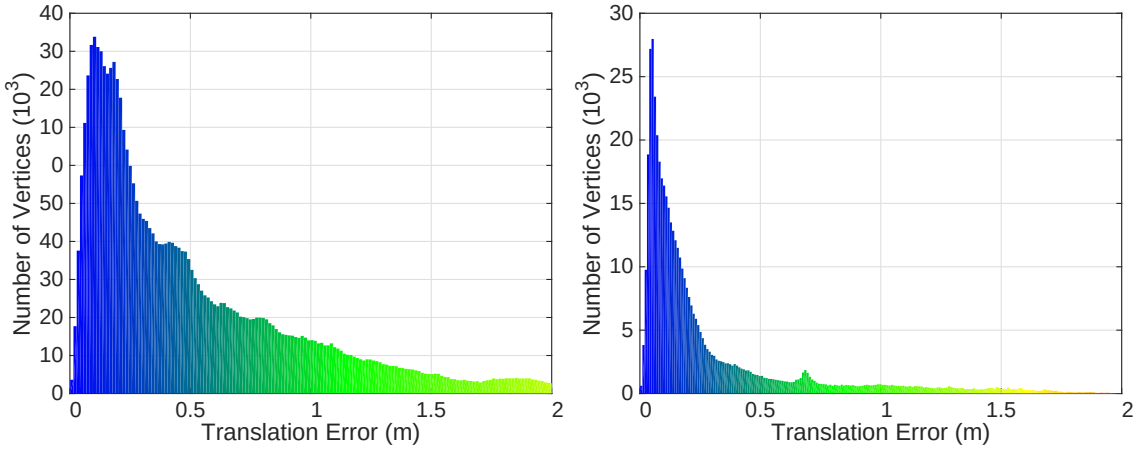
Pose estimation was processed in real time (20 Hz), depth-map estimates (1 Hz) and data fusion (5 Hz) were at interactive rates, while the regularisation could only be achieved via an off-line process since it required approximately 11 minutes to converge in each scenario (see Table 3.1 on the facing page). This is to be expected since the optimiser must, over the course of approximately 100 iterations, converge upon the minimal energy for 134,217,728 voxels (512^3). This requires significant amount of computing and memory resources, all of which are dedicated to solving the energy minimisation algorithm outlined in Section 3.3 on page 59.

Table 3.2 summarises the metric error comparison between the Curless and our BOR^{2}G methods. Figure 3.17 on the next page and Figure 3.18 on page 73 provide a visual overview of the error distributions in the resulting reconstruction while Figure 3.19 on page 74 shows the final, coloured dense reconstructions.

These results were favourable: when comparing the Curless method to BOR^{2}G, our TV regulariser reduced the median metric error by 61% (from 37.30 cm to 14.41 cm) in the outdoor Woodstock dataset and 51% (from 31.02 cm to 15.08 cm) in the indoor Acland dataset. These are small but precise reconstructions; in Chapter 4 we will address scale.



(a) Woodstock Data Set: Comparison of Point Clouds. The Curless implementation (left) produced a large range of spurious data points when compared to our BOR^{2}G method (right). The white vertices are truth data and the colour vertices correspond to the histogram bins in (b).

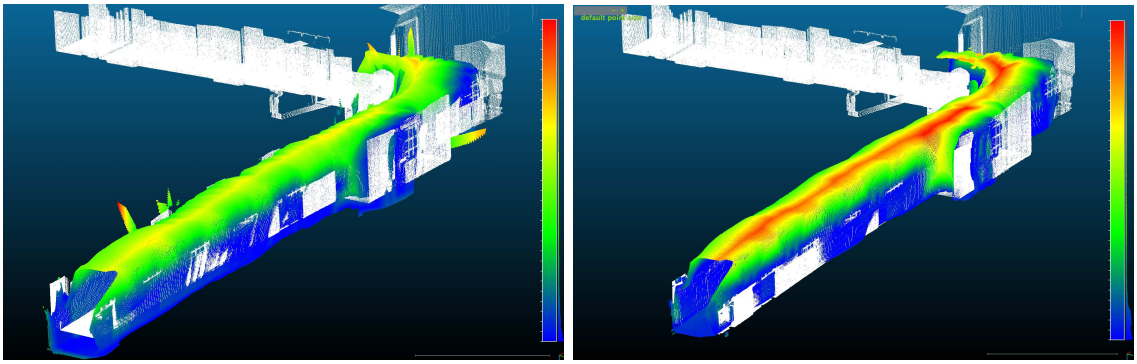


(b) Woodstock Data Set: Histograms of per-vertex-error when compared to laser-generated point clouds. The Curless (left) has a median error of 37.3 cm (σ = 57.1 cm) while our BOR^{2}G (right) method has a median error of 14.4 cm (σ = 3.64 cm). Note that the BOR^{2}G method requires fewer vertices to represent the same scene.

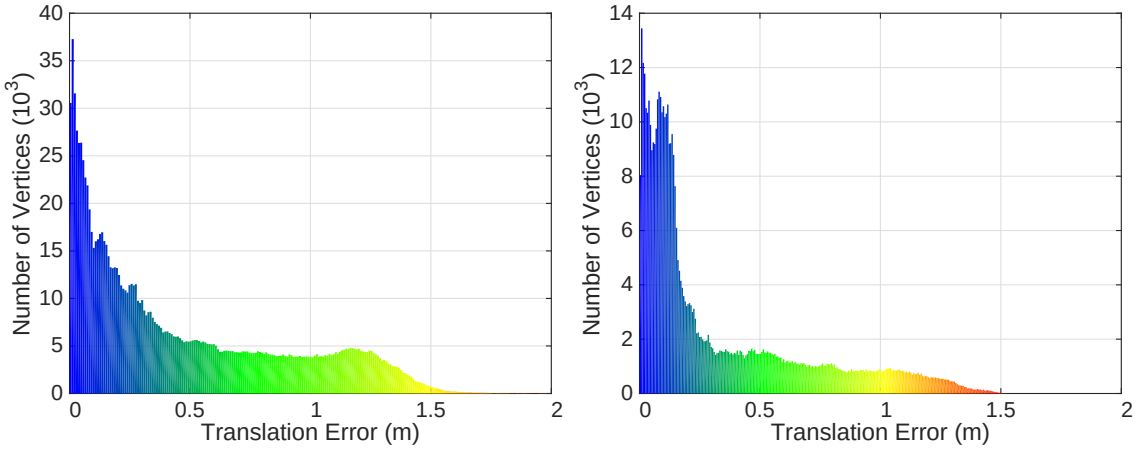
Figure 3.17: Woodstock Dataset: Comparison of the Curless (left) and BOR^{2}G (right) dense reconstruction techniques. The Curless has a larger number spurious outlier segments and requires more than twice the number vertices to represent the structure due to its irregular surfaces. The BOR^{2}G method's median error is approximately half that of the Curless method.

3.7 Conclusions

In this chapter we presented the theory and system required to regularise noisy depth data from real-world environments. The key challenge to overcome was that a naïve 3D regulariser operates on all voxels, whereas mobile-robotics collections result in a voxel grid containing only a subset of voxels with directly observed surface data. Ignoring this fact results in the generation of spurious surfaces.



(a) Acland Data Set: Comparison of Point Clouds. The BOR^{2}G (right) method again outperformed the Curless implementation (left). The white vertices are truth data and the colour vertices correspond to the histogram bins in (b).



(b) Acland Data Set: Histograms of per-vertex-error when compared to laser-generated point clouds. The Curless (left) has a median error of 31.0 cm (σ = 57.1 cm) while our BOR^{2}G (right) method had a median error of 15.1 cm (σ = 35.4 cm). Note that the BOR^{2}G method requires fewer vertices to represent the same scene.

Figure 3.18: Acland Data Set: Comparison of the Curless (left) and BOR^{2}G (right) dense reconstruction techniques. Note that the laser truth data was only measured depth data for the lower-half of the hallway. This results in the spurious errors for the upper-half where our depth maps produced estimates but for which there was no truth data. These errors dominate the right tail of the histograms in (b). As with the Woodstock data set, the BOR^{2}G method's median and standard deviation are approximately half that of the Curless method.

By introducing the new $\boldsymbol{\Omega}$ labelling concept and restricting our regularisation to operate only upon voxels with valid SDF data, we successfully demonstrated a TV regulariser reduces the noise in both synthetic and real-world datasets. In the synthetic datasets, the regulariser noticeably smoothed out all surfaces and aligned the surface normals. In the real-world datasets, the regulariser reduced the metric error (when compared to the raw, noisy, fused reconstruction) by 61%

(a) Acland Reconstruction (b) Woodstock Reconstruction

Figure 3.19: The final dense 3D reconstructions of the experiments using BOR^{2}G with the Acland building (a) and Woodstock, UK (b). In addition to the lack of spurious surfaces, note the high level of detail achieved with the forward-facing monocular camera in both indoor and outdoor environments.

in outdoor environments and 51% in indoor environments, with final respective median errors of 14.41 cm and 15.08 cm.

In the next chapter, we expand upon the **Ω**-labelling concept to build a complete dense reconstruction system that operates in large-scale, autonomous-driving environment with multiple sensor-modality inputs.

The foremost cartographers of the land have prepared this for you; it's a map of the area that you'll be traversing. [The map is blank.] They'll be very grateful if you could just fill it in as you go along.

— Blackadder, *Potato (Blackadder II)*

4

Large-Scale, Multi-Sensor Reconstructions

Abstract

We extend the methodology of Chapter 3 to utilise a *compressed* 3D data structure that enables large-scale scene capture. Our BOR^{2}G system fuses data from multiple sensor modalities (*stereo* cameras, lidars, or both) and regularises the resulting 3D model. Because of the paucity of surface observations by the camera and lidar sensors, we again regularise both the 2D (camera depth maps) and 3D (voxel grid) to provide a local contextual prior for the reconstruction.

We evaluate our system using the Stanford Burghers of Calais, Oxford Broad Street (released with this thesis), and the KITTI datasets. We provide metric-error statistics by comparing our surface models to 3D lidar scans.

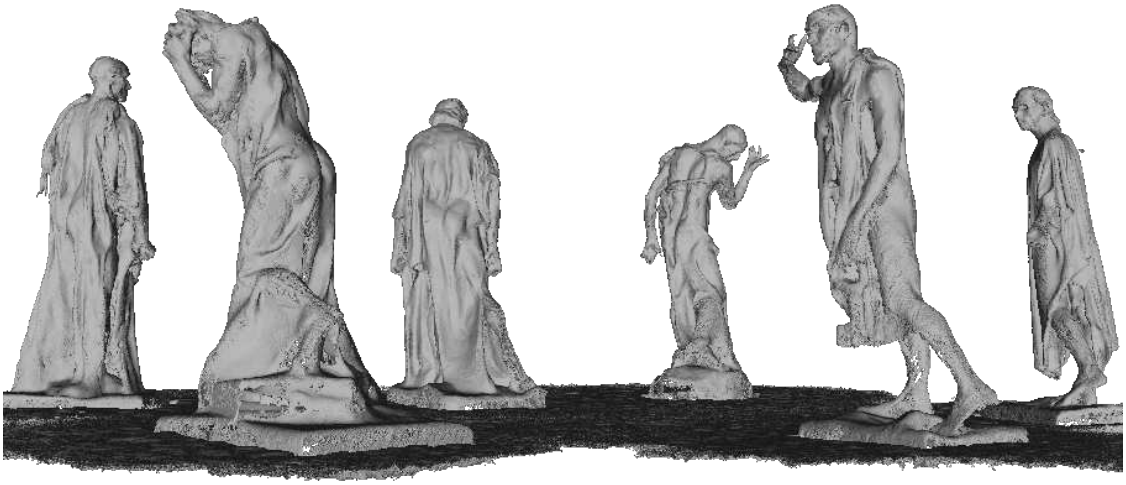
Our regulariser reduces the median error between 27% to 36% in 7.3 km of dense reconstructions with a median accuracy between 4 cm to 8 cm. Qualitatively, fusing both stereo-camera-based depth images with lidar data produces reconstructions that are both higher quality and more comprehensive than either sensor achieves independently.

Contents

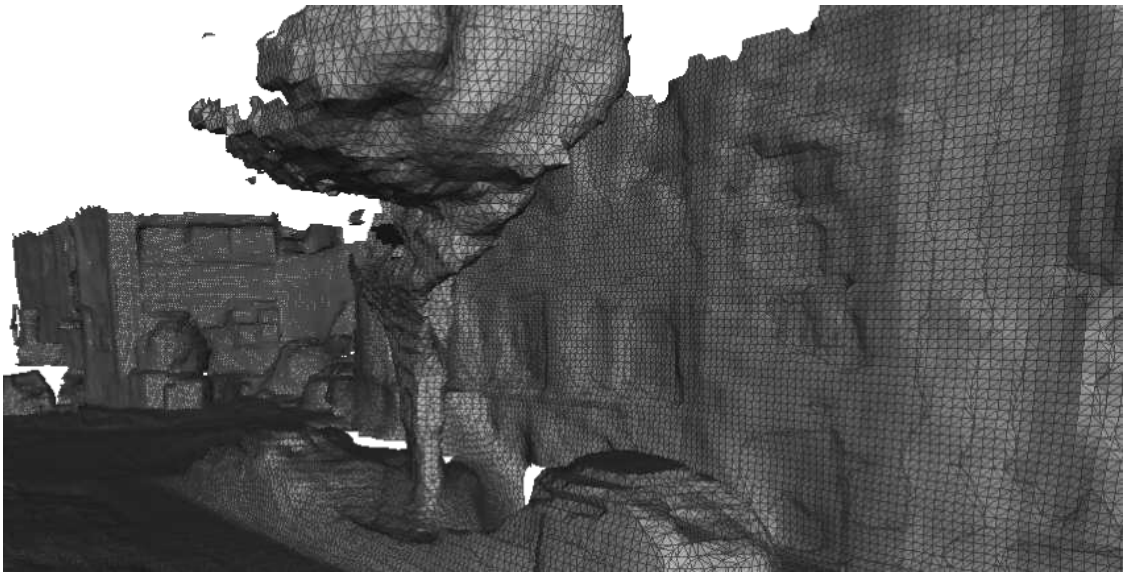
This chapter is about the efficient generation of dense, coloured models of very-large-scale environments from stereo cameras, laser data, or a combination thereof. These models are used for a myriad of autonomous-driving applications– including visualisation, localisation, planning, segmentation, and distraction suppression. However, the computational and memory requirements of dense 3D models have limited their use in real-world, large-scale environments.

Dense 3D reconstruction research has significantly increased over the past half decade because of improvements in computer hardware and mathematical theory. Modern commodity GPUs support massive parallel processing that was only previously available in super-computer environments. When a problem is framed correctly, this GPGPU programming realises multiple orders of magnitude performance improvements over a pure Central Processing Unit (CPU) implementation.

Early reconstruction work focused on processing a set of unstructured images of a 3D environment [83][84][85] while leveraging traditional Structure from Motion (SfM) techniques. However, as mathematical models improved [77][86], more sophisticated optimisation and regularisation techniques became feasible to process the notoriously noisy input 2D depth maps and resulting 3D reconstructions. Unfortunately, many recent techniques focused primarily on mobile and tablet products [87][88][89][90][87] and do not scale for our large-scale mobile-robotics applications.



(a) Stanford Burghers of Calais: 155.97 Depth Images/m^2



(b) KITTI-VO 00: 0.049 Stereo Images/m^2

Figure 4.1: RGB-D vs KITTI reconstruction detail comparison. Both reconstructions were created with the BOR^{2}G pipeline, but the Stanford [66] reconstruction has over 3,100 times (Table 4.3 on page 92) more depth observations per square metre than the KITTI reconstruction. In addition, each the RGB-D depth maps are more than an order of magnitude more accurate than the stereo-camera-generated depth estimates. This chapter is focused on creating high quality reconstructions that gracefully handle KITTI-level low observation densities over kilometres of observations, but also works on the conventional RGB-D use cases. In addition, our method accepts sensor input from RGB-D, stereo images, lidar, or a combination thereof. When compared to the model provided by [66], the above BOR^{2}G Burghers reconstruction has a median point-to-surface difference of 0.5 cm. Section 4.3 on page 92 provides a more detailed quantitative analysis of our reconstruction precision.

Some researchers have worked with data and sensors more applicable for autonomous vehicles. Most notable, in 2010 Google released an academic article detailing their "Street View" application that utilises laser and camera data to create dense 3D reconstructions of cities across the world [91]. However, their algorithms over-emphasize the laser data and assume all depth maps only contain piecewise planar objects. In 2014, Xiao and Furukawa proposed a system to model large-scale *indoor* environments with laser and image inputs, but their modified Manhattan-world assumption restricted reconstructions from modelling anything other than vertical and horizontal planes [92]. Bok et al. built upon the prior state of the art to create large-scale 3D maps using camera and lasers [93]. Their final reconstructions were *sparse* and only utilised the cameras for odometry and loop closure, not as an additional depth sensor to improve the dense reconstructions.

In 2012, Stewart demonstrated a system with capabilities similar to those provided by Xiao and Furukawa's system [94][95][25]. We extensively used Stewart's city-scale, *sparse* 3D models as ground-truth data to evaluate the performance of the algorithms detailed in this thesis.

In Chapter 3, we introduced the $\boldsymbol{\Omega}$ labelling concept that allows one to target optimisation and regularisation to a focused subset of 3D space. We extend that concept in this chapter to build a dense mapping system, targeted to large-scale autonomy and scene understanding applications, that meets the following requirements:

***R1**: Operate in multiple-kilometres-scale environments*

***R2**: No range limitations for input sensor observations*

***R3**: Cope with a paucity of surface observations*

***R4**: Fuse data from multiple sensor modalities*

A survey of the literature found no systems currently exist that meet these requirements.

Four seminal systems (Table 4.1 on the facing page) in this area are DTAM by Newcombe et al. [44]; Kinect Fusion by Newcombe et al. [2]; Kintinuous by Whelan et al. [55]; and Voxel Hashing by Nießner et al. [59].

Before RGB-D cameras were widely available, DTAM presented a method to produce high-quality depth maps with a monocular camera. A cost volume is constructed in front of the camera's focal plane and continually updated with 2D-regularised depth estimates from sequential image frames. The final reconstructions provide fine details, but this system limits the range of the monocular camera to near-field reconstructions.

In 2010, the first commodity RGB-D camera was released. RGB-D cameras provide a centimetre-level-accurate depth measurement for each pixel in the image — 640×480 resolution at 30 Hz in this first device. Newcombe et al. extended their work via the Kinect Fusion system to take advantage of this stream of high-frequency and high-quality depth maps. Leveraging the TSDF [54], depth observations are stored in a voxel grid where each voxel contains a positive number when in front of a surface and a negative number behind the surface. Solving for the zero-value level set results in a dense model of the original surface. Thus, Kinect Fusion could generate unprecedented-quality dense 3D reconstructions in real time (i.e., at the input sensor's frame rate) for workspaces approximately 7 m^3 in size.

In contrast to Kinect Fusion where the voxel grid is fixed to one location in space, Kintinuous sought to extend the size of reconstruction scenes by allowing the voxel grid to move with the camera. A continual stream of previously-observed regions are streamed to disk as a mesh, but can be reloaded back into the GPU if that region is observed again. This system infinitely (within numerical constraints) extended the reconstruction workspace size. However, it cannot leverage sensor observations

*2D Regularisation
†2D and 3D Regularisation

Table 4.1: System Capabilities Comparison

Technique	R1: Env.	R2: Range	R3: Data	R4: Sensors
DTAM*	1 m^2	1 m	100+ img/m^2	Mono
Kinect Fusion	7 m^2	2 m	—"—	RGB-D
Kintinuous	∞	3 m	—"—	RGB-D
Voxel Hashing	∞	4 m	—"—	RGB-D
BOR^{2}G†	∞	50 m	0.05 img/m^2	Mono, Stereo or Lidar

further than 3 m from the RGB-D camera because it is still fundamentally based upon a conventional fixed-size voxel grid.

Nießner's HVG also extended the size of reconstructions but did so by only allocating voxels in regions where surfaces were observed. This removed memory wasted storing free space. When combined with streaming of data to/from the GPU and hard disk when surfaces are "far" away from the sensor, there is essentially no limit on the size of reconstructions. This implementation restricts the sensor range to 4 metres as that is near the maximum effective range of the Kinect camera, however the range can be trivially extended.

All the above techniques focused on object-centred reconstructions. In other words, the operator selected a small scene to reconstruct (e.g., desk, courtyard, etc.) and a single sensor was moved through the environment in a way that observed all surfaces multiple times from a variety of angles. This results in a fine-detailed final reconstruction, as shown in Figure 4.1 on page 77a. In autonomous driving scenarios where the sensor is mounted on a vehicle and the path is not known *a priori*, the surfaces are observed for fewer times. In fact, we found that RGB-D scenarios have over 3,100 times more depth images per square metre than our autonomous vehicle applications (Table 4.3 on page 92).

In addition, mobile-robotics sensor suites typically include lidars and forward-facing monocular or stereo cameras. Since RGB-D cameras viewing range is so short (5 metres) and their accuracy degrades outdoors, they are not useful when reconstructing an urban environment from a mobile-robotics platform. In short, mobile-robotics platforms have significantly less accurate sensors that observe surfaces far fewer times than in a typical RGB-D scenario. Therefore the reconstructions inherently have less detail, as shown in Figure 4.1 on page 77b, that requires us to apply contextual priors via a local regulariser to improve the final surface reconstructions. Table 4.1 on the preceding page, Table 4.3 on page 92, and Figure 4.1 on page 77 summarise the discussion in the preceding literature review.

Our contributions in this chapter are as follows:

1. We present in Sections 4.1 on the next page - 4.2 on page 83 the theory required to construct a state-of-the-art dense-reconstruction pipeline for mobile-robotics applications (i.e., satisfies R1-R4 previously defined in this section). Our implementation includes a compressed data structure (R1/R2) of a sensor-agnostic voxel grid (R4). Since the input data may be noisy and the observations per m^2 are very low, we utilise a regulariser in both 2D and 3D to both serve as a prior and reduce the noise in the final reconstruction (R3).
2. We present in Section 4.2 on page 83 a method to regularise 3D data stored in a compressed volumetric data structure thereby enabling regularisation of significantly larger scenes (fulfils R1 and R3). The key difficulty (and hence our vital contribution) in regularising within a compressed structure is the presence of many additional boundary conditions introduced between regions that have and have not been directly observed by the range sensor. Accurately computing the gradient and divergence operators, both of which are required to minimise the regulariser's energy, becomes a non-trivial problem. We ensure the regulariser only operates on meaningful data and doesn't extrapolate into unobserved regions.
3. We provide, by way of our quantitative results in Section 4.3 on page 92 and the models released in the supplementary materials, a new KITTI benchmark for the community to compare dense-map reconstructions. In addition, we include the ground truth (GT) data used to evaluate our system. This includes the optimised vehicle trajectory, consolidated GT point cloud, and final dense reconstructions with different sensor modalities.
4. We release (Section 4.3.2 on page 93) the 1.6 km Oxford Broad Street dataset as a real-world reconstruction scenario for a mobile-robotics platform with a variety of camera and lidar sensors.

Our hope is that our dataset and benchmarks become valuable tools of comparison within the community.

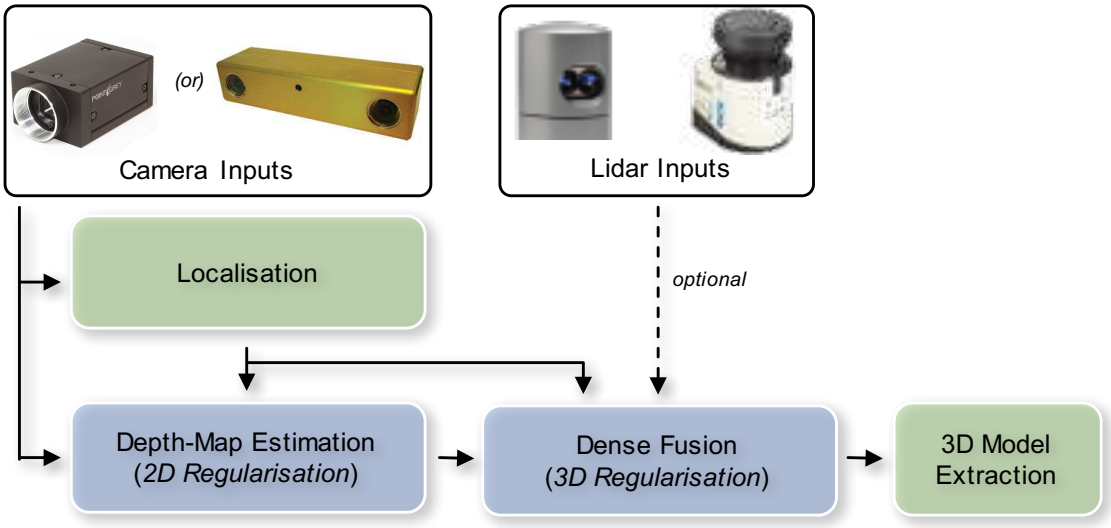


Figure 4.2: An overview of our software pipeline. Our Dense Fusion module accepts data from either laser sensors or depth maps created from mono or stereo cameras. We regularise the 3D model, extract the surface, and then provide the final 3D model to other components on our autonomous robotics platform — e.g., segmentation, localisation, planning, or visualisation.

4.1 System Overview

This section provides a brief overview of our system's architecture (Figure 4.2) before we proceed to a detailed discussion in Section 4.2 on the next page.

At its core, our system consumes a stream of stereo camera images ($\boldsymbol{I}$) or laser scans ($\boldsymbol{L}$) and produces a 3D model. The pipeline consists of a Localisation module that provides sensor pose estimates. We do not restrict the trajectory of source sensors *a priori*. In fact, we demonstrate (Section 4.3 on page 92) our method using a forward-facing stereo camera and lidar sensor on an automobile in an urban environment. The design of this module is versatile allowing us to use any SLAM implementation [96][97][98][99][100][101][102][103][88][104]; we use ORB-SLAM2 [105][106] as it is the current state-of-the-art for sparse localisation and loop closure with monocular, stereo, and RGB-D cameras.

The Depth-Map Estimation module (Section 2.3 on page 24) processes a stream of stereo or monocular frames into a stream of 2D regularised depth maps, $\boldsymbol{D}$. Because of the nature of a passive camera and the large scale at which we operate, we utilise

a sophisticated regulariser – specifically, the Total Generalised Variation (TGV) regulariser – to improve the accuracy of these depth maps.

4.2 Hashing Voxel Grid Modifications

To improve our system from the small-scale single-sensor-modality reconstructions in Chapter 3, we need to make a variety of changes to our fusion algorithm and data structures. Specifically, the methods described in this section build upon the HVG data structure discussed in Section 2.4.1.2 on page 32. First, we must articulate the algorithms to fuse both depth-map data from cameras and laser range data from lidar (Section 4.2.1). Next, in Section 4.2.2 on page 89, we extend the $\mathbf{\Omega}$ labelling construct (Chapter 3) to work with the Hashing Voxel Grid approach introduced in Section 2.4.1.2 on page 32. Finally, in Section 4.2.3 on page 90, we provide an overview of the details required to implement the optimisation with both SDF and histogram data terms.

4.2.1 Multi-Sensor Fusion

Fusing data into the voxel grid varies in method depending on whether one utilises a camera-based sensor (Section 2.4.2 on page 34) or a laser-based sensor (Section 4.2.1).

4.2.1.1 Laser Fusion

Fusing laser data into the voxel grid is accomplished via an efficient ray casting implementation [81]. Consider the case of a single laser ray ($\vec{\boldsymbol{\rho}}$) with an origin at $\boldsymbol{\rho}_o$ and a termination (i.e., where the laser reflects off of a surface) point at $\boldsymbol{\rho}_r$. $\vec{\boldsymbol{\rho}}$ is fused into the voxel grid by tracing the ray and updating each voxel with the signed distance to $\boldsymbol{\rho}_r$. Specifically, the following operations are performed on all voxels that intersect the ray from $\boldsymbol{\rho}_o$ to $\boldsymbol{\rho}_r$:

1. Calculate the voxel's global-frame centre $\mathbf{p}_g = [x_g, y_g, z_g]^T$
2. Compute the vector from the laser scan's origin to the current voxel centre: $\boldsymbol{v}_1 = \mathbf{p}_g - \boldsymbol{\rho}_o$

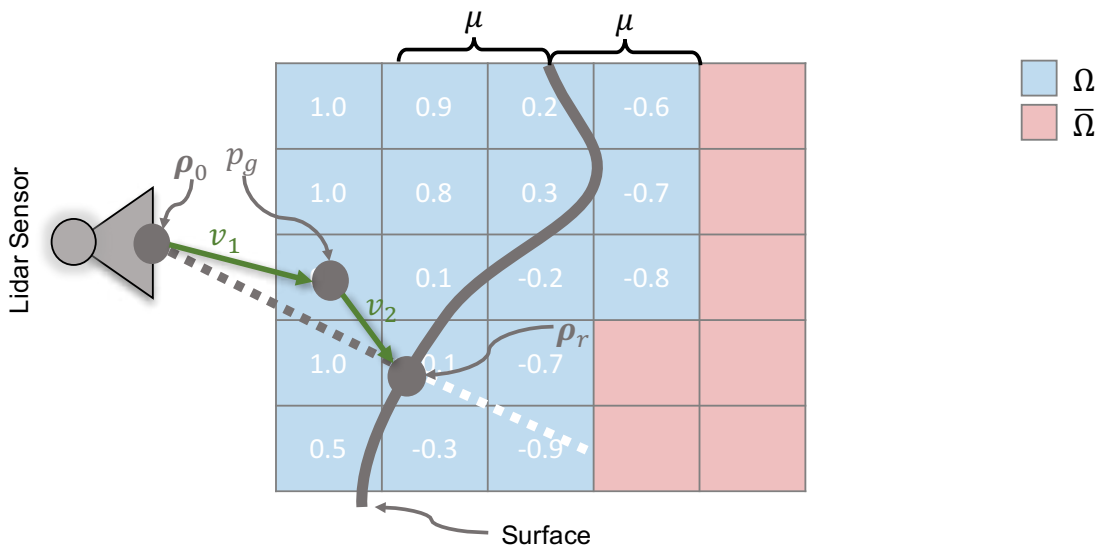


Figure 4.3: A graphical depiction of how a single ray-traced laser is fused into a two-dimensional 'voxel' grid. To compute the SDF value $\boldsymbol{\rho}$ creates in the voxel with centroid p_g, one must compute v_1 and v_2. The dot product of these vectors indicate whether the voxel is in front (positive) or behind (negative) the observed surface. The magnitude of v_2 is equal to the magnitude of the SDF.

3. Compute the vector from the current voxel centre to the laser scan's termination point: $\boldsymbol{v}_2 = \boldsymbol{\rho}_r - \mathbf{p}_g$
4. Compute the voxel's SDF value:

$$u_{SDF} = \text{sgn}(\boldsymbol{v}_1 \cdot \boldsymbol{v}_2)||\boldsymbol{v}_2||_2 \tag{4.1}$$

   If $u_{SDF} > 0$, the voxel is between the surface and the laser sensor, $u_{SDF} < 0$ indicates the surface occludes the laser sensor's view of the voxel, and $u_{SDF} = 0$ indicates the voxel's centroid exactly intersects the surface.
5. Update the voxel's current f (SDF value) and w (weight) via Equation 2.33 on page 35.

The key variables of these steps are graphically depicted in Figure 4.3. As with the depth-map fusion, these calculations are highly data-independent and thus suitable for parallel processing. However, in contrast to depth-map fusion, one must ensure the memory update operations are atomic since multiple laser rays may simultaneously intersect any given voxel.

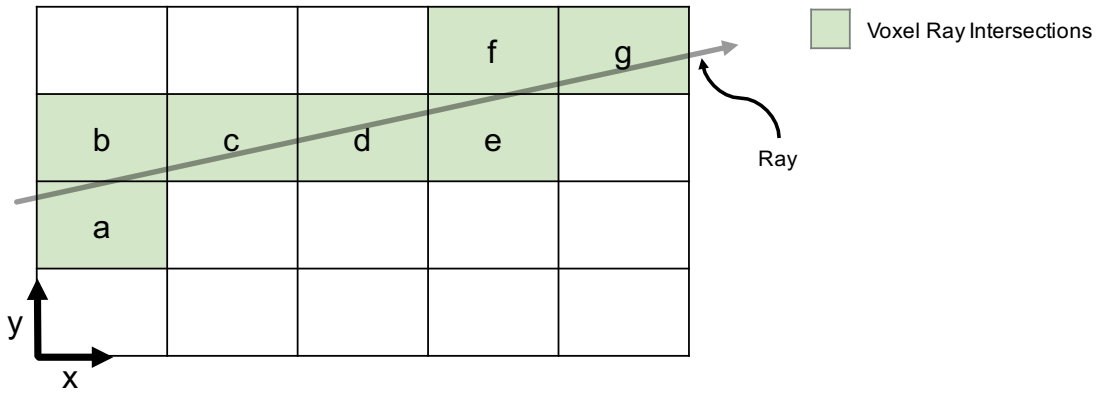


Figure 4.4: Ray casting problem overview. The goal of ray casting is to compute the in-order sequence of voxels intersected by a given ray ($\vec{\boldsymbol{\rho}}$), from its origin ($\boldsymbol{\rho}_o$) to termination point ($\boldsymbol{\rho}_r$). In the above example, the ray intersects 'voxels' in the order: a, b, c, d, e, f, g. Figure inspired by [81].

To conserve space, voxel blocks are allocated in the region $\pm\mu$, but the ray trace starts at the sensor so that free-space regions may be updated. In other words, in addition to the memory allocated immediately around the surface, we ray trace all free space to update any voxel blocks that may have previously been allocated when fusing depth or range data. For example, if an automobile appeared in a previous depth map or laser scan but it has since moved, the full ray trace will update the SDF values in those now-empty voxels. This removes free-space violations without increasing the memory requirements.

4.2.1.2 Efficient Ray Casting

We utilise an efficient implementation [81] for ray-casting operations in our system—these operations include laser fusion, voxel-block memory initialization to fuse depth maps or laser scans, and presenting a rendering of the reconstruction in a OpenGL window. In the last case, the 3D model is still extracted via Marching Cubes method (Section 2.4.3 on page 36) for processing by other applications in our mobile-robotics system pipeline (e.g., planning, segmentation, etc.).

The goal of ray casting is to compute the in-order sequence of voxels intersected by a given ray ($\vec{\boldsymbol{\rho}}$), from its origin ($\boldsymbol{\rho}_o = (x_o, y_o, z_o)$) to termination point ($\boldsymbol{\rho}_r = (x_r, y_r, z_r)$). To simplify notation, we denote the ray's normalized vector

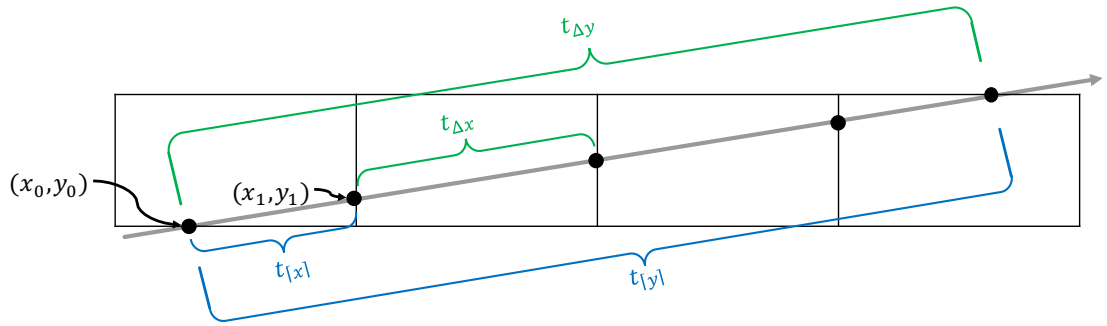


Figure 4.5: In an efficient implementation of ray casting, three key variables (per dimension) must be computed for each ray. The $t_{\Delta x}$ variable indicates the distance along the ray required to travel one voxel in the x direction. $t_{\lceil x \rceil}$ is the distance from the distance from the ray's entry to the voxel grid to the first time it crosses a 'vertical' boundary (i.e., x increments). x_d is 1 when the ray moves in the positive x direction, and -1 otherwise. $t_{\Delta y}$, $t_{\lceil y \rceil}$, and x_d are the same as their respective x variables, but for the y dimension. These values must only be computed once for each ray.

components as $\hat{\boldsymbol{\rho}} = (\hat{x}, \hat{y}, \hat{z})$. Figure 4.4 on the preceding page depicts a simple 2D ray-casting scenario.

Three variables (per dimension) must be computed to initialise the ray-casting algorithm,

$$\begin{aligned} t_{\Delta x} &= \left| \frac{v_x}{\hat{x}} \right| \\ t_{\lceil x \rceil} &= \frac{x^+ - x_o}{\hat{x}} \\ x^+ &= c_x + 0.5 v_x \operatorname{sgn}(\hat{x}) \end{aligned} \tag{4.2}$$

where v_x is the width of the voxel in the x dimension and c_x is the current voxel's centroid. The y- and z-equivalent variables can be found by substitution in the above equations. Each of these variables is computed with respect to t—i.e., a distance along the ray. $t_{\Delta x}$ is the travel distance along the ray to move one voxel in the x dimension, whereas $t_{\lceil x \rceil}$ is the travel distance along the ray *from the entry point* to the next voxel in the x dimension. Figure 4.5 provides graphical depictions of these variables in a 2D scenario.

As all variables are with respect to a distance travelled along the ray, one must transform all points from the Voxel Grid's reference frame to the ray's reference frame via,

Algorithm 1: Ray casting implementation for a 2D grid

```
1  while inGrid(x, y) do
2  |   operationOnCell(x, y);
3  |   if t_⌈x⌉ < t_⌈y⌉ then
4  |   |   t_⌈x⌉ = t_⌈x⌉ + t_Δx;
5  |   |   x = x + sgn(x̂);
6  |   else
7  |   |   t_⌈y⌉ = t_⌈y⌉ + t_Δy;
8  |   |   y = y + sgn(ŷ);
9  |   end
10 end
```

$$
\begin{aligned}
T_{VoxelGrid\_Ray} &= \begin{bmatrix} \vdots & \vdots & \vdots & x_o \\ \hat{r}_x & \hat{r}_y & \hat{r}_z & y_o \\ \vdots & \vdots & \vdots & z_o \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
r_x &= r_y \times r_z \\
r_y &= [x_o, y_o, z_o]^T \times r_z \\
r_z &= [x_r - x_o, y_r - y_o, z_r - z_o]
\end{aligned} \tag{4.3}
$$

Once these variables are initialised, Algorithm 1 presents the process to step through each grid cell in a 2D scenario. Note that each iteration in this implementation requires only two floating-point operations: one comparison and one addition. In concert with coalesced memory operations, this executes efficiently on a GPU.

The 3D algorithm used in the BOR^{2}G implementation is shown in Algorithm 2 on the next page.

Finally, a brief note on numerical precision is appropriate since data fusion in HVG requires two stages. In the first stage, the depth map or laser scan is projected into the voxel grid to allocate memory in the regions where surfaces are observed. The ray-casting algorithm operates on the courser voxel *blocks* for this initialisation. In the second stage, when processing laser scans, the ray caster steps through the voxels (not voxel *blocks*) to update each SDF value.

After processing real-world data to fuse both depth-map and laser data, we found that the system occasionally crashed. This occurred very rarely (in approximately

Algorithm 2: Ray casting implementation for a 3D voxel grid

```
1  while inVoxelGrid(x, y, z) do
2      operationOnVoxel(x, y, z);
3      if t_⌈x⌉ < t_⌈y⌉ then
4          if t_⌈x⌉ < t_⌈z⌉ then
5              t_⌈x⌉ = t_⌈x⌉ + t_Δx;
6              x = x + sgn(x̂);
7          else
8              t_⌈z⌉ = t_⌈z⌉ + t_Δz;
9              z = z + sgn(ẑ);
10         end
11     else
12         if t_⌈y⌉ < t_⌈z⌉ then
13             t_⌈y⌉ = t_⌈y⌉ + t_Δy;
14             y = y + sgn(ŷ);
15         else
16             t_⌈z⌉ = t_⌈z⌉ + t_Δz;
17             z = z + sgn(ẑ);
18         end
19     end
20 end
```

0.0001% of the laser scans) and only when processing laser scans. Further analysis of the problem revealed that numerical precision limitations on the GPU – specifically, cumulative rounding errors – resulted in stage 1 (*block* ray casting) reserving a different memory block than was required by stage 2 (*voxel* ray casting).

This is expanded upon in Figure 4.6 on the facing page where the ray passes near a region where four voxel blocks (eight in 3D) intersect. A slight rounding error results in a different ray-casting solutions depending on the step size—the voxel block ray-casting step size is exactly eight times larger.

This may be avoided by always using a single step size (e.g., voxel) for the ray-casting implementation. Alternatively, a small threshold value may be evaluated at each step to implement a margin of error to "round down" the solution to the same voxel or voxel block.

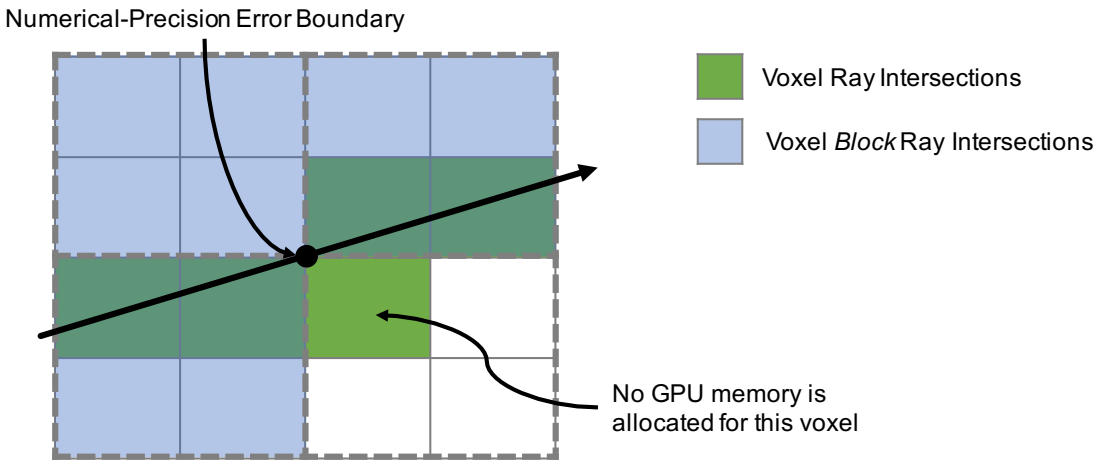


Figure 4.6: Ray casting numerical precision error. In 2D, when a ray passes through or near the intersection of four voxel blocks, a numerical-precision rounding error sometimes results in a different ray-casting solutions. In this instance, the blue voxel blocks were reserved in memory during stage 1 of HVG data fusion, but one green voxel will cause a runtime exception in Stage 2 since its voxel block never had memory reserved.

4.2.2 Voxel Labels

As discussed in Section 2.4.1.2 on page 32, HVG enables reconstruction of an arbitrary location in space by breaking the infinite (within numerical limits) space into voxel *blocks*. Each voxel block is composed of $8 \times 8 \times 8$ (512) voxels. These voxel blocks only occupy GPU memory if a surface is observed by the range sensor.

We augment the HVG data structure with an indicator variable (1_Ω) that designates whether or not a given voxel or voxel block is included in the domain or regularisation,

$$1_\Omega(v) = \begin{cases} 0 & v \in \Omega \\ 1 & v \notin \Omega \end{cases} \tag{4.4}$$

where v is a given voxel or voxel block and Ω is the set of voxels that were directly observed by the range sensor (i.e., the domain across which to apply regularisation). All voxels and voxel blocks are initialized as members of $\bar{\Omega}$ until an individual voxel's SDF value is updated via the data fusion stage. Figure 4.7 on the following page presents the Ω labels and GPU memory model after one range-sensor scan.

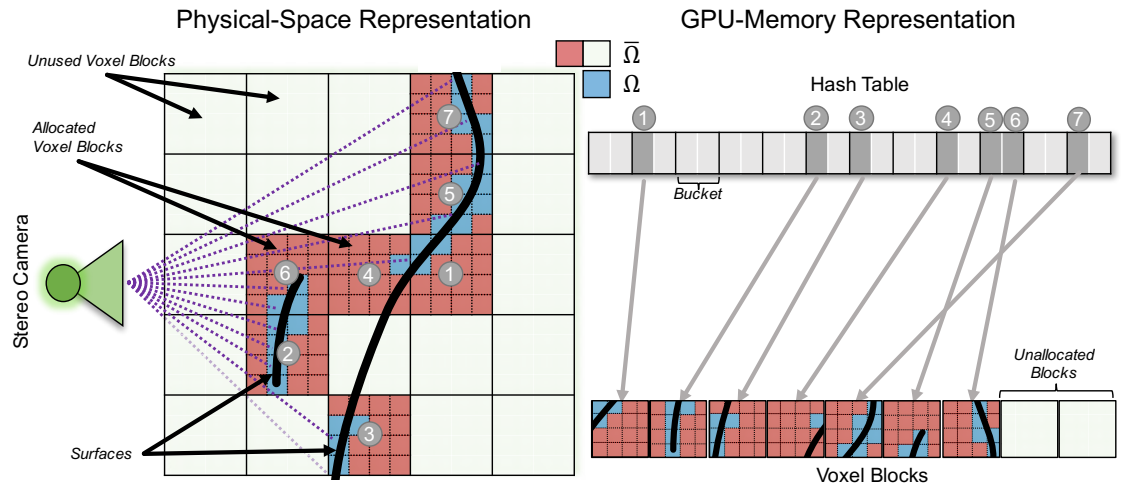


Figure 4.7: Hashing Voxel Grid and $\boldsymbol{\Omega}$ Labels. A depiction of our new combination of the HVG data structure with regularisation to fuse depth observations from the environment. The HVG enables us to reconstruct large-scale scenes by only allocating memory for the regions of space in which surfaces are observed (i.e., the coloured blocks). To avoid generating spurious surfaces, we mark each voxel with an indicator variable ($\mathbf{1}_{\boldsymbol{\Omega}}$) to ensure the regulariser only operates on voxels in which depth information was directly observed. This same approach is used independent of the range sensor — e.g., stereo depth maps or laser. Figure inspired by [59].

4.2.3 Optimisation

$\boldsymbol{\Omega}$ labels naturally extend from the conventional voxel grid (Chapter 3) to the HVG (Section 2.4.1.2 on page 32). Unlike non-homogeneous data structures like octrees, all voxels in the HVG that contain data are identically sized. This greatly simplifies the computation of neighbourhood lookups in the gradient and divergence operations. In fact, once the gradient and divergence operators are implemented, the optimisation implementations are *identical* to those presented in Section 3.3 on page 59.

With respect to GPGPU processing, a simpler implementation is usually faster. Applied to this application, rather than trying to optimise for performance and anticipate when a neighbour-lookup voxel is in the same voxel block, we instead simply fetch (or re-fetch) the voxel block from memory. Empirically, this is faster since it allows memory operations to be coalesced.

Thereafter, the optimisation implementation follows the form presented in Section 3.3 on page 59.

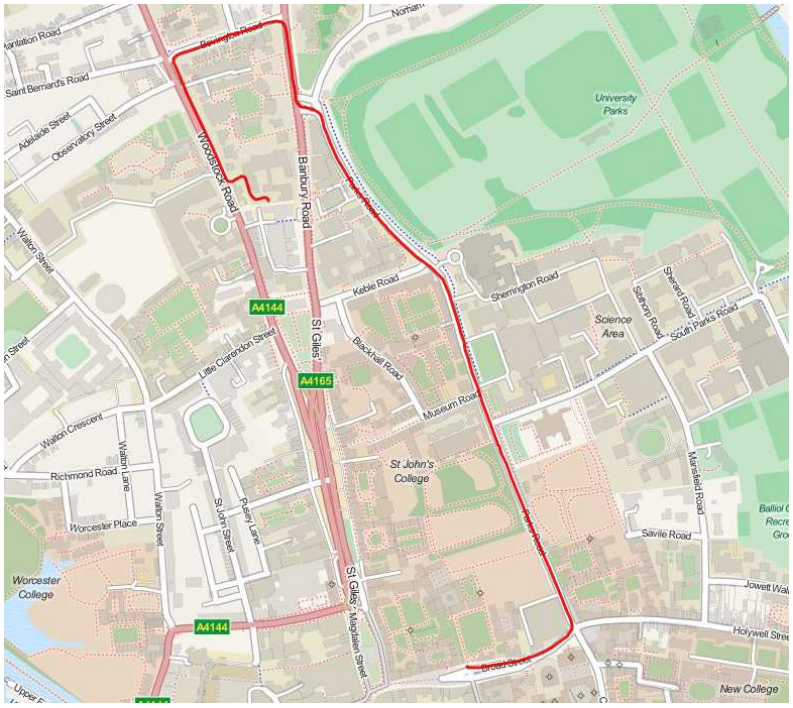


Figure 4.8: GPS trace for the Oxford Broad Street dataset. The dataset includes data from 1x Bumblebee XB3, 1x Bumblebee2, 4x Grasshopper2, 2x Velodyne HDL-32E lidar, and 3x SICK LMS-151 lidar sensors.

Table 4.2: Summary of Parameters Used in System

Symbol	Value	Description
λ_{3D}	0.8 (10 cm) / 0.4 (20 cm)	The TV weighting of the data term versus regularisation
μ_{3D}	1.0 m (10 cm) / 1.6 m (20 cm)	The maximum voxel distance behind a surface in which to fuse negative signed-distance values
σ_p, θ	0.5, 1.0	3D regulariser gradient-ascent/descent step sizes
τ	$1/6$	3D regulariser relaxation-step weight
N	20	Number of histogram bins
λ_{2D}	0.5	The TGV weighting of the data term versus regularisation
α_1, α_2	1.0, 5.0	Relative weights of the affine-smooth/piecewise-constant TGV terms
β, γ	1.0, 4.0	Exponent and scale factor respectively modifying the influence of the image gradient

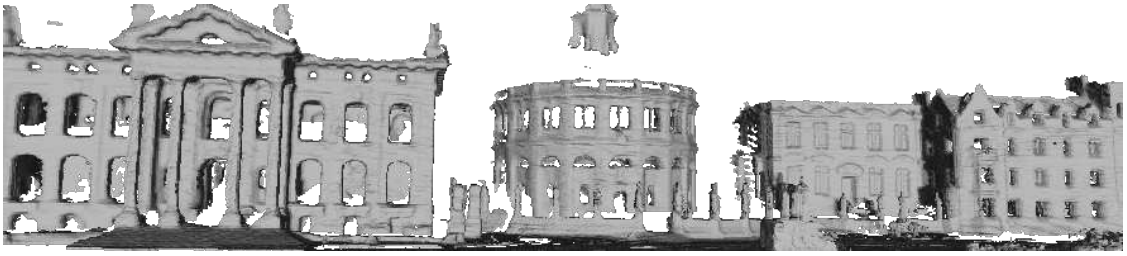


Figure 4.9: This chapter is about the efficient generation of dense models of multiple-km-scale environments from stereo and/or laser data. Pictured is an example reconstruction from the 1.6 km Oxford Broad Street dataset released with this thesis.

4.3 Results

This section provides an extensive analysis of our system, the parameters for which are provided in Table 4.2 on the preceding page. In the first portion of our results, we present how our system's reconstruction quality compares with prior work (Stanford Burghers of Calais). Next, we demonstrate its performance on our Oxford Broad Street dataset. Finally, we use the publicly available KITTI dataset [1] to evaluate the reconstruction performance of stereo-only, laser-only, and multi-sensor fusion methods.

Three KITTI sequences (00, 05, and 06) were selected based on their length, number of loop closures, and urban structure visible throughout the sequences. A summary of the physical scale of each is provided in Table 4.3.

All experiments were performed on a GeForce GTX TITAN with 6 GB memory.

4.3.1 Stanford Burghers of Calais

We first validate that our dense-fusion system creates high-quality reconstructions with the standard Stanford Burghers of Calais RGB-D dataset [66]. We performed a

Table 4.3: Summary of Reconstruction Scenarios

Dataset	Cam. Dist.	# Frames	Surface	Frames/m^2
KITTI-VO 00	3.8 km	4,541	93,546 m^2	0.049
KITTI-VO 05	2.2 km	2,761	55,909 m^2	0.049
KITTI-VO 06	1.2 km	1,101	23,585 m^2	0.047
Burghers	0.2 km	11,230	72 m^2	155.97

point-to-surface comparison between BOR^{2}G's (0.5 cm voxels) and [66]'s (unknown voxel size) reconstructions. BOR^{2}G's median reconstruction difference was 0.5 cm with a 1.0 cm 75-percentile difference. Note that this is a "difference" not an "error" metric as there is no ground-truth data against which we can compare.

As can be seen in Figure 4.1 on page 77, our level of detail compares favourable with the reconstructions in [66]. We accurately depict folds in clothing, muscle tone, and facial features.

However, this does not reveal the full capabilities of our framework as the low-noise, high-observation, and small-scale RGB-D dataset does not require sophisticated 2D and 3D regularisation to create high-quality reconstructions. We are not aware of any other system that both operates at scale and accepts a variety of sensor inputs — hence the remaining quantitative analysis compares only against ground-truth data.

4.3.2 Oxford Broad Street

In practice, we found dense reconstructions are the most complete and highest quality (with our mobile robotics platform) when fusing data from multiple Velodyne, SICK LMS-151, and stereo cameras using our own datasets. We are releasing such a dataset with this thesis to provide a realistic mobile-robotics platform with a variety of sensors—a few of which are prime candidates for ground truth when testing the accuracy of reconstructions with other sensors. For example, the push-broom laser sensor (which excels at 3D urban reconstructions) can be used to compare the reconstruction quality of monocular versus stereo camera versus Velodyne versus a combination thereof.

This dataset is ideal to benchmark and evaluate large-scale dense reconstruction frameworks. It was collected in Oxford, UK at mid-day, thus it provides a representative urban environment with numerous pedestrians, bicycles, and vehicles visible to all sensors throughout the 1.6 km trajectory (Figure 4.8 on page 91).

It includes data from the following sensors that collectively provide a continuous 360° view around the vehicle:

- 1x Point Grey Bumblebee XB3 Stereo Camera (Colour)
- 1x Point Grey Bumblebee2 Stereo Camera (Grayscale)
- 4x Point Grey Grasshopper2 Monocular Cameras (Colour, Fisheye Lens)
- 2x Velodyne HDL-32E 3D lidars
- 3x SICK LMS-151 2D lidars

In addition, the following is provided to aid in processing the raw sensor data:

- Optimised $\mathbb{SE}(3)$ vehicle trajectory
- Undistorted, rectified stereo image pairs
- Undistorted mono images
- Camera intrinsics
- Extrinsic $\mathbb{SE}(3)$ transforms for all sensors

Finally, we provide example depth maps — using the techniques described in this chapter — for the Bumblebee XB3 to enable users to rapidly utilise the dataset with their existing dense reconstruction pipelines.

All data is stored in a similar format as KITTI along with MATLAB development toolkit. Videos of the included data are available at:

`http://ori.ox.ac.uk/dense-reconstruction-dataset/`.

For brevity, we provide more detailed analysis of the stereo-only, laser-only, and multi-sensor fusion performance on the KITTI dataset in the following subsections. However, an example Velodyne-based reconstruction of a small segment of this dataset is shown in Figure 4.9 on page 92 and a larger qualitative analysis is included in the video on the dataset website.

4.3.3 KITTI

The KITTI dataset [1] was created to provide a benchmark system for the mobile robotics and computer vision communities. Geiger et al. mounted a 360° Velodyne lidar, GPS/INS sensor, and colour/grayscale stereo camera rigs composed of Point Grey Flea 2 cameras. After driving 39.2 km through a built-up, urban environment, they released the data in 2012 to provide a common benchmark. One of the

popular benchmarks within the KITTI dataset is the "Visual Odometry" datset (KITTI-VO): a sequence of 11 sequences with ground-truth camera-pose data and another 11 sequences for evaluation.

In this chapter, we propose utilising this dataset to quantitatively evaluate our dense reconstruction system. It is an excellent dense reconstruction benchmark because it: (1) operates in a real-world urban environment, (2) provides multiple sensor modalities, (3) the Velodyne sensor may be used as ground-truth when fusing depth-map data, and (4) is publicly available.

Unfortunately, the optimised GPS/INS ground-truth poses provided by KITTI proved to be highly inaccurate, especially when poses are provided after revisiting the same location multiple times. Empirically, we observed up to 3 m of drift upon loop closures. We selected three KITTI-VO sequences in ORB-SLAM2 to generate optimised pose trajectories with proper loop closures. These optimised pose chains empirically improved reconstruction performance when revisiting the same region, as we discuss in Section 4.3.3.3 on page 98.

4.3.3.1 Multi-Sensor Reconstruction Analysis

Many mobile platforms have a variety of sensors, yet conventionally much dense reconstruction work is isolated to a single sensor—nearly exclusively a camera sensor (RGB-D, monocular, or stereo). The KITTI dataset provides both camera images and Velodyne laser scans, so we decided to evaluate the quality of reconstruction achieved by fusing data from *both* sensors.

A snapshot of our qualitative results are shown in Figure 4.10 on page 97, but the video on the dataset website provides a fly-through of each sequence to visualise the quality of our final regularised 3D reconstructions.

The stereo-camera-only reconstructions were moderately detailed, but a number of false surfaces were created between neighbouring objects (e.g., automobiles, buildings) due to the limitations of stereo-based depth maps. The camera reconstructions included the full height of the buildings, in contrast to the laser reconstructions that could only reconstruct the bottom 2.5 m of objects. However, the laser-only

reconstruction was much more detailed and could see surfaces that were occluded to the camera (e.g., behind automobiles) because the Velodyne was mounted higher on the data-collection vehicle and it has a continuous 360° field of view.

Figure 4.10 on the next page shows that the combination of both sensors produced a reconstruction that was higher quality and more comprehensive. The comprehensiveness is measured in Table 4.4, where the multi-sensor fused reconstruction has 40% to 50% more surface area coverage than the best sensor was able to reconstruct by itself. We leave it to future work to quantitatively demonstrate the effects of cost, field of view, and sensor fidelity (e.g., higher-quality stereo cameras or lasers) on the resulting reconstruction quality and surface-area coverage.

In addition to traditional qualitative analysis, KITTI stereo-only reconstructions may be quantitatively analysed by using laser data as ground truth. We dedicate the remainder of our results write-up to this quantitative analysis.

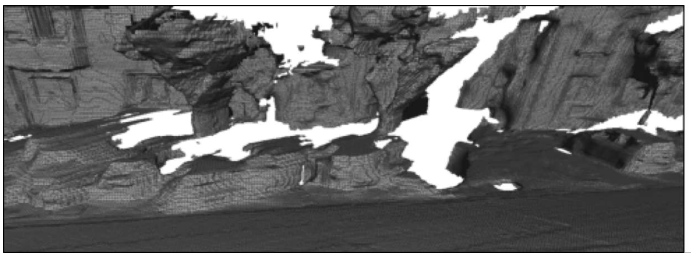
4.3.3.2 SDF Versus Histogram Data Terms Analysis

We consolidated all Velodyne HDL-64E laser scans into a single reference frame using poses provided by ORB-SLAM2. We found that KITTI's "ground-truth" GPS/IMU poses were not accurate enough to produce high-quality dense reconstructions — in particular when revisiting regions. There was as much as 3 metres of drift from one pass of a region to the second pass with the KITTI ground-truth poses, while ORB-SLAM2 poses were accurate within a few centimetres.

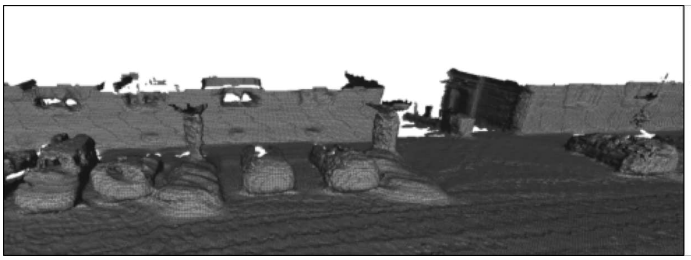
The following three subsections perform analysis on the stereo-only reconstruction performance for SDF versus histogram data terms, comparing multiple stereo camera passes through the same environment, and the quantitative quality of 7.3 km of stereo-only reconstructions.

Table 4.4: Summary of Surface Area Coverage by Sensor Type

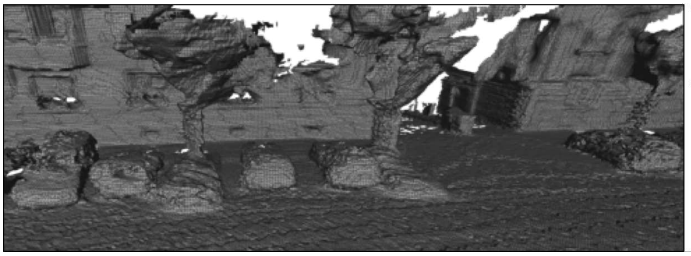
Dataset	Image Only	Laser Only	Image & Laser
KITTI-VO 00	93,546 m^2	117,087 m^2	176,624 m^2
KITTI-VO 05	55,909 m^2	77,292 m^2	111,188 m^2
KITTI-VO 06	23,585 m^2	29,617 m^2	41,401 m^2



(a) Stereo Camera Only



(b) Laser Only



(c) Stereo Camera and Laser

(d) Final Coloured Reconstruction

Figure 4.10: Comparison of reconstruction quality with stereo-camera only, laser only, and stereo-with-laser fusion. The stereo camera has a higher field of view than the laser sensor (i.e., the building/trees are cut half way up in the laser reconstruction), but the laser sensor is much more accurate and can see into regions that were occluded for the stereo camera (e.g., behind the automobiles). Fusing data from both sensors into the same voxel grid produces a more comprehensive (Table 4.4 on the facing page) and higher-quality result than either sensor can achieve alone.

Zach demonstrated the superior histogram data term performance in small-scale, object-centred scenarios [3]. We compare SDF and histogram optimisation performance so dense-reconstruction system developers may make an informed decision based on their target application.

After fusing stereo-based depth images into the voxel grid, the resulting reconstruction is indeed noisy, as shown in Figure 4.11 on the next page. The 2D TGV^2 regulariser significantly reduced the noise in the input depth maps, but these passively-generated depth maps inherently must infer depth of large portions of the image.

Both the SDF and Histogram optimisers smooth the noisy surfaces (e.g., buildings) in the reconstruction, but the SDF regulariser subjectively appears to do a better job. When compared against ground-truth in Table 4.5, the SDF and histogram optimisers are nearly indistinguishable: both reconstructions have a 5.9 cm median point-to-surface error with a 28 cm standard deviation.

Based on Zach's previous results, we initially expected the histogram to perform better. However, it appears that when a camera travels through the environment (rather than observing a single object from many angles) there are not enough observations of surfaces to provide a complete PDF in each voxel. Since the SDF regulariser is both simpler to implement, faster to execute, and provides similar results, we believe it can be a better choice for mobile-robotics platforms and we use it exclusively for the remainder of our experiments.

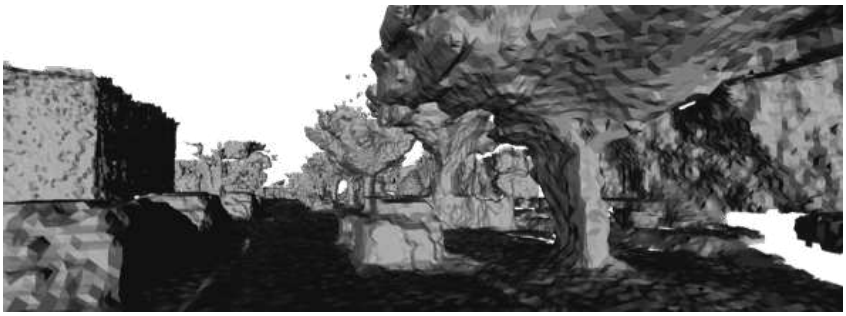
4.3.3.3 Multiple Passes Analysis

In theory, revisiting a region should result in a more complete and accurate reconstruction, assuming one has accurate localisation and loop closures. Noisy depth maps from the stereo camera preclude the traditional point-cloud alignment

Table 4.5: SDF Versus Histogram Regularised Reconstruction Errors

Fusion Method	Median	75%	Std. Dev.
Signed Distance Function (SDF)	5.9 cm	21.0 cm	28.0 cm
Histogram	5.9 cm	21.5 cm	27.9 cm

(a) Raw Reconstruction



(b) Histogram Data Term Reconstruction

(c) SDF Data Term Reconstruction

(d) Final Coloured Reconstruction

Figure 4.11: Comparison of reconstruction improvement with histogram and SDF data terms on stereo-camera-only reconstructions. Both optimisation methods noticeably smooth the surfaces (e.g., buildings, trees, automobiles) while also removing low-certainty surfaces (e.g., speckled surface in the sky near the trees). The SDF data term qualitatively produces "smoother" surfaces, but quantitatively (Table 4.5 on the facing page) SDF and histogram methods are nearly indistinguishable.

approaches used in Kinect Fusion-based approaches. However, ORB-SLAM2 provides accurate loop closures and locally-consistent pose estimates.

In fact, ORB-SLAM2 performs better in our applications than did the GPS/INS ground truth — the latter resulted in trees and walls being observed in the middle of a road when revisiting a location.

In Figure 4.12 on the next page, we compare the quality of reconstruction between a single and two passes of a region. The second pass noticeably improves the detail in previously-observed surfaces and increases the surface area reconstructed by 11% (6,044 m^2 → 6,713 m^2).

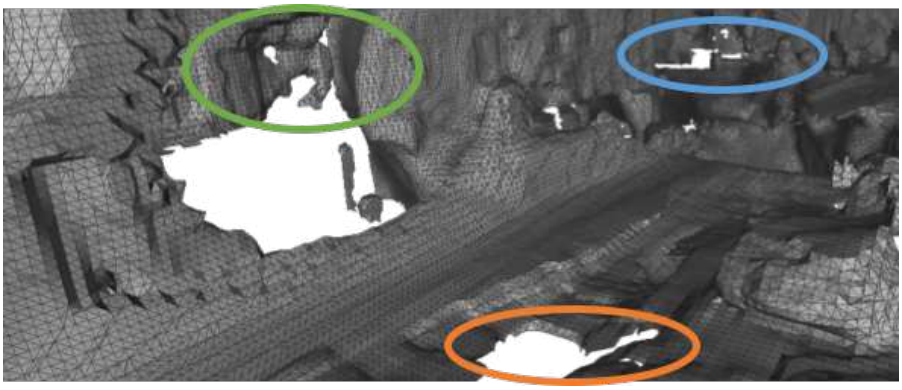
4.3.3.4 Full-Length Error Metrics Analysis

Using only the stereo camera as input, we first processed all three KITTI-VO with 10 cm voxels and compared the dense reconstruction model, both before and after regularisation, to the laser scans. In these large-scale reconstructions, the compressed voxel grid structure provides efficient fusion performance while vastly increasing the size of reconstructions. For the same amount of GPU memory, the conventional voxel grid was only able to process 205 m, in stark contrast to the 1.6 km reconstructed with the HVG—though that may be extended to an infinite-size reconstruction by utilising bidirectional GPU-CPU streaming.

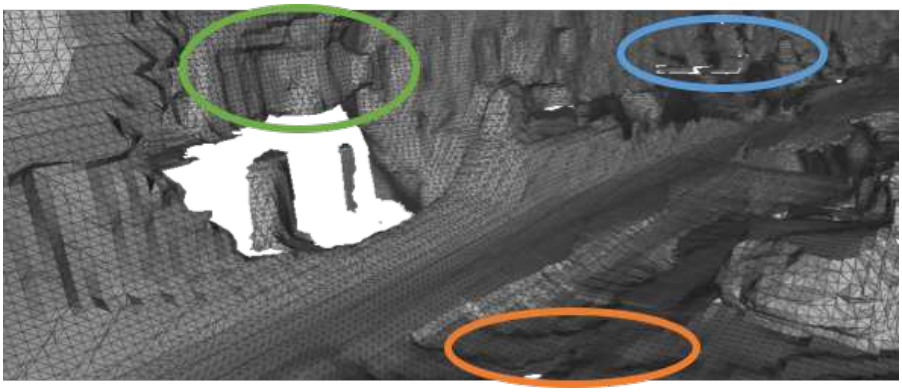
As shown in Table 4.6 on page 105, the SDF regulariser reduced the median error by 27% to 36%, the 75-percentile error by 32% to 50%, and the surface area by 26%. The final median error for the scenarios varied between 4.0 cm and 8.1 cm.

In Figure 4.13 on page 103, it becomes clear that errors in the initial "raw" fusion largely come from false surfaces created in the input depth map caused by sharp discontinuities in the original image. The SDF regulariser removes many of these surfaces, that dominate the tail of the histogram plots and are visible as red points in the point-cloud plots.

When processed at 20 cm voxel resolution, the results are similar, though with slightly higher error metrics, as shown in Table 4.6 on page 105. However, even though the spacial resolution was reduced by a factor of two (and memory



(a) Stereo Camera Pass 1



(b) Stereo Camera Pass 2

(c) Final Coloured Reconstruction

Figure 4.12: Comparison of fusion after two passes of the same region. The second pass results in a more complete model since, with accurate localisation, the stereo-camera depth maps observe surfaces from a slightly different angle and lighting conditions. The first pass reconstructed 6,044 m^2, while the second pass increased that by 11% to 6,713 m^2. Note that the hole in the centre-left of the image is due to object occlusions at the viewing angle of the input stereo images.

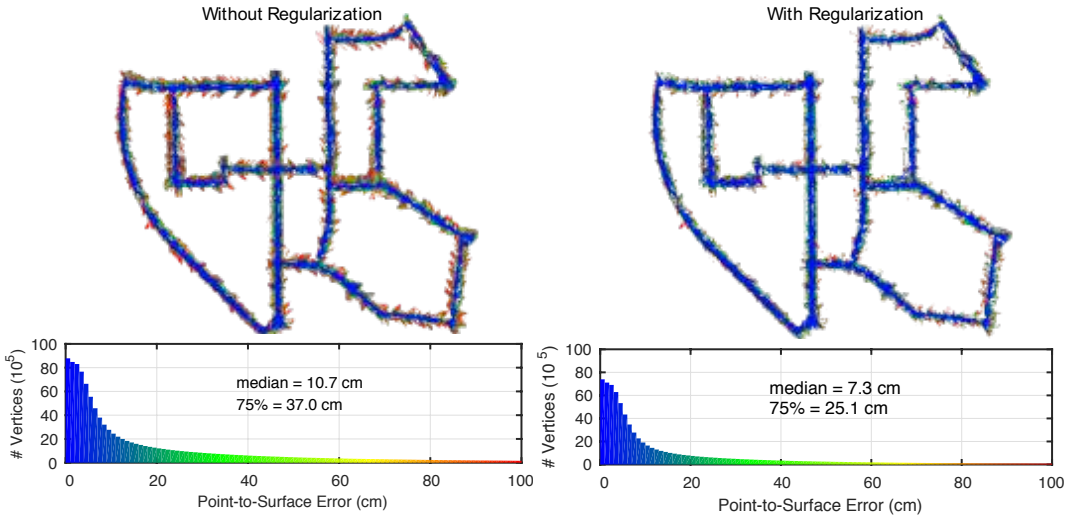
requirements by a factor of 8), the final reconstruction accuracy was only reduced by about 10%. This is because the urban environments are largely dominated by planar objects (e.g., roads, building façades) that, by the very nature of SDF, can be nearly-perfectly reconstructed with course voxels. Smaller voxels are only beneficial in regions with fine detail (e.g., automobiles, steps, curb).

Figure 4.14 on page 104 shows the bird's-eye view of each sequence with representative snapshots of the 10 cm stereo-camera-only reconstructions. To illustrate the quality, we selected several snapshots from various camera viewpoints that were offset in both translation and rotation to the original stereo camera position, thereby providing a representative depiction of the 3D structure. Overall, the reconstructions are quite visually appealing; however, some artefacts such as holes are persistent in regions with poor texture or with large changes in illumination. This is an expected result since, in these cases, no depth map can be accurately inferred.

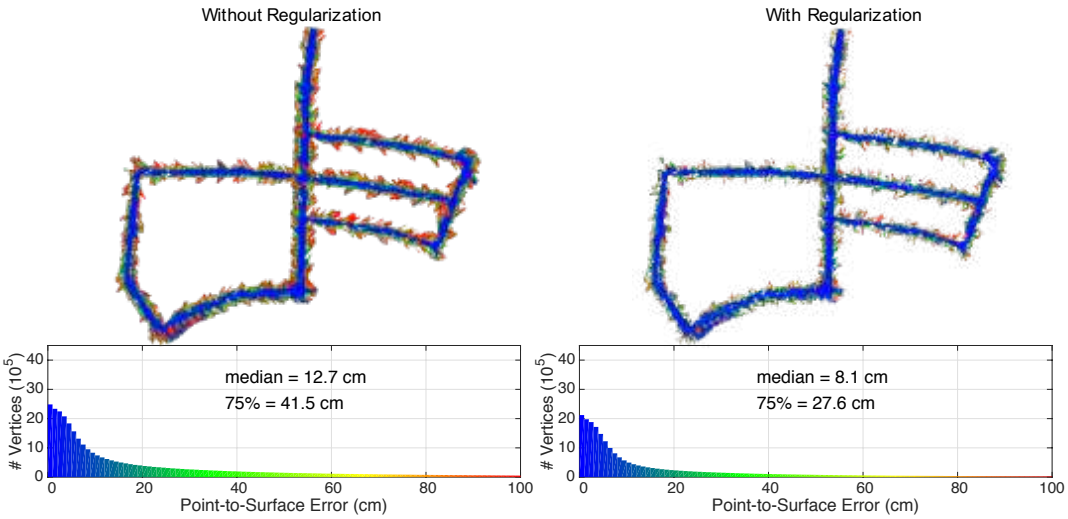
4.4 Conclusions

We presented our state-of-the-art BOR^{2}G dense mapping system for large-scale dense reconstructions. When compared to object-centred reconstructions, mobile-robotics applications have 3,100 times fewer depth observations per square metre, thus we utilised regularisers in both 2D and 3D to serve as priors that improve the reconstruction quality. We overcame the primary technical challenge of regularising voxel data in the compressed 3D structure by redefining the gradient and divergence operators to account for the additional boundary conditions introduced by the data structure. This both enables regularisation and prevents the regulariser from erroneously extrapolating surface data.

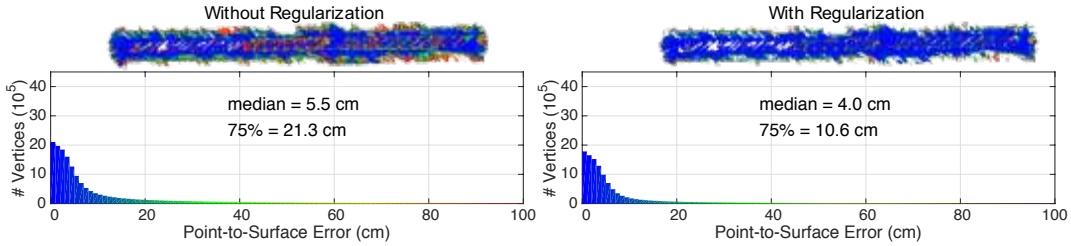
We know of no other dense reconstruction system that is *quantitatively* evaluated at such a large scale. When qualitatively compared to the Stanford Burghers of Calais RGB-D dataset, BOR^{2}G reconstructed the same fine detail. For our large-scale reconstructions, we used the 3D lidar sensor as ground truth to evaluate the



(a) KITTI-VO 00: Stereo Camera Reconstruction Error Metrics

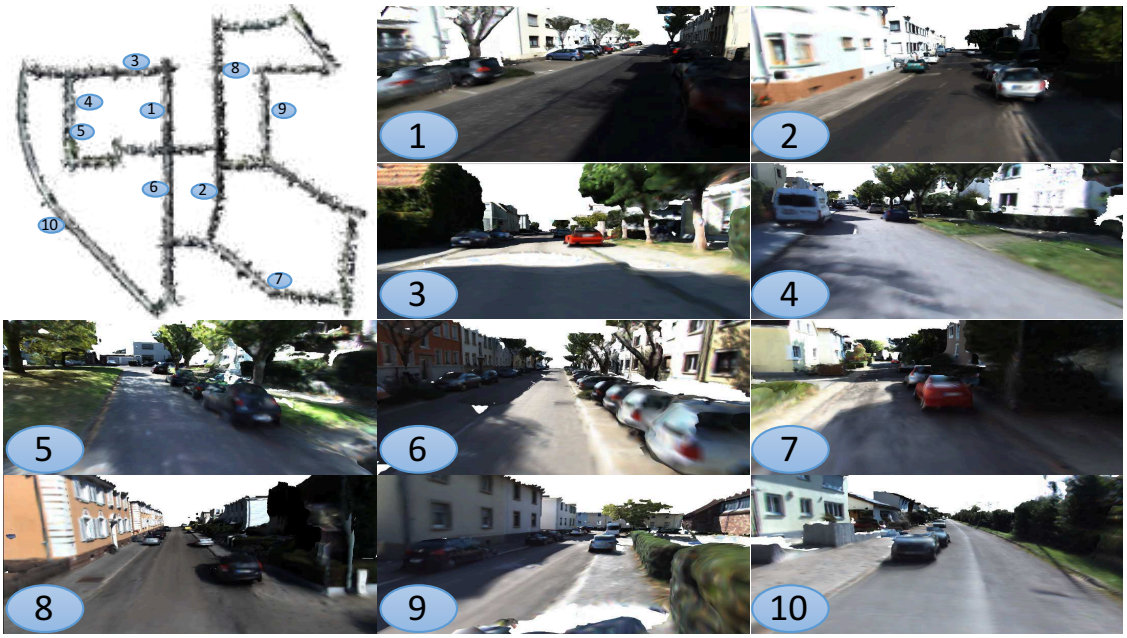


(b) KITTI-VO 05: Stereo Camera Reconstruction Error Metrics

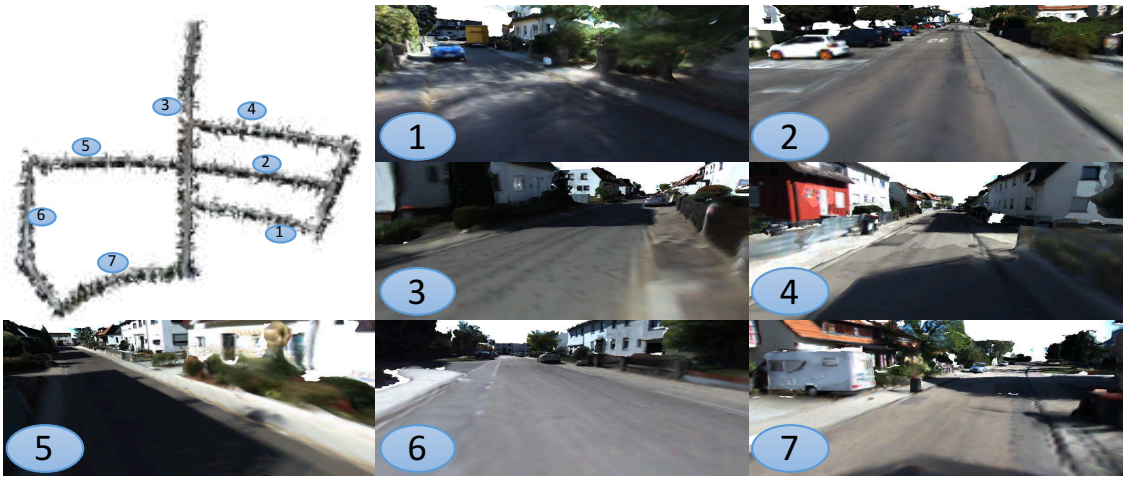


(c) KITTI-VO 06: Stereo Camera Reconstruction Error Metrics

Figure 4.13: A summary of the stereo-camera-only dense reconstruction quality for three scenarios from the KITTI-VO public benchmark dataset. The left side are the results before regularisation and the right side are after regularisation. Above each histogram of point-to-surface errors are the top-view, coloured reconstruction errors corresponding to the same colours in the histogram. The regulariser reduces the reconstruction's error approximately a third, primarily by removing uncertain surfaces — as can be seen when you contrast the raw (left) and regularised (right) reconstruction errors.



(a) KITTI-VO 00: Stereo Camera Reconstruction Samples



(b) KITTI-VO 05: Stereo Camera Reconstruction Samples

(c) KITTI-VO 06: Stereo Camera Reconstruction Samples

Figure 4.14: A few representative sample images for various points of views (offset from the original camera's position) along each trajectory. All sample images are of the final regularised reconstruction with 10 cm voxels using only stereo images as the input.

Table 4.6: Summary of Stereo-Camera-Only Reconstruction Quality and Error Statistics

Dataset	Type	Res.	Median	75%	GPU Memory	Surface Area	# Voxels	Time Per Iteration (mm:ss)
KITTI-VO 00	Raw	20 cm	11.4 cm	38.3 cm	1,222 MiB	126,251 m^2	$107{\cdot}10^6$	—
		10 cm	10.7 cm	37.0 cm	7,565 MiB		$661{\cdot}10^6$	—
	Regularized	20 cm	7.8 cm	25.9 cm	1,222 MiB	93,546 m^2	$107{\cdot}10^6$	1:46
		10 cm	7.3 cm	25.1 cm	7,565 MiB		$661{\cdot}10^6$	11:20
KITTI-VO 05	Raw	20 cm	14.2 cm	45.0 cm	756 MiB	76,880 m^2	$66{\cdot}10^6$	—
		10 cm	12.7 cm	41.5 cm	4,533 MiB		$396{\cdot}10^6$	—
	Regularized	20 cm	9.1 cm	30.2 cm	756 MiB	55,909 m^2	$66{\cdot}10^6$	1:00
		10 cm	8.1 cm	27.6 cm	4,533 MiB		$396{\cdot}10^6$	3:45
KITTI-VO 06	Raw	20 cm	6.1 cm	23.8 cm	356 MiB	32,241 m^2	$31{\cdot}10^6$	—
		10 cm	5.5 cm	21.3 cm	2,591 MiB		$226{\cdot}10^6$	—
	Regularized	20 cm	4.5 cm	12.4 cm	2,591 MiB	23,585 m^2	$31{\cdot}10^6$	0:34
		10 cm	4.0 cm	10.6 cm	356 MiB		$226{\cdot}10^6$	3:44

quality of our reconstructions, for different granularities, against 7.3 km of stereo-camera-only reconstructions. Our regulariser consistently reduced the reconstruction error metrics by a third, for a median accuracy of 7 cm over $173,040$ m^2 of reconstructed area. Qualitatively, fusing both stereo-camera-based depth images with lidar data produces reconstructions that are both higher quality and more comprehensive than either sensor achieves independently. We hope the release of our ground-truth KITTI point clouds, ground-truth KITTI 3D models, and the comprehensive Oxford Broad Street dataset will become helpful tools of comparison for the community.

This chapter applied the $\mathbf{\Omega}$ labelling concept (originally presented in Chapter 3) to a compressed 3D data structure that enabled real-world, large-scale reconstructions. In the next chapter we utilize the $\mathbf{\Omega}$ labelling to interpolate the underlying 3D urban structure that is occluded by dynamic or ephemeral objects.

The subjectivist (i.e. Bayesian) states his judgements, whereas the objectivist sweeps them under the carpet by calling assumptions knowledge, and he basks in the glorious objectivity of science.

— I. J. Good

5

Recovering Hidden Structure and Filling in the Gaps

Abstract

The generation of accurate static maps for mobile-robotics applications is encumbered by ephemeral objects such as vehicles, pedestrians, or bicycles. Building upon the theory presented in Chapter 3, we propose a method to process a sequence of laser point clouds and back-fill dense surfaces into gaps caused by removing objects from the scene – a valuable tool in scenarios where resource constraints permit only *one* mapping pass in a particular region. Our method processes laser scans in a 3D voxel grid using the Truncated Signed Distance Function (TSDF) and then uses a Total Variation (TV) regulariser with a Kernel Conditional Density Estimation (KCDE) "soft" data term to interpolate missing surfaces. Using four real-world scenarios captured with a push-broom 2D laser, our technique infills approximately 20 m^2 of missing surface area for each removed object. Our median reconstruction error ranges between 5.64 cm - 9.24 cm with standard deviations between 4.57 cm - 6.08 cm.

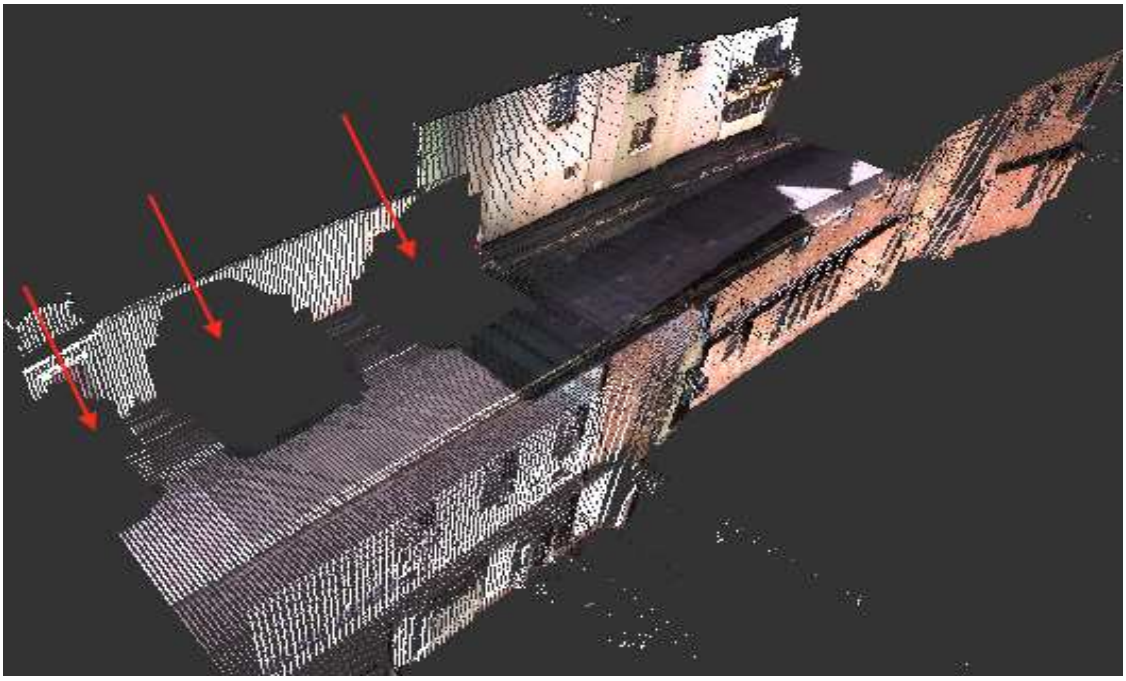
Contents

In robotics applications, it is vital to maintain an accurate map of the local environment. When in the presence of ephemeral objects (e.g., vehicles, pedestrians, bicycles, etc.), this map can quickly become cluttered and inconsistent, especially when there is only a single mapping pass—as is the case in many surveying applications. These inconsistencies make the map difficult to use.

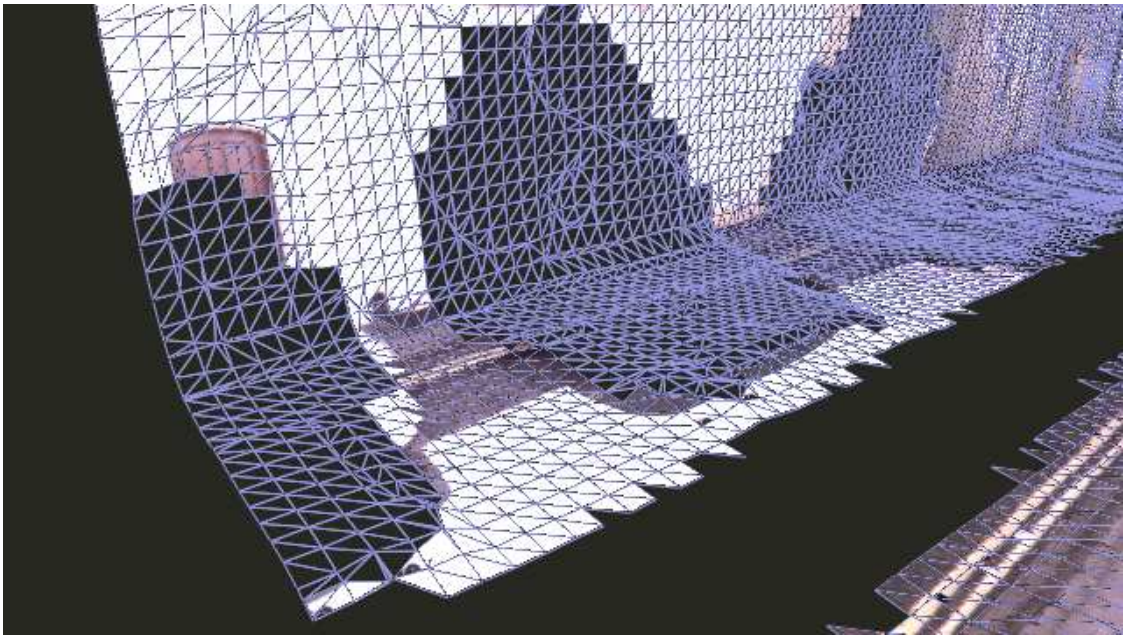
Ephemeral objects are sometimes dynamic (e.g., pedestrian walking on the pavement) but also may be static (e.g., automobile parked on the side of a road). If a complete background map has been created from a previous data run, these difficulties can be avoided by detecting and removing the differences between the maps. However, in real-world data collections we encounter in our autonomous-driving applications, it is uncommon to ever observe the world with no ephemeral objects.

Nonetheless, it is desirable to remove these objects from the map and infill the missing background data as this creates a more accurate static map. Figure 5.1 on the facing page illustrates a representative map created in an urban data collection. Even after we remove the ephemeral objects (automobiles, in this example), holes still remain in the map. If these holes are filled in, the resulting map is more complete, better representing the underlying 3D urban structure.

Of what use is this more-complete static map? Our work was motivated by one specific application: low-cost urban surveys. When competing a large-scale (e.g., city size or larger) dense map for a survey, it is time-prohibitive to traverse every



(a) Before: The *sparse* laser point cloud input.



(b) After: Our *dense*, interpolated reconstruction.

Figure 5.1: Example scenario (a) where three large gaps (highlighted by red arrows) appear in the laser data after vehicles are removed. These gaps must be filled in (b) to generate an accurate map that could later be used for algorithms such as localisation or ephemeral-object detection.

area a sufficient number of times to ensure no surface is occluded by ephemeral objects. Rather, we seek to mount a push-broom oriented laser sensor on an automobile, traverse each street exactly once, and rely upon post-processing to remove ephemeral objects and infer the occluded urban structure.

Another use for the map is to provide a computationally-efficient method to detect objects—for example, by comparing the static map to live laser scans to identify new ephemeral objects. This approach is useful in distraction suppression [107] scenarios where a mask is applied to the input data to avoid bias in dependent localisation or perception algorithms. Alternatively, to complement recent "hard negative" mining research [108], these object detections may be used to automatically generate "hard positives" for standard detector or classification training.

However, removing ephemeral objects traditionally requires manually labelling positive and negative data sets to train a detector. And, once removed, it is even more difficult to back-fill the gaps (Figure 5.1 on the previous page) left in the laser data because, without a prior, there is ambiguity as to which areas should be free space and which should contain interpolated surfaces. We present our approach to these two problems later in this chapter, but what proceeds are methods currently proposed in the literature for hole filling and dense 3D map creation.

There has been a variety of work to address the "hole filling" problem. Wang and Oliveira [109] use a least-squares method to estimate a locally-smooth surface for holes. However, this requires user intervention to identify the holes and can only reconstruct simple surfaces. Vasudevan *et al.* [110] use a neural-network based Gaussian process to model large-scale, outdoor environments. They produce visually appealing results for landscape-type environments. However, this is not applicable in the city-street environments typical for our mobile robotics platform. Davis *et al.* [111] use a small, local signed distance function to represent the surface around the hole. Then, based on a number of heuristics, they select a "class model" to approximate the missing surfaces. Their algorithm was designed and tested in a small scale environment and only operates on gaps that are surrounded on all sides by distance data - the latter assumption is routinely violated in our data

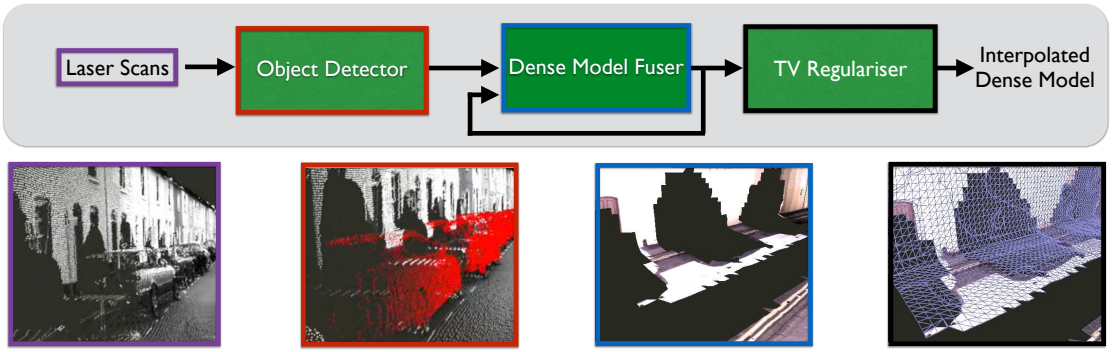


Figure 5.2: Software Pipeline Overview: Our software pipeline consists of three stages to convert a sequence of laser scans into an interpolated dense model. First, we process the laser scans to automatically detect ephemeral objects in the 3D point cloud. Next, the laser rays not associated with an object are fused into a dense 3D model (Section 5.2 on page 115). Finally, we apply our KCDE-modified TV regulariser to interpolate surfaces behind the previously-removed objects (Section 5.4 on page 123).

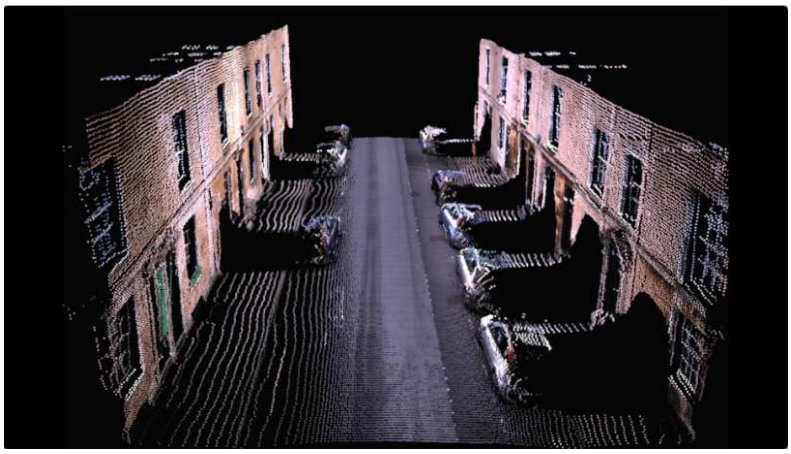
runs with tall ephemeral objects or when objects pass near to the laser sensor. Häne *et al.* [112] use a regularisation term to both smooth out noisy range data and to interpolate small holes in the dense reconstruction with the assumption that the surfaces are piecewise planar.

Much of the recent research in the field of general dense reconstruction is based on RGB or RGB-D (e.g., Microsoft Kinect) inputs [2][67][68][113]. However, we seek to produce an accurate, dense model solely from sparse range-data input (laser) and then infill ephemeral-object gaps via a post-processing step.

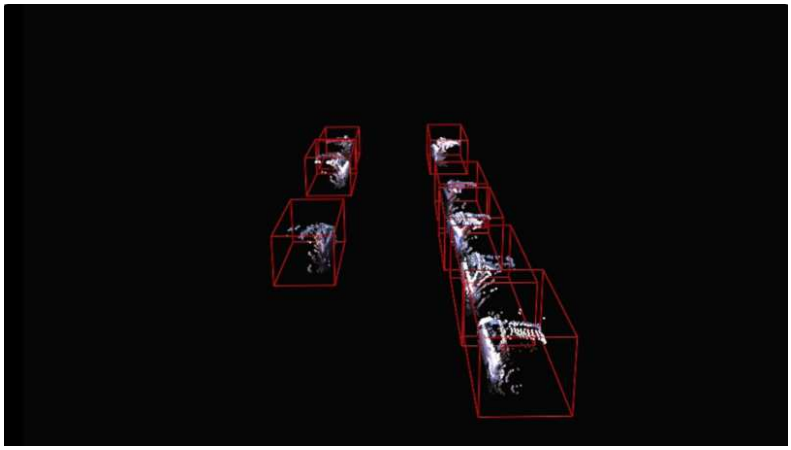
Figure 5.2 provides a graphical overview of the framework presented in this chapter. Our input is a sequence of push-broom laser scans from a SICK LMS-151 mounted on an automobile platform. The laser scans are consolidated into a cohesive 3D sparse point cloud and processed as follows:

1. *Object Detector* — We automatically detect and remove ephemeral objects by using the 3D sliding-window technique presented in [114]. Figure 5.3 on the next page presents an example of automobile object detection.

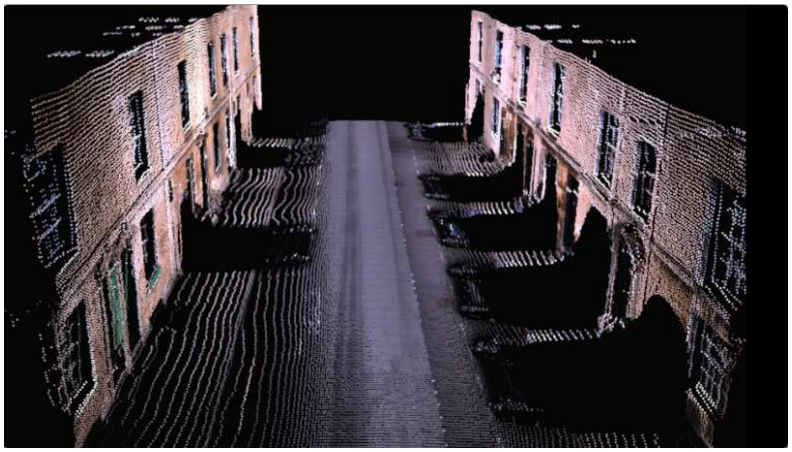
2. *Dense Model Fuser* — As discussed in Chapter 4, the laser scans are fused into a voxel grid to create a dense model of the environment.



(a) Raw laser point cloud



(b) Detected automobiles



(c) Processed point cloud after automobile removal

Figure 5.3: Object Detection Example. An example scenario in which the automatic object-detection pipeline identifies ephemeral objects (automobiles) and removes them from the laser point cloud.

3. *TV Regulariser* — We apply our proposed KCDE TV optimisation method (Section 5.3 on page 117) to interpolate all occluded surfaces.

The final output of our system is a dense, 3D model of the underlying urban architecture.

This chapter's contribution is the formulation of a new KCDE data term for the energy functional in dense reconstructions. With no directly observed data in the interpolation regions, the KCDE makes the energy sympathetic to local structure by using a Gaussian-based likelihood model dependent upon neighbouring surfaces. This helps avoid the tendency for TV to produce a saddle-shaped surface when reconstructing multiple planes (see Section 5.3.1 on page 117).

A naive approach to missing surface reconstruction might use pure interpolation or a simple affine prior. However, the sophisticated regularisation approach we use allows us to model more complicated surfaces without explicitly enumerating all possible priors.

We begin in Section 5.1 with a Bayesian interpretation of the TV method previously described in Chapter 3. Section 5.2 on page 115 provides the $\mathbf{\Omega}$-label (Chapter 3) extensions required for our problem formulation. In Section 5.3 on page 117, we explore the possible interpolation methods that may be utilised by our regulariser. The algorithm to implement our proposed energy-minimisation optimisation approach is presented in Section 5.4 on page 123. Finally, quantitative and qualitative analysis of our regulariser's performance in an urban environment are presented in Section 5.5 on page 126.

5.1 Variational Method Regularisation

In contrast to the derivation in Chapter 3, we use a probabilistic approach to derive the energy function for TV. This provides a better intuition into the rationale as to why variational methods are an appropriate mathematical model for the hole-filling problem.

At its core, our method calculates the *maximum a posteriori* (MAP) estimator,

$$\underset{u}{\operatorname{argmax}} \left(P(u|f) \right) \tag{5.1}$$

with $P(u|f)$ given via Bayes' theorem as,

$$P(u|f) = \frac{P(f|u)P(u)}{P(f)} \tag{5.2}$$

where the scalar u is the denoised or interpolated result, f is the noisy TSDF data, $P(f)$ is a constant, $P(u)$ is the prior or "regularisation" term, and $P(f|u)$ is our likelihood model or "data" term.

From this foundation, we derive a traditional TV energy functional (Section 5.3.1 on page 117), implement a naïve-data-term regulariser (Section 5.3.1 on page 117), and finally incorporate a KCDE to create a more sophisticated regulariser (Section 5.3.2 on page 120) for interpolation purposes.

$P(u)$ and $P(f|u)$ can be modelled, respectively, as Laplace and Gaussian distributions,

$$P(u) \propto \prod_{u_i \in \Omega} \exp\left(\frac{||\nabla u_i||_1}{2\sigma_u^2} \right) \tag{5.3}$$

$$P(f|u) \propto \prod_{(f_i,\ u_i) \in \Omega} \exp\left(\frac{||f_i - u_i||_2^2}{2\sigma_f^2} \right) \tag{5.4}$$

The numerator in the Laplace and Gaussian exponential's fraction are the L_1 and L_2 norms. The L_1 norm is useful as a prior since it is robust to outliers and allows piecewise consistency. If these distributions are substituted into Equation 5.1, we can reframe this as a minimising optimisation problem,

$$\begin{gathered} \underset{u}{\operatorname{argmin}} \left(-\log(P(u|f)) \right) = \\ \underset{u}{\operatorname{argmin}} \sum_{(f_i,\ u_i) \in \Omega} ||\nabla u_i||_1 + \lambda ||f_i - u_i||_2^2 \end{gathered} \tag{5.5}$$

where λ encapsulates the ratio of the constant scalar values in Equations 5.1, 5.3, and 5.4.

Equation 5.5 on the preceding page can be mapped to the continuous domain as a variational method's cost functional in the form [115],

$$\begin{aligned} E(u) &= E_{regularisation}(u) + E_{data}(u, f) \\ E(u) &= \int_{\Omega} ||\nabla u||_1 d\Omega + \lambda \int_{\Omega} ||f - u||_2^2 d\Omega \end{aligned} \tag{5.6}$$

where $E(u)$ is the energy (that we seek to minimise) of the denoised (u) and noisy (f) data. The *data* energy term seeks to minimise the difference between the u and f. The *regularisation* energy term–this specific L_1-norm case is commonly known as a TV regulariser–seeks to fit the solution (u) to a specified prior. When data is non-existent, the regularisation term acts an interpolator and its form serves as a prior. This is a convex energy minimisation problem that can be solved using Primal-Dual techniques [77].

Before discussing a modified data term that improves interpolation performance, first we must define the operating domains of this problem. This will enable us to distinguish the regions that must be interpolated, in contrast to the typical noise-reduction goal of optimisation with regularisation.

5.2 Ω Label Inclusion

In Chapter 3, Ω labels were originally conceived as a method to prevent spurious surfaces from appearing when regularising 3D voxels where only a subset were directly observed by a range sensor. Ω label's defining property is enabling regularisation to target a specific subset of 3D space. It operates on a principal similar to Neumann boundary conditions - i.e., carefully define and limit an operation (regularisation) by taking into account special boundary conditions for the operating domain.

In this chapter, we use the usually-undesirable property (i.e., surface extrapolation) of regularising unobserved voxels with the targeting ability of Ω labels to *interpolate* structure in specific regions of the dense reconstruction. All other regions ($\bar{\Omega}$) must still be excluded to avoid spurious surface generation by the regulariser.

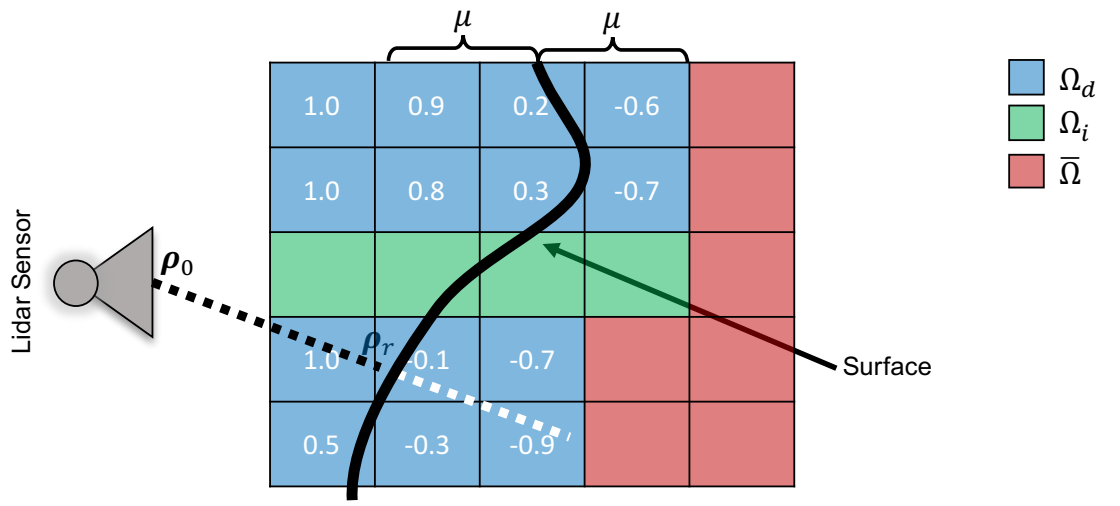


Figure 5.4: A graphical depiction of how a single ray-traced laser is fused into a two-dimensional 'voxel' grid. The TSDF values represent the zero-crossing surface (black line) that has only been partially observed (Ω_d) while the remaining portion of the surface must be interpolated (Ω_i). All voxels with TSDF values are part of Ω_d (blue voxels), voxels near the interpolated surface are in Ω_i (green voxels), and the remaining voxels are in $\bar{\Omega}$ (red voxels). These TSDF values $f \in [-1, 1]$ are a linear mapping from the signed distance a surface is from a given voxel centroid. There is no f value when the signed distance is less than $-\mu$, however when the signed distance is greater than μ (i.e., observed free space in front of a surface) all those voxels are updated with $f = 1$.

To leverage the Ω labels within our regulariser, we subdivide our voxel grid into three distinct sets:

1. Ω_d – These voxels were directly intersected by a laser ray, so the regulariser will only smooth the already-existing data.

2. Ω_i – These voxels were not intersected, but they are in a region in that we will interpolate missing surfaces. For example, a automatic object detector such as [114] indicates where ephemeral objects exist, we then remove the objects and add those areas to Ω_i to interpolate the surfaces. In other words, for each laser ray that intersected an ephemeral object, we extend the ray further outwards to include the background area in Ω_i.

3. $\bar{\Omega}$ – These voxels were neither directly intersected by a laser ray nor were they in a region targeted for interpolation. These voxels are excluded from regularisation.

Therefore, our Ω label domain is defined as $\Omega = \Omega_d \bigcup \Omega_i$. These subsets are graphically depicted in Figure 5.4 on the facing page.

5.3 Potential Interpolation Approaches

In Section 5.1 on page 113, the probabilistic noise-reduction approach presented faced one difficult challenge: What should be done when optimising a region in which no data was directly observed? In other words, how should the energy minimisation problem be posed in regions not directly observed by the sensor?

For noise reduction, the formulation takes the form,

$$E(u) = \int_\Omega ||\nabla u||_1 d\Omega + \lambda \int_\Omega ||f - u||_2^2 d\Omega \tag{5.7}$$

where the data term penalises a final solution that deviates too far from the originally observed data. However, as our goal in this chapter to accurately *interpolate* regions that were occluded by an object, the data term essentially disappears (is zeroed out since there is no f) and therefore only the regularisation term remains.

We evaluate two potential solutions to this problem. First, in Section 5.3.1, we present a naïve approach where the data term remains zero and the TV or TGV term are the sole contributors to the energy minimisation. Second, in Section 5.3.2 on page 120, we modify Equation 5.4 on page 114 with a bipartite likelihood, one computation for smoothing and another for interpolation.

5.3.1 Naïve Likelihood Function

In a naïve approach, denoised and interpolated regions are treated identically in the energy minimisation. However, the input data significantly differs. Equation 5.4 on page 114 can be expanded to detail its form based the voxel's (v operating domain),

(a) Single-Plane Input

(b) Single-Plane TV Naïve Regularisation Interpolation

Figure 5.5: Single-Plane TV Naïve Regularisation: This simple example demonstrates a naïve TV regulariser serving to *interpolate* hidden structure. In this case, the input (a) is a plane with a hole in the centre. The TV regulariser's interpolated surfaces (b) converge on the correct result.

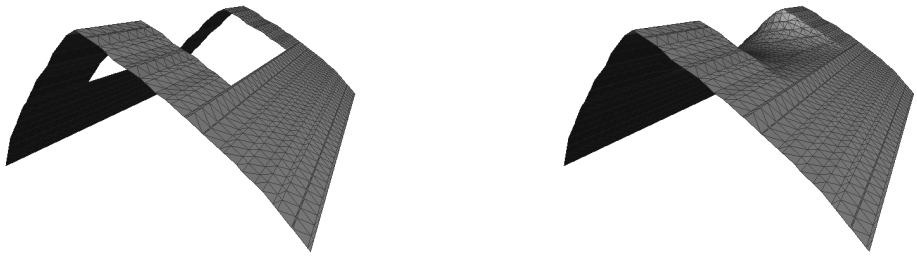
$$
\begin{aligned}
P(f|u,v) &\propto \begin{cases} \exp\left(\frac{||f-u||_2^2}{2\sigma_u^2}\right) & v \in \Omega_d \\ \exp\left(\frac{0}{2\sigma_u^2}\right) & v \in \Omega_i \end{cases} \\
&\propto \begin{cases} \exp\left(\frac{||f-u||_2^2}{2\sigma_u^2}\right) & v \in \Omega_d \\ k_1 & v \in \Omega_i \end{cases}
\end{aligned}
\tag{5.8}
$$

When surface data (f) is directly observed ($v \in \Omega_d$), the data term takes form of a normal distribution centred about the average TSDF value. A penalty ($||f-u||$) is applied when the estimated ("denoised") TSDF value differs from the average observed value.

However, when the surface data is occluded ($v \in \Omega_i$) and the surface must be interpolated, there is no observed f. Without a meaningful penalty to apply for any given estimated (u) TSDF value, the numerator in the exponential becomes 0, and the likelihood is a constant k_1: all states are equally probable.

The "minimisation" version of Equation 5.8 now becomes,

$$
\begin{aligned}
\tilde{\delta}(u,f) &\propto \begin{cases} ||f-u||_2^2 & v \in \Omega_d \\ -\ln(k_1) & v \in \Omega_i \end{cases} \\
&\propto \begin{cases} ||f-u||_2^2 & v \in \Omega_d \\ k_2 & v \in \Omega_i \end{cases}
\end{aligned}
\tag{5.9}
$$



(a) Two-Plane Input

(b) Two-Plane TV Naïve Regularisation Interpolation

Figure 5.6: Two-Plane TV Naïve Regularisation: A slightly more complicated scenario than Figure 5.5 on the facing page, but more representative of the two-plane intersections dominant in urban reconstructions. In this case, the missing data in the centre of the plane intersection (a) causes difficulty in the interpolated result (b). The final converged solution is approximately an average of all possible solutions—thus creating a saddle-shaped interpolated surface.

Since k_2 is a constant, in the minimisation there is effectively no data-term penalty for unobserved voxels. The energy function becomes,

$$E(u) = \int_{\Omega} ||\nabla u||_1 d\Omega + \lambda \int_{\Omega} \tilde{\delta}(u, f) d\Omega \tag{5.10}$$

Before performing large-scale experiments, we chose to evaluate this approach on a simple synthetic scenarios. Planes dominate 3D urban environments. Therefore, it stands to reason that if the interpolation does not perform well when portions of one or two planes are occluded in synthetic data, it will not perform well on even more complex, noisier real-world data.

We first evaluate the result of occluding a part of just a single plane, as depicted in Figure 5.5 on the preceding page. The implementation (Section 5.4 on page 123) of the naïve TV approach (Equation 5.10) is quite good—the occluded region is correctly interpolated as a continuous plane (Figure 5.5 on the facing page).

However, urban structure is composed of more than just a single plane. Therefore, in our second evaluation, we show the more typical case of two planes intersecting at a 90° (Figure 5.6). Again, we occlude the interior portion of the intersecting planes—this type of scenario occurs quite frequently in our real-world datasets

(Section 5.5 on page 126) when a car parked on the side of the road occludes the pavement-building plane intersections.

The naïve TV approach breaks down in this scenario. The final interpolated surface (Figure 5.6 on the previous page) saddle-shaped. Essentially, the TV regulariser approximates the average of all possible planes that could form a convex hull to fill the occluded region. Even switching to a TGV regulariser [116][75] only slightly dampens the saddle, but the saddle is still clear in the final solution.

However, as presented in the next section, it is possible to significantly improve the performance of the optimiser by incorporating knowledge of the neighbouring surface data. Rather than diffusing this data through a gradient term, we can explicitly encode it in the data term.

5.3.2 Proposed Likelihood Function

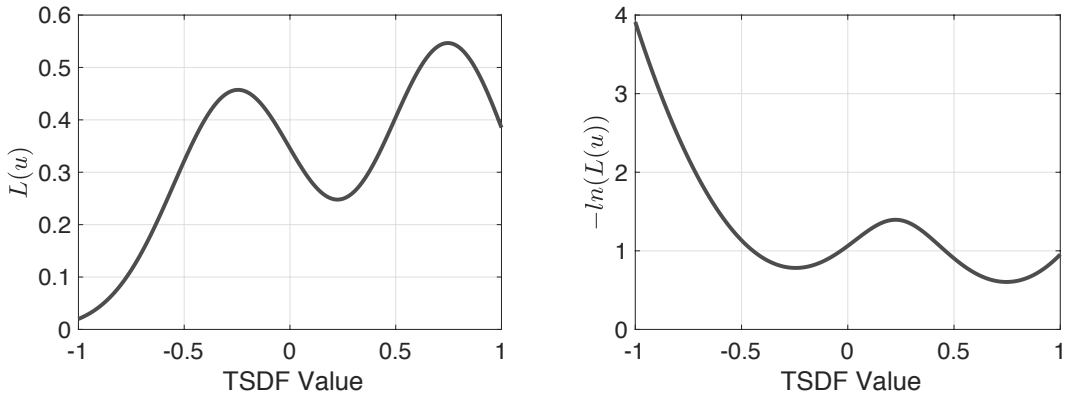
The naïve approach presented in the previous section only propagated neighbouring surface data through a gradient term. The optimiser converged upon a saddle-shaped solution, shown in Figure 5.6 on the preceding page, when presented with an occluded 90° two-plane intersection.

A human visually evaluating the solution sees it is incorrect because it does not match the form of the non-occluded surfaces. In this section, we seek to explicitly encode that knowledge into the data term of the energy function in a probabilistic manner.

As described previously, for voxels where TSDF data (f) is missing (i.e., Ω_i), we cannot calculate the corresponding $P(f|u)$. Our new data term adds a KCDE likelihood estimator for the interpolated voxels (Ω_i),

$$P(f|u,v) \propto \begin{cases} \exp\left(\frac{||f-u||_2^2}{2\sigma_u^2}\right) & v \in \Omega_d \\ L(u) & v \in \Omega_i \end{cases} \tag{5.11}$$

where v is the current voxel and $L(u)$ is a KCDE that provides a prior data term to guide the regularisation as it interpolates surfaces [117],



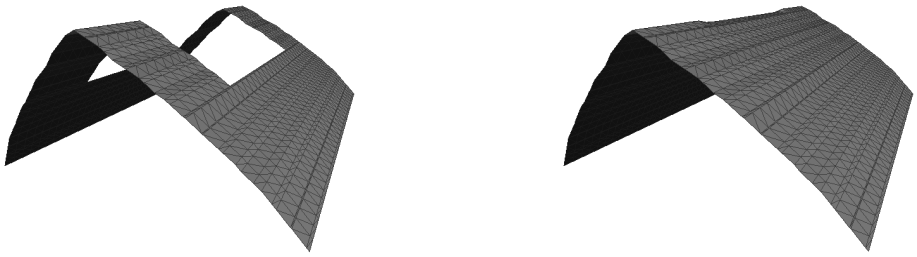
(a) KCDE Likelihood (b) KCDE Log-Likelihood

Figure 5.7: Example KCDE likelihood plots. These plots provide an illustration of the TSDF likelihood distribution for a single voxel in a simple scenario where two neighbours (hence a bimodal distribution) contain valid surface observations. The neighbour voxels contained TSDF values of 0.25 and 0.75 and were respectively 10 and 8 voxels away. As the 0.25 TSDF value is 2 voxels closer, it has a higher likelihood than the 0.75 neighbour. The log-likelihood plot (b) is the inverse of the likelihood that is used in the energy-minimisation optimisation formulation. ($\sigma_\alpha = 0.3$, $\sigma_\beta = 10$)

$$
\begin{aligned}
L(u) &= \frac{\sum_{i=1}^{M} K_u(u - u_i) K_d(d_i)}{\sum_{i=1}^{M} K_d(d_i)} \\
K_u(\alpha) &\propto \exp\left(\frac{-\alpha^2}{2\sigma_\alpha^2}\right) \\
K_d(\beta) &\propto \exp\left(\frac{-\beta^2}{2\sigma_\beta^2}\right)
\end{aligned}
\tag{5.12}
$$

where M is the number of cardinal neighbouring voxels in Ω_d, u_i is the neighbour's TSDF value, and d_i is the distance between the current voxel and the neighbour. The $K_u(\cdot)$ kernel incorporates the neighbour's TSDF value while $K_d(\cdot)$, in concert with the denominator, provides a confidence in the neighbour's value based on the distance between voxels. This can be thought of as a memory-efficient method to accomplish a similar effect as the approach described in [38].

Figure 5.7 provides an example $L(U)$ for a single voxel with only two neighbours with valid TSDF values. Two valid neighbours results in a bimodal distribution with the nearest neighbour's TSDF peak higher than the other peak. This plot makes



(a) Two-Plane Input

(b) Two-Plane TV KCDE Regularisation Interpolation

Figure 5.8: Two-Plane TV KCDE Regularisation: Using the same two-plane input (a) as Figure 5.6 on page 119, our proposed KCDE data term successfully (b) interpolates reasonable surfaces in the interpolated region. Note the stark contrast between this solution (b) and the naïve approach's saddle-shaped solution in Figure 5.6 on page 119. Further analysis with real-world, multi-plane (up to six intersecting planes) scenarios are presented in Section 5.5 on page 126.

it clear that the KCDE formulation elegantly incorporates an arbitrary number of neighbour voxels, applies appropriate weights (based on distance), and then provides a single, meaningful metric that can be used to inform the energy data term.

The minimisation version (Figure 5.7 on the preceding page) of Equation 5.11 on page 120 now becomes,

$$\delta(u, f) = \begin{cases} ||f - u||_2^2 & v \in \Omega_d \\ -\ln(L(u)) & v \in \Omega_i \end{cases} \tag{5.13}$$

that results in the energy function,

$$E(u) = \int_\Omega ||\nabla u||_1 d\Omega + \lambda \int_\Omega \delta(u, f) d\Omega \tag{5.14}$$

However, the KCDE term makes $\delta(u, f)$ non-convex and cannot be solved with traditional techniques. We therefore approximate it with a second-order Taylor Series expansion that guarantees convergence when the Hessian is a positive-semidefinite matrix [118]. In addition, the non-convexity of the data term–as illustrated in the example in Figure 5.7 on the preceding page–contains multiple local minima that we account for in Section 5.4.1 on page 125.

As in the previous section, we now evaluate this new TV KCDE optimisation approach with an occluded synthetic two-plane intersection. The input and output of this approach are shown in Figure 5.8 on the facing page. Note the stark contract between Figure 5.8 on the preceding page (this KCDE approach) and Figure 5.6 on page 119 (the naïve approach). In the naïve approach, there was a distinct saddle-shape to the interpolated surfaces. In this improved KCDE approach, the additional neighbour information provided in the data term helps guide the optimiser to a more reasonable solution.

Based on this modest success on synthetic data, we next discuss implementation details before presenting the results of applying this KCDE approach to real-world data.

5.4 Implementation

Figure 5.2 on page 111 is a summary of our overall software pipeline, outlining the steps required to process a sequence of laser scans to ultimately generate an interpolated dense model. In this section, we describe the algorithm to solve Equation 5.14 on the facing page that is required in the last stage of our software pipeline. The optimisation implementation is similar to that presented in Section 3.3 on page 59, however, the KCDE introduces enough differences to justify explicitly enumeration of the process.

The L_1 term in Equation 5.14 on the facing page is not differentiable; therefore, it cannot be minimised using standard techniques. We use the Legendre-Fenchel Transform [76] [77] to transform it into a differentiable form,

$$\underset{u}{\operatorname{argmin}} \int_{\Omega} ||\nabla u||_1 d\Omega = \underset{u}{\operatorname{argmin}} \underset{||\mathbf{p}||_\infty \leq 1}{\operatorname{argmax}} \int_{\Omega} u \nabla \cdot \mathbf{p} d\Omega \tag{5.15}$$

where the primal scalar u is the current denoised and interpolated TSDF solution and $\nabla \cdot \mathbf{p}$ is the divergence of the dual vector field $\mathbf{p}$ defined as $\nabla \cdot \mathbf{p} = \nabla p_x + \nabla p_y + \nabla p_z$. Applying this transformation to Equation 5.14 on the preceding page, the original energy minimisation problem now becomes a saddle-point (min-max) problem with a new dual variable $\mathbf{p}$ along with the original primal variable u,

$$\underset{u}{\operatorname{argmin}} \underset{||\mathbf{p}||_\infty \leq 1}{\operatorname{argmax}} \int_\Omega u \nabla \cdot \mathbf{p} + \lambda \int_\Omega \delta(u, f) d\Omega \tag{5.16}$$

The solution to this regularisation problem is found with a Primal-Dual optimisation algorithm [77] that we briefly summarise in the following steps:

1. $\mathbf{p}$, u, and $\hat{u}$ are initialised to 0. $\hat{u}$ is a temporary variable that reduces the number of optimisation iterations required to converge.

2. To solve the maximisation, we update the dual variable $\mathbf{p}$,

$$\begin{aligned} \mathbf{p}_k &= \frac{\tilde{\mathbf{p}}}{\max(1, ||\tilde{\mathbf{p}}||_2)} \\ \tilde{\mathbf{p}} &= \mathbf{p}_{k-1} + \sigma_p \nabla \hat{u} \end{aligned} \tag{5.17}$$

   where σ_p is the dual variable's gradient-ascent step size.

3. For the minimisation problem, the primal variable u is updated by,

$$u_k = \begin{cases} \psi(\tilde{u}, w, f) & v \in \Omega_d \\ \xi(\tilde{u}, u_{k-1}, w, f) & v \in \Omega_i \end{cases} \tag{5.18}$$

   with,

$$\begin{aligned} \tilde{u} &= u_{k-1} - \tau \nabla \cdot \mathbf{p} \\ \psi(\cdot) &= \frac{\tilde{u} + \tau \lambda w f}{1 + \tau \lambda w} \\ \xi(\cdot) &= \frac{\tilde{u} + \tau \lambda_i (\delta''(u_{k-1,\cdot}) u_{k-1} - \delta'(u_{k-1}, \cdot))}{1 + \tau \lambda_i \delta''(u_{k-1}, \cdot)} \end{aligned} \tag{5.19}$$

   where τ is the gradient-descent step size and w is the weight of the f TSDF value.

   The $\xi(\cdot)$ calculation is a numerical approximation of the second-order Taylor series expansion of $\delta(u, f)$ from the Ω_i term in Equation 5.13 on page 122. This uses a modified λ_i that is further described in Section 5.4.1 on the next page. To meet the positive-semidefinite matrix constraint, since $\delta''(u_0, \cdot)$ is a scalar, we restrict its value to ≥ 0.

4. Finally, to converge in fewer iterations, we apply a "relaxation" step,

$$\hat{u} = u + \theta(u - \hat{u}) \tag{5.20}$$

where θ is a parameter to adjust the relaxation step size.

While the derivation may seem complex at first glance, the preceding four steps are all that a user must implement to take advantage of our approach.

5.4.1 Adaptive Regularisation

As previously mentioned, the non-convexity of the KCDE data term contains multiple local minima. We initially set λ_i to a small value and then increase it after the regulariser converges.

Initially setting λ_i to a value orders of magnitude lower than λ ensures the KCDE data term only guides rather than dominates the regularisation process. This effectively removes the non-convexity of the energy functional and allows the regulariser to select solutions outside of the current local minimum. Once the regulariser converges, λ_i is increased to further refine the solution. This two-stage process minimises the global energy and therefore reduces the impact of non-convexity of the KCDE data term.

Our approach of utilising different values for λ_i, based on confidence in the local data, was inspired by image pyramids and the 2D depth-map regularisation in [38].

Table 5.1: Scenario error statistics

#	Median Error (cm)	σ (cm)	Eval Dims (m)	Vol. (m^3)
1	8.48	6.00	3.8 x 2.8 x 11.4	121.3
2	9.24	6.08	3.7 x 2.1 x 10.7	83.1
3	5.64	4.64	5.6 x 2.8 x 22.9	359.1
4	5.97	4.57	3.8 x 3.7 x 13.8	194.0

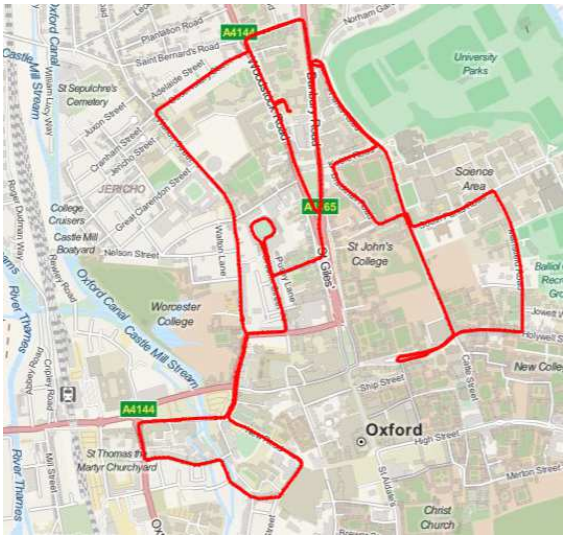


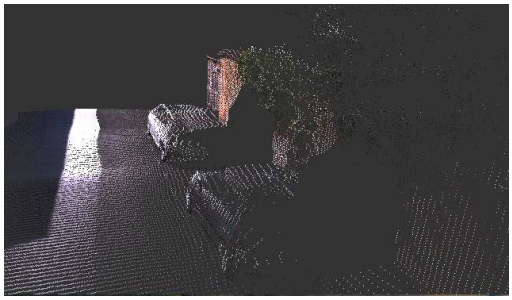
Figure 5.9: The 10 km data collection route was traversed twice: once for reference with few vehicles on the roads and a second time for evaluation data with a large variety of vehicles visible. Map generated with OpenStreetMap [119].

5.5 Results

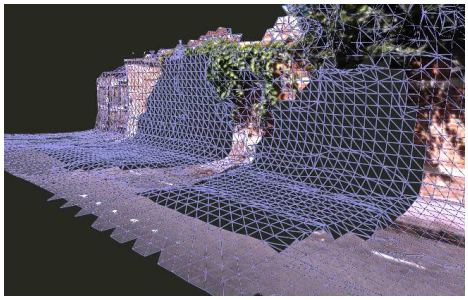
To evaluate the performance of our algorithm, we ran an automated ephemeral object detector on a 10 km data-collection route in Oxford, UK, as depicted in Figure 5.9. Our vehicle had a Bumblebee 2 stereo camera for VO [18]; a Ladybug 2 for colouring the laser points [94] and to add textures to the dense reconstructions; and a SICK LMS-151 2D laser oriented for push-broom collection.

The data-collection route was driven on two separate occasions. The reference data run was accomplished before dawn to minimise the number of vehicles parked on the roadside. The evaluation data run took place near noon on the same day, a time where there were a maximum number of vehicles on the roads or parked nearby.

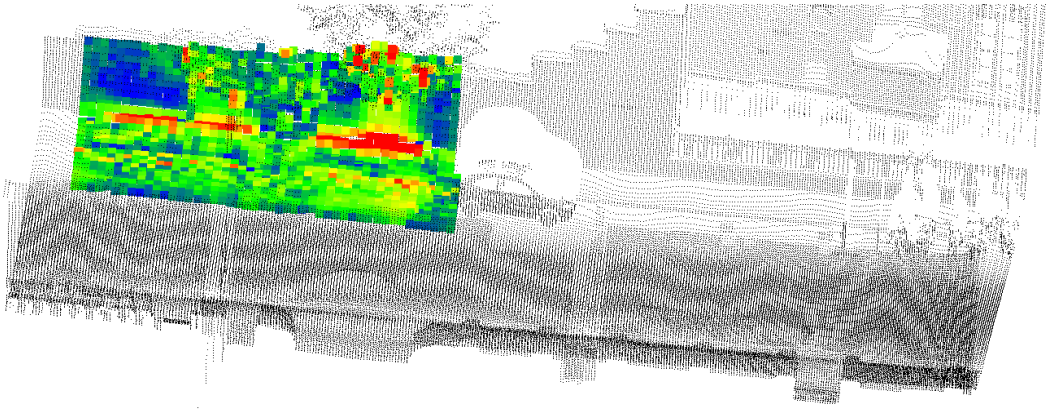
We selected four 30 m segments from the two routes to access our dense reconstruction. The segments contained two or more vehicles in the second data run with no overlapping vehicles in the reference data run. This allowed us to automatically detect cars in the second data run, remove them, apply our dense reconstruction technique, and then compare our results with the original data run. We used the object detector described in [114].



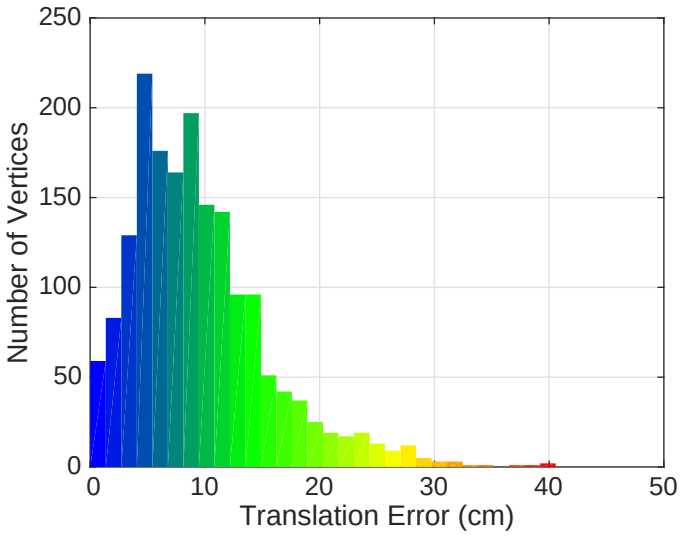
(a) The raw point cloud used as the input to our regularisation algorithm.



(b) The dense reconstruction with infilled background data. Since there is no texture data for the obstructed background, a triangle mesh is overlaid to visualise the reconstruction's 3D model.

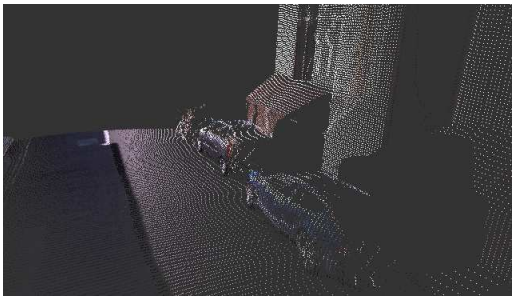


(c) A point cloud of the translation errors for the dense reconstruction. Each colour in this figure correspond to the same translation error in (c).

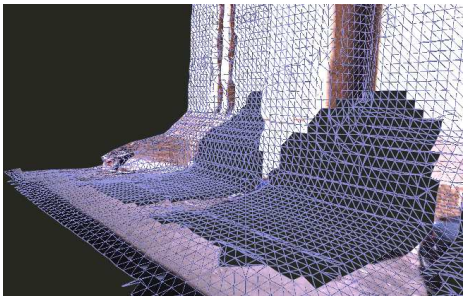


(d) The distribution of the translation error for the isosurface extracted from the dense reconstruction.

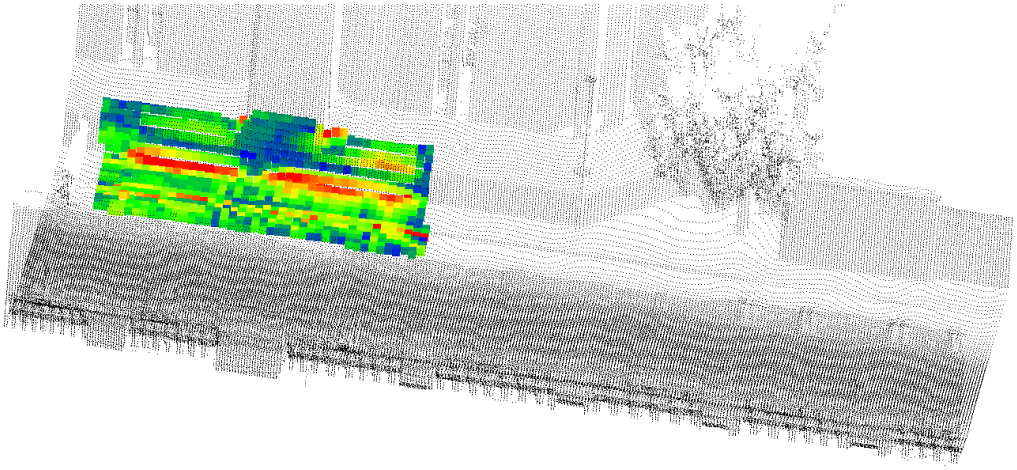
Figure 5.10: Scenario 1: Reconstruction of area behind two automobiles where the surrounding laser data includes a tree (top right of reconstructed area). Error: median = 8.48 cm, σ = 6.00 cm



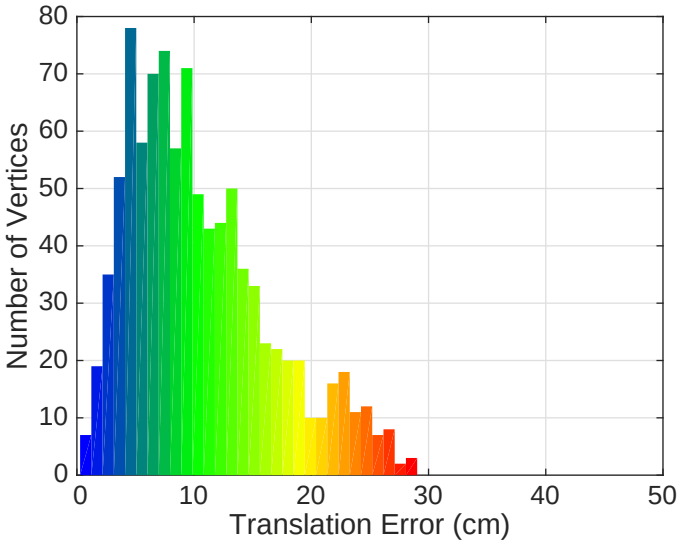
(a) The raw point cloud used as the input to our regularisation algorithm.



(b) The dense reconstruction with infilled background data. Since there is no texture data for the obstructed background, a triangle mesh is overlaid to visualise the reconstruction's 3D model.



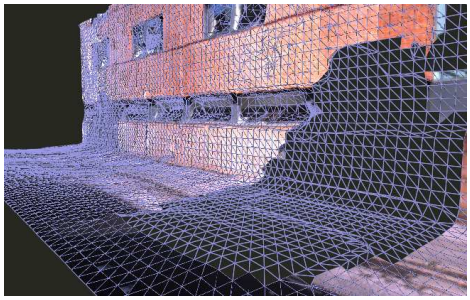
(c) A point cloud of the translation errors for the dense reconstruction. Each colour in this figure correspond to the same translation error in (c).



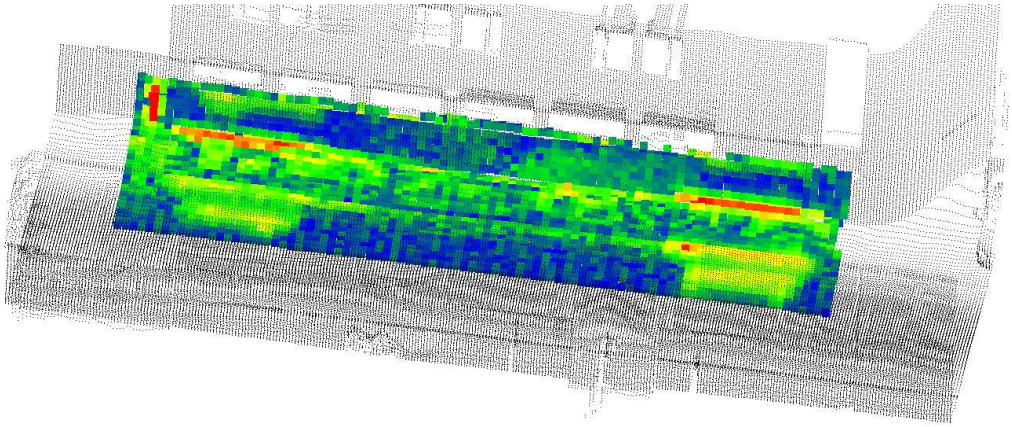
(d) The distribution of the translation error for the isosurface extracted from the dense reconstruction.

Figure 5.11: Scenario 2: Reconstruction of area behind two automobiles where six separate intersecting planes must be interpolated. Error: median = 9.24 cm, σ = 6.08 cm

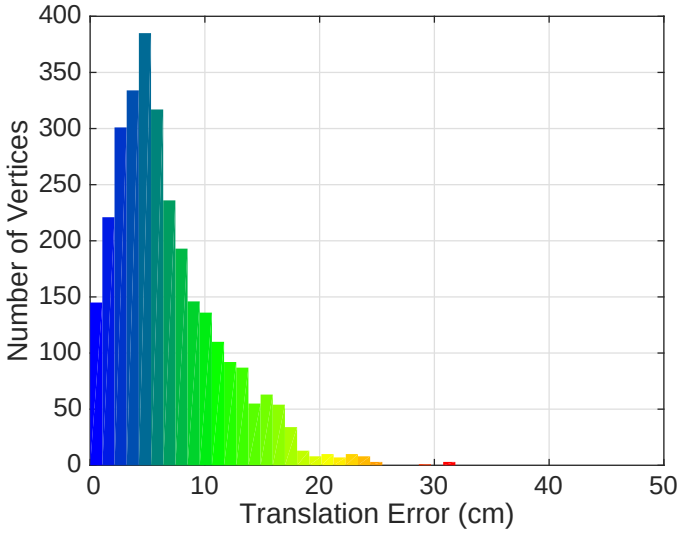
(a) The raw point cloud used as the input to our regularisation algorithm.



(b) The dense reconstruction with infilled background data. Since there is no texture data for the obstructed background, a triangle mesh is overlaid to visualise the reconstruction's 3D model.

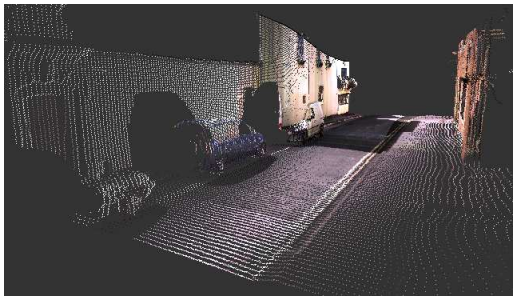


(c) A point cloud of the translation errors for the dense reconstruction. Each colour in this figure correspond to the same translation error in (c).

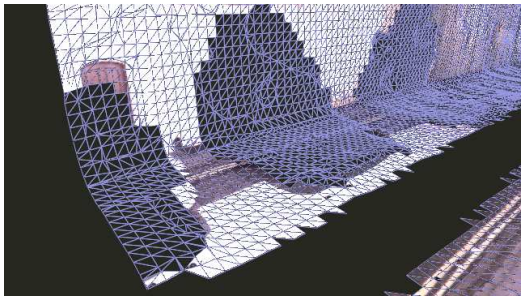


(d) The distribution of the translation error for the isosurface extracted from the dense reconstruction.

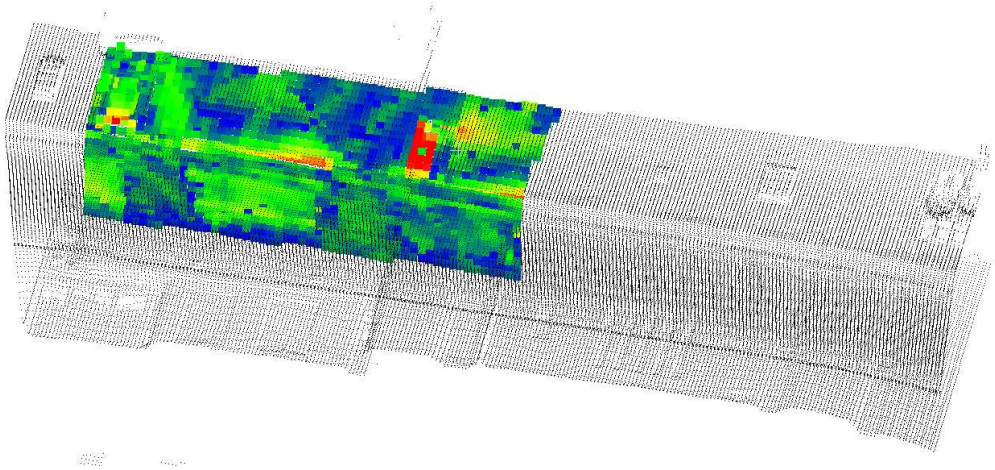
Figure 5.12: Scenario 3: Reconstruction of area behind two automobiles in a typical two-plane-dominated street environment. Error: median = 5.64 cm, σ = 4.64 cm



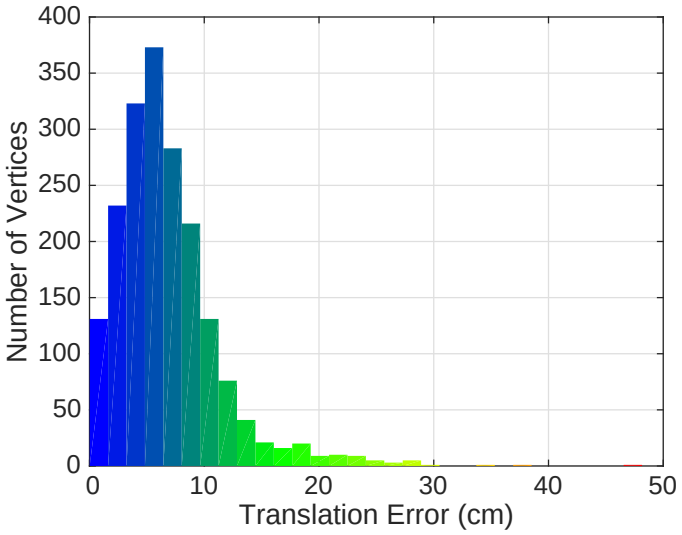
(a) The raw point cloud used as the input to our regularisation algorithm.



(b) The dense reconstruction with infilled background data. Since there is no texture data for the obstructed background, a triangle mesh is overlaid to visualise the reconstruction's 3D model.



(c) A point cloud of the translation errors for the dense reconstruction. Each colour in this figure correspond to the same translation error in (c).



(d) The distribution of the translation error for the isosurface extracted from the dense reconstruction.

Figure 5.13: Scenario 4: Reconstruction of area behind three automobiles in a typical two-plane-dominated street environment. Error: median = 5.97 cm, σ = 4.57 cm

As in Section 3.6 on page 68, the ground-truth data ($\boldsymbol{M}_\ell$) was generated by composing the VO-estimated poses, interpolating the correct laser-to-world pose ($\boldsymbol{T}_{w\ell} \in \mathbb{SE}(3)$) for each 2D push-broom laser scan, and plotting the resulting scan (in 30 m segments) in the correct world position. To this end, we utilise the city-scale, sparse point-cloud method created by Stewart in [94]. The resulting 3D point cloud was used to access the accuracy of our reconstructions.

From the output of our reconstruction algorithm, we extracted a point-cloud sampling of the zero-crossing isosurface in our voxel grid [62] and compared this to the reference data with CloudCompare [65]. These results are quantitatively summarised in Table 5.1 on page 125, Figure 5.10 on page 127, and Figure 5.13 on the facing page.

Each automobile removed required approximately 20 m^2 of surface area to be interpolated by the regulariser. Our reconstructions had median accuracies ranging from 5.64 cm - 9.24 cm with standard deviations between 4.57 cm - 6.08 cm.

Based on the synthetic results presented in Section 5.3.2 on page 120, we are not surprised at the highly accurate reconstruction on the simple reconstruction scenarios. In these typical city-street environment were the reconstruction is dominated by two planes (the pavement and building façade) the interpolated reconstructions (Figure 5.12 on page 129 and 5.13 on the preceding page) contained the least amount of error. However, our system even performed well interpolating more complex structures such as trees (Figure 5.10 on page 127), six separate intersecting planes (Figure 5.11 on page 128), or features (e.g., door or window) obscured by the removed vehicle (Figure 5.13 on the facing page).

Previous work [78] demonstrated other regularisation terms can produce better results than TV. However, once the KCDE data term was added to our pipeline, the TV and TGV regularisers converged upon nearly identical solutions - therefore the additional computational and memory requirements of TGV perhaps are not justified.

Finally, while the algorithm presented in this chapter worked well for the most common scenarios we encountered in our datasets, there are areas in which it would struggle—most notable in the selection of the "neighbour" voxels that guide the

regulariser. For example, if an automobile is parked in front of an alleyway, then the "neighbours" would include the orthogonal walls on either side of the alley, the pavement below, and the free space in the alleyway. Our algorithm converges upon the "neighbour" that results in a low-energy solution (i.e., minimise gradients with respect to the originally-observed surfaces), but this would not be desirable if the regulariser creates a wall where there should be free space. If this is a common occurrence in a given application or dataset, then one must pursue an alternative hole-filling strategy. We leave it to future work to address these potential limitations.

5.6 Conclusions

In this chapter, we presented our new approach that reconstructs large-scale environments from laser scans, removes ephemeral objects, and then back-fills the gaps in the laser data. We created accurate reconstructions by fusing laser data into a voxel grid and isolating the regions for TV regularisation (utilising Ω labelling method presented in Chapter 3) with a KCDE data term to assist when interpolating the missing surfaces. When reconstructing approximately 20 m^2 of surface area, our proposed method's median accuracy was under 10 cm with a standard deviation of 6 cm or less.

The high degree of accuracy achieved in detecting sensor occlusions (e.g., pedestrians, bicycles, automobiles, etc.) and interpolating the hidden surfaces is an enabler for autonomous-driving applications. In concert with the large-scale system presented in Chapter 4, we envision this "hole-filling" system being used as part of an automobile's "dream state". After returning from a day of driving around the city, the dense 3D maps are updated to remove all ephemeral objects thus revealing the underlying urban structure.

After removing ephemeral objects and interpolating the background structure, it may be desirable to further improve the reconstruction quality by identifying and removing error-prone regions of the dense reconstruction. In Chapter 6, we present a machine-learning approach to help identify these areas.

In God we trust. All others must bring data.

— W. Edwards Deming

6

Towards (Deep) Learning Where Dense Reconstructions Go Wrong

Abstract

Dense reconstructions contain errors. We sought to minimise these errors in previous chapters by using high quality sensors and regularising the output, but errors still persisted. In this chapter, we present early-stage results around a machine learning technique to identify and highlight the error-prone regions of reconstructions. This enables a system pipeline to remove or repair these regions to improve the reconstruction accuracy.

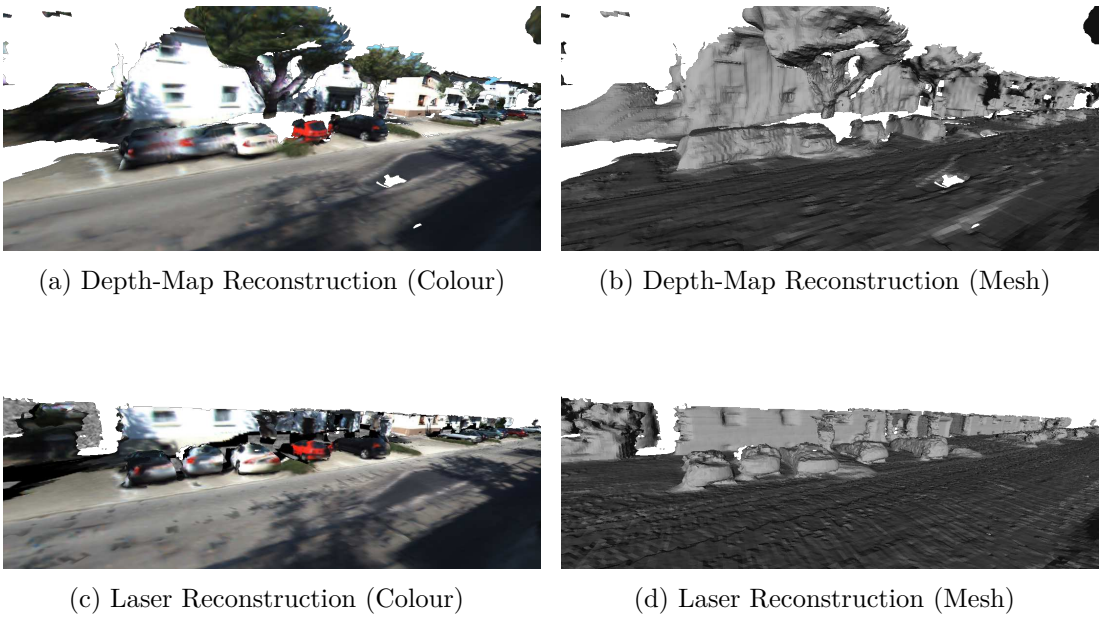
We trained and evaluated our network with data from the BOR^{2}G dense reconstruction pipeline. High-quality laser reconstructions provide ground truth against which we compare our depth-map reconstructions. We train on a 3.8 km sequence from the KITTI dataset. Through qualitative analysis of a separate 2.2 km KITTI sequence, we demonstrate the network correctly identifies the most error-prone regions in the depth-map reconstruction. In addition, we provide a quantitative benchmark against which future work may compare.

Contents

PREVIOUS chapters sought to create accurate 3D maps by regularising the camera depth maps, using high-quality sensors, and regularising the dense-reconstruction output. However, as seen in the results presented in Chapter 4, the reconstructions still contain errors. In this chapter, we present a method to identify the most defect-prone portions. Our goal is to facilitate the removal or repair of these mesh regions to improve overall mapping accuracy.

Defect detection has been a robust area of research for the past 25 years. In 1993, Abe proposed an automated system to detect cracks in real time using a custom-built image processor [120]. This was a significant improvement for the industry; previously they relied solely upon human evaluation of images, whereas the new system improved efficiency by highlighting high-probability "crack" images. Later research by Kumar sought to apply similar 2D image processing and filtering techniques to identify defects in manufacturing processes—specifically targeted to textile, plastic, and paper manufacturing [121].

In the 2000s, research transitioned from analysing individual 2D images to creating a feature-rich 3D reconstruction. The additional spacial information enabled the application of mesh-quality metrics based on explicitly-encoded geometric and gradient priors [122]. Civil architecture applications drove the research during this period. Gordon sought to evaluate the accuracy of a building under construction by comparing the "as-built" 3D laser model to the design blueprints [123]. Natural disaster Building Damage Assessment (BDA) applications required an automated mobile-robotics system to survey buildings and identify regions requiring repair [124][125][126]. These methods heavily relied upon computer vision approaches



(a) Depth-Map Reconstruction (Colour)

(b) Depth-Map Reconstruction (Mesh)

(c) Laser Reconstruction (Colour)

(d) Laser Reconstruction (Mesh)

Figure 6.1: Depth-Map vs. Laser Reconstruction Comparison. A human operator, if given a set of colour and mesh images for a reconstruction, can intuitively recognise the regions that likely contain errors. In the top example ((a) and (b)) one can recognise the "smearing" between the cars and the holes in the road are incorrect. The goal of this chapter is to train a neural network to automatically recognise these regions in depth-map reconstructions by training it on data from laser reconstructions (e.g., (c) and (d)).

to create a 3D model (sparse or dense) of the environment from a sequence of images. An overview of these civil architecture defect detection approaches may be found in Koch's 2015 survey paper [127].

We base our approach on a convolutional neural network using feature data generated from both laser-only and depth-map-only dense reconstructions. As demonstrated in Chapter 4, though laser reconstructions usually cover less surface area, they create more accurate reconstructions and therefore serve as our ground-truth when training the network. We use the laser reconstructions to train the network to evaluate the quality of the depth-map reconstructions. A simple example of this process is shown in Figure 6.1 which compares the same scene reconstructed by laser and camera input data. A human operator, after observing the quality of the laser reconstruction's colour and mesh images, could readily identify regions in the camera reconstruction that are likely incorrect—e.g., holes in the road and the "smearing" area between the automobiles. In this chapter, we present our method

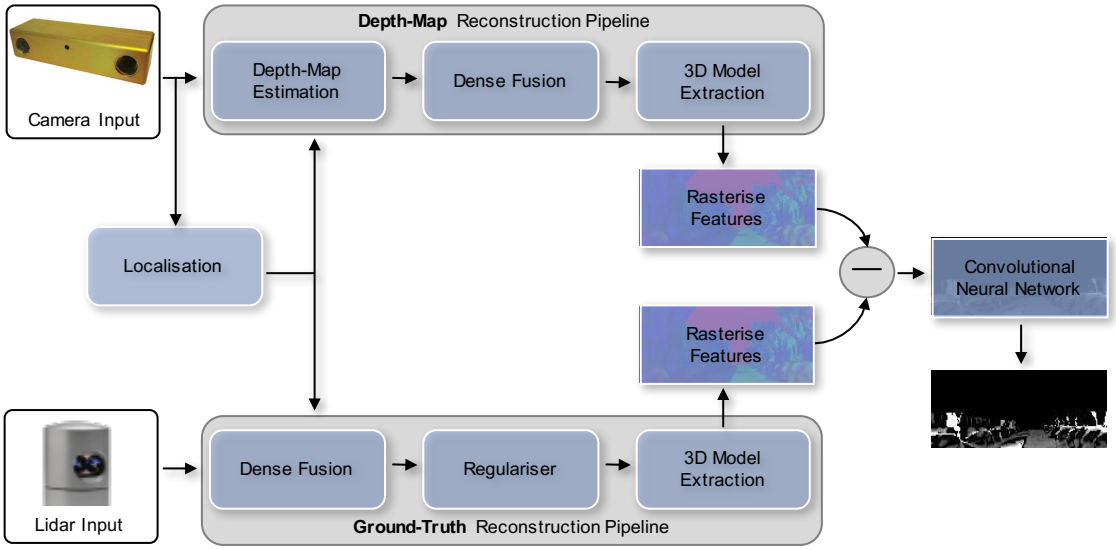


Figure 6.2: Machine Learning Data-Flow Pipeline. To train our network, we create separate depth-map and laser dense reconstructions (using the techniques discussed in Chapter 4). We create ground-truth data by subtracting the rasterised depth maps from each reconstruction. The neural network trains on rasterised feature images (see Figure 6.3 on page 138) to learn to generalise the ground-truth error in new scenarios.

to train a neural network to recognise and highlight these regions.

Machine learning, in particular deep neural networks, has permeated the computer vision community over the past decade due to advances in GPGPU computing and technique. With just image inputs, neural networks have been trained to identify a 6-DoF pose within a small-town-sized region [128], create 3D models of objects [129][130], and estimate per-pixel depth [40][41][131]. Our contribution is *the proposal of the dense reconstruction and machine learning system pipeline that enables the network to identify poorly-reconstructed regions of a given 3D model* and to point the way to future dense reconstruction repair work. Rather than focus on the dense-reconstruction system's inputs (e.g., laser or image), we operate directly on its output (i.e., 3D model). In essence, this is a dual of our work in Chapter 3 and Chapter 4 since we are now focused on learning the objective function rather than adapting a regulariser to improve reconstruction performance.

6.1 System Pipeline

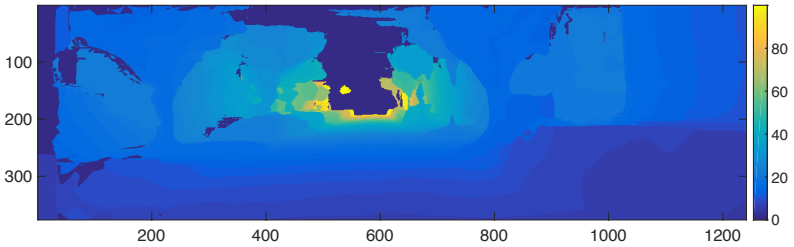
Our system pipeline to train the neural network is presented in Figure 6.2 on the facing page. We create two dense reconstructions: one with camera-only (depth-map) inputs and another with laser inputs. We then extract the 3D model from each reconstruction pipeline and rasterise feature images (Section 6.2). Since the active laser sensor is significantly more accurate than the passive cameras, the laser feature images are used as the ground-truth reference (Section 6.3 on page 139) to train our convolutional neural network (Section 6.4 on page 139).

6.2 Feature Creation

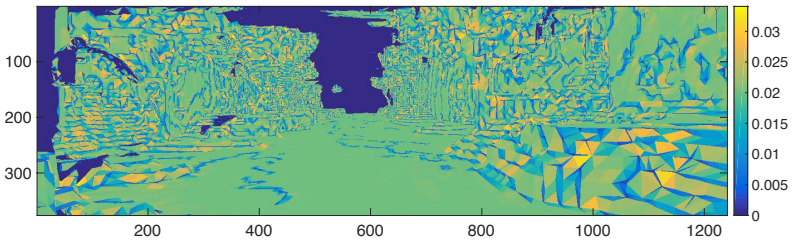
Much work in the deep learning literature relies upon simple RGB images as the input [40][41][131]. However, as our problem formulation assumes a dense 3D model is available, we can extract a much richer set of features from the reconstruction. High-quality features are important as they result in a more robust network that takes less time to train because the network has less to learn. In our scenario where we wish to find *errors* in the reconstruction, if we only provided an RGB image to the network, it might first learn to infer the correct depth before learning to predict the errors. However, if we directly provide the estimated depth as a feature input, the initial learning time is reduced.

We use GLSL to "fly" a virtual camera throughout the 3D models to extract features. Our `vertex` shader transforms the 3D model's vertices into the camera reference frame (Section 2.2.1 on page 15) to compute the depth of each vertex and the normalised camera vector. These camera-frame points are then grouped into triangles and passed along to a `geometry` shader that computes the triangle's area, surface normal, and edge lengths. Finally, the `fragment` shader processes each of the preceding feature values and rasterises them into individual pixels in feature images, as shown in Figure 6.3 on the next page. As a post-processing step, we normalise all feature vectors to approximately fit within the range $[0, 1]$. This GLSL process enables us to extract 11 unique feature images: 4 per-pixel

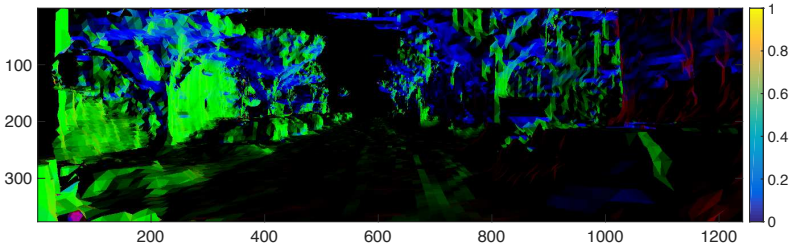
(a) Colour Reconstruction



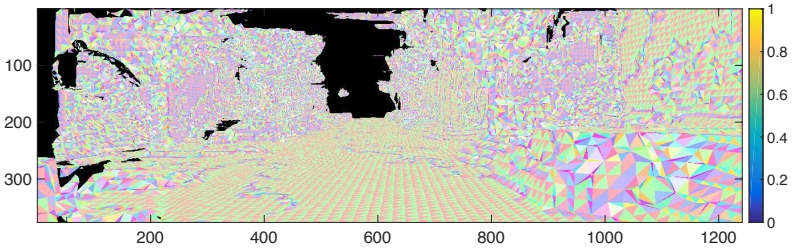
(b) Pixel Depth (m)



(c) Triangle Area (m^2)



(d) Triangle Surface Normal



(e) Triangle Edge Length Ratios

Figure 6.3: Example Input Features. As our dense reconstructions provide a 3D model of our operating environment, we can extract a variety of features that are useful to train our neural network. Pictured above are example features (colour reconstruction, pixel depth, triangle area, triangle surface normal, and triangle edge length ratios) our GLSL pipeline currently extracts. The per-triangle features significantly improve the feature quality as they directly provide contextual and geometric information from which the network can learn higher-order features.

features (RGB and depth) and 7 per-triangle features (area, surface normal, and edge lengths). The triangle-based features are critically important as they provide both contextual and geometric information as patches in the image. This propagates swaths of meaningful data throughout the image, enabling the network to learn higher-order image and scene features.

6.3 Ground-Truth Data

Two identical virtual camera trajectories are used to create a sequence of feature images from both the depth-map and laser reconstructions. However, as the goal is to train a neural network to recognise errors in the depth-map reconstruction, we must first compute *ground-truth errors* with which to train the network. This ground truth must characterise differences between estimated depth values in the laser and depth-map reconstructions. Rather than directly differencing depths, we instead elect to use inverse depth maps so that small errors are heavily penalised by the network. Specifically, we compute the per-pixel ground-truth (δ_{gt}) as,

$$\delta_{gt} = A \left\| \frac{d_i - d_\ell}{d_i d_\ell} \right\|_1 \tag{6.1}$$

where d_i and d_ℓ are the depth-map and laser reconstruction per-pixel depth features, respectively, and A is a constant. In disparity images from stereo cameras, $A = f_x b$ where f_x is the camera's x focal length and b is the baseline distance between the left and right image sensors (Section 2.2.1 on page 15). However, since our approach is agnostic of the input sensor, A is arbitrary but should be standardised when evaluating performance (Section 6.5 on page 143).

6.4 Network Architecture

When developing a machine learning implementation, one must consider four vital components: (1) input features, (2) network architecture, (3) robustness factors, and (4) cost function.

As discussed in Section 6.2 on page 137, our 3D models of the operating environment enable us to extract a much richer feature set than usually available

in image-based machine-learning applications. Of particular utility are the *depth* and triangle-based features since they directly provide the network with geometric and contextual information about the scene.

We down-selected from 11 to 3 features to reduce the time required to train the network. By providing training data to human evaluators, we found the most features most "intuitive" to help detect errors the inverse depth, intensity, and camera-to-surface angle. The inverse depth map directly provides information about the estimated 3D structure of the image. This, in concert with the gradients of an intensity image and the viewing angle of the observed surfaces, were the most salient features to allow the evaluators to estimate the errors after a very short training period—thus making it more likely that a deep neural network can recognise similar patterns if provided enough training data. The evaluators found the triangle edge length to be the least useful since, as a result of the voxel representation and Marching Cubes surface extraction, all triangles were similarly shaped and sized.

As it is the current state-of-the-art neural-network approach to generate depth maps from an image, we selected Lania et al.'s Fully Convolutional Residual Network [131] as the basis for our network architecture. Lania's network, a high-level representation of which is shown in Figure 6.4 on the next page, was originally designed to infer per-pixel depth from a single RGB image, a task which is highly correlated to our aim to compute estimated depth *error* given a set of feature inputs. The general approach is to pass the input feature images through a series of decreasing-sized convolutional layers. Each convolutional layer performs small convolutions (1×1 and 3×3) that are added as features for the next layer. By using the convolutional layers to add features, rather than directly modify the input feature vector, higher-order feature gradients are better preserved throughout the network, which improves the networks learning rate.

A network's *robustness* refers to its ability to generalise. If a high-dimensionality network is trained too long on a small dataset, then the network may become *overtrained*, performing very well on the training dataset but producing poor results

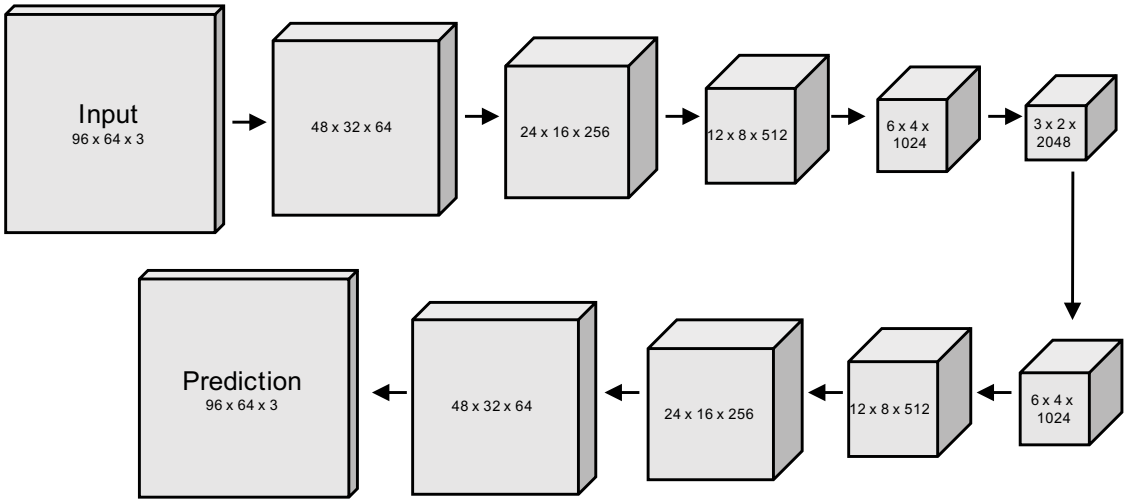
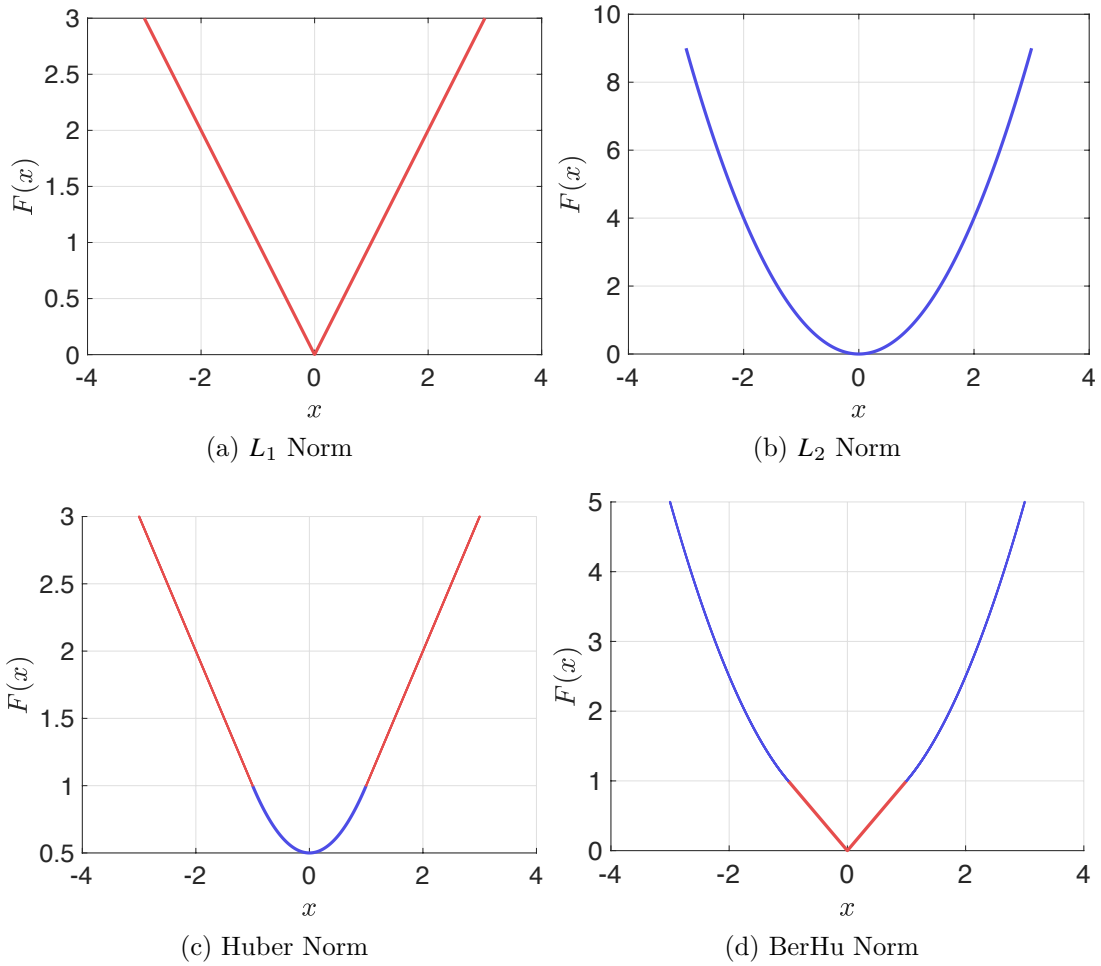


Figure 6.4: Fully Convolutional Residual Network. This network architecture, based on [131], processes each input feature vector by downscaling/projecting the features to a smaller image space but larger feature space (to preserve higher-order gradients). The final prediction is generated by reversing the process. The convolutions in conjunction with image-pyramid approach ensures the network can better generalise features across input scenarios.

when presented new scenarios. A robust network performs reasonably on training data and also generalises well to new datasets.

We seek to improve the robustness of our network through three techniques: cropping, decimation, and regularisation. First, we randomly perturb and crop all input feature images before inputting them into the network. After each epoch of training (i.e., the network has viewed input image), the next epoch will receive a slightly different cropped region of the input image. Without this, the network may learn features that are only associated with a specific pixel location in the training data, thus poorly generalising. Second, we decimate the feature images in multiple stages of the network—similar to, but the reverse of, the image pyramid approaches popular in computer vision. This provides the network with the chance to learn higher-order features on the image that take into account information from larger "patch" regions. Third, we implement a regulariser to prevent the network from over-relying on the cost function at the expense of generalisation performance. Additionally, dropout may be implemented to randomly remove network connections during training to ensure the network builds redundancy and robustness into its parameters [132].

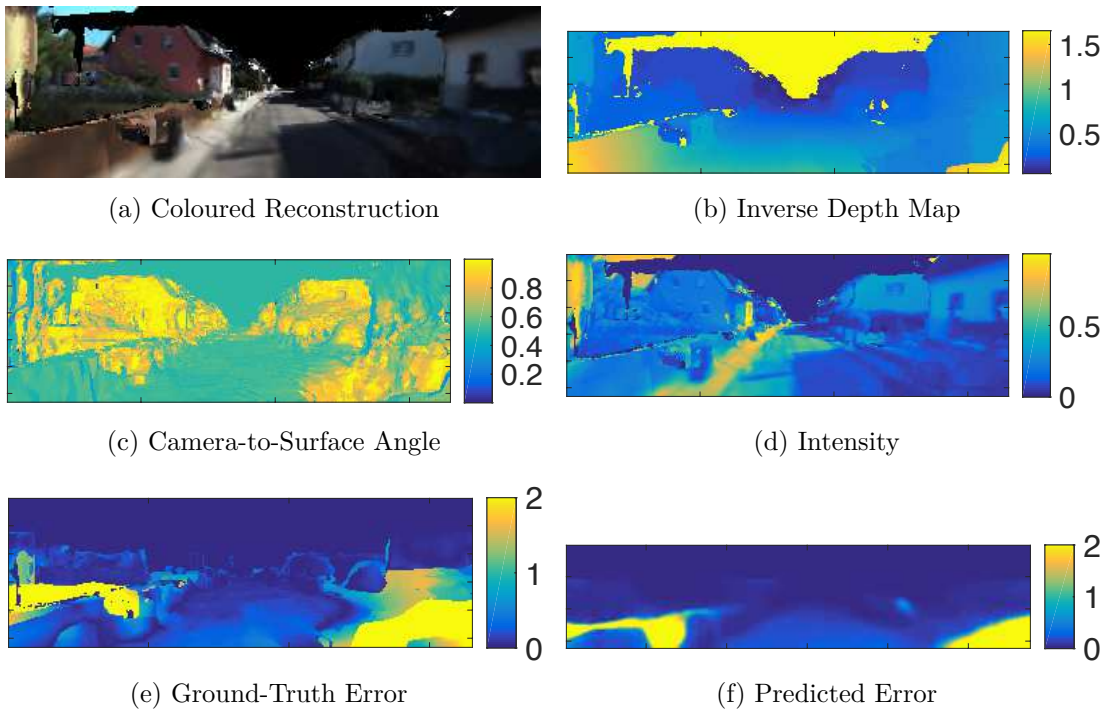


(a) L_1 Norm (b) L_2 Norm

(c) Huber Norm (d) BerHu Norm

Figure 6.5: Comparison of L_1, L_2, Huber, and BerHu Norms. Pictured above are four common norms used in the optimisation literature. We extensively used the L_1 norm (a) throughout earlier portions of this thesis, however in this chapter we transition to the BerHu norm (d) as it enforces a larger penalty on small errors—a desirable feature when learning disparity errors—while still providing a high penalty on larger errors.

A few cost (frequently called "loss" in the deep learning community) functions are widely used in machine learning applications, as shown in Figure 6.5. The L_2 norm is traditionally popular because it heavily penalises large errors and is smooth, thus simplifying the process of optimisation. However, as discussed in Section 3.2.1 on page 54, the L_1 norm emphasises small error penalties while reducing the impact of outliers, but this is at the expense of a more complicated optimisation implementation as L_1 is not differentiable. Over the years, researchers have proposed alternative norms which combine the "best" (based on application) features of each of these norms. The Huber norm uses L_2 near the origin and L_1



(a) Coloured Reconstruction (b) Inverse Depth Map

(c) Camera-to-Surface Angle (d) Intensity

(e) Ground-Truth Error (f) Predicted Error

Figure 6.6: Example Network Prediction (Partial Fence Reconstruction). When provided the feature vectors from (b), (c), and (d), our network recognises and highlights high-probability error regions (f). Of particular note, it recognises the missing portions of the fence (bottom-left of image) and automobile (bottom-right of image). During training, the predicted error image is cropped and randomly perturbed to improve the network's robustness. This is why the "Predicted Error" image (f) is smaller than the "Ground-Truth Error" image (f).

elsewhere, thus making the optimisation implementation simpler. The BerHu uses the L_1 norm near the origin and the L_2 elsewhere [133]. We choose this as our cost function since the L_1 term places a higher penalty on small errors (when compared to Huber or L_2) while still providing strong penalties for larger errors.

6.5 Results

This section presents the qualitative and quantitative prediction performance of our network. We trained the network on 4,541 feature images from KITTI-VO Sequence 00 (3.8 km) and evaluated the network's performance on KITTI-VO Sequence 05 (2.2 km). In the *train* stage we use knowledge of the laser reconstruction to improve network predictions, however in the *evaluate* stage we only use feature inputs from the depth-map reconstruction. Both sequences are predominantly

in urban environments with small amounts of visible vegetation. Using OpenGL shaders, we created a virtual camera to project the dense reconstruction into feature images to train and evaluate the network. We used three salient features: inverse depth, camera-to-surface angle, and pixel intensity.

As discussed in Section 6.3 on page 139, per-pixel ground truth data (δ_{gt}) was computed by comparing the inverse depths of the laser and depth-map reconstructions. For Equation 6.1 on page 139, we select $A = 500$ to appropriately scale the error metrics (Table 6.1).

Example input features, ground-truth data, and predicted errors are presented in Figure 6.6 on the previous page The network successfully highlights the high-likelihood error regions—most noticeable the borders around the fence (bottom-left of input image) and car (bottom-right of input image). More examples of the network's qualitative prediction performance are presented in Figure 6.7 on the facing page, which presents a set of input RGB images and the corresponding ground-truth error and predicted error. The trends in the prediction image are similar to those in the ground-truth images.

Quantitatively, the RMS, median, 75%, and standard deviation errors are recorded in Table 6.1 to serve as a benchmark for future work.

Future work should explore improved methods of detecting reconstruction defects—for example, enhance accuracy by directly inputting 3D data (rather than the current 2D projection) into the network. Additionally, the problem of defect *repair* should be addressed—perhaps through a simple "hole filling" strategy as discussed in Chapter 5 or a more sophisticated method that manipulates a mesh to conform to a learned prior.

Table 6.1: Error (Inverse Depth) Metrics Summary

Method	KITTI Train	KITTI Test	RMS	Median	75%	Std. Dev.
BOR^{2}G	VO Seq 00	VO Seq 05	9.40	1.14	4.40	8.24

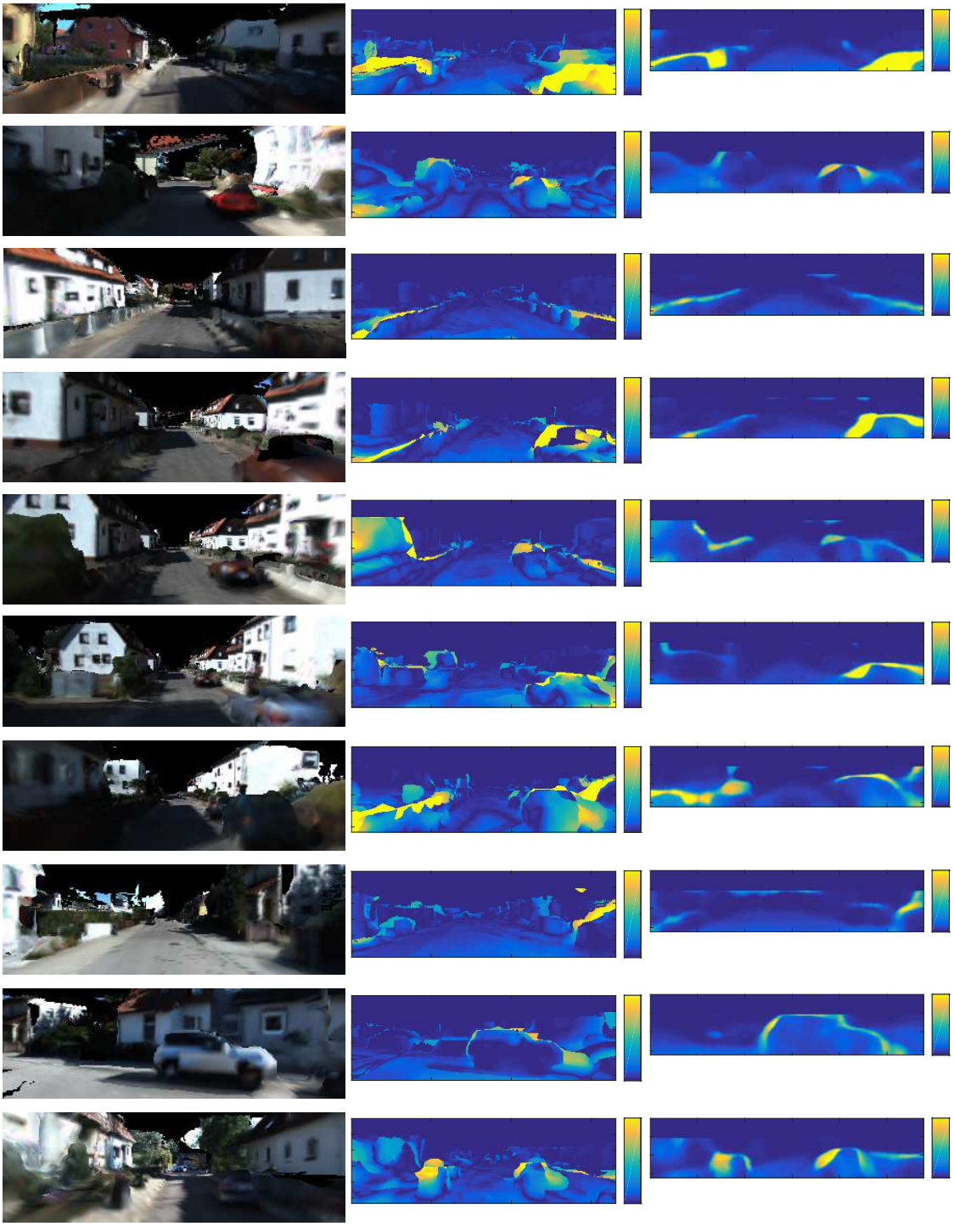


Figure 6.7: Example Ground-Truth vs. Network Prediction. Given the scene present in (a), our network successfully recognises (c) the most error-prone regions of the reconstruction—e.g., unlikely borders around cars, fences, and buildings.

6.6 Conclusions

In this chapter we presented an automated, supervised learning technique to train a neural network to detect error-prone regions of a dense reconstruction. Although external time constraints placed on the author necessitate this research thread remain as an early-stage component, we have not seen work which explicitly *learns* to evaluate dense reconstruction. These nascent results indicate this approach indeed shows promise. We extracted 11-channel feature images, including channels for inverse depth, camera-to-triangle angle, triangle area, and pixel intensity. Using a GLSL pipeline, we positioned a virtual camera at various positions and orientations throughout the reconstruction to generate both input feature data and (by comparing the laser and depth-map reconstruction features) ground-truth data to train the network.

Trained on the reconstructions created in Chapter 4 for the 3.8 km KITTI-VO Sequence 00, our network's performance was qualitatively and quantitatively analysed on the 2.2 km KITTI-VO Sequence 05. The network consistently recognised and highlighted the similar problem areas as those found in the ground-truth data. Thus, the approach proposed in this chapter provides an automated method to evaluate the quality of image-based dense 3D reconstructions created by mobile-robotics platforms.

This is not the end. It is not even the beginning of the end. But it is, perhaps, the end of the beginning.

— Winston Churchill

7 Conclusion

Contents

We began in Chapter 1 by introducing the concept of *dense* 3D reconstructions, establishing their utility in scene segmentation, planning, localisation, augmented reality, material prediction, and hazardous remote inspection. Until recently, computational and memory constraints restricted the quality and scale of reconstructions—they mostly focused on *sparse* point-based maps. With the introduction of low-cost RGB-D sensors and massively-parallel GPGPU computing, Newcombe's KinectFusion framework re-ignited dense 3D reconstruction research [2].

In Chapter 2, we introduced the fundamental research and tools required to understand our contributions. We reviewed $\mathbb{SE}(3)$ coordinate transformations and reference frames. This provides the framework to describe the relative position of multiple sensors and platforms in 3D space at any point in time—a critical prerequisite when fusing data from multiple sensors to produce a coherent model of the environment. As input sensors chosen for this body of work are passive cameras and active lidar sensors, we reviewed the mathematical models for each, thus enabling us to process raw 2D scans and project them into 3D space, and vice

versa. To simultaneously process multiple sensors, it is critical to continually update the *extrinsic* calibration—i.e., relative poses between the sensors and the mobile-robotics platform. We presented two methods from the literature to produce depth maps from passive monocular and stereo cameras, respectively using Newcombe's DTAM [44] and Geiger's LIBELAS [42] algorithms. Rather than project all depth maps and laser scans into 3D space, where memory consumption grows linearly with the each frame, we review variants of the voxel grid along with the TSDF formulation to fuse data into a fixed-size data structure. Finally, the open-source CloudCompare provides a method to quantitatively compare the quality of our dense reconstructions to high-quality ground truth data, usually 3D lidar in this thesis.

Chapter 3 began with a gentle overview of energy-minimising optimisation problems. The energy functions in this thesis exclusively took the form,

$$E(u) = E_{regularisation}(u) + E_{data}(u, f) \tag{7.1}$$

where the *regularisation* term seeks to find a solution which conforms to a prior (e.g., "smooth", "piecewise-constant", etc.); the *data* term ensures the final solution does not vary significantly from original sensor measurements. Though the TV regulariser was used throughout this thesis, we introduced two alternatives for the data term: SDF and histogram. The SDF data term stored two summary statistics (mean and variance) for each voxel, while the histogram provided richer historical information for the optimiser by storing the data observations as a PDF. In our optimiser implementation, we focused on the venerable gradient descent (and ascent) solution for convex problems, but we introduced the Legendre-Fenchel Transformation to reframe a non-smooth energy minimisation function into a smooth min-max saddle problem—thus providing the foundation to derive our TV regulariser implementation. Unfortunately, a straight-forward regulariser implementation produced unexpected results as it both interpolated and extrapolated spurious surfaces. *Our introduction of* Ω*-labels accounted for the special borders conditions between observed and unobserved voxels.* We demonstrated our improved regulariser on synthetic and real-world mobile-robotics datasets. When compared to raw

data fusion of monocular camera depth maps, our regulariser reduced the metric reconstruction errors by 61% in an outdoor scenario and 51% in an indoor scenario, with final respective median errors of 14.41 cm and 15.08 cm.

We built upon this foundation in Chapter 4 by expanding our system to enable data fusion from multiple sensor modalities while operating in the large-scale environments typically seen in mobile-robotics applications. *Our system: (1) operated in multiple-kilometres-scale environments, (2) had no range limitations for input sensor observations, (3) coped with a paucity of surface observations, and (4) fused data from multiple sensor modalities.* We leveraged Nießner's HVG compressed voxel grid [59] to create the large scale reconstructions and then updated our **Ω**-labelling concept to this compressed data structure. Data from both passive stereo cameras and lidar sensors, with proper extrinsic calibration, were efficiently fused into our **Ω**-augmented HVG. To evaluate the system trade-offs between various parameters, our end-to-end dense reconstruction pipeline was robustly analysed with three datasets: Stanford Burghers of Calias [66], Oxford Broad Street (released with this thesis), and KITTI [1]. Fusing multiple sensors provided between 40% to 50% more surface-area coverage. We found the SDF and histogram data terms produced nearly identical results for the low depth observations per m^2 that is characteristic of autonomous-automobile applications. Also, both the surface reconstruction area and quality improved after subsequent passes of the same region. Over a distance of 7.3 km, our regulariser reduced the raw stereo camera depth map fusion reconstruction metric error between 27% to 36% with final median errors between 4 cm to 8 cm.

In Chapter 5, with an eye on single-pass automobile-based surveying applications, we augmented our system to remove dynamic or ephemeral objects (e.g., automobiles, bicycles, pedestrians) from the input sensor data and interpolated the occluded urban structure. To this end, we utilised **Ω**-labelling to target the regulariser to interpolate occluded structure. However, as no range data was directly observed in these regions, the lack of a *data* term caused the optimiser to converge upon a solution that compromised amongst all possible solutions that were consistent with

our piecewise-constant prior assumption. *We formulated a new KCDE "soft" data term for the energy functional to guide the optimiser to a solution that conforms with neighbour surfaces.* When evaluated on four real-world scenarios requiring 20 m^2 of surface infilling, our final reconstruction errors ranged between 5.64 cm and 9.24 cm.

Finally, in Chapter 6, we presented a *machine learning framework that learns to recognise error-prone regions of our dense 3D reconstructions.* The network was trained by comparing 3.8 kilometres of dense reconstructions created by image depth maps versus laser scans. Our system successfully identified depth-map reconstruction areas which contained errors.

7.1 Future Work

The work presented in this thesis contributes to the dense reconstruction and robotics communities, however there are a variety of ways it can be improved upon in the future. We anticipate five specific extensions that should be pursued:

1. *Augmented* $\mathbf{\Omega}$ *Labelling.* When we introduced $\mathbf{\Omega}$-labelling in Chapter 3, we proposed a simple indicator variable for each voxel. Rather than restricting $\mathbf{\Omega}$ to a binary value, the regulariser may be better aided by providing additional information—for example the number of sensors that have observed a voxel or the angels at which a voxel has been observed. This data attests to the confidence of a given voxel's TSDF value, which may be used to guide the optimiser to a more consistent solution.

2. *Characterise Sensor Cost vs. Reconstruction Accuracy.* In Chapter 4, we provided reconstruction quality analysis on a variety of sensors and data-term techniques. We found, when compared to a typical RGB-D dataset, passive camera sensors mounted on an autonomous-vehicle platform have 3,100 *fewer* observations per square metre of surface reconstruction. A system developer, however, must operate with the design trade space between sensor cost, effective range, and reconstruction accuracy. Qualitative and quantitative metrics in these areas for a variety of sensors (e.g., Velodyne, SICK LMS-151, global-shutter mono and stereo

cameras, rolling-shutter cameras, RGB-D, etc.) would be a valuable resource for the mobile-robotics and dense-reconstruction communities.

3. *Infill "Neighbourhood" Definition.* With Chapter 5's KCDE "soft" data term, we found cardinal neighbours effectively guided the optimiser to an accurate solution. There are instances, however, where this simple approach might struggle—for example, if an ephemeral object occludes the lower portion of an alleyway's entry, the neighbour voxels would include both a wall (left/right), pavement (below), and free space (above). Ambiguous scenarios such as these should either be flagged by the algorithm for user review, or a more sophisticated prior may be introduced based on the type of reconstruction (urban, country, indoor, etc.).

4. *Machine-Learning Defect Detection.* In Chapter 6, we leveraged the recent advances in machine learning to build a deep neural network that processes dense reconstructions and "learn" where the reconstructions are incorrect. The neural network was trained by comparing the surface models from camera-only and laser-only reconstructions—it learned which features in the input RGB image, depth map, or resulting reconstruction result in poor surface models (e.g., large holes in the road, uneven walls, etc.). However, defect-detection performance could potentially be further improved by allowing a neural network to train directly on 3D data (e.g., mesh or TSDF voxel grid) rather than the current 2D-only feature set.

5. *Machine-Learning Defect Repair.* In addition to improving defect detection, future work should also address defect *repair*. This might be accomplished via a simple hole infilling method (e.g., Chapter 5) or a machine-learning approach that performs both defect detection and correction in a single neural network.

7.2 Closing Remarks

This thesis sought to contribute to the art of dense 3D reconstruction for mobile-robotics applications. Our primary contribution is the introduction of Ω-labels,

which enable us to target a regulariser to a specific subset of a voxel grid. In Chapter 3 and Chapter 4, we used this to prevent the optimiser from interpolating and extrapolating spurious surfaces in regions unobserved by the input sensors. However, in Chapter 5, we targeted the regulariser to interpolate surfaces in regions occluded by ephemeral and dynamic objects.

The algorithms and systems presented are substantial, requiring more than 480K lines of supporting C++, CUDA, GLSL, and Python source code — code that is already under license for commercial applications. Our system generates clean, optimised, large-scale, and dense 3D reconstructions that may be used by localisation, planning, or any of a myriad of other subsystems on a mobile-robotics platform. We hope these contributions serve as valuable tools to the robotics and dense reconstruction communities.

nos esse quasi nanos gigantium humeris incidentes

— Bernard of Chartres (c. 1159)

References

[1] Andreas Geiger, Philip Lenz, and Raquel Urtasun. "Are We Ready for Autonomous Driving? The KITTI Vision Benchmark Suite". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE. 2012, pp. 3354–3361.

[2] Richard A. Newcombe et al. "KinectFusion: Real-Time Dense Surface Mapping and Tracking". In: *Mixed and Augmented Reality (ISMAR), 2011 10th IEEE International Symposium on*. IEEE, 2011, pp. 127–136. (Visited on 01/14/2015).

[3] Christopher Zach. "Fast and High Quality Fusion of Depth Maps". In: *Proceedings of the International Symposium on 3D Data Processing, Visualization and Transmission (3DPVT)*. Vol. 1. 2008.

[4] Christopher Zach, Thomas Pock, and Horst Bischof. "A Globally Optimal Algorithm for Robust TV-L 1 Range Image Integration". In: *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*. IEEE, 2007, pp. 1–8. (Visited on 04/22/2015).

[5] Geoffrey Pascoe et al. "NID-SLAM: Robust Monocular SLAM Using Normalised Information Distance". In: *Conference on Computer Vision and Pattern Recognition*. Honolulu, Hawaii, July 2017.

[6] Michael Tanner et al. "BOR2G: Building Optimal Regularised Reconstructions with GPUs (in Cubes)". In: *International Conference on Field and Service Robotics (FSR)*. Toronto, ON, Canada, June 2015.

[7] Michael Tanner et al. "DENSER Cities: A System for Dense Efficient Reconstructions of Cities". In: *ArXiv e-prints* (Apr. 2016). arXiv: `arXiv:1604.03734 [cs.CV]`.

[8] Michael Tanner et al. "Keep Geometry in Context: Using Contextual Priors for Very-Large-Scale 3D Dense Reconstructions". In: *Robotics: Science and Systems, Workshop on Geometry and Beyond: Representations, Physics, and Scene Understanding for Robotics*. June 2016.

[9] Michael Tanner et al. "BOR2G: Building Optimal Regularized Reconstructions with GPUs". In: *IEEE Transactions on Robotics* In Review (TBD).

[10] Michael Tanner et al. "What Lies Behind: Recovering Hidden Shape in Dense Mapping". In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. Stockholm, Sweden, May 2016, pp. 979–986.

[11] Gilbert Strang. *Linear Algebra and Its Applications*. Brooks Cole, Feb. 1988.

[12] D. Zill, W.S. Wright, and M.R. Cullen. *Advanced Engineering Mathematics*. Jones and Bartlett Publishers series in mathematics: Advanced engineering mathematics. Jones & Bartlett Learning, 2011.

[13] P. Erdős. “Extremal Problems in Graph Theory”. In: *Theory Of Graphs and Its Applications,” Proc. Sympos. Smolenice*. 1964, pp. 29–36.

[14] James Diebel. “Representing Attitude: Euler Angles, Unit Quaternions, and Rotation Vectors”. In: *Matrix* 58.15-16 (2006), pp. 1–35.

[15] Larry Davis. *Coordinate Systems for the Space Shuttle Program*. Tech. rep. TM X-58153. National Aeronautics and Space Administration, Oct. 1974, p. 19.

[16] David Portree. *Mir Hardware Heritage*. Reference Publication 1357. National Aeronautics and Space Administration, Mar. 1995.

[17] Xavez. *Sketching my Camera*. Nov. 2009. URL: `https://www.flickr.com/photos/xavez/4101398348` (visited on 02/13/2017).

[18] Winston Churchill and Paul Newman. “Practice Makes Perfect? Managing and Leveraging Visual Experiences for Lifelong Navigation”. In: *Proc. IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. Minnesota, USA, May 2012, pp. 4525–4532.

[19] Richard Szeliski. *Computer Vision: Algorithms and Applications*. Springer, 2010. (Visited on 01/14/2015).

[20] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Second. Cambridge University Press, ISBN: 0521540518, 2004.

[21] David A. Forsyth and Jean Ponce. *Computer Vision: A Modern Approach*. Prentice Hall Professional Technical Reference, 2002.

[22] Zhengyou Zhang. “A Flexible New Technique for Camera Calibration”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22.11 (Nov. 2000), pp. 1330–1334.

[23] G. Bradski. “OpenCV”. In: *Dr. Dobb’s Journal of Software Tools* (2000).

[24] Alastair Harrison and Paul Newman. “TICSync: Knowing When Things Happened”. In: *Robotics and Automation (ICRA), 2011 IEEE International Conference On*. IEEE, 2011, pp. 356–363. (Visited on 11/25/2015).

[25] Alexander D. Stewart. “Localisation Using the Appearance of Prior Structure”. PhD thesis. University of Oxford, Oct. 2014.

[26] Winston Churchill. “Exerience Based Navigation: Theory, Practice and Implementation”. PhD thesis. University of Oxford, Oct. 2012.

[27] Will Maddern et al. “1 Year, 1000km: The Oxford RobotCar Dataset”. In: *The International Journal of Robotics Research (IJRR)* 36.1 (2016), pp. 3–15.

[28] Terry Scott et al. “Exploiting Known Unknowns: Scene Induced Cross-Calibration of Lidar-Stereo Systems”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. Hamburg, Germany, 2015, pp. 3647–3653.

[29] Terry Scott et al. “Choosing a Time and Place for Calibration of Lidar-Camera Systems”. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. Stockholm, Sweden, May 2016, pp. 4349–4356.

[30] Zhengyou Zhang. “Microsoft Kinect Sensor and Its Effect”. In: *IEEE multimedia* 19.2 (2012), pp. 4–10. (Visited on 02/17/2017).

[31] Kourosh Khoshelham and Sander Oude Elberink. “Accuracy and Resolution of Kinect Depth Data for Indoor Mapping Applications”. en. In: *Sensors* 12.12 (Feb. 2012), pp. 1437–1454. (Visited on 11/20/2015).

[32] Daniel Scharstein and Richard Szeliski. “A Taxonomy and Evaluation of Dense Two-Frame Stereo Correspondence Algorithms”. en. In: *International Journal of Computer Vision* 47.1-3 (Apr. 2002), pp. 7–42. (Visited on 04/28/2017).

[33] Jan Stühmer, Stefan Gumhold, and Daniel Cremers. “Real-Time Dense Geometry from a Handheld Camera”. In: *Pattern Recognition*. Springer, 2010, pp. 11–20. (Visited on 01/14/2015).

[34] Heiko Hirschmüller. “Semi-Global Matching-Motivation, Developments and Applications”. In: *Photogrammetric Week 11* (2011), pp. 173–184.

[35] Stuart Geman and Donald Geman. “Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images”. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 6.6 (Nov. 1984), pp. 721–741. (Visited on 04/28/2017).

[36] G. Sandini and M. Tistarelli. “Active Tracking Strategy for Monocular Depth Inference over Multiple Frames”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 12.1 (Jan. 1990), pp. 13–27.

[37] Javier Civera, Andrew J. Davison, and J. Montiel. “Inverse Depth Parametrization for Monocular SLAM”. In: *Robotics, IEEE Transactions on* 24.5 (2008), pp. 932–945. (Visited on 01/14/2015).

[38] Pedro Piniés, Lina Maria Paz, and Paul Newman. “Dense Mono Reconstruction: Living with the Pain of the Plain Plane”. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. Seattle, WA, USA, May 2015, pp. 5226–5231.

[39] Ashutosh Saxena, Sung H. Chung, and Andrew Y. Ng. “Learning Depth from Single Monocular Images”. In: *Advances in Neural Information Processing Systems*. Vol. 18. 2005, pp. 1–8. (Visited on 01/14/2015).

[40] David Eigen, Christian Puhrsch, and Rob Fergus. “Depth Map Prediction from a Single Image Using a Multi-Scale Deep Network”. In: *Advances in Neural Information Processing Systems*. 2014, pp. 2366–2374.

[41] Fayao Liu, Chunhua Shen, and Guosheng Lin. “Deep Convolutional Neural Fields for Depth Estimation from a Single Image”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 5162–5170.

[42] Andreas Geiger, Martin Roser, and Raquel Urtasun. “Efficient Large-Scale Stereo Matching”. In: *Asian Conference on Computer Vision (ACCV)*. Springer. 2010, pp. 25–38.

[43] Der-Tsai Lee and Bruce J Schachter. “Two Algorithms for Constructing a Delaunay Triangulation”. In: *International Journal of Computer & Information Sciences* 9.3 (1980), pp. 219–242.

[44] Richard A. Newcombe, Steven J. Lovegrove, and Andrew J. Davison. “DTAM: Dense Tracking and Mapping in Real-Time”. In: *Computer Vision (ICCV), 2011 IEEE International Conference on*. IEEE, 2011, pp. 2320–2327. (Visited on 01/14/2015).

[45] E.B. Olson. “Real-Time Correlative Scan Matching”. In: *IEEE International Conference on Robotics and Automation, 2009. ICRA ’09*. May 2009, pp. 4387–4393.

[46] J. Fossel, J. Sturm, and K. Tuyls. “2D-SDF-SLAM: A Signed Distance Function Based SLAM System for Laser Scanners”. In: *Proc. of the International Conference on Intelligent Robot Systems (IROS)*. IEEE. Sept. 2015, pp. 1949–1955.

[47] Ryan W. Wolcott and Ryan M. Eustice. “Fast LIDAR Localization Using Multiresolution Gaussian Mixture Maps”. In: *Proceedings of the IEEE International Conference on Robotics and Automation*. Seattle, WA, USA, May 2015, pp. 2814–2821.

[48] David Nistér, Oleg Naroditsky, and James Bergen. “Visual odometry for ground vehicle applications”. In: *Journal of Field Robotics* 23.1 (2006), pp. 3–20.

[49] Stefan Kohlbrecher et al. “A Flexible and Scalable Slam System with Full 3d Motion Estimation”. In: *Safety, Security, and Rescue Robotics (SSRR), 2011 IEEE International Symposium On*. IEEE, 2011, pp. 155–160.

[50] Mingyang Li and Anastasios I. Mourikis. “High-Precision, Consistent EKF-Based Visual–inertial Odometry”. en. In: *The International Journal of Robotics Research* 32.6 (May 2013), pp. 690–711. (Visited on 03/04/2015).

[51] Mingyang Li et al. “High-Fidelity Sensor Modeling and Self-Calibration in Vision-Aided Inertial Navigation”. In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2014, pp. 409–416. (Visited on 10/05/2016).

[52] Stefan Leutenegger et al. “Keyframe-based visual–inertial odometry using nonlinear optimization”. In: *The International Journal of Robotics Research* 34.3 (Dec. 2015), pp. 314–334.

[53] Georg Klein and David Murray. “Parallel Tracking and Mapping for Small AR Workspaces”. In: *Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium On*. IEEE, 2007, pp. 225–234. (Visited on 01/14/2015).

[54] Brian Curless and Marc Levoy. “A Volumetric Method for Building Complex Models from Range Images”. In: *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*. ACM, 1996, pp. 303–312. (Visited on 01/14/2015).

[55] T. Whelan et al. “Kintinuous: Spatially Extended KinectFusion”. In: *RSS Workshop on RGB-D: Advanced Reasoning with Depth Cameras*. Sydney, Australia, July 2012.

[56] Ming Zeng et al. “Octree-Based Fusion for Realtime 3D Reconstruction”. In: *Graphical Models* 75.3 (2013), pp. 126–136. (Visited on 01/14/2015).

[57] Ming Zeng et al. “A Memory-Efficient Kinectfusion Using Octree”. In: *Computational Visual Media*. Springer, 2012, pp. 234–241. (Visited on 01/14/2015).

[58] F. Steinbrucker et al. “Large-Scale Multi-Resolution Surface Reconstruction from RGB-D Sequences”. In: *2013 IEEE International Conference on Computer Vision (ICCV)*. Dec. 2013, pp. 3264–3271.

[59] Matthias Nießner et al. “Real-Time 3D Reconstruction at Scale Using Voxel Hashing”. In: *ACM Trans. Graph.* 32.6 (Nov. 2013), 169:1–169:11. (Visited on 07/14/2015).

[60] Olaf Kahler et al. “Hierarchical Voxel Block Hashing for Efficient Integration of Depth Images”. In: *IEEE Robotics and Automation Letters* 1.1 (Jan. 2016), pp. 192–197. (Visited on 02/15/2016).

[61] Steven Parker et al. “Interactive Ray Tracing for Isosurface Rendering”. In: *Proceedings of the Conference on Visualization ’98*. VIS ’98. Los Alamitos, CA, USA: IEEE Computer Society Press, 1998, pp. 233–238. (Visited on 01/15/2015).

[62] William E. Lorensen and Harvey E. Cline. “Marching Cubes: A High Resolution 3D Surface Construction Algorithm”. In: *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH ’87. New York, NY, USA: ACM, 1987, pp. 163–169. (Visited on 01/15/2015).

[63] Gregory M. Nielson and Bernd Hamann. “The Asymptotic Decider: Resolving the Ambiguity in Marching Cubes”. In: *Proceedings of the 2Nd Conference on Visualization ’91*. VIS ’91. Los Alamitos, CA, USA: IEEE Computer Society Press, 1991, pp. 83–91. (Visited on 10/08/2015).

[64] Evgeni V. Chernyaev. *Marching Cubes 33: Construction of Topologically Correct Isosurfaces*. Tech. rep. 1995.

[65] *CloudCompare*. `http://www.danielgm.net/cc/`. (Visited on 08/07/2015).

[66] Qian-Yi Zhou and Vladlen Koltun. “Dense Scene Reconstruction with Points of Interest”. In: *ACM Trans. Graph.* 32.4 (July 2013), 112:1–112:8. (Visited on 01/19/2015).

[67] Thomas Whelan et al. “Real-Time Large-Scale Dense RGB-D SLAM with Volumetric Fusion”. en. In: *The International Journal of Robotics Research* (Dec. 2014), pp. 598–626. (Visited on 01/21/2015).

[68] V. Pradeep et al. “MonoFusion: Real-Time 3D Reconstruction of Small Scenes with a Single Web Camera”. In: *2013 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*. Oct. 2013, pp. 83–88.

[69] Olaf Kahler et al. “Very High Frame Rate Volumetric Integration of Depth Images on Mobile Devices”. In: *IEEE Transactions on Visualization and Computer Graphics* 21.11 (Nov. 2015), pp. 1241–1250. URL: `http://dx.doi.org/10.1109/TVCG.2015.2459891`.

[70] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. New York, NY, USA: Cambridge University Press, 2004.

[71] H. Bauschke and Yves Lucet. “What Is a Fenchel Conjugate”. In: *Notices of the AMS* 59.1 (2012). (Visited on 05/30/2016).

[72] Ankur Handa et al. “Applications of Legendre-Fenchel Transformation to Computer Vision Problems”. In: *Department of Computing at Imperial College London. DTR11-7* 45 (2011), pp. 1–35. (Visited on 05/30/2016).

[73] Hugo Goncalves. *Legendre and Legendre-Fenchel Transforms.* `http://www.onmyphd.com/?p=legendre.fenchel.transform`.

[74] C. Zach, T. Pock, and H. Bischof. "A Duality Based Approach for Realtime TV-L1 Optical Flow". en. In: *Pattern Recognition*. Springer, Berlin, Heidelberg, Sept. 2007, pp. 214–223. (Visited on 04/28/2017).

[75] K. Bredies, K. Kunisch, and T. Pock. "Total Generalized Variation". In: *SIAM Journal on Imaging Sciences* 3.3 (Jan. 2010), pp. 492–526. (Visited on 07/14/2016).

[76] R. T. Rockafellar. *Convex Analysis*. Princeton, New Jersey: Princeton University Press, 1970.

[77] Antonin Chambolle and Thomas Pock. "A First-Order Primal-Dual Algorithm for Convex Problems with Applications to Imaging". In: *J. Math. Imaging Vis.* 40.1 (May 2011), pp. 120–145.

[78] Pedro Piniés, Lina Maria Paz, and Paul Newman. "Too Much TV Is Bad: Dense Reconstruction from Sparse Laser with Non-Convex Regularisation". In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. Seattle, WA, USA, May 2015, pp. 135–142.

[79] Thomas Pock and Antonin Chambolle. "Diagonal Preconditioning for First Order Primal-Dual Algorithms in Convex Optimization". In: *Computer Vision (ICCV), 2011 IEEE International Conference On*. IEEE, 2011, pp. 1762–1769. (Visited on 07/20/2015).

[80] Yingying Li and Stanley Osher. "A New Median Formula with Applications to PDE Based Denoising". EN. In: *Communications in Mathematical Sciences* 7.3 (Sept. 2009), pp. 741–753. (Visited on 03/04/2015).

[81] John Amanatides, Andrew Woo, et al. "A Fast Voxel Traversal Algorithm for Ray Tracing". In: *Eurographics*. Vol. 87. 1987, pp. 3–10. (Visited on 10/08/2015).

[82] A. Handa et al. "A Benchmark for RGB-D Visual Odometry, 3D Reconstruction and SLAM". In: *IEEE Intl. Conf. on Robotics and Automation, ICRA*. IEEE. Hong Kong, China, May 2014, pp. 1524–1531.

[83] M. Pollefeys et al. "Detailed Real-Time Urban 3D Reconstruction from Video". In: *International Journal of Computer Vision* 78.2-3 (July 2008), pp. 143–167.

[84] Sameer Agarwal et al. "Building Rome in a Day". In: *Twelfth IEEE International Conference on Computer Vision (ICCV 2009)*. Kyoto, Japan: ACM, Sept. 2009, pp. 105–112.

[85] Yasutaka Furukawa et al. "Towards Internet-Scale Multi-View Stereo". In: *The Twenty-Third IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2010, San Francisco, CA, USA, 13-18 June 2010*. 2010, pp. 1434–1441.

[86] Bastian Goldluecke, Evgeny Strekalovskiy, and Daniel Cremers. "The Natural Vectorial Total Variation Which Arises from Geometric Measure Theory". In: *SIAM Journal on Imaging Sciences* 5.2 (2012), pp. 537–563. (Visited on 05/05/2015).

[87] Matthew Klingensmith et al. "Chisel: Real Time Large Scale 3D Reconstruction Onboard a Mobile Device". In: *Robotics Science and Systems 2015*. July 2015.

[88] Jakob Engel, Jorg Stuckler, and Daniel Cremers. “Large-Scale Direct SLAM with Stereo Cameras”. In: *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*. IEEE, 2015, pp. 1935–1942.

[89] Thomas Schöps et al. “3d Modeling on the Go: Interactive 3d Reconstruction of Large-Scale Scenes on Mobile Devices”. In: *3D Vision (3DV), 2015 International Conference on*. IEEE, 2015, pp. 291–299.

[90] Victor Adrian Prisacariu et al. “Real-Time 3D Tracking and Reconstruction on Mobile Phones”. In: *IEEE Transactions on Visualization and Computer Graphics* 21.5 (May 2015), pp. 557–570. (Visited on 09/29/2015).

[91] Dragomir Anguelov et al. “Google Street View: Capturing the World at Street Level”. In: *Computer* 43.6 (2010), pp. 32–38.

[92] Jianxiong Xiao and Yasutaka Furukawa. “Reconstructing the World’s Museums”. In: *International journal of computer vision* 110.3 (2014), pp. 243–258.

[93] Yunsu Bok, Dong-Geol Choi, and In Kweon. “Sensor Fusion of Cameras and a Laser for City-Scale3D Reconstruction”. en. In: *Sensors* 14.11 (Nov. 2014), pp. 20882–20909.

[94] Alex Stewart and Paul Newman. “LAPS - Localisation Using Appearance of Prior Structure: 6-DOF Monocular Camera Localisation Using Prior Pointclouds”. In: *Proc. IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. Minnesota, USA, May 2012, pp. 2625–2632.

[95] W. Maddern, A.D. Stewart, and P. Newman. “LAPS-II: 6-DoF Day and Night Visual Localisation with Prior 3D Structure for Autonomous Road Vehicles”. In: *2014 IEEE Intelligent Vehicles Symposium Proceedings*. June 2014, pp. 330–337.

[96] M.W.M.G. Dissanayake et al. “A Solution to the Simultaneous Localization and Map Building (SLAM) Problem”. In: *IEEE Transactions on Robotics and Automation* 17.3 (June 2001), pp. 229–241.

[97] Renato F. Salas-Moreno et al. “Slam++: Simultaneous Localisation and Mapping at the Level of Objects”. In: *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference On*. IEEE, 2013, pp. 1352–1359. (Visited on 01/14/2015).

[98] Tim Bailey and Hugh Durrant-Whyte. “Simultaneous Localisation and Mapping (Slam) Part 2: State of the Art”. In: *Robotics and Automation Magazine* (2006). (Visited on 01/14/2015).

[99] Andrew J. Davison. “Real-Time Simultaneous Localisation and Mapping with a Single Camera”. In: *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference On*. IEEE, 2003, pp. 1403–1410. (Visited on 05/30/2016).

[100] Etienne Mouragnon et al. “Real Time Localization and 3d Reconstruction”. In: *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference On*. Vol. 1. IEEE, 2006, pp. 363–370. (Visited on 01/14/2015).

[101] Richard A. Newcombe. “Dense Visual SLAM”. PhD thesis. Imperial College London, Dec. 2012.

[102] Michael Bosse et al. “An Atlas Framework for Scalable Mapping”. In: *International Conference on Robotics and Automation*. Vol. 2. IEEE. 2004, pp. 1899–1906. (Visited on 01/14/2015).

[103] Lu Ma, Juan Falquez, and Steve McGuire. “Large Scale Dense Visual Inertial SLAM”. In: *Field and Service Robotics*. Springer. Toronto, ON, Canada, June 2016, pp. 141–155.

[104] Jakob Engel, Vladlen Koltun, and Daniel Cremers. “Direct Sparse Odometry”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2017).

[105] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardós. “ORB-SLAM: A Versatile and Accurate Monocular SLAM System”. In: *IEEE Transactions on Robotics* 31.5 (Oct. 2015), pp. 1147–1163.

[106] Raul Mur-Artal and Juan D. Tardos. “ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo and RGB-D Cameras”. In: *arXiv:1610.06475 [cs]* (Oct. 2016). arXiv: `1610.06475 [cs]`. (Visited on 10/24/2016).

[107] Colin McManus et al. “Distraction Suppression for Vision-Based Pose Estimation at City Scales”. In: *Proc. IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. Karlsruhe, Germany, May 2013, pp. 3762–3769.

[108] Jeffrey Hawke et al. “Wrong Today, Right Tomorrow: Experience-Based Classification for Robot Perception”. In: *Field and Service Robotics (FSR)*. Springer. Toronto, ON, Canada, June 2015, pp. 173–186.

[109] Jianning Wang and Manuel M. Oliveira. “A Hole-Filling Strategy for Reconstruction of Smooth Surfaces in Range Images”. In: *Computer Graphics and Image Processing, 2003. SIBGRAPI 2003. XVI Brazilian Symposium on*. IEEE, 2003, pp. 11–18. (Visited on 08/06/2015).

[110] S. Vasudevan et al. “Gaussian Process Modeling of Large Scale Terrain”. In: *IEEE International Conference on Robotics and Automation, 2009. ICRA ’09*. May 2009, pp. 1047–1053.

[111] James Davis et al. “Filling Holes in Complex Surfaces Using Volumetric Diffusion”. In: *3D Data Processing Visualization and Transmission, 2002. Proceedings. First International Symposium on*. IEEE, 2002, pp. 428–441. (Visited on 08/06/2015).

[112] Christian Häne et al. “A Patch Prior for Dense 3d Reconstruction in Man-Made Environments”. In: *3D Imaging, Modeling, Processing, Visualization and Transmission (3DIMPVT), 2012 Second International Conference on*. IEEE, 2012, pp. 563–570. (Visited on 09/10/2015).

[113] Maik Keller et al. “Real-Time 3d Reconstruction in Dynamic Scenes Using Point-Based Fusion”. In: *3DTV-Conference, 2013 International Conference on*. IEEE, 2013, pp. 1–8. (Visited on 01/14/2015).

[114] D.Z. Wang, I. Posner, and P. Newman. “What Could Move? Finding Cars, Pedestrians and Bicyclists in 3D Laser Data”. In: *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE. May 2012, pp. 4038–4044.

[115] Daniel Cremers. *Multiple View Geometry*. Summer 2013. (Visited on 02/17/2015).

[116] Kristian Bredies, Florian Knoll, and Christian Langkammer. “TGV Regularization for Variational Approaches to Quantitative Susceptibility Mapping”. In: (2013). (Visited on 10/03/2016).

[117] Michael P. Holmes, Alexander G. Gray, and Charles Lee Isbell. "Fast Nonparametric Conditional Density Estimation". In: *arXiv preprint arXiv:1206.5278* (2012). eprint: `arXiv:1206.5278`. (Visited on 08/06/2015).

[118] David Ferstl et al. "CP-Census: A Novel Model for Dense Variational Scene Flow from RGB-D Data". In: *Proceedings of the British Machine Vision Conference*. BMVA Press, 2014.

[119] *OpenStreetMap*. `http://www.openstreetmap.org`. (Visited on 08/07/2015).

[120] Satoshi Abe et al. "System Integration of Road-Crack Evaluation System". In: *Proc. SPIE* 1907 (1993), pp. 38–48.

[121] Ajay Kumar and Grantham KH Pang. "Defect Detection in Textured Materials Using Optimized Filters". In: *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)* 32.5 (2002), pp. 553–570. (Visited on 05/04/2017).

[122] Patrick M. Knupp. "Remarks on Mesh Quality". In: *45th AIAA Aerospace Sciences Meeting and Exhibit*. 2007, pp. 7–10. (Visited on 05/04/2017).

[123] Chris Gordon et al. "Combining Reality Capture Technologies for Construction Defect Detection: A Case Study". In: *EIA9: E-Activities and Intelligent Support in Design and the Built Environment, 9th EuropIA International Conference*. 2003, pp. 99–108. (Visited on 05/04/2017).

[124] Matthew M. Torok, Mani Golparvar-Fard, and Kevin B. Kochersberger. "Image-Based Automated 3D Crack Detection for Post-Disaster Building Assessment". en. In: *Journal of Computing in Civil Engineering* 28.5 (Sept. 2014). eprint: `http://ascelibrary.org/doi/pdf/10.1061/%28ASCE%29CP.1943-5487.0000334`. URL: `http://ascelibrary.org/doi/abs/10.1061/%28ASCE%29CP.1943-5487.0000334` (visited on 05/04/2017).

[125] Yu-Fei Liu et al. "Concrete Crack Assessment Using Digital Image Processing and 3D Scene Reconstruction". en. In: *Journal of Computing in Civil Engineering* 30.1 (Jan. 2016), p. 04014124. (Visited on 05/04/2017).

[126] Zixiang Zhou, Jie Gong, and Mengyang Guo. "Image-Based 3D Reconstruction for Posthurricane Residential Building Damage Assessment". en. In: *Journal of Computing in Civil Engineering* 30.2 (Mar. 2016), p. 04015015. (Visited on 05/04/2017).

[127] Christian Koch et al. "A Review on Computer Vision Based Defect Detection and Condition Assessment of Concrete and Asphalt Civil Infrastructure". en. In: *Advanced Engineering Informatics* 29.2 (Apr. 2015), pp. 196–210. (Visited on 05/04/2017).

[128] Alex Kendall, Matthew Grimes, and Roberto Cipolla. "Posenet: A Convolutional Network for Real-Time 6-Dof Camera Relocalization". In: *Proceedings of the IEEE International Conference on Computer Vision*. 2015, pp. 2938–2946.

[129] Christopher Bongsoo Choy et al. "3D-R2N2: A Unified Approach for Single and Multi-View 3D Object Reconstruction". In: *CoRR* abs/1604.00449 (2016), pp. 628–644.

[130] Gernot Riegler, Ali Osman Ulusoys, and Andreas Geiger. "OctNet: Learning Deep 3D Representations at High Resolutions". In: *arXiv preprint arXiv:1611.05009* (2016). eprint: `arXiv:1611.05009`.

[131] Iro Laina et al. "Deeper Depth Prediction with Fully Convolutional Residual Networks". In: *3D Vision (3DV), 2016 Fourth International Conference On*. IEEE, 2016, pp. 239–248.

[132] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting." In: *Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.

[133] Art B Owen. "A robust hybrid of lasso and ridge regression". In: *Contemporary Mathematics* 443 (2007), pp. 59–72.