Learnt Error Correction in 3D Reconstructions



Ștefan Săftescu
Somerville College
University of Oxford

A thesis submitted for the degree of
Doctor of Philosophy

Michaelmas 2020

Abstract

This thesis is about reducing the cost of building good 3D reconstructions, which we achieve by correcting 3D meshes built from data obtained at low costs – such as by using cameras or by performing only sparse scans of scenes. Specifically, we propose a method that learns to correct reconstructions of outdoor scenes built from stereo-camera data collected from a car driving on city streets. To capture geometric and appearance priors, we use a convolutional neural network model that takes mesh features – 2D views rendered from the reconstruction – as input, and predicts corrected inverse-depth images. This model is supervised by a high-quality lidar reconstruction of the same scene, and its predictions are used to build a new, corrected reconstruction.

To allow for large amounts of training data and novel viewpoints, we replace the input colour images with synthetic ones rendered from the low-quality mesh, at virtually no cost to performance. We ensure the geometry of predictions matches between neighbouring views by using a geometric consistency loss during training. By adding introspective capabilities in the form of a learnt soft mask applied to the output of our network, we only correct parts of the reconstruction with a high degree of certainty. When deployed on 21.8 km of driving data, our system reduces total reconstruction error by up to 39.1%.

Finally, we propose a novel architecture for aggregating information from multiple views during inference. We make the observation that features within the network depend on viewpoint and therefore are difficult to aggregate when views are disparate. We address this by learning a mapping from relative poses to a feature-space transform which we use prior to aggregation. In our experiments, we show that this additional step is essential to effectively use

information from multiple views. Beyond using it for mesh correction, such an architecture is useful in general for multi-view tasks and future work will explore some of its other applications.

Contents

Contents i

1 Introduction 1

1.1 Thesis Overview . . . 4

1.2 Contributions . . . 6

1.2.1 Publications . . . 6

2 Background 7

2.1 Convolutional Neural Networks . . . 7

2.2 Camera Geometry . . . 10

2.2.1 Pinhole Camera Model . . . 10

2.2.2 Camera Location and Orientation . . . 12

2.2.3 Point Correspondence . . . 13

2.2.4 Disparity, Depth Maps, and Inverse-Depth . . . 14

2.3 Dense 3D Reconstruction . . . 16

2.4 Mesh Correction . . . 17

2.4.1 Measuring Mesh Correctness . . . 19

3 Mesh Correction System 21

3.1 Overview . . . 21

3.2 3D Reconstruction Pipeline . . . 23

3.2.1 Localisation . . . 23

3.2.2 Depth Estimation . . . 24

3.2.3 3D Fusion . . . 24

3.3 Rendering Mesh Features . . . 24

3.3.1 Rendering Pipeline . . . 27

3.3.2 OpenGL Implementation . . . 28

3.4 TensorFlow Implementation . . . 31

3.4.1 Input Pipeline . . . 31

3.4.2 CNN Model . . . 33

4 **Meshed Up: Learnt Error Correction in 3D Reconstructions** 35
Introduction . . . 35
Paper . . . 36
4.1 Further Experiments . . . 43
4.1.1 Input Size . . . 43
4.1.2 Multi-view Correction . . . 44
4.1.3 Per-Sequence Breakdown . . . 44
4.2 Conclusion . . . 45

5 **Large Scale Outdoor Scene Reconstruction and Correction with Vision** 47
Introduction . . . 47
Paper . . . 48
5.1 Discussion . . . 74
5.2 Conclusion . . . 74

6 **Learning Geometrically Consistent Mesh Correction** 76
Introduction . . . 76
Paper . . . 77
6.1 Further Results . . . 88
6.1.1 Per-View Breakdown . . . 88
6.1.2 Post-Correction 3D Reconstructions . . . 89
6.2 Conclusion and Extensions . . . 93

7 **Large Scale Results** 94
7.1 The KITTI Odometry Dataset . . . 95
7.2 Experiments . . . 96
7.2.1 Inverse-Depth Refinement (2D Results) . . . 97
7.2.2 Fusing Refined Inverse-Depth (3D Results) . . . 99
7.3 Future Work . . . 102
7.4 Conclusion . . . 103

8 **Learning to Correct 3D Reconstructions from Multiple Views** 104
Introduction . . . 104
Paper . . . 105

8.1 Extensions and Future Work 115
8.2 Conclusion 116

9 Summary **117**
9.1 Future Work 118

References **120**

Acronyms

BRIEF	Binary Robust Independent Elementary Features
CNN	convolutional neural network
FAST	Features from Accelerated Segment Test
FCN	fully-convolutional network
GLSL	OpenGL Shading Language
GNSS	Global Navigation Satellite System
GPU	graphics processing unit
MLP	multi-layer perceptron
ORB	Oriented FAST and Rotated BRIEF
PNG	Portable Network Graphics
RMSE	root mean square error
SGD	stochastic gradient descent
SLAM	Simultaneous Localisation and Mapping
TGV	total generalised variation
TPU	tensor processing unit
TSDF	truncated signed distance function
TV	total variation

1 Introduction

This thesis is about reducing the cost of building dense 3D reconstructions. Dense reconstructions are central to many applications in robotics and beyond. They reduce the risk of surveying potentially dangerous industrial facilities (such as nuclear power plants) by allowing remote inspection. They are essential in cultural heritage, archaeology, and digital preservation, where digital replica projects such as Open Heritage[1] and The 3D Museum[2] make historically important sites and artefacts more accessible and facilitate research. In robotics, dense reconstructions are useful across the autonomy stack: they help robots localise, plan, and navigate within as well as interact with their operating environments.

Despite many advances in algorithms as well as hardware – both in terms of sensing, with availability of consumer-grade depth cameras and cheaper lidar scanners, and in terms of compute, with ever-faster and bigger graphics processing units (GPUs) – building good 3D reconstructions remains an expensive, challenging task. For example, when using cameras as a source of data, reconstructions contain visible errors where depth estimation is ambiguous. Furthermore, even if the estimated depth is of good quality, there are parts of the scene that are unobservable from the camera (e.g. behind cars, as illustrated in Figure 1.1), which will result in incomplete reconstructions. This is usually addressed by careful scanning of a scene, but at an increased cost – scanning large ancient sites for digital preservation can take a team of people days or weeks of work, and requires tens of thousands of photos and laser scans.[3]

Key to mitigating the high cost of dense reconstructions is the observation that human-made environments and artefacts exhibit a high degree of geometric regularity. For example, indoor areas contain largely planar surfaces and common types of objects – chairs, tables, sofas, etc.; in urban scenes, the road is typically horizontal, and buildings vertical; human-made objects often contain aesthetically pleasing symmetries and repetitions. Such regularities can be leveraged to handle noisy, sparse, or missing data, and indeed

[1] The Open Heritage project, a collaboration between Google Arts & Culture and CyArk, provides online access to 3D reconstructions of world heritage sites such as the ancient city of Corinth. The project can be accessed at https://artsandculture.google.com/project/openheritage.

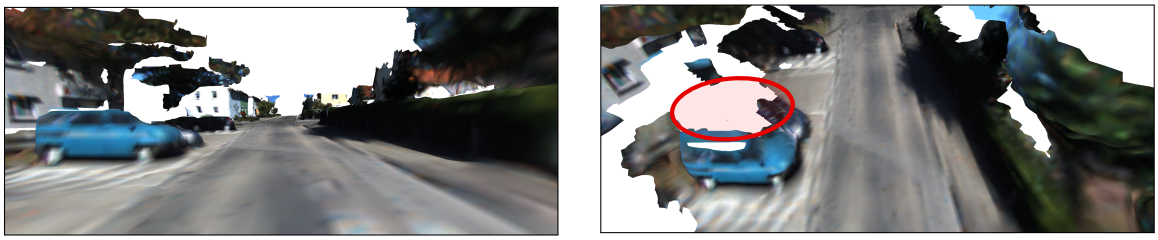
[2] The 3D Museum are preserving objects of cultural significance in a library of 3D models. The museum is available at http://www.the3dmuseum.com/.

[3] For example, the effort to preserve an ancient site in the Peruvian Andes has been documented by CyArk at https://cyark.org/about/documenting-a-site-at-10000-feet.

many of today's 3D reconstruction systems use filters and regularisation to encode this prior knowledge.

Instead of focusing on a particular dense reconstruction system, in this thesis we look at the *output* of such a system and ask how we could correct an arbitrarily generated mesh. In particular, we look at how we can correct meshes of large-scale urban scenes generated from a forward-facing road-level stereo camera. This is an especially challenging scenario for several reasons:

- There are very few observations per surface area;
- Stereo cameras yield noisy depth measurements, especially in poorly textured areas that abound in urban scenes;
- The camera viewpoint leads to missing observations in many parts of the scene, as illustrated in Figure 1.1.



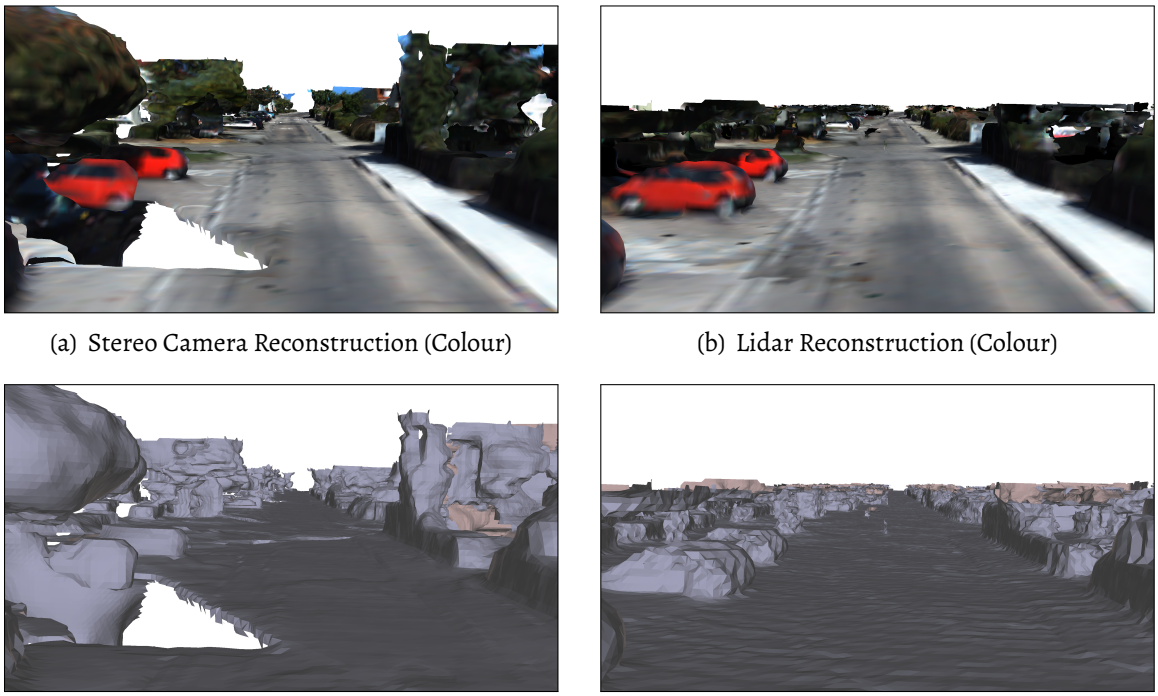
(a) Road-level View (b) Bird's-eye View

Figure 1.1: Unobservable regions from camera viewpoint. This figure shows two images of the same reconstruction that was built from road-level depth estimated from stereo pairs. The estimated depth is of high-quality, which results in a good reconstruction from the input road-level viewpoint (a). However, when other viewpoints are considered, such as the bird's-eye view in (b), unobserved regions can be seen, such as behind cars (highlighted in red). Instead of having to collect more data, we propose to fix such issues and more with a learned model capable of capturing geometric priors.

Existing mesh correction algorithms are not designed to operate in such scenarios – local, scalable methods focus primarily on filling relatively small holes or smoothing the mesh surface without taking much context into account, and global methods are generally object-centric and produce watertight meshes which are not suitable for large-scale outdoor scenes. Furthermore, many existing methods rely on some user interaction to

achieve good results. The method we propose, in contrast, is fully automatic, and is capable of correcting errors of varying scale and severity.

Our learning-based mesh correction method proceeds as follows. At training time, we start with two reconstructions of the same scene: a stereo camera reconstruction that we learn to correct, and a high-quality lidar reconstruction that serves as the supervising signal. An illustration of the two types of reconstructions is shown in Figure 1.2. From each reconstruction, we render 2D images capturing the appearance and geometry of the mesh (as detailed in Section 3.3), and train a convolutional neural network (CNN) to predict corrected depth. When deployed, the predictions are used to create a new, corrected reconstruction, reducing error by up to 39.1%.



(a) Stereo Camera Reconstruction (Colour)

(b) Lidar Reconstruction (Colour)

(c) Stereo Camera Reconstruction (Mesh)

(d) Lidar Reconstruction (Mesh)

Figure 1.2: Illustration of two reconstructions of the same urban scene from the KITTI Odometry Dataset (Andreas Geiger, Philip Lenz, et al., 2012). The reconstruction on the left was generated from stereo camera data, while the one on the right was generated from lidar data. The stereo camera reconstruction contains more errors, such as the whole in the road. For the method we present in this thesis, we use the lidar reconstruction to train a CNN capable of fixing errors in stereo camera reconstructions.

1.1 Thesis Overview

This thesis proposes and refines a method for mesh correction that reduces the cost of building good 3D reconstructions. Researching this topic has resulted in several academic articles that are reproduced and discussed in their own respective chapters, as indicated in Section 1.2.1. In the following paragraphs we give an overview of how this thesis develops the idea of mesh correction.

Background

We start in Chapter 2 by providing a context of relevant literature and the geometrical tools necessary for understanding this thesis. In particular, we discuss convolutional neural networks and architectural choices relevant to our work. We introduce the camera model used in our work and present existing dense 3D reconstruction and mesh correction algorithms.

Mesh Correction System

Our mesh correction method is part of a system we designed and implemented to enable our research. Chapter 3 gives an overview of how different components interact to produce better reconstructions. It details training data generation and storage, and discusses the practical aspects of training and evaluating a mesh correction CNN.

Meshed Up: Learnt Error Correction in 3D Reconstructions

The first instantiation of our mesh correction work, presented in Chapter 4, asks whether we can improve stereo camera reconstructions given the images used to build them and the resulting mesh. We experiment with the different types of information that can be extracted from a mesh, and investigate a simple method for processing multiple views, which prompts our later research. For this initial work, we evaluate the ability of our method to refine inverse-depth as a proxy for improving 3D reconstructions.

Large Scale Outdoor Scene Reconstruction and Correction with Vision

Chapter 5 demonstrates our method in practice, in a large scale reconstruction system. It addresses the overly-smooth predictions of our previous work by using a better-suited

CNN architecture, and, unlike previous work, evaluates the *meshes* that result from our method, instead of the predicted inverse-depth.

Learning Geometrically Consistent Mesh Correction

Since we learn to correct views from a static mesh, predictions from our method should yield the same geometry. In Chapter 6 we make two contributions that improve on our previous work. First, we introduce a loss that ensures predictions on nearby views result in consistent geometry. Second, we notice that the real colour images we used in previous work can be replaced with synthetic ones without a decrease in performance. This enables us to generate vastly more training data from entirely novel viewpoints which benefit from the geometric consistency loss.

Large Scale Results

Previous chapters used several CNN architectures, viewpoints, 2D or 3D evaluation, and either real or synthetic colour images. Chapter 7 consolidates this and provides a shared framework for evaluating mesh correction, to better understand the individual contributions of different parts of our research. Furthermore it extends the dataset to reconstructions built from 21.8 km of driving, demonstrating that our method scales to large scenarios.

Learning to Correct 3D Reconstructions from Multiple Views

The approach to multiview geometric consistency introduced in Chapter 6 is limited to a supervisory signal during the training of a CNN, while when deployed predictions are still made one view at a time. In Chapter 8 we present a novel method for aggregating information from arbitrary views in a CNN. We observe that significantly different viewpoints result in view-dependent representations within the network and introduce a method for transforming features between viewpoints that can be used both during training and when the method is deployed.

Summary

Chapter 9 summarises this thesis. It discusses possible directions for future work and provides the concluding remarks.

1.2 Contributions

This thesis presents and refines the idea of mesh correction as a supervised learning problem. In developing this idea, we make the following contributions:

1. A method for reducing the cost of building good dense 3D reconstructions by correcting *meshes*;
2. The development of a system for rendering training data from pairs of meshes, which we use to train and evaluate a CNN that corrects reconstructions;
3. An evaluation on a real-world large-scale dataset (21.8 km of driving) of our mesh correction system, showing that it reduces reconstruction error by up to 39.1%;
4. A novel method for aggregating information from arbitrary viewpoints in multiview CNN architectures.

1.2.1 Publications

The core findings of this thesis are presented as publications that follow the idea of mesh correction introduced at the beginning of this chapter:

- Tanner, M., Săftescu, Ș., Bewley, A., & Newman, P. (2018). Meshed up: Learnt error correction in 3D reconstructions, In *Proceedings of The IEEE International Conference on Robotics and Automation (ICRA)*, Brisbane, Australia. Presented in Chapter 4.

- Tanner, M., Piniés, P., Paz, L. M., Săftescu, Ș., Bewley, A., Jonasson, E., & Newman, P. (2020). Large scale outdoor scene reconstruction and correction with vision. *The International Journal of Robotics Research (IJRR)*. Presented in Chapter 5.

- Săftescu, Ș., & Newman, P. (2020). Learning geometrically consistent mesh corrections, *Proceedings of The International Conference on Computer Vision Theory and Applications (VISAPP)*, Valletta, Malta. Presented in Chapter 6.

- Săftescu, Ș., Gadd, M., & Newman, P. (n.d.). Look here: Learning geometrically consistent refinement of inverse-depth images for 3D reconstruction [Under review.]. *The International Journal of Pattern Recognition and Artificial Intelligence*

- Săftescu, Ș., & Newman, P. (2021). Learning to correct 3D reconstructions from multiple views, *Proceedings of The International Conference on Computer Vision Theory and Applications (VISAPP)*. Presented in Chapter 8.

2 Background

This thesis presents a mesh correction system that reduces the cost of building large-scale reconstructions. Our contributions are elaborated as papers in Chapters 4 to 6 and 8, and each comes with a self-contained review of literature, which we will not fully reproduce here. This chapter will instead focus on the background concepts required for an understanding of the publications.

We start in Section 2.1 by presenting CNNs, which we use to capture prior knowledge of large-scale scenes. We introduce in Section 2.2 the camera model theory required for rendering and processing images, and for training our CNN models, with emphasis on depth and disparity images. In Section 2.3, we provide an overview of 3D reconstruction systems that are used to fuse depth data into 3D meshes. Finally, Section 2.4 looks at existing mesh repair methods and introduces the evaluation metrics necessary for comparing large-scale meshes.

2.1 Convolutional Neural Networks

At the core of our mesh correction method lies a CNN that processes mesh features rendered from a 3D reconstruction and produces refined inverse-depth images. This section provides the context and introduces the main neural network concepts we use in our system.

Neural networks have been studied for decades, with early work on artificial neurons by McCulloch and Pitts (1943) and on the perceptron model by Rosenblatt (1957). A typical feed-forward network consists of artificial neurons organised in layers, with weighted connections between the neurons on adjacent layers. While some learning methods were developed for early models (Rosenblatt, 1960), effective training of neural networks was only achieved decades later with the presentation, in several independent formulations, of the error backpropagation algorithm (e.g. LeCun, 1985; 1986; Parker, 1985; Rumelhart, Hinton, et al., 1986a; b; Werbos, 1974).

Neural networks are trained by minimising a loss which encodes the task being solved. This minimisation is usually done using first-order optimisers based on stochastic gradient descent (SGD) (Robbins and Monro, 1951), such as Momentum (Polyak, 1964) or Adam

(Kingma and Ba, 2015). First-order optimisers apply backpropagation to obtain the gradient of the loss function with respect to the neural network weights, and iteratively update the weights in a direction that decreases the loss. For a long time, first-order methods were the only ones employed for optimising neural networks, due to the large number of parameters that need to be updated – second-order solvers based on the Newton's method usually need to invert the Hessian matrix or an approximation of it, and this is intractable for neural networks. However, recent work by Henriques, Ehrhardt, et al. (2019) addresses this limitation and proposes a second-order method that compares favourably to the usual first-order methods, but requires no hyper-parameter tuning. For an up-to-date comparison of optimisers used with neural-networks, we refer the reader to the survey papers by D. Choi, Shallue, et al. (2019), and Schmidt, Schneider, et al. (2020).

Several advances both in theory and in hardware have enabled the type of neural networks seen today in all fields of machine learning (LeCun, Bengio, et al., 2015; Schmidhuber, 2015). Solving the vanishing or exponential gradients problem (Glorot and Bengio, 2010) through the use of rectified-linear units (ReLU) (Nair and Hinton, 2010), or by normalising activations (e.g. Ioffe and Szegedy, 2015; Y. Wu and He, 2018), has allowed for stacking many layers to form very deep neural networks. The implementation of neural networks on GPUs (Cireșan, Meier, Gambardella, et al., 2010; Cireșan, Meier, Masci, et al., 2011; Oh and Jung, 2004) and hardware designed specifically for neural network algorithms like the tensor processing unit (TPU) (Jouppi, Young, et al., 2017) have enabled training and inference of very large models. For computer vision tasks (though not exclusively) CNNs, introduced by LeCun, Boser, et al. (1989), have allowed the transition from algorithms with hand-crafted features to models trained end-to-end in which feature extractors are learned from the data. This specialised type of neural network was shown early on to outperform traditional approaches at the task of handwritten digit recognition (LeCun, Bottou, et al., 1998), and is designed to work directly on image pixels without much preprocessing. When predictions are spatially dense – each input pixel having a corresponding output prediction – the fully-convolutional network (FCN) architecture as proposed by Long, Shelhamer, et al. (2015) has allowed for easy training and inference.

Unlike traditional feed-forward neural networks such as the multi-layer perceptron (MLP), where every neuron on one layer is connected to every other neuron on the previous layer, in a CNN neurons on one layer are connected only to a small set of neurons on

the previous layer. Furthermore, the connection weights are shared between neurons on the same layer. This connectivity pattern is implemented with convolutional filters whose weights can be learned, and neurons on each layer are arranged in multi-channel images called *feature maps* (LeCun, Boser, et al., 1989). A good reference for the convolutional arithmetic and resulting connectivity patterns can be found in the guide written by Dumoulin and Visin (2016).

In addition to convolutional layers, several other architectural patterns have emerged in recent years that we make use of in our work and that we describe in the following paragraphs.

Encoder-Decoder: When using CNN models for per-pixel prediction tasks such as semantic segmentation or depth prediction, it is usual for the architecture to first downsample the feature maps through successive pooling operations, and then successively upsample them to the size of the input image (e.g. Laina, Rupprecht, et al., 2016; Noh, Hong, et al., 2015). The downsampling stage, often called the encoder, increases the size of the receptive field that an output pixel draws information from, and reduces the amount of computation required. The upsampling stage, often called the decoder, recovers the resolution lost while downsampling, and refines the prediction.

U-Nets: One of the drawbacks of downsampling is loss of resolution, which results in blurry predictions. To overcome this, Ronneberger, Fischer, et al. (2015) propose U-Net, an encoder-decoder variant where feature maps from the encoder are concatenated with feature maps at the corresponding scale in the decoder. These "skip" connections preserve the high-frequency information, such as edges, thus preventing blurriness, while maintaining the advantages of downsampling.

Residual Networks: Network depth has been shown to improve predictions considerably (Simonyan and Zisserman, 2015; Szegedy, W. Liu, et al., 2015). However, even with well-behaved gradients, He, Zhang, et al. (2016a) show that training very deep (100+ layers) neural networks is challenging. To solve this issue they propose a structure where each group of successive layers learn a *residual* for the previous shallower network. He, Zhang, et al. (2016b) further improve ResNets by keeping a branch of the network completely free of

non-linearities, thus allowing better flow of information. With this structure, they show that neural networks of up to 1001 layers can be effectively trained.

View Synthesis as Supervision: Multiple views are related by the geometry of the underlying scene. Hence, given the scene geometry and the relative pose between views, one view can be synthesised from another, up to occlusions, dynamic objects, or view-dependent appearance changes (e.g. reflections). Leveraging this, view synthesis has recently been used as a supervisory task to train CNNs for many geometric problems, such as disparity estimation (Godard, Mac Aodha, et al., 2017), depth prediction and odometry (Mahjourian, Wicke, et al., 2018; T. Zhou, Brown, et al., 2017), optical flow (Ahmadi and Patras, 2016; Y. Wang, Yang, et al., 2018), etc. Key to the success of this approach is implementing view synthesis in a differentiable manner compatible with backpropagation, an idea introduced for image sampling by Jaderberg, Simonyan, et al. (2015).

2.2 Camera Geometry

Fundamental to the understanding of our system are the basics of camera geometry. These concepts appear both when building 3D reconstructions from images, as well as when we generate the inputs to our mesh correction CNN, as described in Section 3.3. In the following sections we introduce the single and multiple view geometry necessary for our system, and refer the reader to Hartley and Zisserman (2004) for an in-depth treatment of the subject.

2.2.1 Pinhole Camera Model

With roots in the ancient *camera obscura*, the pinhole camera model is commonly employed in computer vision. Abstractly, this model consists of a point called the camera centre, and a projection plane, also called image plane, onto which the image is formed. The image of an object appears at the intersection between light rays that pass through the camera centre and the projection plane. In the *camera obscura* experiment, the camera centre is a small hole (as those made by the tip of a pin) in a screen that lies between an object and the image plane (a wall or another screen), but this model is more commonly depicted as in Figure 2.1 with the image plane between the object and the camera centre. The only difference between the two representations is the sign of the focal length, f, in the equations

below. Due to its simplicity, the geometry of most modern cameras is described using the pinhole camera model, though other models exist for cameras with wide viewing angles or omnidirectional cameras (e.g. Yagi and Kawato, 1990).

The 2D coordinates, (p_u, p_v), of the image point are related through similar triangles to the 3D coordinates, (p_x, p_y, p_z), of the world point:

$$\frac{p_u}{f} = \frac{p_x}{D} \qquad \text{and} \qquad \frac{p_v}{f} = \frac{p_y}{D}, \tag{2.1}$$

where f is the focal length of the camera, and $D = p_z$ is the depth of the world point. Note that, as illustrated in Figure 2.1, the focal length is measured in the same units as the image points. For example, for digital images it is measured in pixels and is determined not only by the physical distance between the camera centre and the image plane, but also the physical size of a pixel.

To facilitate the use of linear algebra, the image point can be represented in homogeneous coordinates using the following mapping:

$$\mathbf{P}_{2D} = \begin{bmatrix} p_u \\ p_v \end{bmatrix} \mapsto \begin{bmatrix} \lambda p_u \\ \lambda p_v \\ \lambda \end{bmatrix} \qquad (\forall)\, \lambda \in \mathbb{R} \setminus \{0\}, \tag{2.2}$$

where λ is a free scaling factor indicating that homogeneous points which differ only in scale are equivalent. This equivalence arises from the fact that all 3D points on a ray passing through the camera centre project onto the same image point, and allows us to express the ratios in Equation (2.1) in terms of matrix multiplication. Picking $\lambda = 1$ and using the tilde

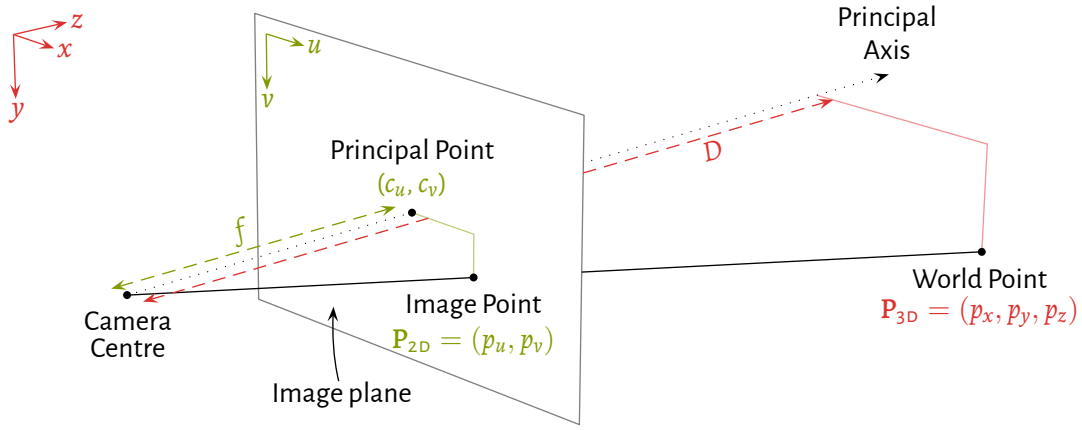


Figure 2.1: Illustration of the pinhole camera model.

to indicate homogeneous coordinates, we obtain the following:

$$\tilde{\mathbf{p}}_{2D} = \begin{bmatrix} p_u \\ p_v \\ 1 \end{bmatrix} = \begin{bmatrix} fp_x/p_z \\ fp_y/p_z \\ 1 \end{bmatrix} \sim \begin{bmatrix} fp_x \\ fp_y \\ p_z \end{bmatrix} = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} = \mathbf{K} \cdot \mathbf{P}_{3D}. \tag{2.3}$$

In the equation above, the intrinsic matrix, $\mathbf{K}$, contains the parameters that map world coordinates to image coordinates. A few generalisations are possible:

- If the principal point in Figure 2.1 is not at the origin of the image coordinate system, then two parameters, c_u and c_v, are introduced to translate image coordinates. This is often necessary when using digital cameras, since the origin of pixel coordinates is usually in the top-left corner of the image.
- If the two image axes do not have equal scale, then two focal lengths, f_u and f_v, are used. This arises in images where pixels are not square, as is the case in some digital image and video formats.
- If the two image axes are not orthogonal, then a skew parameter, s, is introduced.

Taking these generalisations into account, the full camera matrix becomes:

$$\mathbf{K} = \begin{bmatrix} f_u & s & c_u \\ 0 & f_v & c_v \\ 0 & 0 & 1 \end{bmatrix}. \tag{2.4}$$

2.2.2 Camera Location and Orientation

The equations in the previous section all assume that the camera is located at the origin of the 3D space. However, as the camera moves relative to the world, this assumption does not hold. To account for the camera motion, points in the world coordinate frame are transformed in the camera frame with a rotation and a translation:

$$\mathbf{P}_{3D} = \mathbf{R} \cdot \mathbf{P}_{3D}^{world} + \mathbf{t}, \tag{2.5}$$

where $\mathbf{R}$ is a rotation matrix ($\mathbf{R}^{-1} = \mathbf{R}^{T}$, $\det(\mathbf{R}) = 1$), and $\mathbf{t}$ a translation vector.

Analogously to Equation (2.2), we can express world points in homogeneous coordinates:

$$\mathbf{P}_{3D} = \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} \mapsto \begin{bmatrix} \omega p_x \\ \omega p_y \\ \omega p_z \\ \omega \end{bmatrix} \qquad (\forall)\ \omega \in \mathbb{R} \setminus \{0\}, \tag{2.6}$$

where ω is the scaling factor. This allows us combine the rotation and translation in Equation (2.5) into a single transformation matrix, $\mathbf{T}$:

$$\tilde{\mathbf{P}}_{3D} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix} \tilde{\mathbf{P}}_{3D}^{world} = \mathbf{T} \cdot \tilde{\mathbf{P}}_{3D}^{world}. \tag{2.7}$$

An additional advantage of using homogeneous coordinates is that points infinitely far away are handled no differently than points finitely far away. Such points take the form $[x, y, z, 0]^T$ in homogeneous coordinates and are only affected by the rotation component of $\mathbf{T}$ and not the translation.

Finally, combining Equations (2.3) and (2.7), we can obtain the image coordinates of a world point directly:

$$\tilde{\mathbf{P}}_{2D} = \begin{bmatrix} \mathbf{K} & \mathbf{0} \end{bmatrix} \cdot \mathbf{T} \cdot \tilde{\mathbf{P}}_{3D}^{world}, \tag{2.8}$$

where the zero column added to $\mathbf{K}$ bridges 3D and 2D homogeneous coordinates.

2.2.3 Point Correspondence

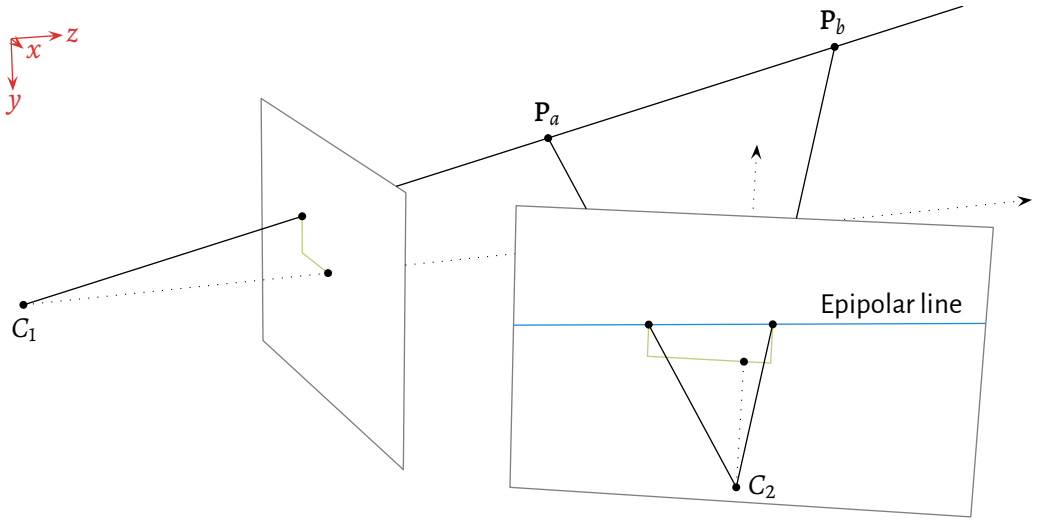


Figure 2.2: Image point correspondence between two cameras, C_1 and C_2. Points $\mathbf{P}_a$ and $\mathbf{P}_b$ are colinear with C_1 and therefore project onto the same image point in the first view – as do all other points on this line. When searching for corresponding image points in C_2, these will lie along an epipolar line.

Often we deal with multiple images of the same scene, either captured by the same camera as it moves, or captured by multiple cameras. Establishing a correspondence between points in two images allows us to recover the underlying 3D structure that is pro-

jected onto the images – 3D points will lie at the intersection of the rays passing through corresponding 2D image points.

Due to the pinhole camera geometry described in Section 2.2.1, all points on a ray passing through the camera centre will project onto the same image point. As illustrated in Figure 2.2, corresponding points from another image will lie on an epipolar line – the projection of the ray onto the other image. This geometric relation means that searching for point correspondences is a 1D problem.

A common simplification occurs when the two images are aligned such that epipolar lines align with image rows and corresponding points are found on the same row. Such a setup is referred to as a rectified stereo image pair, and is found in stereo cameras.

2.2.4 Disparity, Depth Maps, and Inverse-Depth

Often, point correspondences are computed densely for every pixel in an image. When the images are rectified, point correspondences are found at offsets on the same row, so can be expressed as disparities, δ:

$$\begin{bmatrix} p_u^1 \\ p_v^1 \end{bmatrix} = \begin{bmatrix} p_u^2 + \delta \\ p_v^2 \end{bmatrix}, \tag{2.9}$$

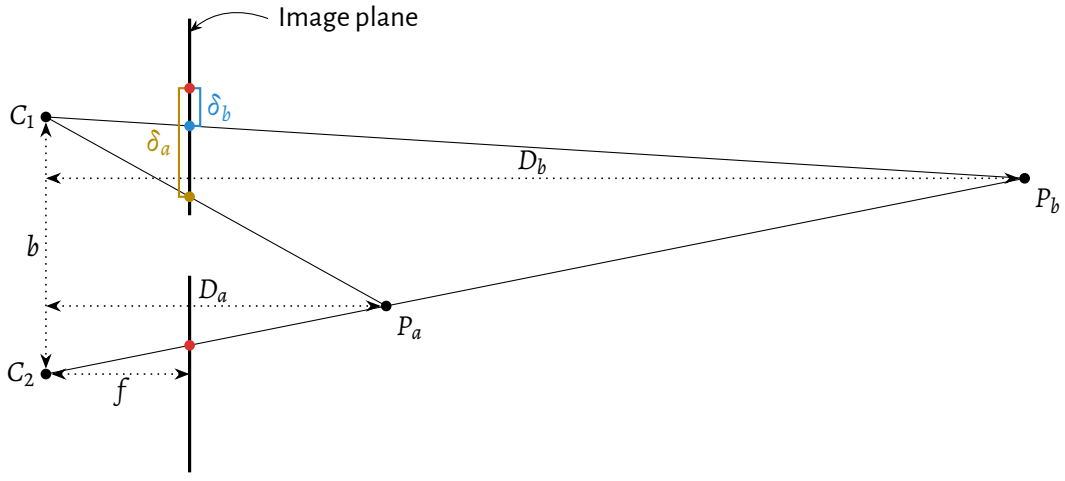


Figure 2.3: A one-dimensional illustration of a rectified stereo camera setup and the relation between disparity and depth. The two cameras, C_1 and C_2, are facing the same way and have a baseline distance between them, b. Two points, P_a and P_b, that project on the same place of the C_2 image plane (•), will project in different places on the image plane of C_1 (• and •) depending on how far away they are from the cameras.

which form a *disparity map* – an image whose intensity values represent disparities. As illustrated in Figure 2.3, disparity is proportional to the focal length, f, the (baseline) distance between cameras, b, and inversely proportional to depth, D:

$$\delta = \frac{fb}{D}. \tag{2.10}$$

It is therefore easy to convert between a disparity map and a depth map when the focal length, f, and stereo baseline, b, are known.

The images we use for our method are synthetic, being rendered from a 3D reconstruction as described in Section 3.3 and are therefore not tied to any particular camera setup. However, the properties of disparity maps are advantageous for our method – they assign more weight to foreground objects, and they can be resampled with linear interpolation, a convenient property for the geometric consistency loss in Chapter 6. We therefore use *inverse-depth images*, noting that inverse-depth, d, is proportional to disparity but does not depend on camera parameters:

$$d = \frac{1}{D} = \frac{\delta}{fb}. \tag{2.11}$$

Stereo Matching

Finding point correspondences between pairs of images is known as stereo matching. This is a well-researched area and many different algorithms have been proposed. One common approach for dense methods, as summarised by Scharstein and Szeliski (2002), is to compute a matching cost for each pixel at each disparity value in a fixed range, build a cost volume from which disparity can be computed, and finally refine the initial disparity map estimate. Local algorithms, such as Yoon and Kweon (2006), or Weber, Humenberger, et al. (2009), aggregate the cost volume to remove noise and obtain smoother disparity, while global algorithms, such as Felzenszwalb and Huttenlocher (2004), Kolmogorov and Zabih (2001), Ranftl, Gehrig, et al. (2012), or Woodford, Torr, et al. (2008), make explicit smoothness assumptions and obtain disparity estimates through optimisation.

More recently, the components of stereo matching algorithms have been replaced with learnable models. For example, Žbontar and LeCun (2016), or Luo, Schwing, et al. (2016) trained networks that produced a matching score for a pair of image patches, while the remaining steps in the stereo matching algorithm were kept unchanged, and showed that learning the matching cost computation results in significant improvements in disparity

estimation. Others (e.g. Chang and Y.-S. Chen, 2018; Kendall, Martirosyan, et al., 2017; Khamis, Fanello, et al., 2018; Tulyakov, Ivanov, et al., 2018) keep the steps of the stereo matching algorithm distinct, but replace traditional algorithms for feature extraction and cost volume regularisation with 2D and 3D convolutional layers. In the interest of speed, Yee and Chakrabarti (2020) use fast traditional matching costs to build a cost volume, and train a CNN to predict disparities from this volume, while Mayer, Ilg, et al. (2016) replace the whole stereo matching algorithm with a CNN that directly predicts disparity and is trained end-to-end.

2.3 Dense 3D Reconstruction

This thesis focuses on correcting meshes that are output by dense 3D reconstruction systems. In this section we summarise the context for the system we use in our work. An overview of the system components is given in Section 3.2, and the theory and implementation is described in detail in the paper presented in Chapter 5.

The task of a dense 3D reconstruction system is to fuse depth (and optionally colour or intensity) data into a representation from which a surface can be rendered or extracted. Often, the reconstructed scene is static, or only the static parts of the scene are reconstructed, with dynamic objects being masked out or treated as outliers in the data. The most common approaches are based on the formative volumetric fusion work of Curless and Levoy (1996), who use a voxel grid representation into which depth measurements are accumulated as a truncated signed distance function (TSDF). A surface mesh can be extracted from the TSDF voxel grid using the marching cubes algorithm (Lorensen and Cline, 1987). Algorithmic and hardware advances lead to KinectFusion (Newcombe, Izadi, et al., 2011), the first real-time RGB-D dense reconstruction system, based on the Kinect depth camera. For large-scale reconstructions, a dense voxel grid representation is inefficient – as most of the scene will contain empty space. This limitation is addressed with space-partitioning methods such as octrees (e.g. Zeng, Zhao, et al., 2012; 2013), or using the voxel hashing method introduced by Nießner, Zollhöfer, et al. (2013). These approaches only allocate voxels close to surfaces, where the TSDF is defined. Ensuing methods focus on robustness (S. Choi, Q.-Y. Zhou, et al., 2015), and the ability to dynamically change the pose of depth measurements in the volumetric representation (Dai, Nießner, et al., 2017). In our mesh correction system, we use BOR^{2}G (Tanner, Piniés, et al., 2015), which combines

the voxel hashing method of Nießner, Zollhöfer, et al. (2013) with a 3D total variation (TV) regulariser implemented over the compressed data structure, and demonstrates success in large-scale outdoor scenes.

Recently, learning-based methods that use neural networks have been proposed, although they largely focus on single object or small-scale reconstruction. Early work uses a latent vector representation that is decoded into a voxel grid with convolutional operations (e.g. Wiles and Zisserman, 2017; Z. Wu, Song, et al., 2015). Volumetric representations are also abundant since the regularity of voxel grids is attractive for convolutional operations. Learnt Stereo Machines (Kar, Häne, et al., 2017) learn to perform multi-view stereo and reconstruct objects from multiple RGB views without explicitly requiring depth measurements, while Sitzmann, Thies, et al. (2019) optimise a volumetric representation of a scene that can subsequently be used to render arbitrary views. Larger-scale methods are presented by Cherabier, Schönberger, et al. (2018), who follow the same system structure as traditional depth fusion, but unroll the 3D TV regularisation step into a 3D CNN that learns semantic-dependent regularisation kernels, and by Weder, Schönberger, et al. (2020), who learn the depth-map correction and the TSDF update steps of traditional pipelines. Space partitioning complicates the implementation of convolutional operations, but methods based on the OctNet architecture (Riegler, Ulusoy, and Andreas Geiger, 2017), such as OctNetFusion (Riegler, Ulusoy, Bischof, et al., 2017) can achieve much better resolutions. Another type of representation proposed for learning 3D reconstruction is proposed by Mescheder, Oechsle, et al. (2019) with occupancy networks. In this representation, surface is implicitly defined to be the decision boundary of a neural network classifier that labels arbitrary 3D points as inside or outside an object. Niemeyer, Mescheder, et al. (2020) show how this implicit representation can be differentiably rendered, and train models without explicit 3D supervision. Similarly, Mildenhall, Srinivasan, et al. (2020) propose neural radiance fields, which assign colour and opacity to arbitrary 3D points and viewing directions, and are trained using only 2D supervision, recovering scene structure in remarkable detail after several days of training.

2.4 Mesh Correction

Polygon meshes are a common output format for dense 3D reconstruction systems. They represent surfaces in a 3D space with a set of vertices and polygons (triangles in the sim-

plest form) defined over the vertices.

Meshes built from data often contain errors. Attene, Campen, et al. (2013) classify mesh defects into two categories: *topological* and *geometrical* defects. Topological defects occur when the polygons of a mesh do not describe a 2D surface, for example when an edge is shared by more than two faces. Geometrical errors occur when the surface defined by the mesh does not correspond to the real surface being modelled, for example due to noise, or missing areas in the mesh. In this thesis, we focus on correcting geometrical errors which predominate in large-scale outdoor reconstructions.

A further distinction can be made between *local* methods and *global* methods. Local methods change meshes only around defects, preserving the existing correct structure of the mesh. Global methods, on the other hand, repair the mesh as a whole, and usually involve a different intermediate representation. As a result, such methods often involve remeshing, thus changing existing structure. Local methods are easier to scale, but take into account less context, while global methods can leverage full context. Our method lies in-between the two approaches. We use 2D mesh features (Section 3.3) as an intermediate representation that provides a large but local context, and requires remeshing. However, our mesh features can be rendered at any location, so our method scales to arbitrarily large meshes.

Local Methods: Early local methods for mesh repair aim to fill gaps and holes in meshes either by stitching nearby matching vertices together (Barequet and Sharir, 1995) or by simply merging nearby edges (Barequet and Kumar, 1997). Nguyen, Yuan, et al. (2005) propose a method for hole filling and geometry completion that parametrises the 3D mesh surface in the neighbourhood of defects, and uses 2D texture synthesis to fill in missing surface. To improve consistency with neighbouring geometry, C.-Y. Chen and Cheng (2008) fill holes taking into account sharp features in the mesh, Panchetti, Pernot, et al. (2010) use images to constrain and guide the generated surface when filling in holes, while X. Wang, Hu, et al. (2018) first fill in holes with a smooth surface, then recover high-frequency detail from other similar regions in the mesh. In a similar method to the one proposed by Nguyen, Yuan, et al. (2005), Sarkar, Varanasi, et al. (2017; 2018) use CNNs to learn how to correct local patches of mesh surface parametrised as 2D height maps. Depending on how noisy the input data is and which method is employed to construct a mesh, sharp features such as corners and edges can be flattened, an artefact that the method proposed by Attene,

Falcidieno, et al. (2005) addresses. With the same goal, Fan, Yu, et al. (2010) observe that sharp features are often found at the intersection of piece-wise smooth regions and take that into account when denoising meshes. Early mesh denoising methods, such as Taubin (1995), used a low-pass filter that did not shrink the surface, but was applied equally everywhere, thus removing sharp features. Inspired by image-based denoising, Fleishman, Drori, et al. (2003) propose a denoising method with a bilateral filter that preserves sharp features.

Global Methods: Many global methods try to recover missing surface by assuming the output meshes should be watertight – with a well defined interior and exterior, and no holes or gaps in the surface. To that end, Murali and Funkhouser (1997) propose a mesh-driven space partitioning method aligned with the polygons of the input mesh. This allows them to recover the solid boundary consistent with existing geometry, repairing holes and gaps. Ju (2004) uses an octree grid representation to discriminate between the interior and exterior of solids. Shen, O'Brien, et al. (2004) propose a method that interpolates existing geometry with a continuous surface, producing watertight meshes. Similarly, Centin, Pezzotti, et al. (2015) build a Poisson surface reconstruction (Kazhdan, Bolitho, et al., 2006) from an existing mesh, and sample it to complete missing surface. Zhong and Qin (2016) transform the mesh into a low-dimensional spectral representation from which the missing surface can be recovered. Other global methods include template-based approaches, such as the one proposed by Kraevoy and Sheffer (2005), where meshes are replaced by deformed templates from a mesh library, but such approaches are limited to a small number of object classes.

2.4.1 Measuring Mesh Correctness

There are generally two approaches for comparing meshes – either comparing their volumes or comparing their surfaces. The former set of methods is applicable for meshes that describe closed surfaces and that are usually watertight, while the latter can be used with any type of mesh. Since the focus of our work is correcting large-scale outdoor 3D reconstructions, which are not usually represented by closed surfaces, we will only use the latter approach for evaluating the quality of our meshes.

Surface error: To compute surface error between a mesh, M, and a reference mesh, R, we uniformly sample the surface of M at a rate of 10 samples/m^2. For each sample, we compute the distance to the closest point in R, and calculate the mean and median over all the samples.[1] This method is not symmetric, so for a corrected reconstruction, C, and the ground-truth lidar reconstruction, L, we give the following interpretation to the two directions:

- Correctness Error ($C \to L$). This direction measures how correct *existing* surface is, but fails to capture missing surface. In the extreme case, where C has no surface, this will be 0 and therefore not very informative.
- Completeness Error ($L \to C$). This direction measures how complete the corrected mesh is. Where there is surface missing, the distance from L to C will be high. However, this direction does not capture unnecessary surface.

The two directions are complementary, so we report both of them when comparing meshes.

Total Error: To facilitate model comparison, it is necessary to select an appropriate ordering. Intuitively, models that are both correct and complete should rank better over models that are just very complete or very correct. When computing a measure of total error from the two quantities, any p-norm with $p > 1$ will order the models according to this intuition. We choose $p = 2$, since Euclidean norm provides the best interpretability.

Finally, in addition to these three quantities, we also compute surface area, for a more concrete idea of completeness.

[1] We use `libigl` for sampling meshes and computing point-to-mesh distances. Documentation can be accessed at https://libigl.github.io.

3

Mesh Correction System

This thesis proposes a method to post-process meshes in order to improve their quality. To this end, we have employed an existing 3D reconstruction system and developed additional post-processing components. These components have been gradually improved upon in a common codebase, and our publications (Chapters 4 to 6 and 8) tracks the evolution of that codebase. The resulting system consists of the following components:

1. A large-scale 3D reconstruction pipeline which uses the following sub-components:
    a) A dense stereo depth estimation module, to generate depth data from pairs of stereo images;
    b) A localisation module that keeps track of the depth sensor location as it moves through space;
2. A module for rendering 2D mesh features from 3D meshes;
3. A CNN that uses mesh features to refine inverse-depth.

This chapter describes the resulting system and provides further context for our main contributions (Chapters 4 to 8). We first give an overview of how different components interact, in Section 3.1. We then briefly introduce the 3D reconstruction pipeline that is part of our system in Section 3.2, with a more in-depth description given in Chapter 5, followed by a description of the components that contain our core contributions in Sections 3.3 and 3.4.

3.1 Overview

A typical 3D reconstruction pipeline fuses different types of 3D information, such as depth maps and lidar pointclouds, into a common volume, and then computes the surface of the objects in the scene. Several components come together to enable effective scene reconstruction. Usually, data collected from a single location is not sufficient to fully reconstruct a scene – the sensor needs to move in order to scan previously unobserved regions, and this requires localisation of the sensor. Cameras might not directly provide 3D information, but depth maps can be estimated either from the motion of the camera, or from

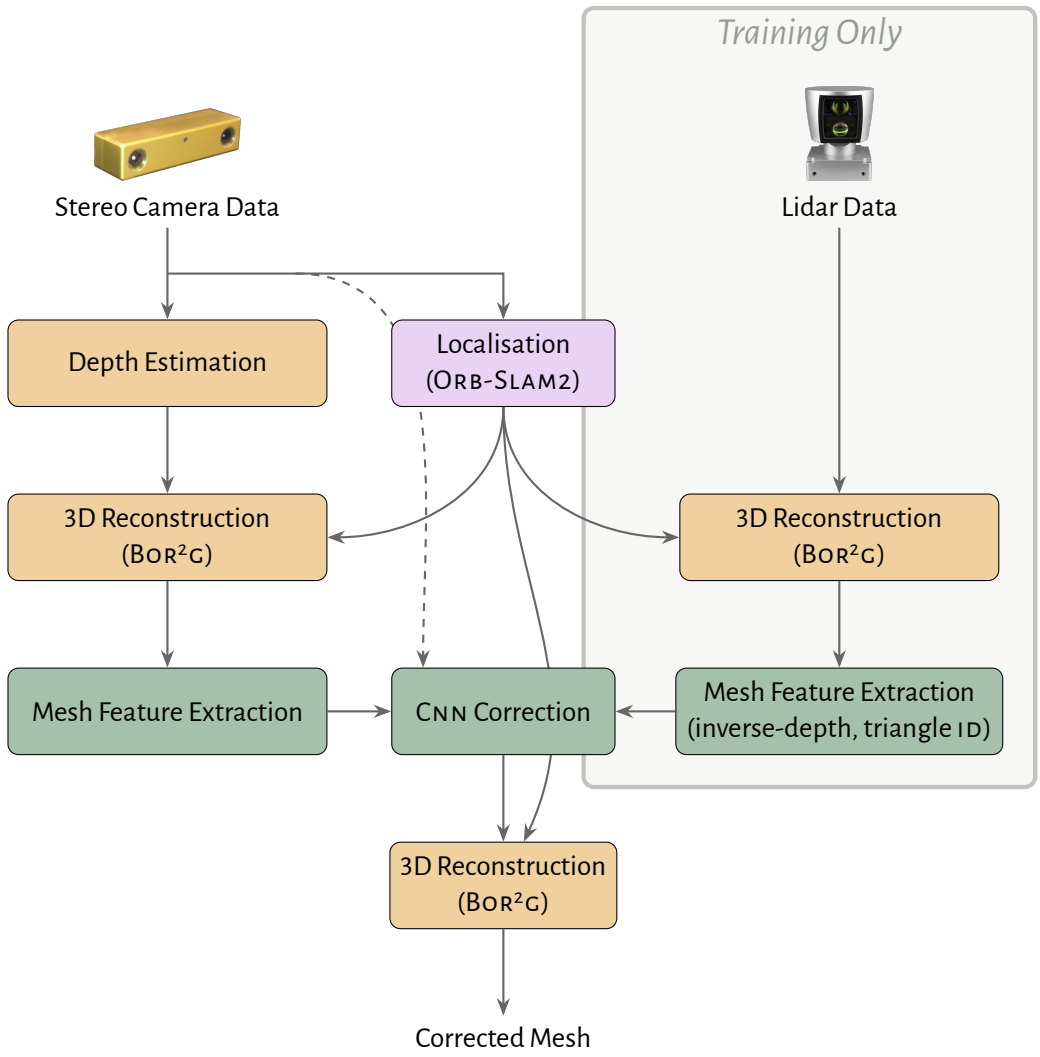


Figure 3.1: Overview of the mesh correction system proposed by this thesis. An initial reconstruction is built from stereo camera data, and then mesh features are rendered from synthetic views. A CNN learns to refine inverse-depth images, which are then fused into a new, corrected 3D reconstruction. During training, a high-quality lidar mesh is available for supervision. In Chapters 4 and 5, real images are used when correcting inverse depth (dashed arrow). In later chapters, these are replaced by synthetic colour images rendered along with the other mesh features. Shown here in green are the modules containing our core contributions. The modules in orange are discussed in detail in Chapter 5, while the purple localisation module is described in detail by Mur-Artal and Tardós (2017).

stereo pairs of images, as an additional data source that can be used for reconstruction. In addition to the usual modules, our system also contains a mesh post-processing step that corrects faults in the reconstructions arising from sensing limitations.

Figure 3.1 shows how the components in our system interact. We use stereo images from the KITTI Odometry Dataset (Geiger, Lenz, et al., 2013), with the depth estimation

method proposed by Ranftl, Gehrig, et al. (2012), and build "cheap" reconstructions – the kind we will learn to correct – with BOR²G (Tanner, Piniés, et al., 2015). In addition to the camera-based reconstructions, we require higher-quality lidar data to build meshes that we use when training our CNN model. For localisation, we use the open-source ORB-SLAM2 (Mur-Artal and Tardós, 2017). The mesh feature extraction module is implemented in OpenGL, and the CNN model is implemented and trained with TensorFlow.[1]

3.2 3D Reconstruction Pipeline

3.2.1 Localisation

Localisation – knowing where a robot is with respect to a map or to where it was before – is a core task in mobile robotics. This is closely related to mapping, another core robotics task – building a map of the operating environment. In unknown environments, where a map to localise against is unavailable, this task is known as Simultaneous Localisation and Mapping (SLAM): both the map and the location within the map are estimated at the same time.

There are many ways in which maps can be represented, depending on the application and the level of detail needed. As we are working with 3D reconstructions, we need our SLAM component to be metric, such that any two parts of a reconstruction maintain the same spatial relations as in the environment we capture.

The SLAM implementation our reconstruction pipeline uses is ORB-SLAM2 (Mur-Artal and Tardós, 2017), which has a publicly available implementation.[2] ORB-SLAM2 is an instance of visual SLAM that uses a stream of images to keep track of how a camera moves in the world. The implementation is based on sparse ORB features (Rublee, Rabaud, et al., 2011), where the metric map stores 3D locations of landmarks extracted from monocular or stereo images. To find the location of a new camera frame, the algorithm matches ORB features in the image to the ones already recorded in the map. A particularly useful element of this implementation when building large-scale reconstructions is support for loop closures – recognising when the same place is visited at a later time and adjusting any drift that might have accumulated in the estimate of the camera location.

[1] TensorFlow is an end-to-end open-source machine learning platform commonly used for training CNNs. Documentation is available at https://www.tensorflow.org.

[2] The ORB-SLAM2 implementation is available at https://github.com/raulmur/ORB_SLAM2.

3.2.2 Depth Estimation

Given a pair of rectified stereo images, disparity maps specify to what extent pixels need to be shifted along their row to align the left and right frames. The depth of the scene can easily be computed from disparity and, as seen in Equation (2.10), the two quantities are inverse-proportional.

Our 3D reconstruction pipeline uses the dense stereo energy-based disparity estimation method of Ranftl, Gehrig, et al. (2012). It proposes a regularisation term based on total generalised variation (TGV) that takes into account the correlation between discontinuities in an image and discontinuities in estimated disparity. In other words, the prior on disparity favours affine surfaces and sudden changes in disparity when they correspond to edges in the input image.

3.2.3 3D Fusion

To turn depth measurements (either depth-maps estimated from a stereo camera, or range readings from lidar) into meshes, we use BOR²G (Tanner, Piniés, et al., 2015). To obtain a mesh, BOR²G fuses depth information into a common volumetric representation: 3D space is discretised into voxels, and the TSDF is stored and updated with every new observation. A mesh is then generated from this representation by using the marching cubes algorithm (Lorensen and Cline, 1987).

Several key aspects allow BOR²G to scale to large outdoor spaces. Firstly, following the observation that most space will be empty and far away from a surface, the volumetric representation uses a data structure that only requires memory for voxels close to a surface. Secondly, it adapts a 3D TV regularisation method to this data structure, enabling efficient regularisation of large volumes. Finally, the independent nature of depth data within a scan lends itself to highly parallelised GPU implementations.

3.3 Rendering Mesh Features

When correcting reconstructions, we do not directly operate in 3D, but rather on 2D projections of meshes. This allows us to borrow from architectures that operate on images.

The meshes we correct have been built by fusing many depth observations as described in Section 3.2.3 and Chapter 5, and we would like to use as much of the existing information

about geometry as possible when refining them. To achieve this, for each view that we process, we render several *mesh features* that encode the properties of the visible portion of the underlying mesh. As we want to ensure good coverage of the mesh surface, we generate mesh features following the trajectory used when building the initial reconstruction.

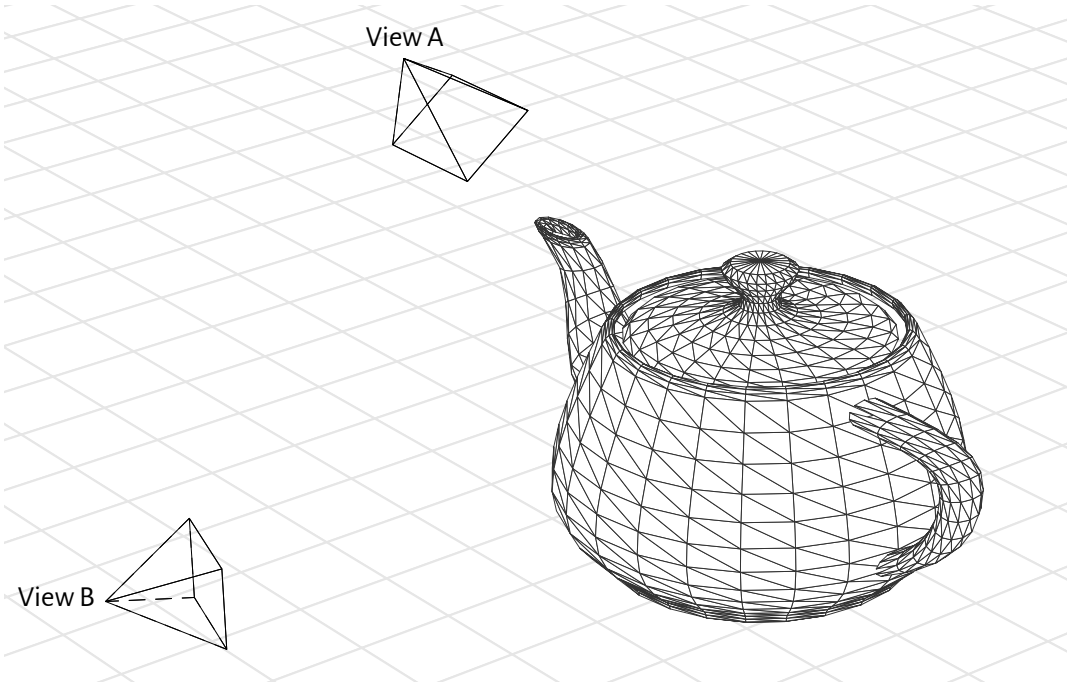


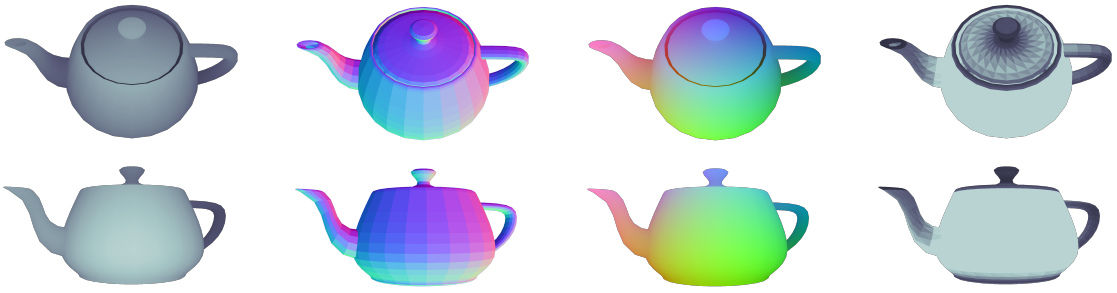
Figure 3.2: A 3D mesh and two views from which we render mesh features.

To illustrate mesh features, we use the two-view setup shown in Figure 3.2, with a model of the Utah Teapot[3] serving as the mesh. For each viewpoint, we render two per-pixel features (inverse-depth and colour), and two per-triangle features (triangle normal w.r.t. the viewpoint, and triangle area), as shown in Figure 3.3.

In the first two iterations of our work (Chapters 4 and 5), we also experimented with two other types of mesh features: triangle edge length ratio, which captures the shape of triangles, and camera-to-surface angle, a quantity that can easily be derived from the normals. Since we concluded that they bring little benefit while requiring storage that could otherwise be used for additional viewpoints, we omitted them from further work.

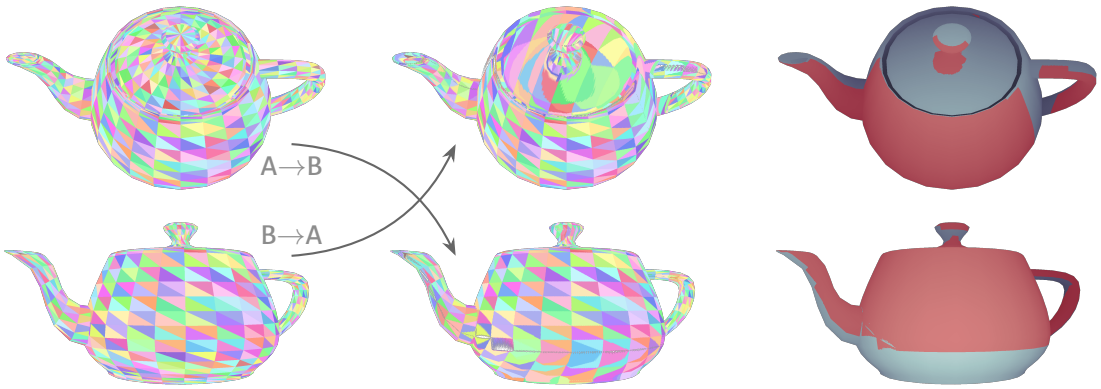
In the later parts of this thesis (Chapters 6 and 8), we operate on multiple views at the same time. One of the issues that arises in multi-view setups is that of occlusions – dif-

[3]The model used here has been downloaded from https://people.sc.fsu.edu/~jburkardt/data/ply/ply.html and is distributed under the GNU LGPL license. The full text of the license can be accessed at https://www.gnu.org/licenses/lgpl-3.0.en.html.



(a) Inverse-depth (b) Normals (c) Colour (d) Triangle area

Figure 3.3: Illustration of the mesh features rendered by our system for the viewpoints show in Figure 3.2. View A is shown in the top row, and View B in the bottom row. For visualisation purposes, we ommit the background.



(a) Triangle IDs (b) Warpped triangle IDs (c) Masked inverse-depth

Figure 3.4: Computing visibility between two views. The triangle ID images (a) are warpped from one view to the other (b) according to the inverse-depth and relative pose between views. The pixels where the warpped triangle IDs match the ones of the target view are the ones where the surface is visible in both views, highlighted here in red (c). For visualisation purposes, we depict triangle IDs with unique random colours and ommit the background.

ferent views capture different parts of the mesh surface, so there will be portions that are visible in one view but not in another. To help identify these regions, we assign each triangle in the mesh an ID, and render an image of triangle IDs from every viewpoint. Together with the inverse-depth images and the relative-pose between two views, we can use these IDs to compute visibility between views. An illustration of this is shown in Figure 3.4.

3.3.1 *Rendering Pipeline*

Generating enough training data can be a time-consuming task – for each viewpoint we need to render up to seven different mesh feature images that we then compress and serialise into a format that is easy to use during training. This process uses three different types of resources: CPU, GPU, and I/O. If we were to process every viewpoint serially from rendering to writing to disk, these resources would generally be idle. Instead, we design a parallelised pipeline (illustrated in Figure 3.5) that better utilises the available resources and allows us to generate training data more rapidly. We achieve this parallelism by using a separate process pool for each type of resource and passing tasks between resources using queues.

To generate mesh features for a given mesh, we first sample along the trajectory of

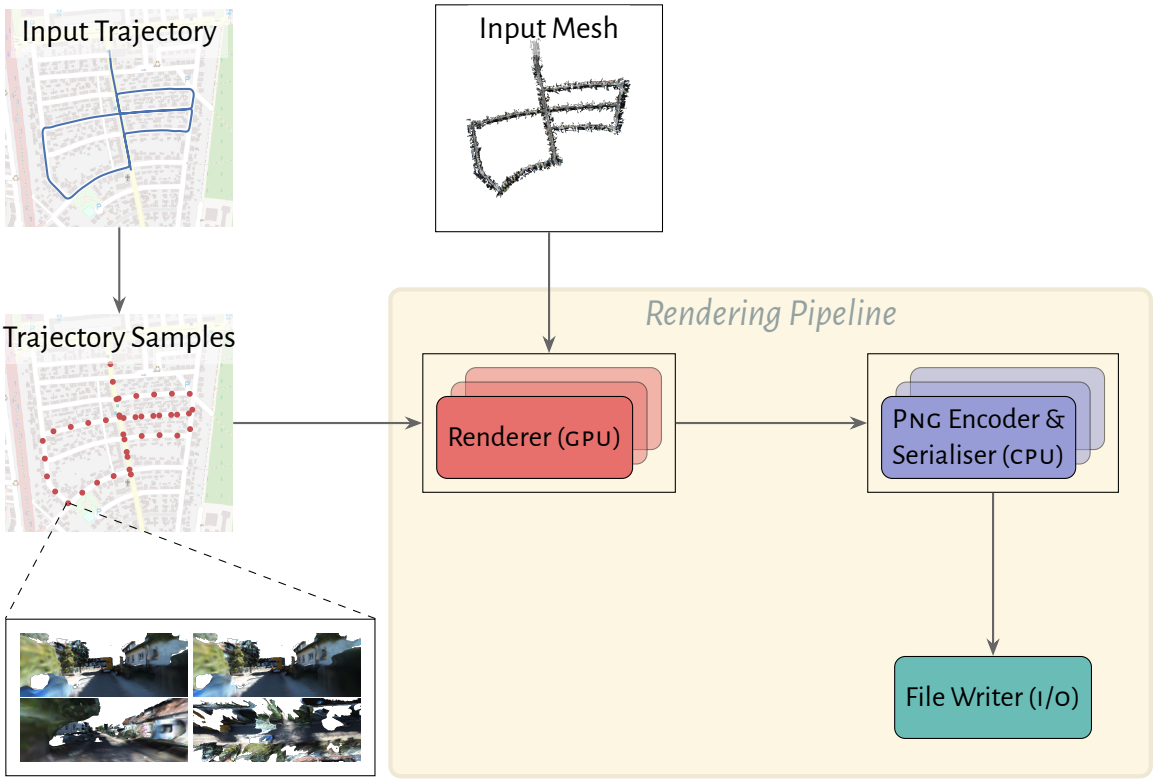


Figure 3.5: Mesh feature rendering pipeline. To generate mesh features from a 3D mesh, we sample views along the original trajectory (—). At each sample location (•), we generate four different viewpoints that we pass on to our rendering pipeline. Several GPU renderers produce mesh feature images that are then encoded to PNG format and serialised using Protocol Buffers by several CPU processes. Finally, the serialised mesh features are written to disk by another process. In addition to the parallelism that multiple processes affords us, the separation of tasks by hardware resource further boosts the speed at which we can generate mesh features.

the vehicle used to collect data. At each location, we generate four viewpoints and feed them into our rendering pipeline. For each viewpoint generated, we generate five random perturbations to improve viewpoint diversity.

The first step in the pipeline is the pool of OpenGL renderers that use the GPU to produce mesh feature images for each viewpoint. The mesh features are then encoded as PNGs and, along with the viewpoint, serialised into Protocol Buffer[4] messages by a pool of CPU processes. Finally, a separate I/O thread collects the serialised data and writes it to disk.

This designs allows for the three resources to be utilised in parallel, minimises idle time, and increases mesh feature rendering throughput, enabling us to quickly generate data, either for training or for correcting new meshes.

3.3.2 *OpenGL Implementation*

We implement mesh feature rendering in OpenGL with OpenGL Shading Language (GLSL) shaders. Our renderer consists of three standard shading stages. The vertex shader transforms geometry from the world frame to the viewpoint currently rendered. The transformed geometry is then passed to a geometry shader, which assembles mesh triangles and computes per triangle features: normals and triangle area. Finally, the fragment shader computes the pixels of the mesh feature images by "filling in" visible triangles and interpolating vertex values.

Triangle IDs

As mentioned in Section 3.3, we assign each triangle in the rendered mesh a unique ID to help us compute visibility between arbitrary views on-the-fly. To compute the ID of a triangle, we hash its *world frame* coordinates, using the algorithm shown in Listing 3.1, during the geometry stage of the GLSL pipeline.

The drawback of this approach is that hash values could collide, which would result in different triangles being assigned the same ID. Indeed, for meshes of the size we are considering, with millions of triangles, and with a hash size of 32 bits, it is almost certain that collisions would occur, even if we were to use a hash function with outputs evenly distributed over the full range of 32 bit numbers (Peyravian, Roginsky, et al., 1998). However,

[4] A description of the format and usage documentation can be accessed at https://developers.google.com/protocol-buffers.

as we are using triangle IDs to compute visibility between views, we only need to consider hash collisions for triangles within a set of views – several thousand. In this scenario, the chance of collisions is sufficiently low (0.08%) to be negligible in practice.

```
uint hash_float(float x) {
    return floatBitsToUint(x);
}

uint add_to_hash(float x, uint hash) {
    return hash * 61U + hash_float(x);
}

int hash_vertices(vec3 vertices[3]){
    uint hash = 0U;
    for (int i = 0; i < 3; ++i) {
        hash = add_to_hash(vertices[i].x, hash);
        hash = add_to_hash(vertices[i].y, hash);
        hash = add_to_hash(vertices[i].z, hash);
    }

    return int(hash);
}
```

Listing 3.1: Shader code used to hash vertex coordinates of a triangle.

Perhaps a cleaner approach would have been to number each of the triangles of a mesh in advance, guaranteeing uniqueness, and rendering this index as triangle ID. However, we found our hashing approach simpler to implement, since OpenGL does not easily allow for each triangle to be assigned a unique ID ahead of rendering – only per-vertex attributes are supported.

Storing mesh features

Since we can render as many views as we want from any given mesh, we are only bound by how many we can store on disk. Except for colour, which can readily be encoded as an RGB image with 24 bits per pixel and stored in compressed PNG format, the other mesh features are floating-point images more difficult to compress with existing imaging tools.

To work around this, we use the OpenGL normalised integer format to render our features. In this format, floating-point values in the range $[0, 1]$ are mapped to the maximum

Table 3.1: Range mapping and storage size for mesh features

Mesh Feature	Floating-Point Range	Integer Size (bits)	# Channels	Pixel Size (bits)	Compressed Pixel Size (bits)
Colour	$[0, 1]$	8	3	24	10.84 ± 1.62
Inverse-dpeth	$[0, 256]$	16	1	16	5.11 ± 1.36
Normals	$[0, \pi]$ (ϑ) $[-\pi, \pi]$ (φ)	16	2	32	19.38 ± 4.66
Area	$[0, 0.04]$	8	1	8	3.91 ± 0.70
Triangle ID	N/A	32	1	128†	33.21 ± 5.47†

range for an integer of a given size (for example, $[0, 2^{16} - 1]$ for 16-bit integers). With additional scaling and translation, any range can be stored (with varying degrees of precision). We summarise the ranges and integer sizes used for our data in Table 3.1.

Triangle normals are usually thought of as 3D unit-length vectors pointing away from the plane that a triangle lies in. The unit-length constraint means that normals are actually on a 2D manifold, which is the surface of the unit sphere. This observation allows us to store normals using two channels – ϑ (inclination), and φ (azimuth) in spherical coordinates, instead of three – x, y, and z in Cartesian coordinates.

Rendering mesh features as integer images allows us to easily compress and store them in PNG format, as long as we correctly rescale their values when decoding. Since PNG does not support 32-bit integers, we store triangle IDs in a 4-channel (RGBA) 8-bit image.

Once mesh features are encoded as PNG images, we serialise them to a `tf.Example` message, a type commonly used to store data for TensorFlow. Each message contains the keys shown in Listing 3.2.

Finally, we store pairs of `tf.Example` messages – one for the stereo-camera mesh and one for the reference lidar mesh – as records in *TFRecord*[5] files. *TFRecord* files can only be read sequentially, so to allow access to validation and test data without having to seek to the appropriate record, we store these separately. We also keep file sizes relatively small by capping the number of records in a single file to 256.

†To allow for better sampling during warping, we render triangle ID images at twice the size of mesh features, which means four times the number of pixels. This four-fold increase in the number of pixels is included in the pixel size for triangle ID images.

[5]Documentation of the *TFRecord* format and tools is available at https://www.tensorflow.org/tutorials/load_data/tfrecord.

timestamp
: The timestamp along the original trajectory corresponding to the location at which the mesh features were rendered.

sample<n_s>/feature/<feature name>
: A list of four byte-strings, containing the PNG image data of the mesh feature for each of the four views.

sample<n_s>/camera/intrinsics
: A list of four intrinsic matrices K_v used to render each views. Usually all intrinsic matrices are identical.

sample<n_s>/camera/T_world_view
: A list of four global poses for each of the views.

Listing 3.2: Keys stored in the `tf.Example` message when serialising mesh data. For $n_s = 0$, each view is rendered in at a fixed pose relative to the location in the trajectory. For $n_s \in \{1, 2, \ldots, 5\}$, the poses of the views are randomly perturbed to obtain more variety in the data.

3.4 TensorFlow Implementation

As mentioned above, to build and train our CNN model, we use the TensorFlow framework. Our implementation consists of two main components, the input pipeline responsible for loading and pre-processing mesh features from disk, and the CNN model that we train with the end-goal of correcting meshes.

3.4.1 Input Pipeline

The first step when training machine learning models is loading data from disk. This process must be be fast enough to provide a constant supply of data to the training loop. Since each of the training examples is decoded and pre-processed in the same way, the process can be parallelised in a straightforward fashion. In our implementation, we use the `tf.data` API[6] provided by TensorFlow for building efficient, parallelised input pipelines.

Our input pipeline can be configured for 3 modes: training, evaluation, and correction. In the following paragraphs, we describe each of these modes.

[6] Documentation for the `tf.data` API can be accessed at https://www.tensorflow.org/guide/data

Training

This mode is used when our model is learned, and input data is loaded and pre-processed following these steps:

1. Read records from *TFRecord* files containing training data, in parallel;
2. Deserialise records and `tf.Example` messages;
3. For each view, select 3 random perturbations;
4. Shuffle data;
5. Decode PNG mesh features and rescale values to the appropriate range, cf. Table 3.1;
6. Compute ground-truth and visibility;
7. Create mini-batches from data.

Shuffling Models trained with some version of SGD learn on mini-batches assembled from random training examples as that is known to improve convergence. Our dataset, like many other modern datasets, is far too large to fully load in memory and to randomly sample. This is usually approximated with an online shuffling algorithm that keeps a buffer of examples and returns one at random when needed. The size of the shuffle buffer affects the quality of the shuffle, so we shuffle *before* decoding the mesh features, to conserve memory and to allow for a larger buffer. Finally, when loading records from several files in parallel, the order in which the records arrive from each file is non-deterministic, so this can be seen as another form of shuffling.

Random sampling As shown in Listing 3.2, each record contains 6 samples for each viewpoint, with pose perturbations. During training, we randomly pick 3 perturbations for each view, resulting in 3 different training examples. This method provides some diversity both in each of the viewpoints, as well as in the relative poses between the viewpoints, which play a role in Chapters 6 and 8, since there are $6^4 = 1296$ different ways to pick the 4 input views.

Computing ground-truth The model we train predicts *error* in inverse-depth, which we compute as the difference between the high-quality lidar inverse-depth and the low-quality one. In this step, we also compute the visibility between views (Figure 3.4) used for the geometric consistency loss presented in Chapter 6.

Evaluation

The evaluation mode is used for computing inverse-depth refinement performance, either on the validation data or on the test data. It differs from the training mode in several ways. Firstly, since the metrics are aggregated over all the data, the order in which examples are processed is inconsequential. We can therefore remove the shuffling step. Secondly, to evaluate performance on all available views, we use all the samples stored in each record, instead of a subset.

Correction

The correction mode is used when our method is deployed, with the predictions made in this mode fused into corrected 3D reconstructions. This mode further differs from the evaluation mode in the following ways. Firstly, we only use one set of views, without random perturbations, from each record, since the perturbed views were only needed for data augmentation. Secondly, computing ground-truth or visibility between views is not necessary. Finally, we read records sequentially, following the trajectory of the data collection vehicle.

3.4.2 CNN Model

At the centre of our mesh correction system lies a CNN that uses mesh features to refine inverse-depth images. It is implemented using TensorFlow's Keras API[7] and split into an encoder and several decoders used in different parts of our work, such as a simple upscaling decoder used in Chapter 4, a decoder with skip connections used from Chapter 5 onwards, and a decoder that predicts both correction and a soft mask, used in Chapter 6.

[7]Documentation of the Keras API can be found at https://www.tensorflow.org/api_docs/python/tf/keras.

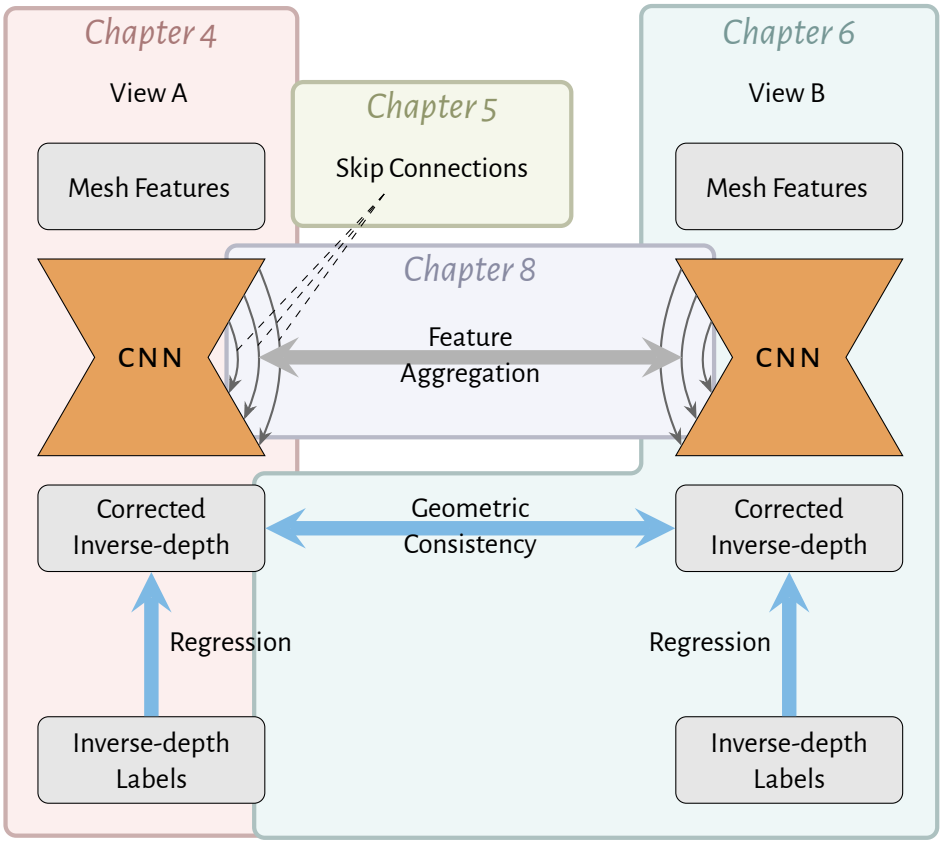


Figure 3.6: Evolution of our CNN architecture. We start in Chapter 4 by adapting a depth-prediction architecture to correct inverse-depth by using mesh features. In Chapter 5 we improve the sharpness of our predictions and show how mesh correction fits into a larger 3D reconstruction pipeline. In Chapter 6 we show how our model can be trained using multiple views. Finally, in Chapter 8 we present a novel method for aggregating multi-view information within the network.

As we develop our system through this thesis, the architecture and training of our network changes, though we work on the same codebase. Figure 3.6 shows an overview of our model and highlights where in this thesis different elements are first introduced.

Except for Chapter 8, all other architectures are designed to operate on one view at a time. To meet this design, the 4 views present in an example are stacked along the batch dimension and presented to the network as 4 separate examples. The predictions of the network are then unstacked and the 4-views-per-example structure is restored, for use by the geometric consistency loss of Chapter 6.

4

Meshed Up: Learnt Error Correction in 3D Reconstructions

Despite many impressive advances, building large-scale outdoor reconstructions with cheap sensors, such as cameras, remains very challenging. Alternatively, there are situations where the survey vehicle needs to travel at high speeds (for example while surveying a motorway) and therefore cannot scan an environment densely.

To address this, we take a post-processing approach to mesh correction. Due to its post-hoc nature, this approach is agnostic to the 3D reconstruction pipeline – we use the BOR²G pipeline of Tanner, Piniés, et al. (2015), and has the key advantage of only addressing errors that remain *after* having used all available data.

One disadvantage of such an approach is that it requires a mesh, so it cannot be easily transformed to operate on live depth data unless the underlying 3D reconstruction system also presents this capability. However the architecture and training methods we propose, due to their 2D nature, can be adapted to refine *depth-maps* online and several approaches along those lines have recently been proposed (Jeon and Lee, 2018; Sterzentsenko, Saroglou, et al., 2019).

In the previous chapter we gave an overview of the system we built to enable our mesh correction research. In this chapter, we present our initial results with the CNN correction module from Section 3.4. Before concluding the chapter, we include several experiments supplemental to the article presented here, which further explain our design and prompt some of our later research.

The following manuscript is a reproduction of the paper presented at the IEEE International Conference on Robotics and Automation (ICRA), 2018.

Meshed Up: Learnt Error Correction in 3D Reconstructions

Michael Tanner[†⋆] Ștefan Săftescu[†⋆] Alex Bewley[†] Paul Newman[†]

***Abstract*— Dense reconstructions often contain errors that prior work has so far minimised using high quality sensors and regularising the output. Nevertheless, errors still persist. This paper proposes a machine learning technique to identify errors in three dimensional (3D) meshes. Beyond simply identifying errors, our method quantifies both the magnitude and the direction of depth estimate errors when viewing the scene. This enables us to improve the reconstruction accuracy.**

We train a suitably deep network architecture with two 3D meshes: a high-quality laser reconstruction, and a lower quality stereo image reconstruction. The network predicts the amount of error in the lower quality reconstruction with respect to the high-quality one, having only viewed the former through its input. We evaluate our approach by correcting two dimensional (2D) inverse-depth images extracted from the 3D model, and show that our method improves the quality of these depth reconstructions by up to a relative 10% RMSE.

I. Introduction

This paper is about detecting and rectifying mess ups in dense reconstructions. Dense reconstruction as a mapping from input images to 3D meshes is a well studied area. However much of that work is feed-forward in the sense that it strives to produce the best mesh in an open loop. In this work we consider how, given an arbitrary generated mesh and the images that were used to create it, one might correct reconstruction errors post-hoc.

Prior efforts towards improving the accuracy of generated 3D meshes often focused on regularising the camera depth maps [1], using expensive high-quality sensors [2], or regularising the dense-reconstruction output [3]. While regularisation based approaches impose structure to smooth local surfaces, gross reconstruction errors still remain. In contrast the focus of this work is to identify error prone areas in 3D reconstructions. The goal is to facilitate the removal or repair of these mesh regions in order to improve overall mapping accuracy. As an example of what we will describe Figure 1 illustrates how the most inaccurate areas in a reconstruction are identified and corrected in a 2D representation of the scene.

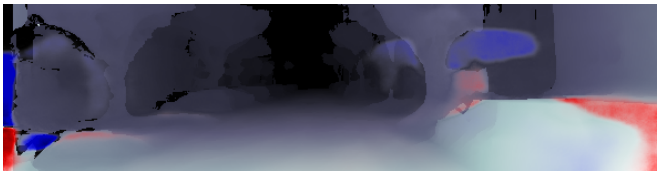
To realise this capability, a mapping is learnt from rasterised geometric and appearance features to the error in the camera-based reconstruction. Accurate laser reconstructions serve as our ground-truth to train a CNN that estimates the quality of depth-map reconstructions. A simple example of this process is shown in Figure 2, which displays the same scene reconstructed by laser and camera input data. Compared to the laser reconstruction's colour and mesh images, the regions in the camera reconstruction that are likely incorrect can readily be identified – *e.g.* holes in the road and the "smearing" area between the cars. Ideally, we would have a similar but automated way of perceiving the mesh outputs to identify erroneous regions. In this paper, we present our method for training a neural network to recognise and highlight these regions.



(a) Camera-only Reconstruction



(b) Inverse-depth image



(c) Predicted amount of error



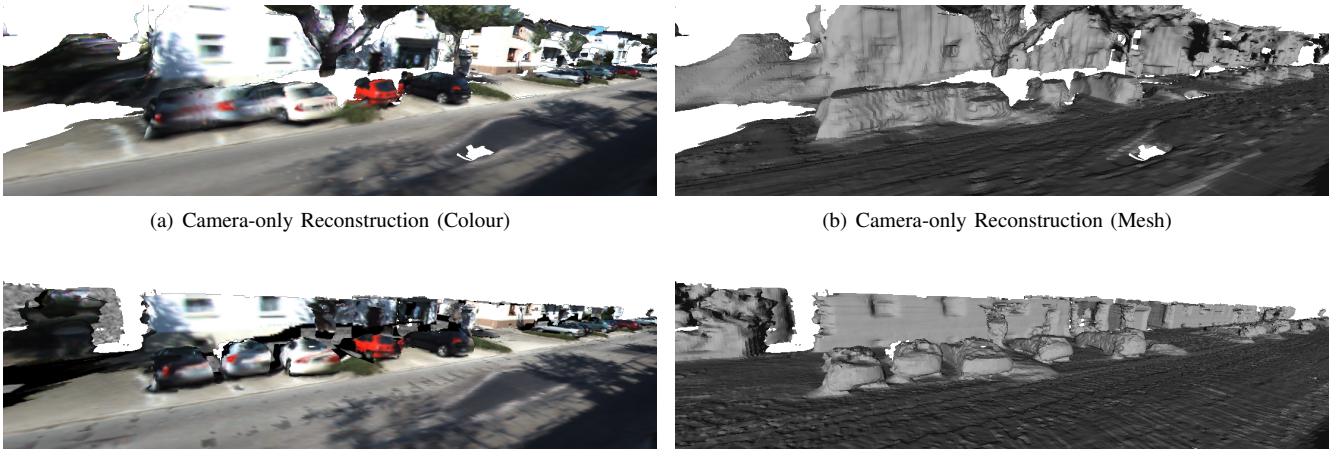
(d) Corrected inverse-depth image

Fig. 1. An illustration of using the CNN presented in this paper to correct an inverse-depth image generated from a dense 3D reconstruction. Structural errors in the reconstruction that are not obvious in (a) due to the lack of colours, are evident in (b), especially around the car. An overlay of the error predicted by the CNN's over (b) is displayed in (c). Note that this predicted error is signed, with positive error shown in red and negative error shown in blue. (d) shows how simply subtracting this error from the inverse-depth image results in a better representation of the scene (*e.g.* the car edges are more defined).

A high-level overview of the proposed system is presented in Figure 3 and the accompanying video, that show the framework for rendering mesh features and using them in training a CNN. Two dense reconstructions are used: one constructed using camera-only depth-maps and another

[†]Oxford Robotics Institute, University of Oxford, United Kingdom; mtanner,stefan,bewley,pnewman@robots.ox.ac.uk

[⋆]Authors contributed equally to this work.



(a) Camera-only Reconstruction (Colour)

(b) Camera-only Reconstruction (Mesh)

(c) Laser Reconstruction (Colour)

(d) Laser Reconstruction (Mesh)

Fig. 2. Camera depth-map and laser mesh reconstructions generated using BOR^{2}G [4], used for network inputs and ground truth respectively. In the top examples ((a) and (b)) one can easily recognise the "smearing" between the cars and the holes in the road are incorrect. The goal of this work is to use a neural network to automatically recognise these regions in depth-map reconstructions by leveraging training data derived from laser reconstructions (*e.g.*, (c) and (d)).

constructed using more accurate laser data. Both geometric and appearance features are extracted from the camera reconstruction and rasterised to a 2D image suitable for the input to a CNN (Section II). Inverse-depth is extracted from the laser reconstruction and used to compute the ground-truth label.

In Section III, we describe our network architecture and loss function based on related work in the area of monocular depth estimation [5] and re-purpose it towards the task of correcting for mesh errors. In particular, we investigate the potential and value of providing a diverse set of features (*e.g.* colour, inverse-depth, triangle edge ratios and surface normals) to the network which are not commonly available without a mesh representation. Finally, in Section IV we demonstrate the efficacy of the proposed system in identifying and correcting errors on three meshes created using sequences from the KITTI visual odometry (KITTI-VO) [6] data set. Within this framework, the following contributions are made:

1) The novel use of a CNN applied to rasterised mesh features is shown to identify error-prone regions in a low quality mesh 3D reconstruction;
2) Using the network predictions to correct for error in the inverse-depth images leads to significant improvements over what is extracted from the original mesh;
3) We analyse which of the rasterised mesh features are most informative to the CNN through an ablation study on the trained network.

II. Mesh Features

Many works in the deep learning literature rely upon simple RGB images as the input [5], [8], [9]. However, as our problem formulation assumes that a dense 3D model is available, we can extract a much richer set of features from the reconstruction. Throughout this work we use the dense mesh reconstruction proposed by Tanner *et al.* [4] referred to as BOR^{2}G, although our framework is agnostic to the 3D mesh pipeline.

A. Feature Creation

To extract dense 2D features suitable for a CNN, we begin by using OpenGL Shading Language (GLSL) to "fly" a virtual camera through the 3D models. Our `vertex` shader transforms the 3D model's vertices into the camera reference frame 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 4. This GLSL process enables us to extract six feature images: two per-pixel features (RGB and depth) and four per-triangle features (area, surface normal, edge length ratio, and surface-to-camera angle). Intuitively, the mesh-triangle-derived features contain a lot of information; they provide geometric cues that have been accumulated from multiple depth-maps and integrated into a 3D reconstruction by BOR^{2}G. We look at whether this intuition is verified in Section IV.

B. Ground-Truth Data

For a ground-truth reference, we construct a mesh using the same reconstruction pipeline as above. The same virtual camera trajectory is used to create a sequence of feature images from both the depth-map and laser reconstructions. Since 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. Rather than directly subtracting depths, we instead elect to use inverse-depth. This way, the network is encouraged to emphasise errors in foreground objects – which are more likely to have complex geometry compared to background surfaces. Specifically, we compute the per-pixel ground-truth

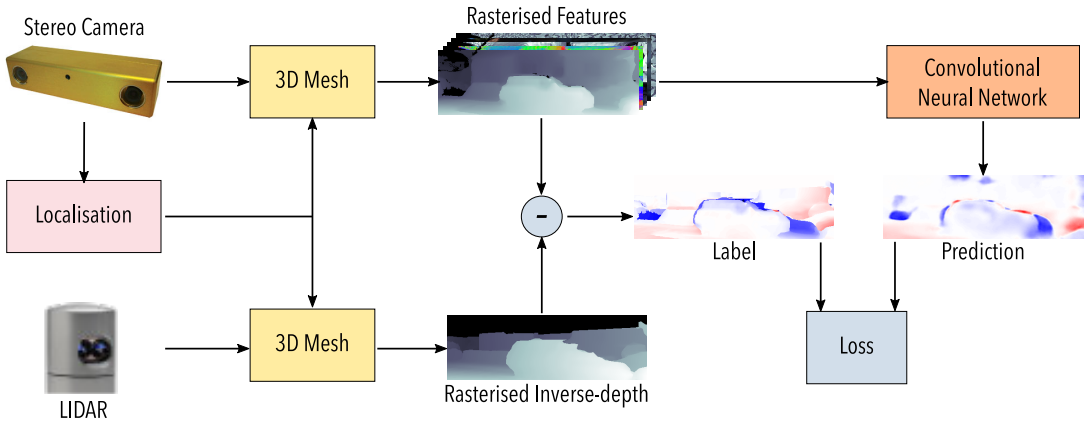


Fig. 3. Machine Learning Data-Flow Pipeline. To train our network, we create separate depth-map and laser dense reconstructions (using the techniques discussed in [7]). We create ground-truth data by subtracting the rasterised inverse-depth images from each reconstruction. The neural network trains on rasterised feature images (see Figure 4) to learn to generalise the ground-truth error in new scenes.

(Δ_{gt}) as:

$$\Delta_{gt} = \frac{A}{d_c} - \frac{A}{d_l}, \tag{1}$$

where d_c and d_l are pixels in the depth-map features for the camera and laser reconstruction, respectively, with A being the scaling 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. Because our approach is agnostic of the input sensor, A can be arbitrary, and we set $A = 1$ when evaluating performance (Section IV).

III. Reconstruction Error Prediction

A. Network Architecture

For our network architecture, we based our network on Lania *et al.*'s Fully Convolutional Residual Network [5] with concatenated ReLU activation functions on the intermediate layers [10]. This network was designed to infer per-pixel depth from a single RGB image, a task highly correlated with our aim to compute estimated depth *error* given a set of feature inputs. In this work, the input layer is generalised to accept F input channels dependant on the number of active features used from the previous section. Table I provides high-level details of the CNN where a series of residual blocks based on the *ResNet-50* architecture [11] is followed by up-projection blocks, proposed in [5]. Note the fractional strides denote spatial upsampling. The final layer is a simple 3×3 convolutional layer which outputs real-valued estimates corresponding to the inverse-depth error at each pixel location.

B. Generalisation Capacity

To be able to deploy a learnt CNN to new meshes created without a high-fidelity laser we need our network generalise to unseen data. To this end, three techniques are employed: cropping, downsampling, and weight regularisation. Firstly, we randomly perturb and crop all input feature images before providing them to the network. After each epoch of training (*i.e.* the network has viewed all the training images), the next epoch will receive a slightly different cropped region of each input image. This prevents the network from associating a specific pixel location in the training data with its corresponding output.

TABLE I

Overview of the CNN architecture for error prediction

Block Type	Filter Size/Stride	Output Size
Input	-	$64 \times 96 \times F$
Convolution	$7 \times 7/2$	$32 \times 48 \times 64$
Max Pool	$3 \times 3/2$	$16 \times 24 \times 64$
Residual, Residual, Projection	$3 \times 3/2$	$16 \times 24 \times 256$
Residual, Residual, Projection	$3 \times 3/2$	$8 \times 12 \times 512$
Residual, Residual, Projection	$3 \times 3/2$	$4 \times 6 \times 1024$
Residual, Residual	$3 \times 3/1$	$2 \times 3 \times 2048$
Up-projection	$3 \times 3/\frac{1}{2}$	$4 \times 6 \times 1024$
Up-projection	$3 \times 3/\frac{1}{2}$	$8 \times 12 \times 512$
Up-projection	$3 \times 3/\frac{1}{2}$	$16 \times 24 \times 256$
Up-projection	$3 \times 3/\frac{1}{2}$	$32 \times 48 \times 128$
Up-projection	$3 \times 3/\frac{1}{2}$	$64 \times 96 \times 32$
Convolution	$3 \times 3/1$	$64 \times 96 \times 1$

Secondly, the feature maps are gradually downsampled via projection blocks (from *ResNet-50*) creating a bottleneck, thus reducing representational capacity. This also provides greater context to the convolutional filters in the later layers of the network by increasing the size of their receptive field.

Thirdly, we implement a L_2 weight regulariser to further constrain the representational capacity by preventing the network from over-relying on the cost function at the expense of generalisation performance.

C. Loss Function

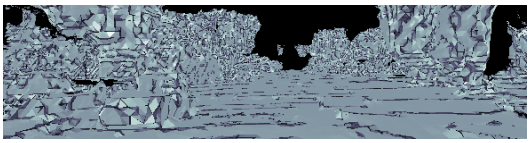
Several loss functions are widely used in machine learning applications. The L_2 norm is traditionally popular because it heavily penalises large errors and is smooth. However, unlike the L_1 norm, it has a near zero gradient for small errors, thus is often unable to drive the error completely to zero. Over the years, researchers have proposed alternative norms which combine the "best" characteristics (based on application) of each of these norms. The Huber norm uses L_2 near the origin and L_1 elsewhere, while BerHu uses the L_1 norm near the origin and the L_2 elsewhere [12]. We choose BerHu as our



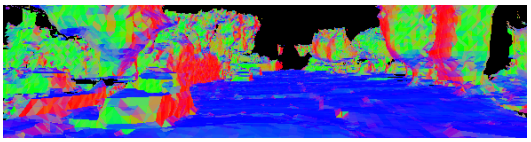
(a) Colour Reconstruction



(b) Inverse-depth



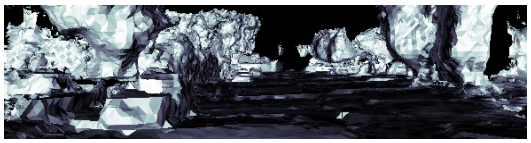
(c) Triangle Area



(d) Triangle Surface Normal



(e) Triangle Edge Length Ratios



(f) Surface to Camera Angle

Fig. 4. Example Input Features. As our dense reconstructions provide a 3D model of our operating environment, we can extract a variety of features. Pictured above are example features our GLSL pipeline currently extracts. The network has the potential to use these low level features in its intermediate representation.

cost function on the networks error prediction 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. This is in contrast to regularising depth maps where the Huber norm is more favourable to capture high gradient edges around objects [1]. These benefits were also observed in the related task of depth prediction [5].

IV. Experiments

A. Experimental Setup

Dataset: Three sequences from the KITTI-VO dataset were used as the input to the reconstruction pipeline. The input of the network consists of features from the camera based mesh, while the ground truth used for both training and evaluation comes from the laser based mesh, as described in Section II-B. For all experiments, the KITTI-VO sequences 00, 05, and 06 were used. Three models are trained, one on each combination of two sequences, with the third sequence used for testing. The results presented are an average over the three models. All three sequences are predominantly in urban environments with small amounts of visible vegetation. Using GLSL, we created a virtual camera to project the dense reconstruction into input feature images. The trajectory of the virtual camera follows the original trajectory in the KITTI-VO sequences. This enables the use of real colour images as input alongside the generated features. Both the camera and laser reconstruction meshes were generated with a fixed voxel width of 0.2 m.

Network Training and Inference: The network module was implemented in Python using the TensorFlow framework. The network weights were optimised using the ADAM solver [13] with a learning rate of 10^{-4} for 250 epochs over two of the sequences. At this point, the value of the loss function appears converged with variance coming form the stochasticity of the mini-batch sampling. To reduce this variance and confirm convergence, the optimisation is run a further 50 epochs with the learning rate reduced to 10^{-5}. Other training hyper-parameters include a batch size of 16 and a L_2 weight decay factor of 10^{-6} per training step. On average, generating mesh features takes 52 ms, and inference 72 ms per half-size KITTI image.

Performance Metrics: This work uses two different metrics to measure the resulting performance. The first metric is the commonly used root mean square error (RMSE) which provides a quantitative measure of per pixel error, computed as follows:

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{X}(d^* - d_l)^2},$$

where d_l is the ground truth depth, d^* is the predicted depth, X is the set of valid pixels, n is the cardinality of X.

Our second metric measures the accuracy of our network's ability to estimate errors under a given threshold, serving as an indication of how often our estimate is correct. The thresholded accuracy measure from [14] is essentially the expectation that a given pixel in X is less than a threshold thr^k:

$$\delta_k = \mathbb{E}_X\left[\mathbb{I}\left(\max\left(\frac{d_l}{d^*}, \frac{d^*}{d_l}\right) < thr^k\right)\right],$$

where $\mathbb{I}(\cdot)$ represents the indicator function. As in [14], $thr = 1.25$, and $k \in \{1,2,3\}$.

B. Correcting Depth Maps

The central question of this paper is whether we can correct 3D mesh reconstructions using a CNN. As a proxy, the mesh surface projected as an inverse-depth image and used to represent the local 3D scene. For our baseline reference, we compare the inverse-depth image from the camera-only reconstruction (created using BOR^{2}G) with the equivalent inverse-depth from the laser-only reconstruction, thus $d^* = d_c$. To obtain the refined depth-map, we use the

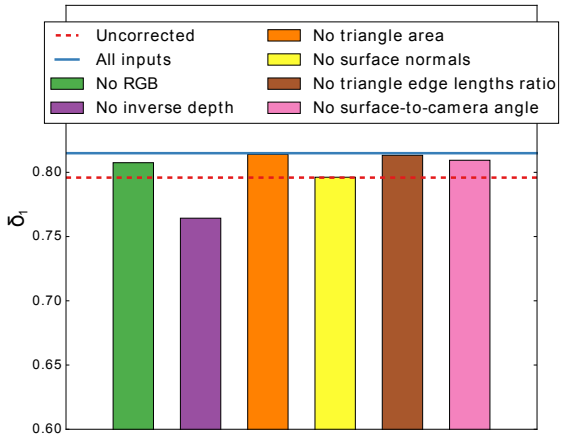


Fig. 5. Analysis of our framework's sensitivity to disabling different input features on the depth correction accuracy δ_1.

network output Δ^* to correct the depth-map reconstruction, giving us $d^* = d_c - 1/\Delta^*$. An example of this process is shown in Figure 6.

Quantitatively, we find our CNN based correction provides a 10% relative RMSE improvement over the baseline generated by BOR^{2}G (presented in Table II). Furthermore, it is consistently more accurate over different thresholds, indicating better depth values both across more pixels. A qualitative visualisation of the network output is provided in Figure 7, along side the ground truth difference in camera to laser inverse-depth maps. This visualisation shows that the network is capable of identifying small errors over broad regions (*e.g.*the red offset in the road level potentially caused by imperfect calibration) as well as large but narrow errors commonly found at object boundaries.

C. Feature Ablation Study

In the previous experiment, all features were provided to the CNN under the assumption that some features may contain more information than others and that the network can choose to ignore certain features if they contain no information. This will be reflected by measuring the sensitivity in the network output when a specific feature is withheld from the input. It is also possible that the full set of features contains redundant information which would result in the network spreading its internal importance weighting across multiple features. Such behaviour is implicitly encouraged through the L_2 weight regularisation placed on the network, meaning that small weights across multiple channels are favoured over large weights on single channels. If the network learns to use redundant features then it should be more robust to the absence of one of the features if the equivalent information could be found elsewhere.

Figure 5 shows the relative feature importance evaluated by disabling one feature at a time and monitoring the influence on the inverse-depth accuracy δ_1. Crucially, the inverse-depth from the camera reconstruction was considered to be the most important. On the other hand, mesh specific features such as triangle area and edge length ratio (Figure 4 (d)(f)) appear to be least useful. To test if redundancy is explaining this lack of sensitivity, we keep only RGB, inverse-depth and normals, to fine-tune the network (Table III). Here, the CNN failed to recover the lost performance confirming our intuition that mesh features provide additional information not easily derived from simple (stereo) camera features.

TABLE II
DEPTH ERROR CORRECTION PERFORMANCE

Model	RMSE	δ_1	δ_2	δ_3
BOR^{2}G [4]	0.051	0.799	0.844	0.870
Our CNN approach	**0.046**	**0.815**	**0.874**	**0.905**

TABLE III
FINE-TUNING WITH REDUCED FEATURE INPUTS

Model	δ_1
CNN with all features	0.815
No area, surface-to-camera angle, edge ratios	0.806
No area, surface-to-camera angle, edge ratios, fine-tuned	0.807

V. CONCLUSIONS

In this paper we present a supervised learning technique for training a neural network to detect and correct error-prone regions of a dense reconstruction. Using a GLSL pipeline, we placed a virtual camera at various positions and orientations throughout the reconstruction to generate both input feature data and, by comparing the laser and camera reconstruction features, ground-truth data to train the network. We demonstrated that the error predictions of our model can be used to make corrections to a mesh projected into an inverse-depth image and provide quantitative evaluation.

ACKNOWLEDGMENT

The authors would like to acknowledge the support of the UK's Engineering and Physical Sciences Research Council (EPSRC) through the Centre for Doctoral Training in Autonomous Intelligent Machines and Systems (AIMS) Programme Grant EP/L015897/1, Programme Grant EP/M019918/1, and the Doctoral Training Award (DTA). Additionally, the donation from Nvidia of the Titan Xp GPU used in this work is also gratefully acknowledged.

REFERENCES

[1] R. A. Newcombe, S. J. Lovegrove, and A. J. Davison, "Dtam: Dense tracking and mapping in real-time," in *Computer Vision (ICCV), 2011 IEEE International Conference on*. IEEE, 2011, pp. 2320–2327.

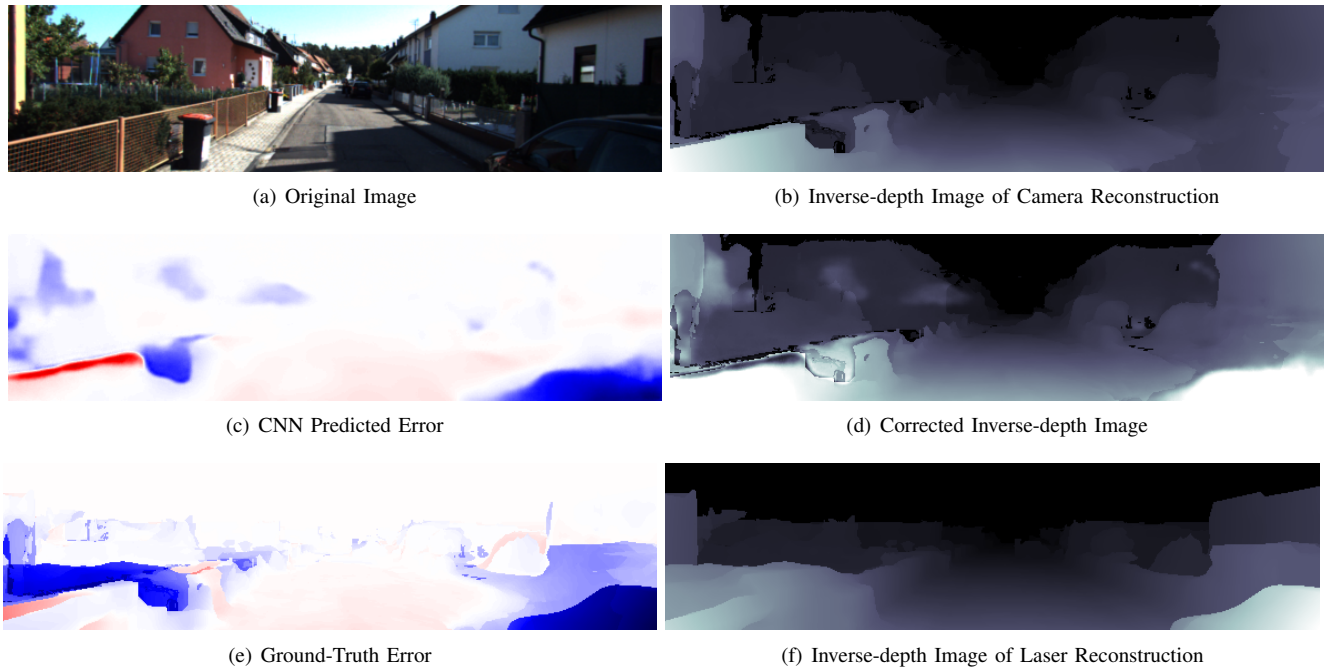
[2] Y. Bok, D.-G. Choi, and I. S. Kweon, "Sensor fusion of cameras and a laser for city-scale 3d reconstruction," *Sensors*, vol. 14, no. 11, pp. 20 882–20 909, 2014.

[3] T. Wright and B. Lennox, "Algorithmic approach to planar void detection and validation in point clouds," in *Conference Towards Autonomous Robotic Systems*. Springer, 2017, pp. 526–539.

[4] M. Tanner, P. Piniés, L. M. Paz, and P. Newman, "BOR^{2}G: Building optimal regularised reconstructions with gpus (in cubes)," in *Field and Service Robotics*. Springer, 2016, pp. 111–124.

[5] I. Laina, C. Rupprecht, V. Belagiannis, F. Tombari, and N. Navab, "Deeper depth prediction with fully convolutional residual networks," in *International Conference on 3D Vision*. IEEE, 2016, pp. 239–248.

[6] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? The KITTI vision benchmark suite," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

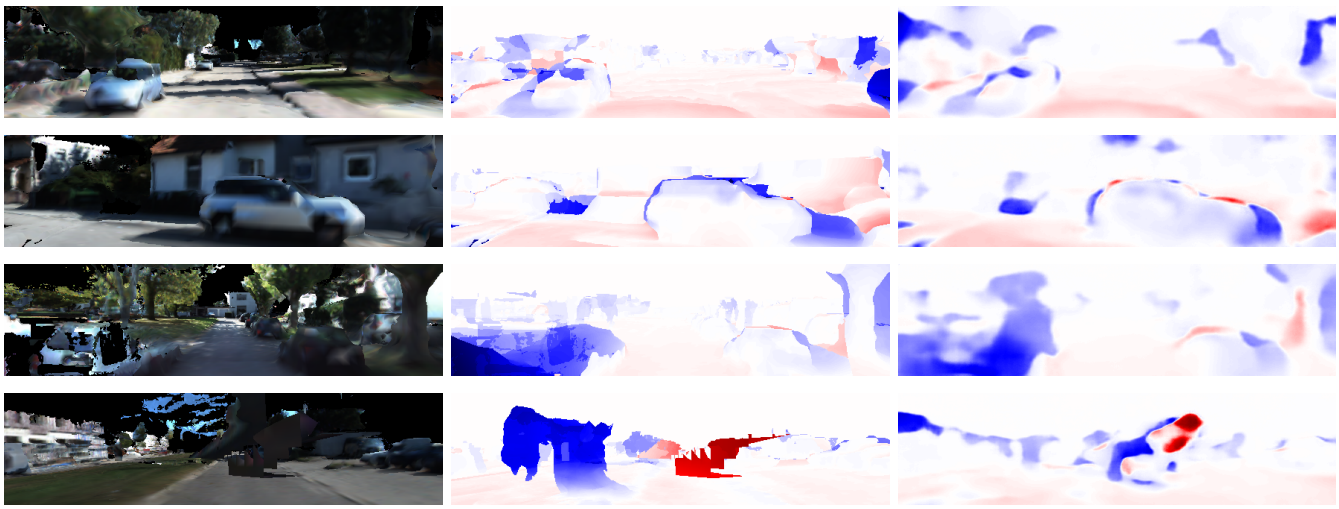


(a) Original Image (b) Inverse-depth Image of Camera Reconstruction

(c) CNN Predicted Error (d) Corrected Inverse-depth Image

(e) Ground-Truth Error (f) Inverse-depth Image of Laser Reconstruction

Fig. 6. An example demonstrating how our network corrects for reconstruction error on a test set image. (a), (b): A scene where gross errors in the fence on the left and car on the right are captured within the output predictions of the network (c). Here the colour intensity shows the magnitude of the errors with red indicating positive (camera depth is too close) and blue for negative (camera depth is too far) depth errors. Trivially applying the output of out network as a correction to the low quality inverse-depth map (b) produces a higher quality inverse-depth estimate (d), capturing structure missed in the original camera based reconstruction.



(a) Colour Reconstruction (b) Ground-Truth Error (c) CNN Predicted Error

Fig. 7. Example Ground-Truth and 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. The last row shows an instance of a failure where the grass on the left of the road appears to be consistent from the camera view point and the car ahead is dynamic and therefore significantly challenges the static world assumption in the reconstruction process.

[7] M. Tanner, P. Piniés, L. M. Paz, and P. Newman, "DENSER Cities: A System for Dense Efficient Reconstructions of Cities," *ArXiv*, 2016.

[8] D. Eigen, C. Puhrsch, and R. 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.

[9] F. Liu, C. Shen, and G. 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.

[10] W. Shang, K. Sohn, D. Almeida, and H. Lee, "Understanding and Improving Convolutional Neural Networks via Concatenated Rectified Linear Units," in *International Conference on Machine Learning*, 2016, pp. 2217–2225.

[11] K. He, X. Zhang, S. Ren, and J. Sun, "Learning for Image Recognition," in *Computer Vision and Pattern Recognition*, 2016.

[12] A. B. Owen, "A robust hybrid of lasso and ridge regression," *Contemporary Mathematics*, vol. 443, pp. 59–72, 2007.

[13] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations (ICLR)*, 2015.

[14] F. Liu, C. Shen, and G. Lin, "Deep Convolutional Neural Fields for Depth Estimation from a Single Image," in *Computer Vision and Pattern Recognition (CVPR)*, 2015.

Statement of Authorship for joint/multi-authored papers for PGR thesis

To appear at the end of each thesis chapter submitted as an article/paper

The statement shall describe the candidate's and co-authors' independent research contributions in the thesis publications. For each publication there should exist a complete statement that is to be filled out and signed by the candidate and supervisor **(only required where there isn't already a statement of contribution within the paper itself).**

Title of Paper	Meshed Up: Learnt Error Correction in 3D Reconstructions
Publication Status	☒ Published ☐ Accepted for Publication ☐ Submitted for Publication ☐ Unpublished and unsubmitted work written in a manuscript style
Publication Details	Tanner, M., **Săftescu, Ș.**, Bewley, A., & Newman, P. (2018). Meshed up: Learnt error correction in 3D reconstructions. In *Proceedings of The IEEE International Conference on Robotics and Automation (ICRA)*, Brisbane, Australia.

Student Confirmation

Student Name:	Ștefan Săftescu		
Contribution to the Paper	• Generation of ideas • Software development • Designed and performed experiments • Wrote the paper with co-authors		
Signature	Ș.Săftescu	Date	29/08/2020

Supervisor Confirmation

By signing the Statement of Authorship, you are certifying that the candidate made a substantial contribution to the publication, and that the description described above is accurate.

Supervisor name and title:	Professor Paul Newman		
Supervisor comments			
Signature	P. Newman	Date	10/09/2020

This completed form should be included in the thesis, at the end of the relevant chapter.

4.1 Further Experiments

In this section we include two additional experiments to support our method. We first look at how the size of the input affects the performance of our model. We then present a simple experiment in which we use two input views at the same time. Finally, we give a more detailed breakdown of results for the three KITTI sequences used in this chapter.

Throughout this section, we employ the same metrics for results as in our paper, but rename the root mean square error (RMSE) metric to iRMSE, to emphasise that this metric is computed over inverse-depth images. In subsequent chapters we will maintain this notation.

4.1.1 Input Size

One factor to consider when designing and training neural networks is input size. The fully-convolutional architecture we chose allows for any size input without requiring modification. However, a model can only be used to make predictions for sizes of inputs it has been trained on.

On one hand, given a fixed FCN architecture, larger input images result in more training data (since what we train on are image patches of a fixed size), which may improve performance. On the other, the context for every predicted pixel becomes smaller, which may decrease performance. Larger inputs also increase training and inference time proportionally with the increase in number of input pixels.

Table 4.1: Summary of correction results for different input scales

Model	Scale	iRMSE (m^{-1})	δ_1 (%)	δ_2 (%)	δ_3 (%)
BOR²G	25%	$5.07 \cdot 10^{-2}$	79.95	84.41	87.03
	50%	$5.13 \cdot 10^{-2}$	79.23	83.86	86.59
Ours	25%	$4.62 \cdot 10^{-2}$	81.53	87.37	90.51
	50%	$4.62 \cdot 10^{-2}$	81.46	87.33	90.44

The model presented in the our paper was trained using inputs 25% the size of real KITTI images (80×290 – the 64×94 input size shown in Table I of the paper is incorrect). When compared to using inputs 50% the size of KITTI images (150×590), the metrics shown in Table 4.1 see a slight decrease for both uncorrected and corrected inverse-depth. The

model trained on the larger inputs performs corrections slightly better, but the four-fold increase in training and inference time does not justify using larger images.

4.1.2 Multi-view Correction

Since we render views from the same mesh, the scene geometry is static and views can easily be related given relative poses and inverse-depth. Intuitively, if more views are available, predictions ought to be better, since parts of the object that may not be fully visible for the target image could be present in a nearby view. For example, a car appearing on one side of an image might not clearly be a car, whereas if looked at from a few meters behind, it would certainly look like one.

Table 4.2: Summary of correction results for single- and two-view models

Model	iRMSE (m^{-1})	δ_1 (%)	δ_2 (%)	δ_3 (%)
BOR²G	$5.07 \cdot 10^{-2}$	79.95	84.41	87.03
Ours (single view)	$4.62 \cdot 10^{-2}$	81.53	87.37	90.51
Ours (two views)	$4.61 \cdot 10^{-2}$	81.72	87.53	90.71

For this work, we conducted a simple multi-view experiment: we took pairs of consecutive input views and concatenated them as input to a two-view model. The results for this naïve approach to multi-view are shown in Table 4.2. While the improvements are consistent across different error scales, they are also not significant. One possible reason is that the relation between the two input views is not taken into account. The views can be farther or closer apart depending on the speed of the vehicle used to capture the original data, so a clear mapping between input and output locations is difficult to obtain.

The limitations of this simple initial experiment prompted us to further explore the use of multiple views, either while training (Chapter 6), or during inference (Chapter 8).

4.1.3 Per-Sequence Breakdown

Throughout this chapter, we have worked with three sequences from the KITTI Odometry Dataset: 00, 05, and 06. As described in our paper, three separate models were trained, with two of the sequences used as training data, and the other one as test to report results on. For completeness, we include in Table 4.3 the results for the three models, broken-down by test sequence.

Interestingly, the sequences with larger errors are the ones that are corrected the most. In particular, sequence 06 has large holes in the input mesh, which our model can more easily correct. This correlation is also observed and further discussed in the larger scale experiments from Chapter 7.

Table 4.3: Full correction results for models in this chapter.

Test Sequence	Model	Scale	iRMSE (m^{-1})	δ_1 (%)	δ_2 (%)	δ_3 (%)
KITTI 00	BOR²G	25%	$3.34 \cdot 10^{-2}$	86.96	91.75	94.26
		50%	$3.37 \cdot 10^{-2}$	86.59	91.52	94.11
	Ours (single view)	25%	$2.92 \cdot 10^{-2}$	86.75	92.93	95.85
		50%	$2.81 \cdot 10^{-2}$	87.47	93.36	96.14
	Ours (two views)	25%	$2.91 \cdot 10^{-2}$	86.89	92.91	95.79
		50%	$2.87 \cdot 10^{-2}$	87.23	93.15	96.02
KITTI 05	BOR²G	25%	$3.96 \cdot 10^{-2}$	84.65	89.79	92.64
		50%	$3.99 \cdot 10^{-2}$	84.03	89.37	92.36
	Ours (single view)	25%	$3.44 \cdot 10^{-2}$	85.90	91.90	94.95
		50%	$3.43 \cdot 10^{-2}$	85.92	92.12	95.26
	Ours (two views)	25%	$3.45 \cdot 10^{-2}$	85.92	91.86	94.91
		50%	$3.45 \cdot 10^{-2}$	85.92	92.01	95.13
KITTI 06	BOR²G	25%	$7.93 \cdot 10^{-2}$	68.23	71.68	74.19
		50%	$8.04 \cdot 10^{-2}$	67.07	70.70	73.30
	Ours (single view)	25%	$7.50 \cdot 10^{-2}$	71.93	77.29	80.73
		50%	$7.63 \cdot 10^{-2}$	70.99	76.51	79.91
	Ours (two views)	25%	$7.49 \cdot 10^{-2}$	72.36	77.83	81.45
		50%	$7.59 \cdot 10^{-2}$	71.45	76.91	80.35

4.2 Conclusion

In this chapter we have introduced a method for correcting meshes by refining 2D renderings of them. We adapted the depth prediction model from Laina, Rupprecht, et al. (2016), and showed that we can improve inverse-depth images by 10% in RMSE. The model we used has the limitation of producing overly-smooth outputs, due to the large downsample factor in the encoder-decoder architecture.

In the next chapter, we address this limitation in two ways. First, we will use skip connections between the encoder and the decoder – as described in Section 2.1, this is a common way to forward high-resolution information from the encoder to the decoder, while preserving the large context that downsampling provides. Second, we will propose

a loss-weighting mechanism to focus training on edges and further reduce smoothness. In addition, we will close the loop and show how our method can be integrated into a 3D reconstruction pipeline to correct meshes.

5

Large Scale Outdoor Scene Reconstruction and Correction with Vision

In the previous chapter, we introduced the idea of correcting meshes with a CNN that operates on 2D features rendered from a mesh. We used metrics over inverse-depth images as a proxy for mesh correction performance, guided by the intuition that better inverse-depth images result in better reconstructions. While the results in the previous chapter were encouraging, for our method to be of practical use we need to demonstrate that this intuition holds.

To this end, this chapter presents how our mesh correction method can be used as part of a 3D reconstruction system to build corrected *meshes*, and we evaluate these meshes against reference lidar pointclouds instead of evaluating inverse-depth network predictions. We conclude that our our mesh correction method improves surface coverage of stereo-camera meshes without increasing metric error.

To achieve this, we address the issue of overly-smooth predictions identified in the previous chapter in two ways. Firstly, we add skip connections between the encoder and decoder of our network to preserve high-frequency information such as edges (as noted in Section 2.1). Secondly, during training we introduce a loss-weighting mechanism that assigns more importance to edges in a prediction.

The following manuscript is a reproduction of the paper accepted for publication in The International Journal of Robotics Research (IJRR). The contributions of this thesis are discussed in Section 6 and Section 7.6 of the following article.

Large Scale Outdoor Scene Reconstruction and Correction with Vision

IJRR
XX(X):1–25
©The Author(s) 2018
Reprints and permission:
sagepub.co.uk/journalsPermissions.nav
DOI: 10.1177/ToBeAssigned
www.sagepub.com/
SAGE

Michael Tanner[1], Pedro Piniés[1], Lina María Paz[1], Ștefan Săftescu[1], Alex Bewley[1], Emil Jonasson[2] and Paul Newman[1]

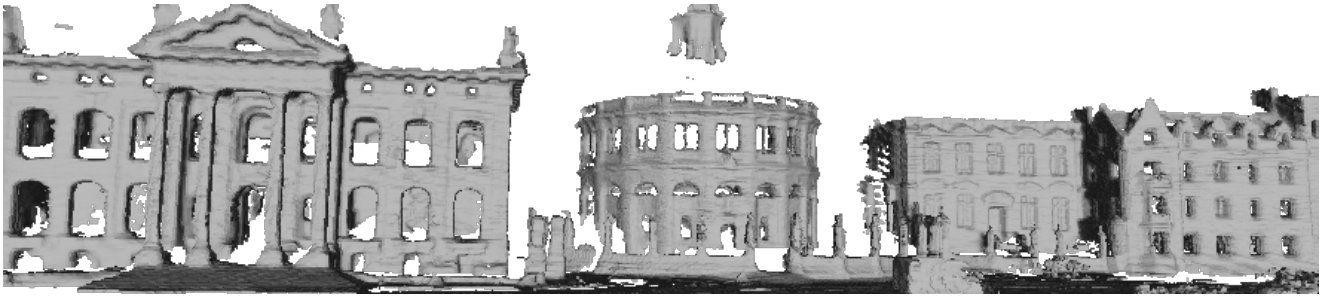


Figure 1. This paper is about the efficient generation of dense models of multiple-km-scale environments from stereo and/or laser data. Pictured is an example reconstruction from the 1.6 km Oxford Broad Street dataset released with this paper.

Abstract

We provide the theory and the system needed to create large-scale dense reconstructions for mobile-robotics applications – this stands in contrast to the object-centric reconstructions dominant in the literature. Our BOR^{2}G system fuses data from multiple sensor modalities (cameras, lidars, or both) and regularizes the resulting 3D model. We use a compressed 3D data structure which allows us to operate over a large scale. In addition, because of the paucity of surface observations by the camera and lidar sensors, we regularize over both the 2D (camera depth maps) and 3D (voxel grid) to provide a local contextual prior for the reconstruction. Our regularizer 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. Our pipeline does not end with regularization. We take the unusual step to apply a learned correction mechanism which takes the global context of the reconstruction and adjusts the constructed mesh, addressing errors that are pathological to the first pass camera-derived reconstruction.

We evaluate our system using the Stanford Burghers of Calais, Imperial College ICL-NUIM, Oxford Broad Street (released with this paper), and the KITTI datasets. These latter data sets see us operating at a combined scale and accuracy not seen in the literature. We provide statistics for the metric errors in all surfaces created compared to those measured with 3D lidar as ground truth. We demonstrate our system in practice by reconstructing the inside of the EUROfusion Joint European Torus (JET) fusion reactor, located at the Culham Centre for Fusion Energy (UK Atomic Energy Authority) in Oxfordshire.

Keywords

Dense Reconstruction, Regularization, Mapping

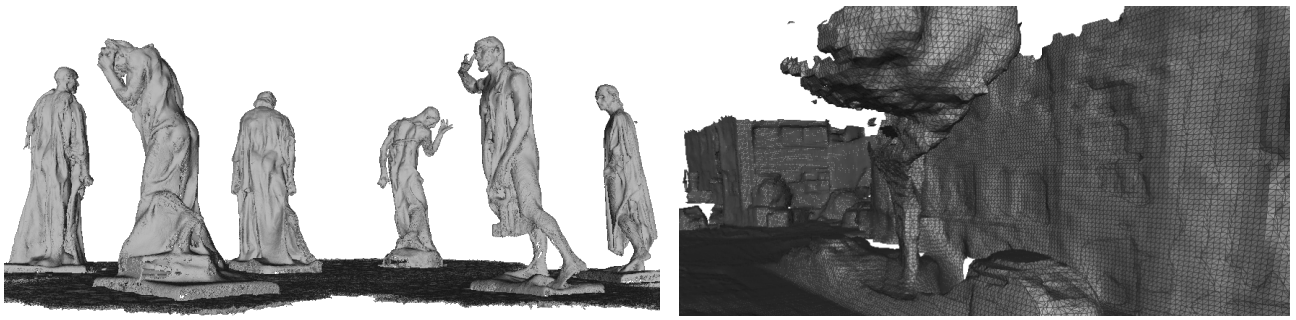
1 Introduction and Previous Work

This paper is about the efficient generation of dense, colored models of very-large-scale environments from stereo cameras, laser data, or a combination thereof. Better maps make for better understanding; better understanding leads to better robots, but this comes at a cost. The computational and memory requirements of large dense models can be prohibitive.

Over the past few years, the development of 3D reconstruction systems has undergone an explosion facilitated by the advances in GPU hardware. Earlier, large-scale efforts such as Pollefeys et al. (2008); Agarwal et al. (2009); Furukawa et al. (2010) reconstructed sections of urban scenes from unstructured photo collections. The ever-strengthening and broadening theoretical foundations of

[1]Oxford Robotics Institute, University of Oxford, UK
{mtanner,ppinies,linapaz,stefan,bewley,pnewman}@robots.ox.ac.uk
[2]RACE, Culham Science Centre, Abingdon, UK
emil.jonasson@ukaea.uk

Corresponding author:
Ștefan Săftescu, Oxford Robotics Institute, University of Oxford, 17 Parks Road, Oxford, OX1 3PJ, UK.
Email: stefan@robots.ox.ac.uk



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

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

Figure 2. RGB-D vs KITTI reconstruction detail comparison. Both reconstructions were created with the BOR^{2}G pipeline, but the Stanford reconstruction (Zhou and Koltun 2013) has over 3,100 times (Table 5) more depth observations per square meter than the KITTI reconstruction. In addition, each of the RGB-D depth maps are more than an order of magnitude more accurate than the stereo-camera-generated depth estimates. This paper is focused on creating high quality reconstructions which gracefully handle KITTI-level low observation densities over kilometers of observations, but also works on the traditional 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 Zhou and Koltun (2013), the above BOR^{2}G Burghers reconstruction has a median point-to-surface difference of 0.5 cm. Section 7 provides a more detailed quantitative analysis our reconstruction precision.

continuous optimization (Chambolle and Pock 2011; Goldluecke et al. 2012), upon which the most advanced algorithms rely, have become accessible for robotics and computer vision applications. Together these strands – hardware and theory – allow us to build systems which create large-scale 3D dense reconstructions.

However, the state of the art of many dense 3D reconstruction systems rarely considers scalability for the practical use in mapping applications such as autonomous driving or inspection. The most general approaches are motivated by the recent mobile phone and tablet development (Klingensmith et al. 2015; Engel et al. 2015; Schöps et al. 2015) with an eye on small-scale reconstruction.

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 (Anguelov et al. 2010). However, their algorithms overemphasize the laser data and assume all depth maps only contain piecewise planar objects. In 2013, Google presented an alternative Structure from Motion approach using only camera sensors (Klingner et al. 2013). 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. Bok et al. (2014) built upon the prior state-of-the-art to create large-scale 3D maps using camera and lasers. 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.

To this end, we propose a dense mapping system which meets the following requirements:

R1: *Operate in multiple-kilometers-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*

Table 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

These were designed to support our autonomous-driving applications. A survey of the literature found no systems currently exist which meet these requirements.

Four seminal systems (Table 1) in this area are Dense Tracking and Mapping (DTAM) by Newcombe et al. (2011b); Kinect Fusion by Newcombe et al. (2011a); Kintinuous by Whelan et al. (2012); and Voxel Hashing by Nießner et al. (2013).

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-regularized 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 centimeter-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 Truncated Signed Distance Function (TSDF) (Curless and Levoy 1996), depth observations are stored in a voxel grid where each voxel contains a positive number when in front of a surface and

*2D Regularization
†2D and 3D Regularization

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 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 theoretically infinitely extended the reconstruction workspace size. However, it cannot leverage sensor observations further than 3 m from the RGB-D camera because it is still fundamentally based upon a conventional fixed-size voxel grid.

Nießner's Hashing Voxel Grids (HVG) also extended the size of reconstructions but did so instead 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 meters as that is near the maximum effective range of the Kinect camera, however the range can be trivially extended. Solutions such as Whelan et al. (2014) leverage a rolling cyclical buffer as the volumetric-reconstruction data structure. This is an interesting approach that allows local volumetric regions to virtually translate as the camera moves through an environment. The correct choice of an efficient volumentric data structure has also gained attention in the deep learning context where it impacts the resolution of 3D tasks, including 3D object classification, orientation estimation, and point-cloud labeling Riegler et al. (2017)

Most of the above techniques focused on object-centered 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 2a. In autonomous driving scenarios where the sensor is mounted on a vehicle and the path is not known *a priori*, the surfaces are observed fewer times. In fact, we found that RGB-D scenarios have over 3,100 times more depth images per square meter than our autonomous vehicle applications (Table 5).

In addition, mobile-robotics sensor suites typically include lidars and forward-facing monocular or stereo cameras. Since the viewing range of RGB-D cameras is so short (5 meters) and their accuracy degrades outdoors, they are not useful when reconstructing an urban environment from a mobile-robotics platform. Cameras on mobile-robotics platforms produce significantly less accurate depth measurements. Over time, lidar sensors usually have a smaller reconstruction field of view and depth-observation density than a similarly placed camera sensor. Both camera and lidar sensors observe surfaces far fewer times than in a typical RGB-D scenario. Therefore the reconstructions inherently have less detail, as shown in Figure 2b, which requires us to apply contextual priors via a local regularizer to improve the final surface reconstructions.

Table 1, Table 5, and Figure 2 summarize the discussion in the preceding literature review.

Our contributions are as follows:

1. We present in Sections 2–6 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 utilize a regularizer in 2D and 3D to both serve as a prior and reduce the noise in the final reconstruction (R3).
2. We present in Sections 5.3–5.4 a method to regularize 3D data stored in a compressed volumetric data structure thereby enabling optimal regularization of significantly larger scenes (fulfills R1 and R3). The key difficulty (and hence our vital contribution) in regularizing within a compressed structure is the presence of many additional boundary conditions introduced between regions which have and have not been directly observed by the range sensor. Accurately computing the gradient and divergence operators, both of which are required to minimize the regularizer's energy, becomes a non-trivial problem. We ensure the regularizer only operates on meaningful data and doesn't extrapolate into unobserved regions.
3. We present in Section 6 a method to adjust reconstructions and correct gross errors with priors learnt from high-fidelity historical data (e.g. roads generally don't have holes in them, cars are of certain shapes, etc.) We show that appearance and geometry features can be extracted from a 3D reconstruction, and better depth maps can be produced with a neural network that applies these priors. We illustrate how this step can be included in our dense-reconstruction pipeline to produce more comprehensive meshes.
4. We provide, by way of our quantitative results in Section 7 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 optimized vehicle trajectory, consolidated GT pointcloud, and final dense reconstructions with different sensor modalities.
5. We release (Section 7.3) the 1.6 km Oxford Broad Street dataset (partial example reconstruction in Figure 1) 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.

This article builds upon our prior work (Tanner et al. 2015, 2018) by enabling regularization in a compressed 3D data structure; evaluating the relative reconstruction quality based on using combinations of camera and laser sensors; documenting the performance of SDF and histogram data

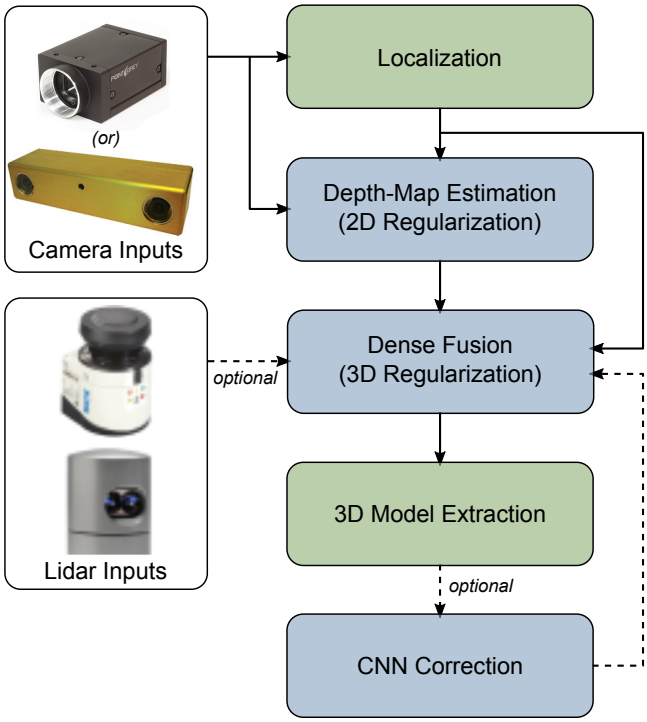


Figure 3. An overview of our software pipeline. Our Dense Fusion module accepts data from either laser sensors or depth maps created from mono or stereo cameras. We regularize the 3D model, extract the surface, and apply learnt corrections to the meshes. Then, we provide the final 3D model to other components on our autonomous robotics platform – e.g., segmentation, localization, planning, or visualization. The blue modules are discussed in further detail in Sections 3–6.

terms for the regularizer; correcting reconstructions using historical priors; and using a full SLAM approach to evaluate the improved reconstruction quality when revisiting a location.

2 System Overview

This section provides a brief overview of our system's architecture (Figure 3) before we proceed to a detailed discussion in Sections 3–6.

At its core, our system consumes a stream of stereo camera images (I) or laser scans (L) and produces a 3D model. The pipeline consists of a Localization module which provides sensor pose estimates. We place no requirements on the trajectory of the source sensors; indeed, we illustrate our method using images captured from a forward-facing camera on a road vehicle – an ill-conditioned use-case which is challenging yet likely given the utility of forward-facing cameras in navigation and perception tasks. The design of this module is versatile allowing us to pick any arbitrary SLAM derivative; we use ORB-SLAM2 (Mur-Artal and Tardós 2015) as it is a widely used, open source benchmark tool for localization and loop closure with monocular, stereo, and RGB-D cameras.

The Depth-Map Estimation module (Section 4) processes a stream of stereo or monocular frames into a stream of 2D regularized depth maps, D. Because of the nature of a passive camera and the large scale at which we operate, we utilize a sophisticated regularizer – specifically, the Total Generalized Variation (TGV) regularizer – to improve the accuracy of these depth maps.

The Dense Fusion module (Section 5) merges the regularized depth maps into a compressed data structure as a Signed Distance Function (SDF), $f : \Omega_3 \rightarrow \mathbb{R}$, where $\Omega_3 \subset \mathbb{R}^3$ represents a subset of points in 3D space. New incoming data can be added at any time from any suitable sensor source. Even with the regularized input depth maps, the resulting reconstruction tends to contain spurious and noisy surfaces. Our 3D Total Variation (TV) regularizer operates over Ω_3 to generate the optimized, denoised SDF field, $u : \Omega_3 \rightarrow \mathbb{R}$. This regularizer also serves as a prior to aid in the reconstruction even with a paucity of direct surface observations (Figure 2).

The CNN Correction step (Section 6) further improves reconstructions created from low quality or scarce data. Our approach corrects the 3D geometry based on prior knowledge of what certain scenarios are expected look like when reconstructed. We achieve this by projecting various features in training 3D reconstructions into 2D images and using a CNN to learn the differences between high and low quality data. Learnt error correction, along with regularization in both 2D and 3D are the salient features of our system.

A final surface model is extracted to be processed in parallel by a separate application (e.g., segmentation, localization, planning, or visualization).

3 Optimization with Total Variation

In this paper we address the inverse problem of estimating the 3D dense structure of an environment given a set of 2D images and/or 3D sparse laser scans. Inverse problems deal with the estimation of unknown quantities u (structure in our case) given their effects f (images or laser data). By their nature, these problems are ill-posed which means that at least one of the following three requirements is not met: existence of a solution, uniqueness of the solution, or stability of the solution (i.e. small changes in input data produce small changes in the result). Clearly the problem we try to solve is ill-posed. As an extreme example, if we want to reconstruct a white wall from a set of images, we cannot guarantee uniqueness and stability (walls at different distances and orientations will produce the same observations).

The properties of the problem can improve if we limit the space of meaningful solutions by imposing regularity assumptions or prior knowledge about the solution. Energy-based minimization approaches model these requirements by using an energy regularization term that favours certain solutions and a data term that models how well the searched solution u explains the set of observations f. Equation 1 synthesizes this approach,

$$\min_u E_{reg}(u) + E_{data}(f, u) \tag{1}$$

where $E_{reg}(u)$ is the regularisation or "smoothness" term and $E_{data}(f, u)$ is the data-fidelity term.

In this paper, we use a first order ($\alpha = 1$) and a second order ($\alpha = 2$) Total Generalised Variation (TGV$^\alpha$) regularization term, respectively, in the optimizations

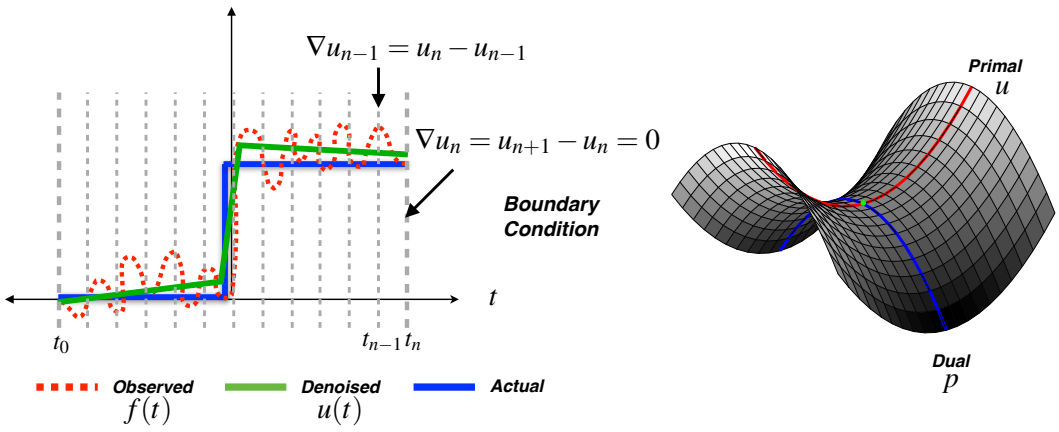


Figure 4. (Left) Example of a denoised 1-D signal with Total Variation. The aim of the continuous energy minimisation is to recover a function $u(t)$ given the observed signal $f(t)$ defined in the continuous domain $\Omega_1 = [t_0, t_n]$. TV regularisation filters out high frequency noise preserving sharp discontinuities. We impose zero Neumann boundary conditions that assume the signal remains constant outside the left and right boundaries. (Right) The original non-differentiable energy minimization is transformed into a smooth Primal-Dual problem by applying the Legendre-Fenchel transform to the TV term. The resulting saddle point problem can be solved by alternating variants of gradient descent and gradient ascent steps for the primal u and dual p variables, respectively.

performed in the Depth-Map Estimation and the 3D Dense Fusion blocks in Figure 3. As we will discuss later, these regularizers allow us to incorporate prior information about the type of surfaces we expect to encounter in the traversed environment.

Let us now present a simple denoising problem that will help us illustrate the Primal-Dual algorithm Chambolle and Pock (2011) implemented to solve Equation 1 in the following sections. For simplicity, we employ a first order TGV[1] regularizer also known as Total Variation (TV). Figure 4 left shows the effect of the TV on a noisy 1D signal $f(t)$ defined in the bounded domain $\Omega_1 = [t_0, t_n]$ where $\Omega_1 \subset \mathbb{R}$. The energy to be minimized is given by,

$$E(u) = \int_{\Omega_1} |\nabla u| d\Omega_1 + \frac{\lambda}{2} \int_{\Omega_1} ||u - f||_2^2 d\Omega_1 \tag{2}$$

where $\int_{\Omega_1} |\nabla u| d\Omega_1$ is the Total Variation of the solution and, in this case, we use a quadratic penalty for the data term. Notice that the TV accumulates the magnitude of local variations in the signal and therefore measures its total length. Intuitively the TV term tends to reduce signal's ripples and therefore filters out high frequency noise while maintaining big discontinuities present in the data term. In the extreme case when no data term is available, TV would produce a constant solution $u = const$ since then $\nabla u = 0$. On the other hand, the data term tries to preserve the shape of the input data $f(t)$. The trade off between both terms is controlled by λ which, when correctly tuned, favors piece-wise constant solutions. To solve this problem we first discretize Equation 2,

$$E(u) = \sum_{i=0}^{n-1} |\nabla u_i| + \frac{\lambda}{2} \sum_{i=0}^{n} ||u_i - f_i||_2^2 \tag{3}$$

where we calculate the gradient using forward differences and we assume zero Neumann boundary conditions – the value of the gradient at the borders is 0, i.e., we assume the signal remains constant outside the left and right boundaries,

$$\nabla u_i = \begin{cases} u_{i+1} - u_i & \text{if } 0 \leq i < n \\ 0 & \text{if } i = n \end{cases} \tag{4}$$

To simplify the explanation let us minimize Equation 3 w.r.t. the i^{th} sample in the summation,

$$\min_{u_i} |\nabla u_i| + \frac{\lambda}{2} ||u_i - f_i||_2^2 \tag{5}$$

Although this problem is convex and therefore its global minimum can be computed, standard optimization algorithms like gradient descent or Newton and quasi-Newton methods cannot be applied directly because they require smooth cost functions to assure convergence (Nocedal and Wright 2006). In this case, the gradient is not defined at the kink of the absolute value. The Legendre-Fenchel transform (Rockafellar 1970) provides an elegant way to convert the L1 norm into a maximization problem that, although more complex, is differentiable (see Figure 5 for an intuitive explanation),

$$|\nabla u_i| = \sup_{|p_i| \leq 1} < \nabla u_i, p_i > \tag{6}$$

where the operator $< \bullet, \bullet >$ computes the inner product of its operands and p_i is known as the dual variable which represents for each u_i the sub-gradient of the L1 norm. Substituting Equation 6 into Equation 5 we obtain,

$$\min_{u_i} \sup_{|p_i| \leq 1} < \nabla u_i, p_i > + \frac{\lambda}{2} ||u_i - f_i||_2^2 \tag{7}$$

which is a saddle point problem with equivalent primal and dual solutions since strong duality holds (Boyd and Vandenberghe 2004). Figure 4 right shows an example of a saddle point problem in u and p with zero duality gap.

An equivalent energy expression, that will be useful in the derivation of the optimization algorithm, can be obtained by calculating the adjoint Adj($\bullet$) of the linear operator ∇. When ∇u, and therefore $\mathbf{p}$, is n-dimensional (with $n > 1$), the adjoint of the gradient is equal to the negative of the divergence operator: $\text{Adj}(\nabla)\mathbf{p} = -\nabla \cdot \mathbf{p} = -(\nabla_1 p^1 +$

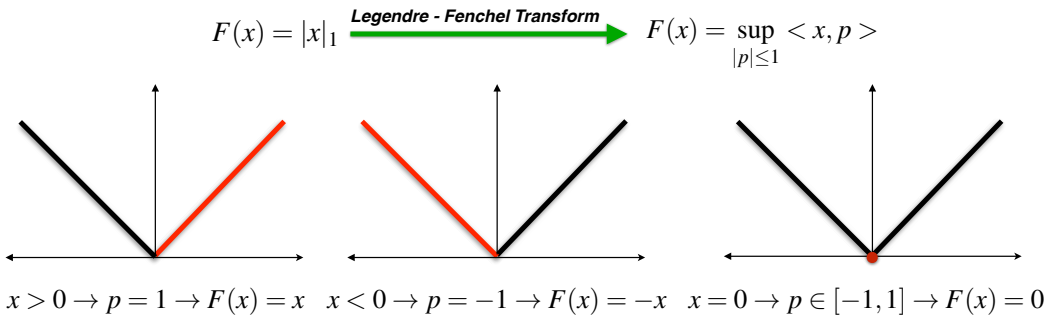


Figure 5. Legendre-Fenchel transform of the non-differentiable L_1 norm. In the 1-D case, the norm is represented by the absolute value $|x|_1$ which is a non-smooth function whose gradient is not determined at $x = 0$. A more convenient representation, that allows us to apply well known optimization techniques, consists in transforming the original function into a maximization problem with linear inequality constraints. To show that both the original and the transformed representation are equivalent, we can demonstrate that the solution of the maximization problem leads indeed to the original L_1 norm: when $x > 0$ (left), the value of p that maximises the expression subject to the constraint $|p| \leq 1$ is $p = 1$, therefore $F(x) = x = |x|$; when $x < 0$ (center), the inner product is maximized when $p = -1$ leading to $F(x) = -x = |x|$. Finally, when $x = 0$, any value of p in the feasible set $[-1, 1]$ gives the same solution $F(x) = 0$. Note that, as a result of applying the Legendre-Fenchel transform, we have transformed a simple but non-differentiable expression $|x|_1$ into a more complex but smooth problem for which we can apply well-known optimization algorithms such as projected gradient ascent.

$\ldots + \nabla_n p^n)$ where $\nabla_j p^j$ is the j^{th} directional derivative of the last component of $\mathbf{p}$, we will use as well $(\nabla \cdot \mathbf{p})^j = \nabla_j p^j$. By definition a linear operator and its adjoint fulfill,

$$< \nabla u_i, \mathbf{p}_i >= - < u_i, \nabla \cdot \mathbf{p}_i > \tag{8}$$

To satisfy this equation, forward differences and Neumann boundary conditions for the gradient become backward differences and Dirichlet boundary conditions for the divergence – the value of the variable at the borders is 0,

$$(\nabla \cdot \mathbf{p}_i)^j = \begin{cases} p_i^j - p_{i-1}^j & \text{if } 0 < i < n \\ p_i^j & \text{if } i = 0 \\ -p_{i-1}^j & \text{if } i = n \end{cases} \tag{9}$$

With a little abuse of notation, we will use the divergence operator for the current 1D example (j can only be equal to 1). Using the adjoint of the gradient we can rewrite Equation 7 as

$$\min_{u_i} \sup_{|p_i|\leq 1} - < u_i, \nabla \cdot p_i > + \frac{\lambda}{2} ||u_i - f_i||_2^2 \tag{10}$$

The Primal-Dual optimization algorithm (Chambolle and Pock 2011) to solve Equation 7 or its equivalent Equation 10 consists of the following steps:

1. p_i, u_i, and $\hat{u}_i$ are initialised to 0. $\hat{u}_i$ is a temporary variable which reduces the number of optimization iterations required to converge.
2. To solve the maximisation we use a projected gradient ascent algorithm. The gradient of Equation 7 with respect to p_i is just $\nabla \hat{u}_i$ (as we explained before we will use $\hat{u}_i$ instead of u_i to speed up the convergence of the algorithm, more details in Chambolle and Pock 2011). Therefore $p_i^{new} = p_i + \sigma \nabla \hat{u}_i$ where σ is the dual variable's gradient-ascent step size. Since $p_i^{new} \in [-1, 1]$ we project the solution into the feasible set using a projection operator $\pi(p_i^{new})$. Therefore the maximization step is given by,

$$\begin{aligned} p_i &= \pi(p_i + \sigma_p \nabla \hat{u}_i) \\ \pi(p_i) &= \frac{p_i}{\max(1, |p_i|)} \end{aligned} \tag{11}$$

3. The minimization is based on a look-ahead gradient descent algorithm. In this case it is easier to calculate the gradient with respect to u_i from Equation 10. With look-ahead we mean that the gradient is not evaluated at the current solution but at the new estimate u_i^{new} after the step is taken. Deriving Equation 10 w.r.t. u_i and evaluating at u_i^{new} the gradient is given by: $-\nabla \cdot p_i + \lambda(u_i^{\text{new}} - f_i)$. Applying a gradient descent step with step size τ we obtain,

$$u_i^{\text{new}} = u_i - \tau(-\nabla \cdot p_i + \lambda(u_i^{\text{new}} - f_i)) \tag{12}$$

which gives an implicit equation in u_i^{new}. Rearranging terms, the primal step is,

$$u_i^{\text{new}} = \frac{u_i + \tau \nabla \cdot p_i + \tau \lambda f_i}{1 + \tau \lambda} \tag{13}$$

4. To reduce the number of required iterations we apply the following "relaxation" step,

$$\hat{u}_i = u_i + \theta(u_i - \hat{u}_i) \tag{14}$$

where θ is a parameter to adjust the relaxation step size.
5. Finally, we repeat steps 2–4 until convergence.

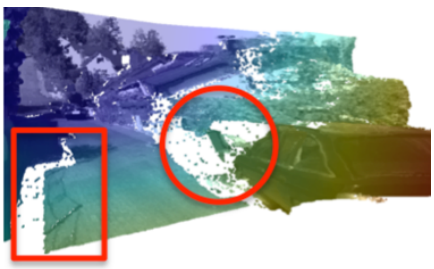
In the following two sections we will make use of the Primal-Dual algorithm to regularize both a stream of 2D depth maps (Section 4) and the final 3D dense reconstruction (Section 5).

4 Depth-Map Estimation

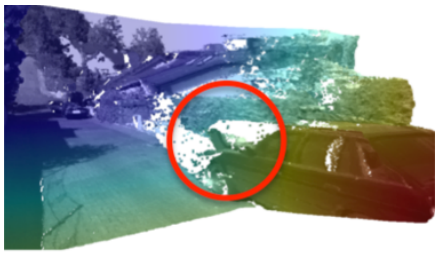
Given a pair of rectified images captured from a stereo camera, we implement an algorithm based on Ranftl



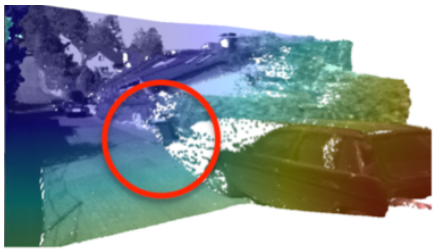
(a) Reference Image



(b) TV Regularizer



(c) TGV Regularizer



(d) TGV-Tensor Regularizer

Figure 6. Comparison of three 2D depth-map regularizers: TV, TGV, and TGV-Tensor. Using the reference image (a), the TV regularizer (b) favours fronto-parallel surfaces, therefore it creates a sharp discontinuity for the shadow on the road (red rectangle) and attaches the rubbish bin to the front of the car (red circle). TGV (c) improves upon this by allowing planes at any orientation, but it still cannot identify boundaries between objects: the rubbish bin is again estimated as part of the car. Finally, the TGV-Tensor (d) regularizer both allows planes at any orientation and is more successful at differentiating objects by taking into account the normal of the color image's gradient. For clarity, the sparse reconstructions have a different viewing origin than the reference image.

et al. (2012) that allows us to calculate very high quality disparity maps. Although many other dense stereo approaches exist in the literature (a nice survey is presented by Scharstein and Szeliski 2002), most of the solutions require explicit strategies to handle occlusions. Instead, we implement an energy minimization approach with an L1-fidelity data term that provides some robustness against outliers from occlusions while handling varying illumination and radiometric differences between the cameras. For regularization we use a second-order regularizer term that favours piecewise planar disparity maps and allows sub-pixel accurate solutions. The energy to be minimized is,

$$E(d) = E_{reg}(d) + E_{data}(d; I_L, I_R) \tag{15}$$

where d is the searched disparity image defined in $\Omega_2 \subset \mathbb{R}^2$.

4.1 Data Term

To improve robustness against outliers we implement an L1-norm penalty term that penalizes photo-consistency differences between matched pixels in the images,

$$E_{data}(d; I_L, I_R) = \int_{\Omega_2} |\nabla C_W(I_L(x+d, y), I_R(x, y))| d\Omega_2 \tag{16}$$

In particular $\nabla C_W(\bullet)$ stands for the difference between the Census Transform Signature (CTS) (Zabih and Woodfill 1994) of two pixels calculated in a window of size W for a candidate disparity d. This metric has been shown to be both illumination invariant and fast to compute in comparison to other metrics such as the Sum of Absolute Distances (SAD), the Sum of Square Distances (SSD) or the Normalized Cross Correlation (NCC) (Zabih and Woodfill 1994). Its computation is simple: given a pixel, the CTS computes a bit string by comparing the chosen pixel with the local window W centered around it. A bit is set to 1 if the corresponding pixel has a lower intensity than the pixel of interest. The difference between two windows $\nabla C_W(\bullet)$ is then given by the Hamming distance between the two bit strings defining them.

4.2 Affine Regularization

As we explained in the previous section, for ill-posed problems, like depth-map estimation, good and apt priors are essential – whether the prior is task-specific and bespoke (Güney and Geiger 2015) or more general. A common choice is to use TV regularization as a prior to favor piecewise-constant solutions. However, its use lends itself to poor depth-map estimates over outdoor sequences because it assumes fronto-parallel surfaces. Figure 6 shows some of the artifacts created after back-projecting the point cloud for planar surfaces not orthogonal to the image plane (e.g. the roads and walls which dominate our urban scenes). Thus we reach for a second order Total Generalized Variation (TGV^2) regularization term (Bredies et al. 2010) which favours affine surfaces,

$$\min_{d \in \mathbb{R}, \mathbf{w} \in \mathbb{R}^2} \alpha_1 \int_{\Omega_2} |\nabla d - \mathbf{w}| d\Omega_2 + \alpha_2 \int_{\Omega_2} |\nabla \mathbf{w}| d\Omega_2 \tag{17}$$

where $\mathbf{w}$ allows the disparity d in a region of the depth map to change at a constant rate and therefore creates planar surfaces with different orientations. α_1 and α_2 are free parameters (Table 2).

To minimize this problem using the Primal-Dual algorithm explained in the previous section, we apply the Legendre-Fenchel transform to both summands in Equation 17 so that

both terms become differentiable,

$$|\nabla d - \mathbf{w}| = \sup_{|\mathbf{p}|_\infty \leq 1} < \nabla d - \mathbf{w}, \mathbf{p} >$$
$$|\nabla \mathbf{w}| = \sup_{|\mathbf{q}|_\infty \leq 1} < \nabla \mathbf{w}, \mathbf{q} > \quad (18)$$

Notice that, this time, two additional dual variables $\mathbf{p} \in \mathbb{R}^2$ and $\mathbf{q} \in \mathbb{R}^4$ are added to the optimization.

4.3 Leveraging Appearance

A common problem that arises during the energy minimization is the resulting tension between preserving object discontinuities while respecting the smoothness prior. Ideally the solutions preserve intra-object continuity and inter-object discontinuity. For example, in Figure 6 we desire to accurately estimate the depths of both the rubbish bin and automobile (intra-object consistency), but we also must account for their position relative to one another (inter-object discontinuity).

One may mitigate this tension by using an inhomogeneous diffusion coefficient $c(|\nabla I|)$ in the energy as an indicator of boundaries between objects,

$$\min_{d \in \mathbb{R}, \mathbf{w} \in \mathbb{R}^2} \alpha_1 \int_{\Omega_2} |c(|\nabla I|)\, \nabla d - \mathbf{w}| d\Omega_2 + \alpha_2 \int_{\Omega_2} |\nabla \mathbf{w}| d\Omega_2 \quad (19)$$

where $c(|\nabla I|)$ is defined as,

$$c(|\nabla I|) = \exp(-\gamma |\nabla I|^\beta) \quad (20)$$

However, though this aids the regularizer, it does not contain information about the *direction* of the border between the objects. To take this information into account we adopt an anisotropic diffusion tensor,

$$\min_{d \in \mathbb{R}, \mathbf{w} \in \mathbb{R}^2} \alpha_1 \int_{\Omega_2} |\mathbf{T}\, \nabla d - \mathbf{w}| d\Omega_2 + \alpha_2 \int_{\Omega_2} |\nabla \mathbf{w}| d\Omega_2 \quad (21)$$

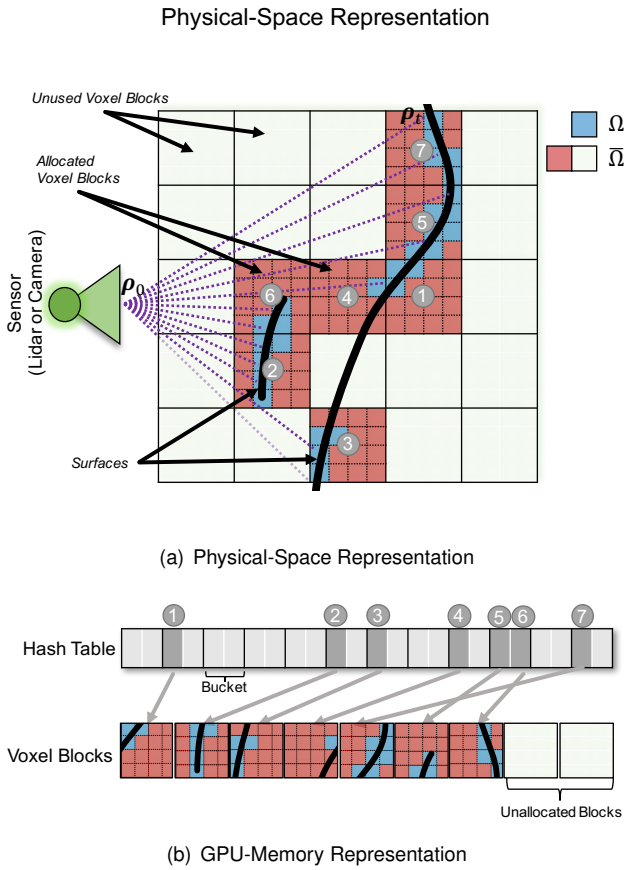
with,

$$\mathbf{T} = \exp(-\gamma |\nabla I|^\beta) n n^T + n^\perp n^{\perp^T} \quad (22)$$

where $n = \frac{\nabla I}{|\nabla I|}$ and $n^\perp$ is its orthogonal complement. γ and β are free parameters (Table 2). The effect of this tensor on Equation 22 is to decompose the components of the disparity gradient ∇d in directions aligned with the gradient of the image n and orthogonal to it $n^\perp$. We do not smooth depth discontinuities aligned with n but we penalize large image gradient components aligned with $n^\perp$. In other words, if there is a discontinuity visible in the color image, then it is highly probable that there is a discontinuity in the depth image. The benefits of this tensor term are visually depicted in Figure 6.

5 Dense 3D Fusion

The core of the 3D dense mapping system consists of a Dense Fusion module which integrates a sequence of depth/range observations (depth maps from Section 4 or laser scans) into a volumetric representation. To smooth noisy surfaces and remove uncertain surfaces, our system carries out an energy optimization on the data volume together with 3D regularization.



(a) Physical-Space Representation

(b) GPU-Memory Representation

Figure 7. A depiction of our novel combination of the Hashing Voxel Grid (HVG) data structure with regularization to optimally 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 colored blocks). A location in the physical operating environment is mapped to a *bucket* index in GPU memory (gray circle number) via a hash function. The *bucket* may be thought of as a linked list to deconflict hash collisions. To avoid generating spurious surfaces, we mark each voxel with an indicator variable ($\mathbf{1}_{\Omega_3}$) to ensure the regularizer 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. The rationale behind Ω_3 and $\bar{\Omega}_3$ is described in Section 5.3. Figure inspired by Nießner et al. (2013).

5.1 Fusing Data

Our aim is to efficiently reconstruct urban outdoor 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. 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} (Curless and Levoy 1996). The TSDF value of each voxel is computed such that one can globally solve for the zero-crossing level set (isosurface) to find a continuous surface model. Even though the voxel grid is a discrete field, because the TSDF value stores the precise distance to the nearest surface, the surface reconstruction is even more precise than the voxel size.

Due to memory constraints, only a small subset of space (a few cubic meters) can be reconstructed using a conventional approach where the voxel grid is fixed in space Newcombe et al. (2011a,b). 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., 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) (Whelan et al. 2012), as shown in Table 1.

In recent years, a variety of techniques have been proposed to remove these limits (Whelan et al. 2012; Nießner et al. 2013; Chen et al. 2013). 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. A compressed data structure only allocates and stores data in voxels which are near a surface.

One of the more prolific compression approaches is Nießner et al.'s (2013) Hashing Voxel Grid (HVG). HVG creates an "infinite" (within numerical limitation) virtual grid that subdivides the environment with a course and fine level of detail (see Figure 7). 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. Applying a hash function to coordinates in world space deterministically computes the index within the hash table, which in turn points to the $8 \times 8 \times 8$ voxel grid of SDF data. We refer the reader to the original HVG paper (Nießner et al. 2013) for further implementation details.

Fusing data into the voxel grid varies depending on whether we are using a camera-based sensor (Section 5.1.1) or a laser-based sensor (Section 5.1.2).

5.1.1 Depth-Map 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 (Amanatides et al. 1987) the depth map and reserving memory from the range $d \pm \mu$.

Once memory is reserved, the data fusion 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 Newcombe et al. (2011a). For each voxel allocated in memory, perform the following operations on every new depth map, D:

1. Calculate the voxel's global-frame center $\mathbf{p}_g = [x_g, y_g, z_g]^T$ 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 which exactly intersect the surface.
4. Update the voxel's current f (SDF value) and w (weight or confidence in f) at time k,

$$w_k = \begin{cases} w_{k-1} + 1 & u_{SDF} \geq -\mu \\ w_{k-1} & u_{SDF} < -\mu \end{cases}$$
$$f_k = \begin{cases} \frac{u_{SDF} + w_{k-1} f_{k-1}}{w_k} & u_{SDF} \geq -\mu \\ f_{k-1} & u_{SDF} < -\mu \end{cases} \quad (23)$$

   where w_{k-1} and f_{k-1} are the previous values of f and w for that voxel.

Note we initially elected to use the SDF rather than the TSDF 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. For example, μ might be large for points far away from the stereo camera but be smaller for laser scans or close stereo observations. However, in practice, we found a fixed μ value based on the sensor accuracy (Table 2) produced better results. When using multiple sensor modalities, we selected μ based on the *least* accurate sensor.

This fusion process is well suited for real-time processing on a GPU (via CUDA or OpenCL) as, once the memory is properly allocated, no atomic operations are required to fuse a deph map into the voxel grid. Each voxel can be allocated an individual GPU thread to compute the updated f and w values.

5.1.2 Laser Fusion Fusing laser data into the voxel grid is accomplished via an efficient ray casting implementation (Amanatides et al. 1987). Consider the case of a single laser ray ($\vec{\rho}$) with an origin at $\boldsymbol{\rho}_o$ and a termination (i.e., where the laser struck a surface) point at $\boldsymbol{\rho}_t$. $\vec{\rho}$ is fused into the voxel grid by tracing the ray and updating each voxel with the signed distance to $\boldsymbol{\rho}_t$. Specifically, the following operations are performed on all voxels which intersect the ray from $\boldsymbol{\rho}_o$ to $\boldsymbol{\rho}_t$:

1. Calculate the voxel's global-frame center $\mathbf{p}_g = [x_g, y_g, z_g]^T$
2. Compute the vector from the laser scan's origin to the current voxel center: $\boldsymbol{v}_1 = \mathbf{p}_g - \boldsymbol{\rho}_o$
3. Compute the vector from the current voxel center to the laser scan's termination point: $\boldsymbol{v}_2 = \boldsymbol{\rho}_t - \mathbf{p}_g$
4. Compute the voxel's SDF value:

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

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 23.

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.

To conserve space, voxel blocks are *allocated* in memory for the region $\pm\mu$, but voxels between the sensor and observed surface are *updated* (if they exist). This removes free-space violations without increasing the memory requirements. In other words, in addition to the memory allocated to store the observed surface, we ray trace between the surface and the sensor to update any voxel blocks which may have previously been allocated when fusing depth or range data. For example, if an automobile appeared in a previous depth map but it has since moved, the full ray trace will update the SDF values in those now-empty voxels (i.e., "delete" the now-spurious automobile reconstruction).

5.2 Energy for 3D Reconstruction

Both inputs to our system pipeline, stereo-image-based depth maps and laser scans, are noisy, especially when compared to the centimeter-level-accuracy of RGB-D systems. As explained in Section 3, we pose the noise-reduction problem as a continuous energy minimization

$$E(u) = E_{reg}(u) + E_{data}(u, f) \tag{24}$$

where f represents noisy SDF data and u is the optimally denoised SDF, both of which are defined in the 3D domain $\Omega_3 \subset \mathbb{R}^3$.

For the regularization term, we first implemented a second order 3D TGV^2 regularizer, – TGV^2 provided the high-quality 2D depth-map results in Section 4 – but we found that the simpler first order TV regularizer produces nearly identical results. We believe this is because the size of the voxels is very small relative to the size of the surfaces being reconstructed. The frontal-parallel and piece-wise discontinuity effects are therefore minimal, so we selected the 3D TV regularizer as it has significantly lower computational and memory requirements while producing similar results. In practice, the TV norm has a two-fold effect: (1) smooths out the reconstructed surfaces, and (2) removes surfaces which are "uncertain" – i.e., voxels with high gradients and few direct observations. This includes removing surfaces with very small surface area, but also serves to smooth out larger surfaces.

We propose two implementations for the data term: the first method (Section 5.2.1) averages all depth measurements, whereas the second method (Section 5.2.2) records the depth measurements as samples from a probability-density function in a histogram data structure.

5.2.1 SDF Data Term

In the method originally proposed by Curless and Levoy (1996) and popularized by Newcombe et al. (2011a), 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 23) are two numbers stored in each voxel: (1) f, the signed metric distance to the nearest surface, and (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 characterize the sampled distribution. As the energy minimization seeks to solve for a denoised-version of f, a simple L2 norm data term ensures the final u is consistent with the observed depths:

$$E(u) = \int_{\Omega_3} |\nabla u|_1 d\Omega_3 + \frac{\lambda}{2} \int_{\Omega_3} ||f - u||_2^2 d\Omega_3 \tag{25}$$

5.2.2 Histogram Data Term

In general, the more meaningful information provided to an optimizer, the better the final results. The SDF data-term method summarized 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 (Zach et al. 2007),

$$E(u) = \int_{\Omega_3} |\nabla u| d\Omega_3 + \frac{\lambda}{2} \int_{\Omega_3} \frac{1}{N} \sum_{t=1}^{N} ||f_t - u||_1 d\Omega_3 \tag{26}$$

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.

A more practical approach is to store depth measurements as a histogram representing the probability-density function (PDF) for that voxel rather than explicitly storing the entire history of observations or the mean and variance (Zach 2008). 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 linear scale-and-clamp f in the interval [-1, 1). The relationship between SDF, TSDF, and histogram bins is graphically depicted in Figure 8.

The energy minimization becomes,

$$E(u) = \int_{\Omega_3} |\nabla u| d\Omega_3 + \frac{\lambda}{2} \int_{\Omega_3} \sum_{b=1}^{n_{bins}} h_b ||c_b - u||_1 d\Omega_3 \tag{27}$$

where the center of the bins are computed as,

$$c_b = \frac{2b}{n_{bins}} - 1 \tag{28}$$

This method has been demonstrated by Zach (2008) on a small-scale, object-centered environment where the final reconstruction was within a millimeter of the ground-truth laser scan with 99% completeness.

Table 2. Summary of Parameters Used in System

Symbol	Value	Description
λ_{3D}	0.8 (10 cm) / 0.4 (20 cm)	The Total Variation (TV) weighting of the data term vs. regularization
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 regularizer gradient-ascent/descent step sizes
τ	$^1/_6$	3D regularizer relaxation-step weight
N	20	Number of histogram bins
λ_{2D}	0.5	The Total Generalized Variation (TGV) weighting of the data term vs. regularization
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

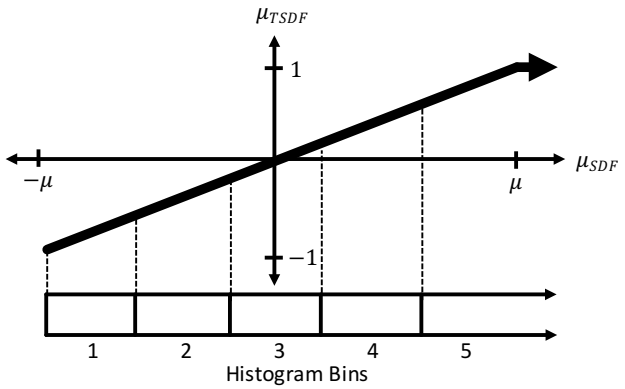
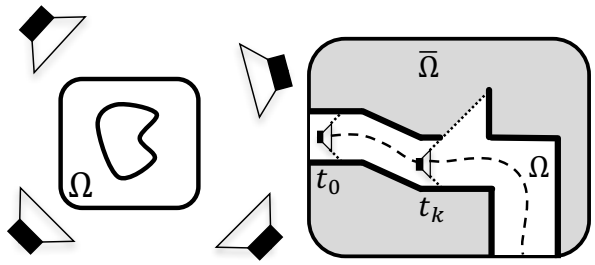


Figure 8. Visual depiction of the relationship between the TSDF, SDF, and histogram bins storage methods for voxel distance values. TSDF has traditionally been the favoured approach, but SDF is useful when fusing multiple sensor modalities so that each each sensor has a different μ based on its precision. The histogram approach is useful to aid the regularizer by directly storing a PDF (vs. the weighted average stored by SDF/TSDF) of noisy surface measurements.



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

Figure 9. Traditional 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 regularization techniques do not make a distinction between Ω_3 (observed regions) and $\bar{\Omega}_3$ (unobserved regions). However, in mobile-robotics applications the world environment is traversed and observed during exploration, requiring voxels which were never directly observed. For example, at camera capture t_x, it is unknown what exists in the camera's upper field of view. Not accounting for $\bar{\Omega}_3$ in regularisation results in incorrect surface generation.

5.3 Omega Domain

Since we are moving at an *a priori*-unknown trajectory through the world, we only observe surfaces in a subset of all allocated voxels and voxel blocks. We therefore present a new technique to prevent the unobserved voxels from negatively affecting the regularisation results of the observed voxels – i.e., to ensure the regularizer doesn't extrapolate into unobserved regions. In order to achieve this, as illustrated in Figure 9, we define the complete voxel grid domain as Λ and use Ω_3 to represent the subset of voxels which have been directly observed and which will be regularised. The remaining subset, $\bar{\Omega}_3$, represents voxels which have never been observed. By definition, Ω_3 and $\bar{\Omega}_3$ form a partition of Λ and therefore $\Lambda = \Omega_3 \cup \bar{\Omega}_3$ and $\Omega_3 \cap \bar{\Omega}_3 = \varnothing$.

To the authors' knowledge, all prior works rely on a fully-observed conventional voxel grid before regularization and they implicitly assume that $\Lambda = \Omega_3$ — i.e., every voxel in the voxel grid is directly observed by the sensor. In our mobile-robotics platform, this assumption is not valid. The robot motion results in unobserved regions caused by object occlusion, field-of-view limitations, and trajectory decisions. Therefore, $\Omega_3 \subset \Lambda$ as Figure 9 illustrates. In practice, we found failure to account for the Ω_3-$\bar{\Omega}_3$ boundaries causes the regulariser to spuriously extrapolate surfaces into undesired regions of the reconstruction.

Our approach introduces a new state variable, $\mathbf{1}_{\Omega_3} : \Lambda \rightarrow \{0, 1\}$, in each voxel indicating whether or not it was directly observed by a range sensor. Ω_3 is therefore the set solely upon which the regluarizer is constrained to operate. In our previous work, we found that failure to account for these boundary conditions creates spurious surfaces in non-valid voxels ($\bar{\Omega}_3$) (Tanner et al. 2015; Tanner et al. 2016). Note that all voxel *blocks* which are *not* allocated are in $\bar{\Omega}_3$. A graphical depiction of how we augmented the HVG data structure with Ω_3 and $\bar{\Omega}_3$ is provided in Figure 7.

Since compressed voxel grid data structures are not regular in space, the proper method to compute the gradient (and its dual: divergence) in the presence of the additional boundary conditions (caused by the Ω_3 domain and voxel-block boundaries) is not straightforward.

We define the gradient as:

$$\nabla_x u_{i,j,k} = \begin{cases} u_{i+1,j,k} - u_{i,j,k} & \text{if } 1 \leq i < V_x \\ 0 & \text{if } i = V_x \\ 0 & \text{if } u_{i,j,k} \in \bar{\Omega}_3 \\ 0 & \text{if } u_{i+1,j,k} \in \bar{\Omega}_3 \end{cases} \tag{29}$$

where $u_{i,j,k}$ is a voxel's TSDF 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 regularizer encourages smoothness across neighbouring voxels which explains why this new gradient definition excludes $\bar{\Omega}_3$ voxels — they have not been observed.

To solve the Primal-Dual optimization (Section 3), we also need to define the corresponding divergence operator:

$$(\nabla \cdot \mathbf{p}_{i,j,k})^x = \begin{cases} \mathbf{p}^x_{i,j,k} - \mathbf{p}^x_{i-1,j,k} & \text{if } 1 < i < V_x \\ \mathbf{p}^x_{i,j,k} & \text{if } i = 1 \\ -\mathbf{p}^x_{i-1,j,k} & \text{if } i = V_x \\ 0 & \text{if } u_{i,j,k} \in \bar{\Omega}_3 \\ \mathbf{p}^x_{i,j,k} & \text{if } u_{i-1,j,k} \in \bar{\Omega}_3 \\ -\mathbf{p}^x_{i-1,j,k} & \text{if } u_{i+1,j,k} \in \bar{\Omega}_3 \end{cases} \tag{30}$$

Each voxel block is treated as a voxel grid with boundary conditions which are determined by its neighbours' Ω_3 indicator function, $\mathbf{1}_{\Omega_3}$. The regularizer operates only on the voxels within Ω_3, the domain of integration, and thus it neither spreads spurious surfaces into unobserved regions nor updates valid voxels with invalid SDF data.

Note that 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.

5.4 3D Energy Minimization

In the following two subsections, we describe the algorithm to solve Equations 25 and 27. We point the reader to Piniés et al. (2015); Chambolle and Pock (2011); Rockafellar (1970); Handa et al. (2011) for a detailed derivation of these steps. We vary from their methods in our new definition for the gradient and divergence operators, as described in Section 5.3.

5.4.1 SDF Optimization Implementation The optimization described in Equation 25 is a 3D version of the Primal-Dual TV algorithm explained in Section 3. We just need to use the new definitions of the gradient and the divergence operator defined previously.

5.4.2 Histogram Optimization Implementation Again, the implementation to minimize the histogram-based approach closely mirrors the implementation described Section 3. The only difference is that the primal update (Equation 13) becomes

$$\begin{aligned} u_k &= \text{median}(c_1, ..., c_{n_{bins}}, b_0, ..., b_{n_{bins}}) \\ \tilde{u} &= u_k - \tau \nabla \cdot \mathbf{p} \\ b_i &= \tilde{u} + \tau \lambda W_i \\ W_i &= -\sum_{j=1}^{i} h_j + \sum_{j=i+1}^{n_{bins}} h_j \\ i &\in [0, n_{bins}] \end{aligned} \tag{31}$$

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

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

5.5 Data Structures

To summarize the key data tracked during the data fusion and 3D regularization stages our pipeline, we define the following three voxel datatypes:

Listing 1: Voxel Data Structures

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

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

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

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 7 provides a more detailed analysis on why this is the case.

Depth-map fusion `color` is read directly from the RGB reference image. When fusing lidar scans, the `color` is the grayscale reflectance intensity value. If *both* lidar and depth images are fused into the same voxel grid, color data always takes priority over grayscale data as to make the final model visually appealing (i.e., surfaces viewed by the camera are colored correctly while surfaces viewed only by the laser are grayscale).

Once all data is fused, the energy minimization 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 (Lorensen and Cline 1987) or Ray Casting (Amanatides et al. 1987) to solve for the zero-crossing level set of u.

6 Learnt Corrections and Prior Application

The final step in our pipeline is a system that uses prior knowledge of scene appearance and geometry to detect and correct missing data in the 3D reconstructions (e.g. holes in the road). To achieve this, we leverage a deep neural network. The subsystem described in this section is illustrated in Figure 10 and operates as follows:

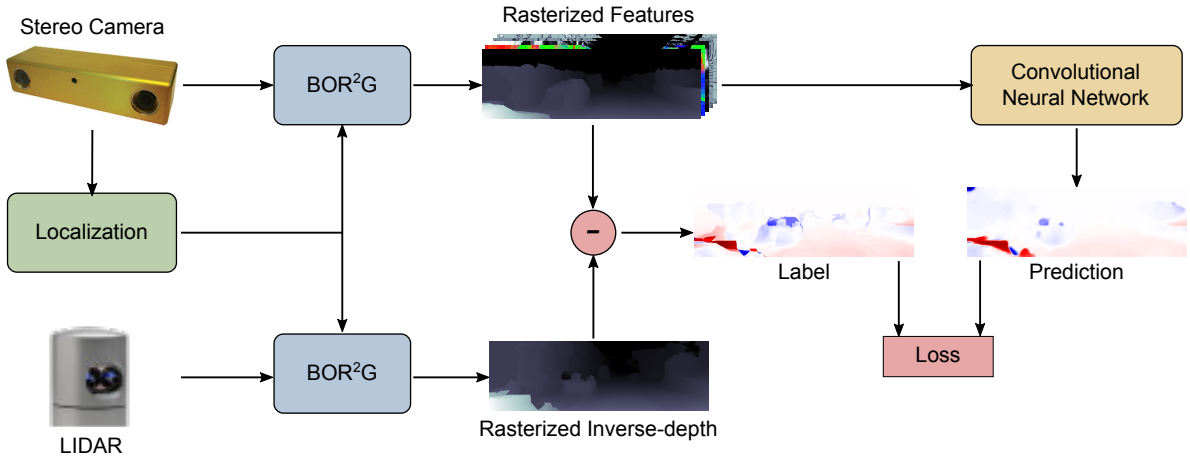
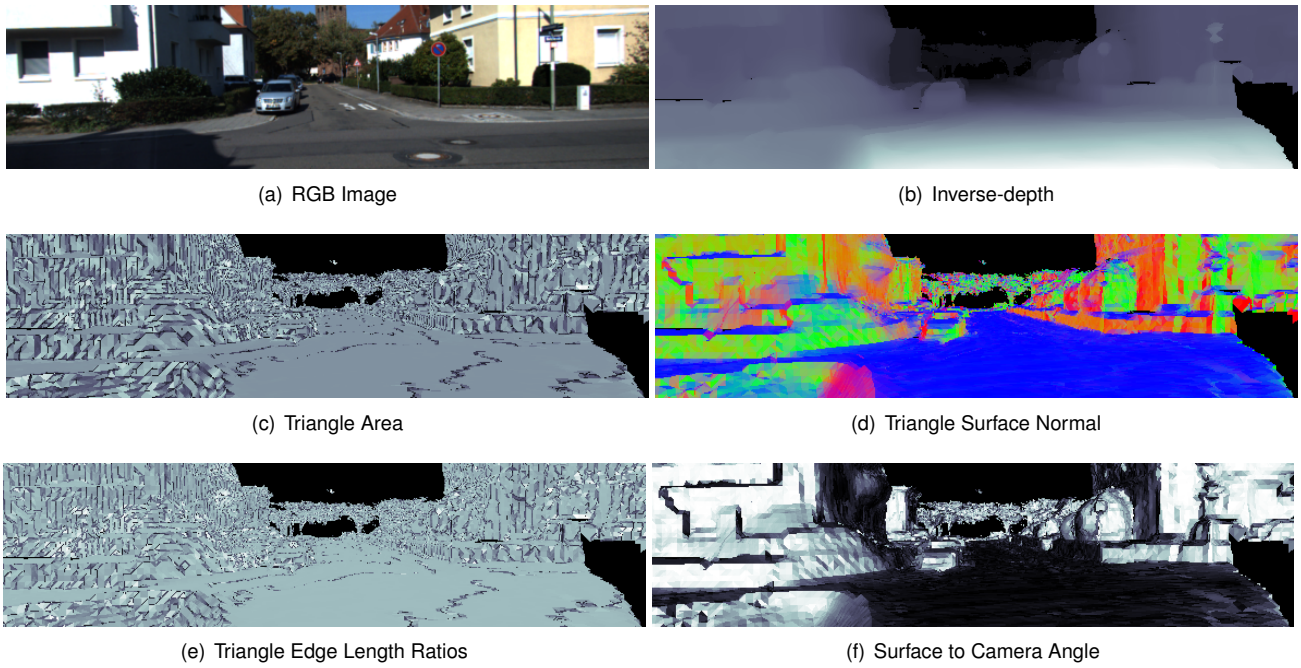


Figure 10. Machine Learning Data-Flow Pipeline. To train our network, we create separate stereo-camera and laser dense reconstructions, using the techniques discussed in Sections 3–5. We create ground-truth data by subtracting the rasterized inverse-depth images from each reconstruction. The neural network trains on rasterized feature images (see Figure 11) to learn to generalize the ground-truth error in new scenes.



(a) RGB Image (b) Inverse-depth

(c) Triangle Area (d) Triangle Surface Normal

(e) Triangle Edge Length Ratios (f) Surface to Camera Angle

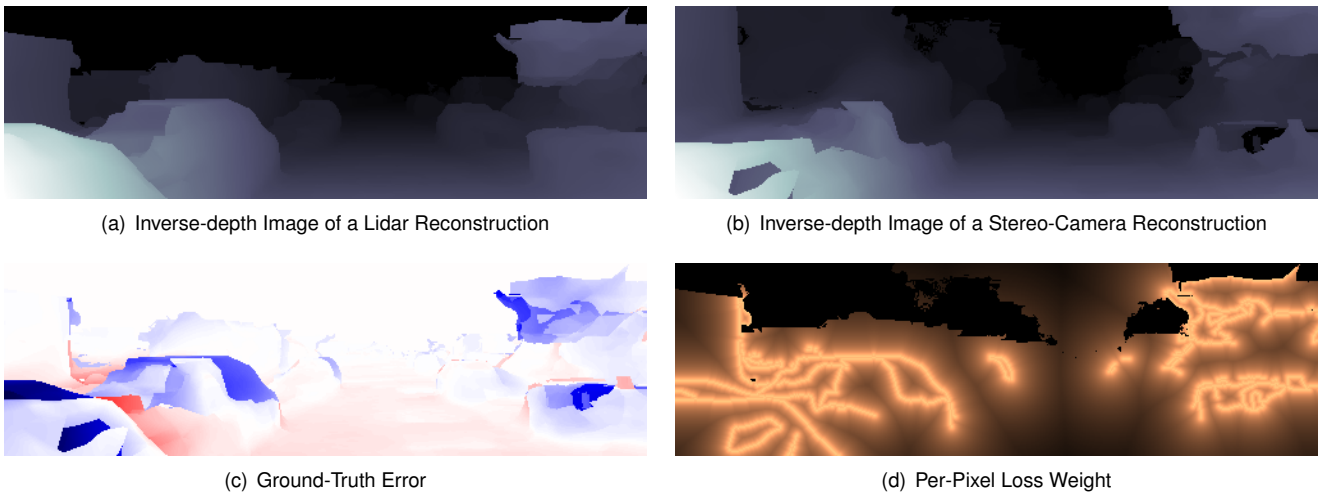
Figure 11. Example Input Features. As our dense reconstructions provide a 3D model of our operating environment, we can extract a variety of features. Pictured above are an example frame from KITTI-VO (a), and the corresponding features our GLSL pipeline currently extracts (b)–(f). The network has the potential to use these low level features in its intermediate representation.

1. Generate a sequence of mesh features (including inverse-depth maps) from an existing reconstruction;
2. Use a CNN trained on high-quality reconstructions to predict the residual error in inverse-depth;
3. Correct the initial inverse-depth maps by subtracting the predicted errors;
4. Create a new reconstruction using the other modules described in this paper.

6.1 Mesh Features

Many works in the deep learning literature deal with RGB images and thus operate in 2D. However, as our problem formulation assumes that a dense 3D model is available, we can extract a much richer set of features from the reconstruction. Our system builds on the method proposed by Tanner et al. (2018), where mesh features are projected into sequences of 2D images. This method enables the use of established CNN architectures, while still leveraging the geometric information that meshes contain.

6.1.1 Feature Creation To extract dense 2D features suitable for a CNN, we begin by using GLSL to "fly" a virtual camera through the 3D models. Our `vertex` shader transforms the 3D model's vertices into the camera reference frame 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 11. This GLSL



(a) Inverse-depth Image of a Lidar Reconstruction

(b) Inverse-depth Image of a Stereo-Camera Reconstruction

(c) Ground-Truth Error

(d) Per-Pixel Loss Weight

Figure 12. Illustration of the edge-based loss weighting used to train our neural network. When learning to predict error for a scene (a,b), edges are extracted from the ground-truth error label (c) using the Canny edge detector, and a weighting based on the distance transform is computed (d). Brighter areas represent higher weights. The per-pixel losses (berHu, bilateral smoothness regularization) are scaled by this weight, increasing penalty especially around sharp edges. The black areas correspond to a weight of 0, where ground-truth data is missing. Note the ground-truth error (c) is signed: blue represents negative values (missing parts) and red represents positive values (extra parts).

process enables us to extract six feature images: two per-pixel features (RGB and depth) and four per-triangle features (area, surface normal, edge length ratio, and surface-to-camera angle). Intuitively, the mesh-triangle-derived features contain a lot of information; they provide geometric cues that have been accumulated from multiple depth-maps and integrated into a 3D reconstruction by BOR^{2}G.

6.1.2 Ground-Truth Data For a ground-truth reference, we construct a mesh using the same reconstruction pipeline as above. The same virtual camera trajectory is used to create a sequence of feature images from both lower-quality (e.g. stereo-camera) and higher-quality (e.g. laser) reconstructions. Since the goal is to train a network to recognize *errors* in the lower-quality reconstruction, we must first compute *ground-truth errors* with which to train the network. Rather than directly subtracting depths, we instead elect to use inverse-depth. This way, the network is encouraged to emphasize errors in foreground objects – which are likely to have more detailed observable geometry compared to background surfaces. Specifically, we compute the per-pixel ground-truth (Δ_{gt}) as:

$$\Delta_{gt} = \frac{1}{d_{lq}} - \frac{1}{d_{hq}}, \tag{32}$$

where d_{lq} and d_{hq} are pixels in the depth-map features for the low-quality and high-quality reconstruction, respectively. An illustration of this is shown in Figure 12c.

6.2 Reconstruction Error Prediction

6.2.1 Network Architecture The architecture we employ in this paper, like the one proposed by Tanner et al. (2018), is based on the Fully Convolutional Residual Network (Laina et al. 2016) with concatenated ReLU activation functions on the intermediate layers (Shang et al. 2016). This network was designed to infer per-pixel depth from a single RGB image, a task highly correlated with our aim to compute estimated depth *error* given a set of feature inputs. In this work, the input layer is generalized to accept F input channels dependant on the number of active features used from the previous section. A notable addition is the U-Net-style (Ronneberger et al. 2015) skip connections between the encoder and the decoder. Table 3 gives a summary of the architecture. The encoder uses a series of residual blocks based on the *ResNet-50* architecture (He et al. 2016), and the decoder uses the up-projection blocks proposed by Laina et al. (2016). The outputs of the residual blocks are padded and concatenated with the inputs to the up-projection blocks of corresponding scale. This architecture allows for high-frequency information (such as edges), to be more easily localized, since it is not compressed all the way through the encoder. The final layer is a simple 3×3 convolutional layer which outputs real-valued estimates corresponding to the inverse-depth error at each pixel location.

Table 3. Overview of the CNN architecture for error prediction.

Block Type	Filter Size/Stride	Output Size
Input	-	150 × 590 × F
Convolution	7×7/2	74 × 295 × 32
Max Pool	3×3/2	38 × 148 × 32
Convolution	3×3/1	38 × 148 × 64
Residual, Residual, Projection	3×3/2	19 × 74 × 256
Residual, Residual, Projection	3×3/2	10 × 37 × 512
Residual, Residual, Projection	3×3/2	5 × 19 × 1024
Residual, Residual	3×3/1	5 × 19 × 1024
Up-projection	3×3/½	10 × 38 × 512
Up-projection	3×3/½	20 × 76 × 256
Up-projection	3×3/½	40 × 152 × 128
Up-projection	3×3/½	80 × 304 × 64
Up-projection	3×3/½	160 × 608 × 32
Convolution, Resize	3×3/1	150 × 590 × 1

6.2.2 Generalisation Capacity To be able to deploy a learnt CNN to new meshes created without a high-fidelity sensor, we need our network to generalize to unseen data. To this end, three techniques are employed: cropping, downsampling, and weight regularization. Firstly, we randomly perturb and crop all input feature images before providing them to the network. After each epoch of training (i.e. the network has viewed all the training images), the next epoch will receive a slightly different cropped region of each input image. This prevents the network from associating a specific pixel location in the training data with its corresponding output.

Secondly, the feature maps are gradually downsampled via projection blocks (from *ResNet-50*) creating a bottleneck, thus reducing representational capacity. This also provides greater context to the convolutional filters in the later layers of the network by increasing the size of their receptive field.

Thirdly, we implement an L_2 weight regularizer to further constrain the representational capacity by preventing the network from over-relying on the cost function at the expense of generalization performance.

6.2.3 Loss Function Several loss functions are widely used in machine learning applications. The L_2 norm is traditionally popular because it heavily penalises large errors and is smooth. However, unlike the L_1 norm, it has a near zero gradient for small errors, thus is often unable to drive the error completely to zero. Over the years, researchers have proposed alternative norms which combine the "best" characteristics (based on application) of each of these norms. The Huber norm uses L_2 near the origin and L_1 elsewhere, while berHu uses the L_1 norm near the origin and the L_2 elsewhere (Owen 2007). We choose berHu as our cost function on the networks error prediction 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. This is in contrast to regularizing depth maps where the Huber norm is more favourable to capture high gradient edges around objects (Newcombe et al. 2011b). These benefits were also observed in the related task of depth prediction Laina et al. (2016).

Similarly to Heise et al. (2013); Godard et al. (2017); Mahjourian et al. (2018), an edge-aware smoothness loss is employed to further regularize the network. It essentially allows for discontinuities in CNN predictions $\widetilde{Y}^{ij}$ for each pixel ij, so long as there are corresponding discontinuities in the label Y^{ij}:

$$L_{sm} = \sum_{i,j} \left|\partial_x \widetilde{Y}^{ij}\right| \cdot e^{-|\partial_x Y^{ij}|} + \left|\partial_y \widetilde{Y}^{ij}\right| \cdot e^{-|\partial_y Y^{ij}|}. \quad (33)$$

Furthermore, inspired by the work of Ronneberger et al. (2015) on U-Nets, we use a loss-weighting mechanism based on the Euclidean Distance Transform (Felzenszwalb and Huttenlocher 2004) to give more importance to edge pixels when regressing to the error in depth (Figure 12). We first extract Canny edges (Canny 1986) from the ground-truth labels. Based on these edges, we then compute the per-pixel weights as:

$$\begin{aligned} d^{ij} &= \ln(1 + \mathrm{EDT}(i,j)) \\ w^{ij} &= (w_{max} - w_{min}) * \left(1 - \frac{d^{ij}}{\max_{i,j} d^{ij}}\right) + w_{min} \end{aligned}, \quad (34)$$

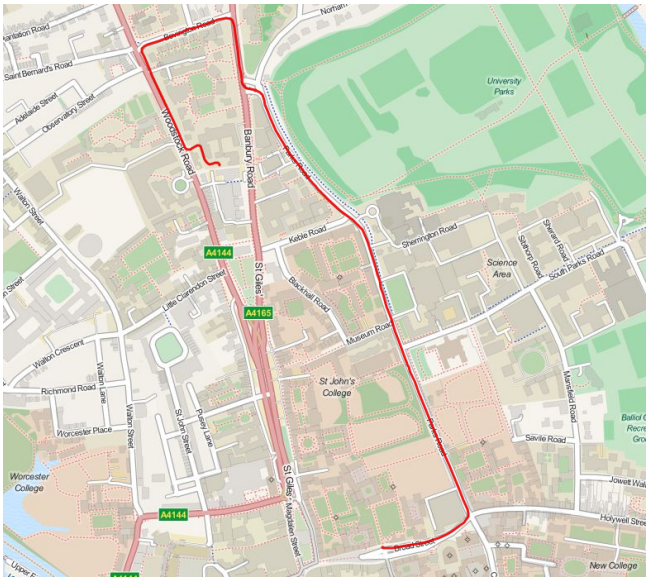


Figure 13. GPS trace for the Oxford Broad Street dataset. The dataset includes data from 1x Bumblebee XB3, 1x Bumblebee2, 4x Grasshopper2, 2x Velodyne HDL-32E lidar, and 3x SICK LMS-151 lidar sensors.

Table 4. Imperial College ICL-NUIM Augmented Results

Scenarios	Mean	St. Dev.	50%	75%
LR1 (Raw)	15.27 cm	14.60 cm	10.10 cm	22.42 cm
(Reg)	**12.18 cm**	**12.39 cm**	**7.64 cm**	**17.29 cm**
LR2 (Raw)	8.11 cm	10.59 cm	4.35 cm	9.32 cm
(Reg)	**6.06 cm**	**7.01 cm**	**3.88 cm**	**7.36 cm**
O1 (Raw)	48.45 cm	177.65 cm	4.40 cm	10.18 cm
(Reg)	**10.70 cm**	**65.65 cm**	**3.34 cm**	**6.63 cm**
O2 (Raw)	28.25 cm	121.53 cm	4.45 cm	11.06 cm
(Reg)	**7.62 cm**	**33.27 cm**	**3.17 cm**	**7.51 cm**

Table 5. 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

where w^{ij} is the loss weight for pixel ij, $\mathrm{EDT}(i,j)$ is the Euclidean Distance Transform at pixel ij, and w_{min} and w_{max} are the desired range of the per-pixel weight. We use $w_{min} = 0.1$ and $w_{max} = 5$.

Finally, since the ground-truth reconstructions might be missing some data, we allow for a per-pixel mask on the loss, to ignore pixels with unavailable data.

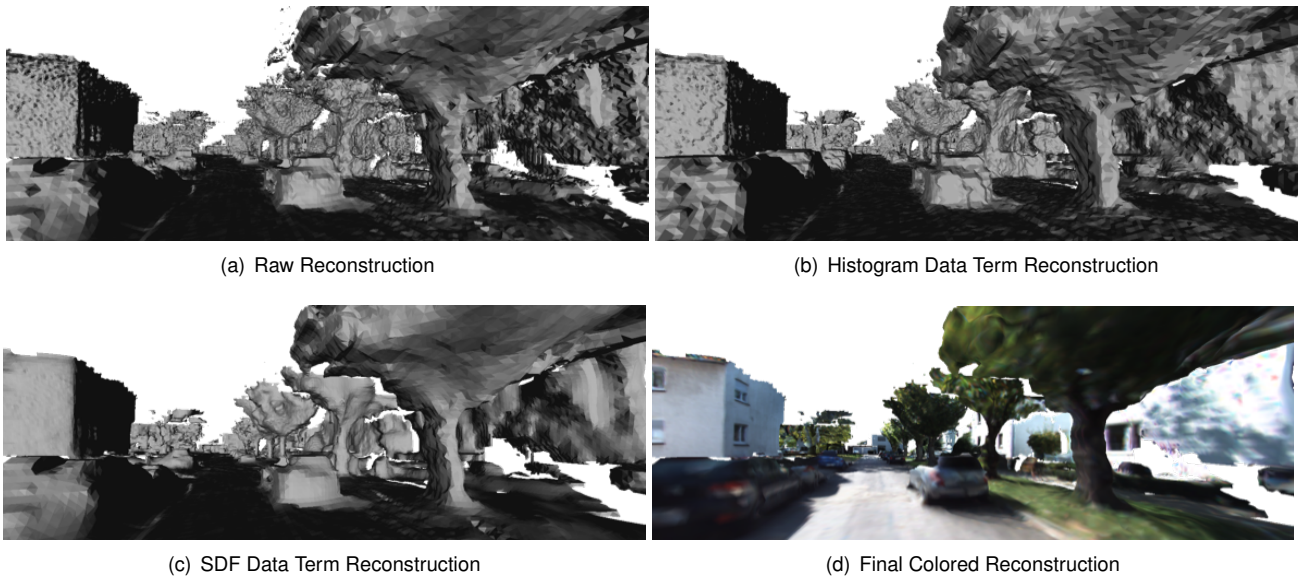
The final loss function can be written as:

$$L = w_\alpha L_{reg} + \sum_{ij} w^{ij} \cdot (w_\beta L^{ij}_{berHu} + w_\gamma L^{ij}_{sm}), \quad (35)$$

where L_{reg} is the weight regularization loss. We use $w_\alpha = 10^{-6}$, $w_\beta = 1$, $w_\gamma = 1$.

7 Results

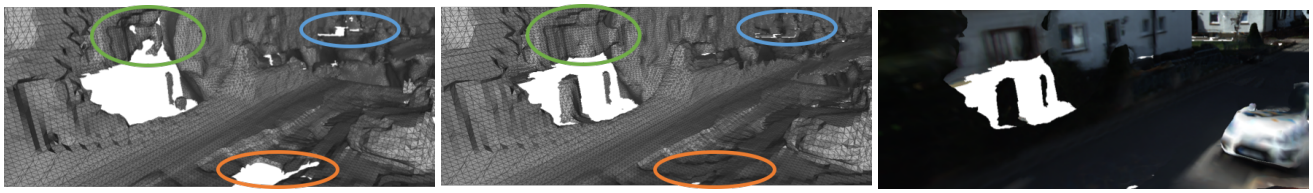
This section provides an extensive analysis of our system, the parameters for which are provided in Table 2. Some



(a) Raw Reconstruction (b) Histogram Data Term Reconstruction

(c) SDF Data Term Reconstruction (d) Final Colored Reconstruction

Figure 14. Comparison of reconstruction improvement with histogram and SDF data terms on stereo-camera-only reconstructions. Both optimization 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 6) SDF and histogram methods are nearly indistinguishable.



(a) Stereo Camera Pass 1 (b) Stereo Camera Pass 2 (c) Final Colored Reconstruction

Figure 15. Comparison of fusion after two passes of the same region. The second pass results in a more complete model since, with accurate localization, 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 center-left of the image is due to object occlusions at the viewing angle of the input stereo images.

Table 6. SDF vs. Histogram Regularized 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

parameters (σ, θ, τ) are standard constants from the literature (Pock and Chambolle 2011). The μ and λ were empirically tuned. As discussed in Section 5.1.1, we found a constant μ value based on the sensor accuracy produced better results across a range of scenarios. When using multiple sensor modalities, we selected μ based on the *least* accurate sensor. We found λ to be dependent on the sensor observation density (number of observations per voxel) and accuracy. Selecting too large λ resulted in minimal changes to the quality of the reconstruction, but a very low λ deletes too many surfaces. If the same platform is used with a similar sensor suite and vehicle velocity between data collections, as is the case in the KITTI dataset, then the same λ is valid for all data collections.

In the first portion of our results, we present how our system's reconstruction quality compares with prior work (Stanford Burghers of Calais and Imperial College ICL-NUIM). Next, we demonstrate its performance on our Oxford Broad Street dataset, that we release with this paper. Then, we use the publicly available KITTI dataset (Geiger et al. 2012) to evaluate the reconstruction performance of stereo-only, laser-only, and multi-sensor fusion methods. Finally, we use the KITTI dataset to present additional improvements that can be made to stereo-only reconstructions, when using a CNN trained on laser data.

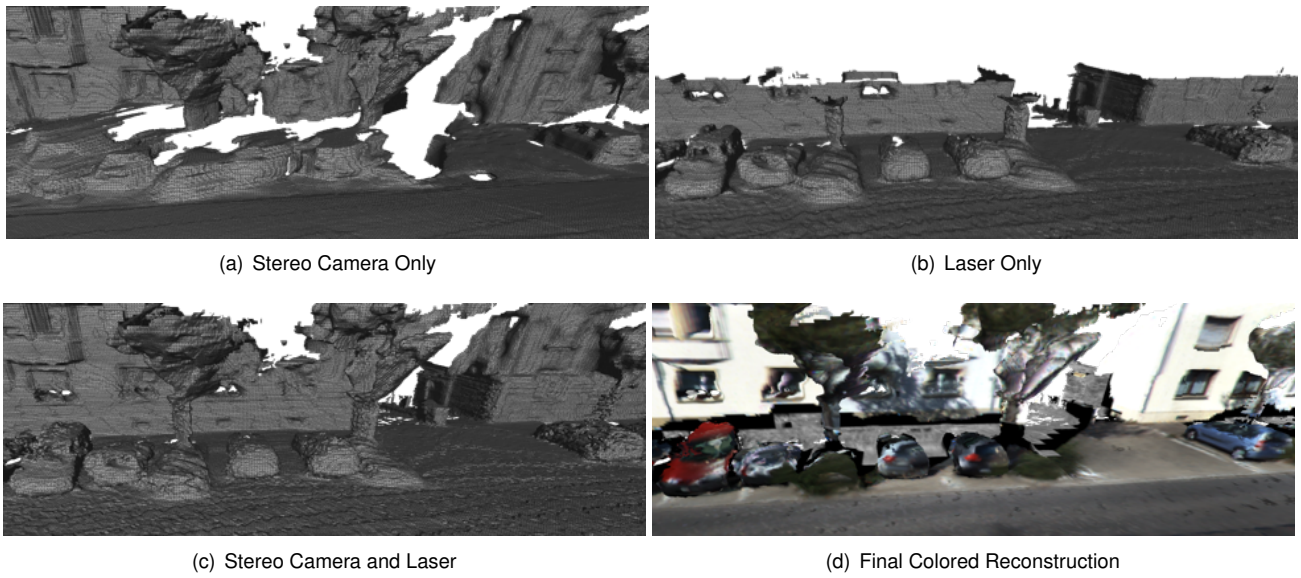
Three KITTI sequences (00, 05, and 06) were selected based on their length, number of loop closures, and urban structure visible throughout the sequences. A summary of the physical scale of each is provided in Table 5.

The dense fusion and reconstruction experiments were performed on a GeForce GTX TITAN with 6 GB memory. The CNN mesh correction experiments were performed on a Tesla K80 with 12 GB memory.

Pose estimation was processed in real time (20 Hz); depth-map estimates (1 Hz), data fusion (5 Hz deph maps, 10 Hz Velodyne), and CNN post-processing (12 Hz) were at interactive rates; while the regularisation could only be achieved via an off-line process since it required multiple minutes to converge in each scenario (see Table 7). With these timing requirements, we use this *reconstruction* (depth

Table 7. Summary of Stereo-Camera-Only Reconstruction Quality and Error Statistics

Dataset	Type	Res.	50%	75%	GPU Memory	Surface Area	# Voxels	Time Per Iter.
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 (mm:ss)
		10 cm	7.3 cm	25.1 cm	7,565 MiB		$661{\cdot}10^6$	11:20 (mm:ss)
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 (mm:ss)
		10 cm	8.1 cm	27.6 cm	4,533 MiB		$396{\cdot}10^6$	3:45 (mm:ss)
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 (mm:ss)
		10 cm	4.0 cm	10.6 cm	356 MiB		$226{\cdot}10^6$	3:44 (mm:ss)



(a) Stereo Camera Only (b) Laser Only

(c) Stereo Camera and Laser (d) Final Colored Reconstruction

Figure 16. 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 which 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 8) and higher-quality result than either sensor can achieve alone.

Table 8. Summary of Surface Area Coverage by Sensor Type

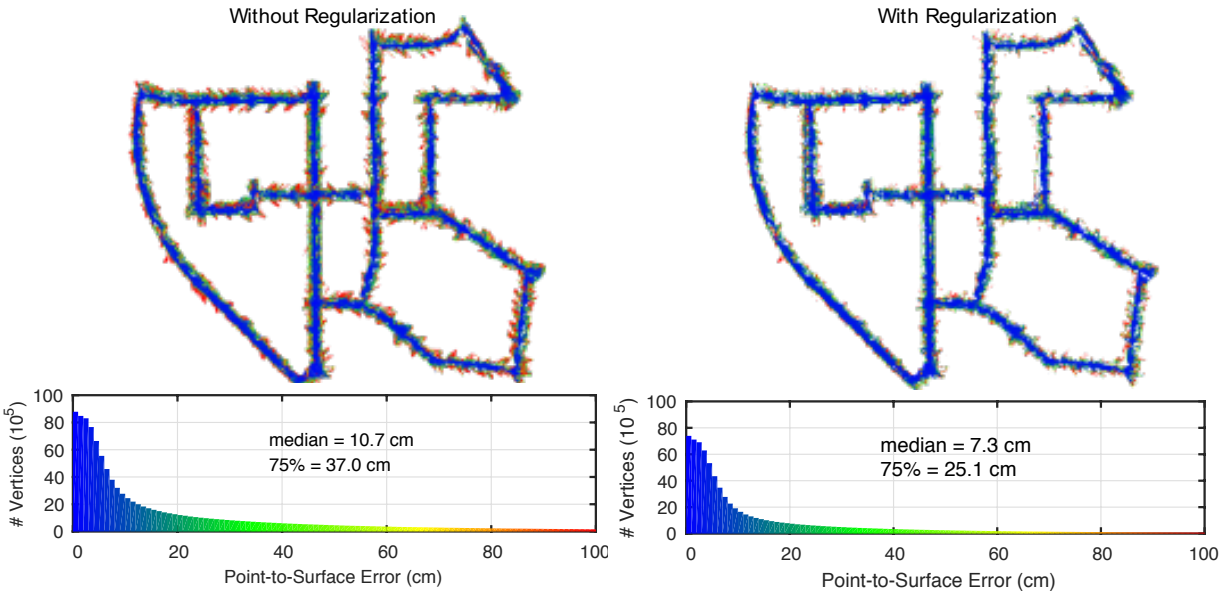
Dataset	Image Only	Laser Only	Image & Laser
KITTI-VO 00	93,546 m^2	117,087 m^2	176,624 m^2
KITTI-VO 05	55,909 m^2	77,292 m^2	111,188 m^2
KITTI-VO 06	23,585 m^2	29,617 m^2	41,401 m^2

map and laser fusion) pipeline in real-time mobile-robotics applications, but then further improve the reconstructions via regularization as a part of the "dream state" robot maintenance between trials. We prefer to create small local maps (as used in Stewart and Newman 2012; Maddern et al. 2014). However, we describe in this section the utility of improving reconstructions after incorporating the loop closures when revisiting a location. After a loop closure, the dense reconstruction may be updated in real-time via graph deformation (Whelan et al. 2014) although some voxel historical data is lost in this process (voxels → mesh → deformed mesh → voxels).

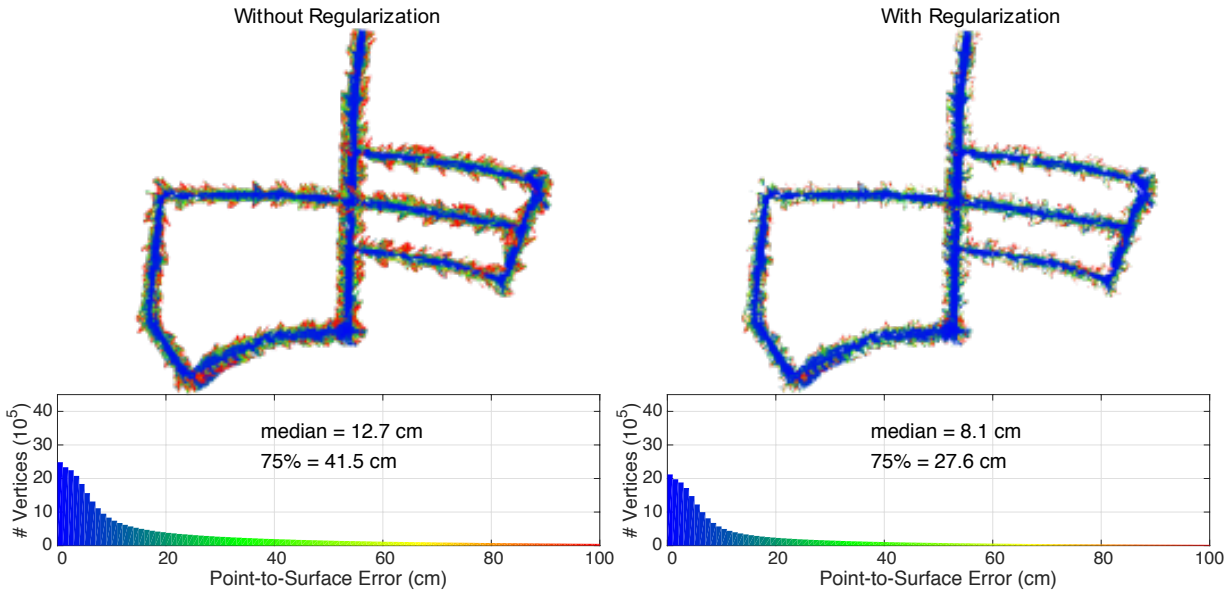
7.1 Imperial College ICL-NUIM

We first evaluated the performance of our dense reconstruction pipeline (fusion and regularization) with the augmented Imperial College ICL-NUIM dataset Choi et al. (2015). The original dataset (Handa et al. 2014) provided a set of indoor scenarios with sequences of ground-truth 3D models, camera poses, and RGB-D images. In 2015, Choi et al. augmented the dataset by applying lens and depth distortion models to make the input more similar to that of real-world data.

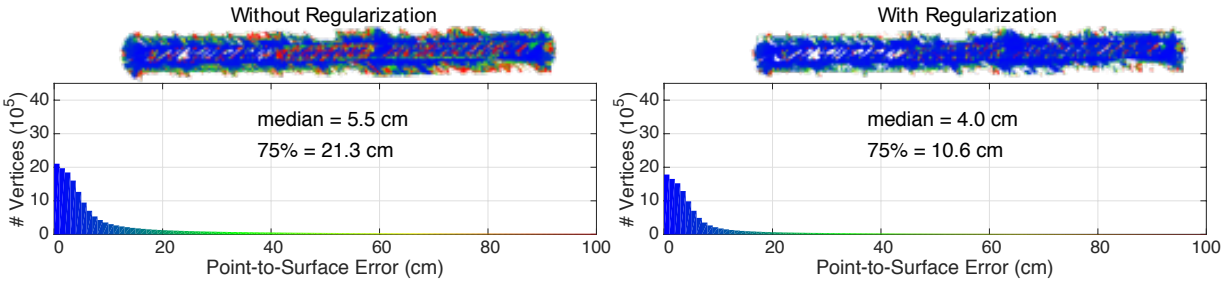
A summary of our reconstruction results are presented in Table 4. The median reconstruction error of the raw fused model ranged from 4.35 cm to 10.10 cm. The regularizer reduced error by 24% to 29%, with final median errors between 3.17 cm and 7.64 cm. The absolute error metrics are not as important as the fact that the regularizer consistently reduced the reconstruction errors – a theme which repeats throughout the quantitative results results presented in this section.



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

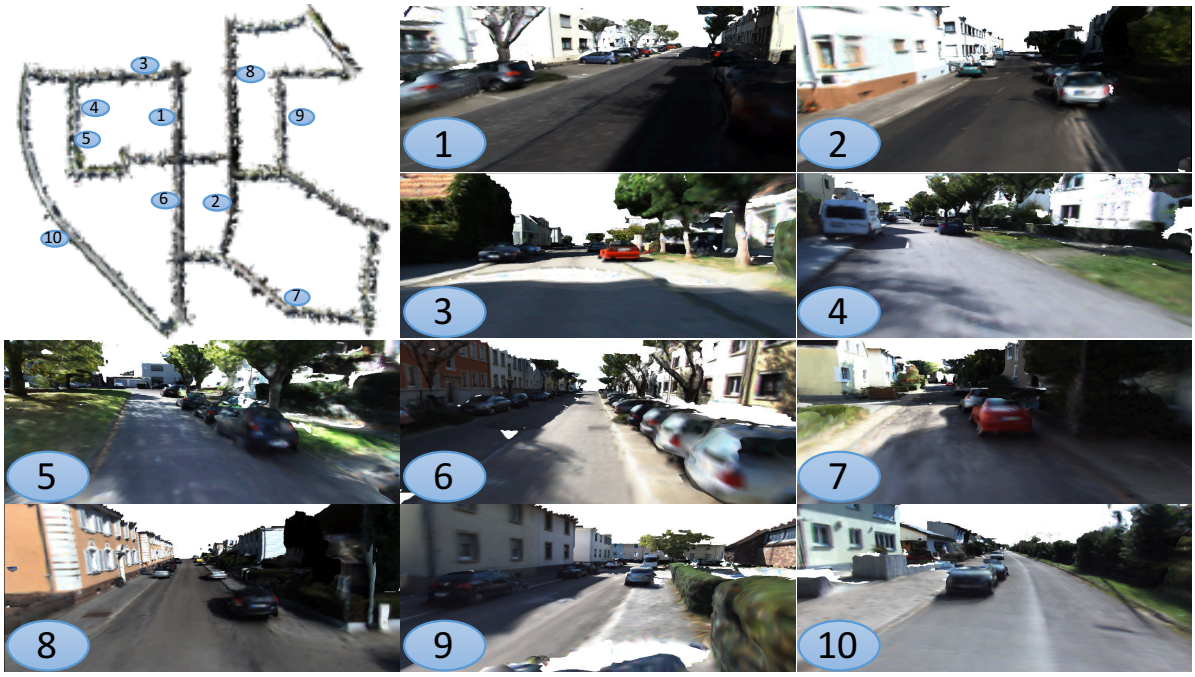


(b) KITTI-VO 05: Stereo Camera Reconstruction Error Metrics



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

Figure 17. 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 regularization and the right side are after regularization. Above each histogram of point-to-surface errors are the top-view, colored reconstruction errors corresponding to the same colors in the histogram. The regularizer reduces the reconstruction's error by approximately a third, primarily by removing uncertain surfaces – as can be seen when you contrast the raw (left) and regularized (right) reconstruction errors.



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

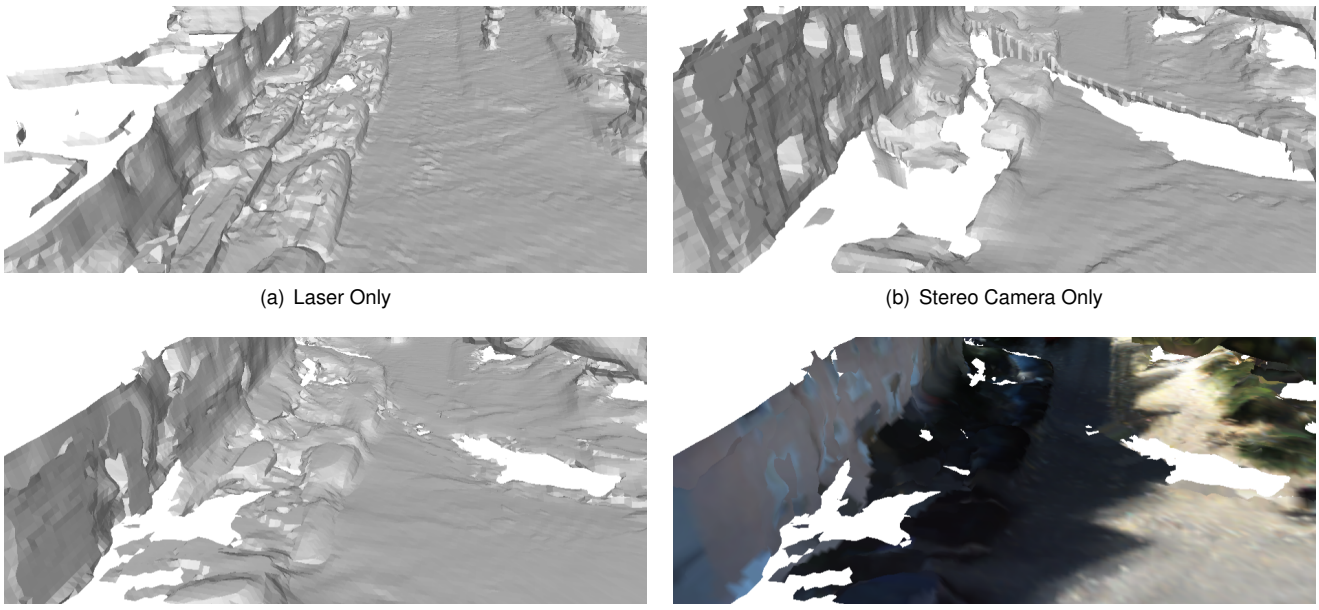


(b) KITTI-VO 05: Stereo Camera Reconstruction Samples



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

Figure 18. A few representative sample images for various points of view (offset from the original camera's position) along each trajectory. All sample images are of the final regularized reconstruction with 10 cm voxels using only stereo images as the input.



(a) Laser Only (b) Stereo Camera Only

(c) Corrected Stereo Camera (d) Final Colored Reconstruction

Figure 19. Comparison of reconstruction quality with laser only, stereo-camera only, and stereo-camera after CNN Correction. The correction step results in meshes with more coverage, though some areas (such as behind the parked cars) are occluded from the camera and therefore are not be corrected. A less obvious fix can be noticed in the road, where the uneven surface in (b), due to the shadows in the original images, is smoothed in (c).

7.2 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 (Zhou and Koltun 2013). We performed a point-to-surface comparison between BOR^{2}G's (0.5 cm voxels) and Zhou and Koltun'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 2, our level of detail compares favorable with the reconstructions by Zhou and Koltun (2013). 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 regularization to create high-quality reconstructions. We are not aware of any other system which both operates at scale and accepts a variety of sensor inputs – hence the remaining quantitative analysis compares only against ground-truth data.

7.3 Oxford Broad Street Dataset

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 paper 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 vs. stereo camera vs. Velodyne vs. 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 13).

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

- 1x Point Grey Bumblebee XB3 Stereo Camera (Color)
- 1x Point Grey Bumblebee2 Stereo Camera (Grayscale)
- 4x Point Grey Grasshopper2 Monocular Cameras (Color, 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:

- Optimized $\mathbb{SE}(3)$ 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 paper – for the Bumblebee XB3 to enable users to rapidly utilize 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 1 and a larger qualitative analysis is included in the Supplemental Material video.

7.4 KITTI Stereo-Only Reconstruction Analysis

In addition to traditional qualitative analysis, KITTI stereo-only reconstructions may be quantitatively analyzed by using laser data as ground truth. 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 meters 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 centimeters.

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

7.4.1 SDF vs. Histogram Data Terms Zach (2008) demonstrated the superior histogram data term performance in small-scale, object-centered scenarios. We compare SDF and histogram optimization 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 14. The 2D TGV^2 regularizer significantly reduced the noise in the input depth maps, but passively-generated depth maps inherently must infer depth of large portions of the image.

Both the SDF and Histogram optimizers smooth the noisy surfaces (e.g., buildings) in the reconstruction, but the SDF regularizer subjectively appears to do a better job. When compared against ground-truth in Table 6, the SDF and histogram optimizers 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 regularizer 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.

7.4.2 Multiple Passes In theory, revisiting a region should result in a more complete and accurate reconstruction, assuming one has accurate localization and loop closures. Noisy depth maps from the stereo camera preclude the traditional point-cloud alignment 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 15, 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).

7.4.3 Full-Length KITTI-VO Error Metrics 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 regularization, 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 utilizing bidirectional GPU-CPU streaming.

As shown in Table 7, the SDF regularizer 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 17, 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 regularizer removes many of these surfaces, which 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 7. However, even though the spatial resolution was reduced by a factor of two (and memory 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) which, 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 18 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 camera viewpoints offset in both translation and rotation to the original stereo camera position, thereby providing a depiction of the 3D structure. Overall, the reconstructions are quite visually appealing; however, some artifacts 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.

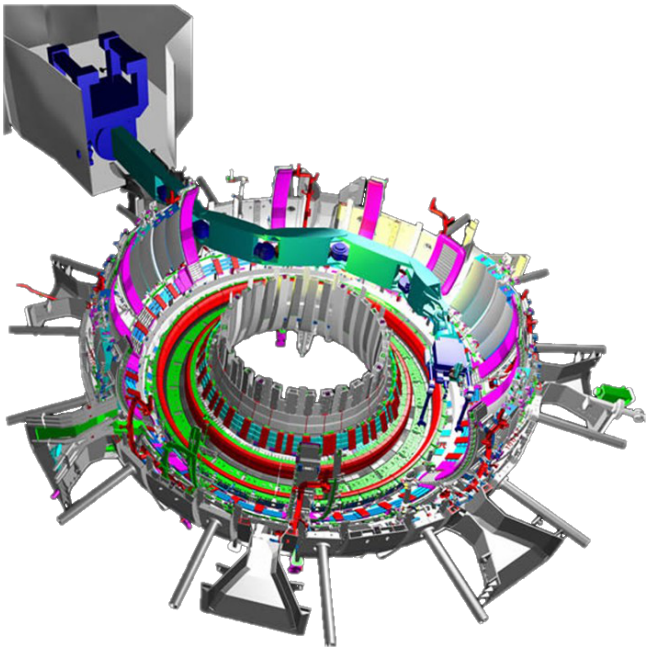


Figure 20. Schematic view of the Joint European Torus (JET) fusion device, showing the Remote Handling systems (Boom and Mascot) deployed inside the torus. Image: EFDA-JET.

7.5 KITTI Multi-Sensor Reconstruction Analysis

Many mobile platforms have a variety of different sensors, yet conventionally much dense reconstruction work is isolated to a single sensor, and almost exclusively a camera of some sort (RGB-D, monocular, or stereo). The KITTI dataset provides both camera images and Velodyne laser scans, so we decided to see what quality of reconstructions could be achieved by fusing data from *both* sensors.

A snapshot of our qualitative results are shown in Figure 16, but the supplemental video provides a fly-through of each sequence to visualize the quality of our final regularized 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 which could only reconstruct the bottom 2.5 m of objects. However, the laser-only reconstruction was much more detailed and could see surfaces which 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 16c shows the combination of both sensors produced a reconstruction that was higher quality and more comprehensive. The comprehensiveness is measured in Table 8, where the multi-sensor fused reconstruction has 40% to 50% more surface area coverage than the best sensor was able to reconstruct by itself.

7.6 Mesh Correction with CNNs

Correcting meshes post-hoc is particularly useful when dealing with reconstructions obtained in mobile robotics scenarios, where the amount of data is small relative to the surface area of the reconstruction.

Table 9. Parameters for CNN Training

Parameter	Value	Notes
Batch size	14	
Training epochs	400	
Learning rate	10^{-4}; 10^{-5}	10^{-4} for the first 300 epochs
w_α	10^{-6}	L_2 weight decay factor
w_β	1	berHu loss factor
w_γ	1	smoothnes loss factor
w_{min}	0.1	loss scaling factor for pixels farthest from an edge
w_{max}	5	loss scaling factor for edge pixels

Table 10. Summary of Surface Area Coverage with Learnt Prior

Dataset	Image Only	Image & Laser	Corrected Image Only
KITTI-VO 00	93,546 m^2	176,624 m^2	123,625 m^2
KITTI-VO 05	55,909 m^2	111,188 m^2	71,453 m^2
KITTI-VO 06	23,585 m^2	41,401 m^2	26,835 m^2

In our experiments, we used a regularized laser reconstruction with 20 cm^3 voxels as our high-quality prior, along with 20 cm^3 regularized stereo-camera-only reconstructions to train the CNN described in Section 6. We generate mesh features along the original trajectories from each of the three KITTI sequences (00, 05, and 06), and train three models, one on each pair of sequences. When evaluating our approach on a sequence, we use the model trained on the other two sequences.

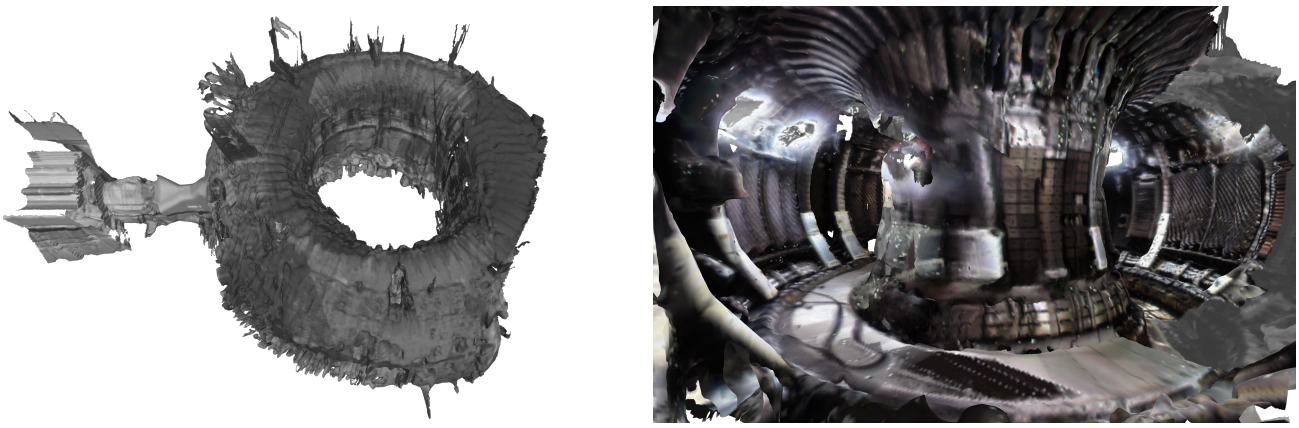
The network module was implemented in Python using the TensorFlow framework. The network weights were optimized using the ADAM solver (Kingma and Ba 2015) with a learning rate of 10^{-4} for 300 epochs over two of the sequences. At this point, the value of the loss function appears converged with variance coming form the stochasticity of the mini-batch sampling. To reduce this variance and confirm convergence, the optimization is run a further 100 epochs with the learning rate reduced to 10^{-5}. All training parameters are documented in Table 9.

The reconstructions obtained by fusing laser and stereo-camera data are the most complete we can obtain given the sensor setup. As shown in Table 10, our learnt correction closes the coverage gap between the stereo-camera reconstruction and the combined reconstruction by 18–36%, without increasing the metric error.

7.7 EUROfusion Joint European Torus Reconstruction

We also demonstrate a very different practical application of our work at the Joint European Torus (JET), which is operated by the UK Atomic Energy Authority (UKAEA) under contract from the European Commission, and exploited by the EUROfusion consortium of European Fusion Laboratories[1].

Together with the UKAEA remote handling division, RACE (Remote Applications in Challenging Environments), an inspection was carried out inside the JET device using the remote handing system (Figure 20), in order to collect lidar and camera data. Our inspection platform (the "NABU") is equipped with two push-broom lidars that are used in the



(a) Outside View of JET (Laser Only)

(b) Inside View of JET (Laser and Stereo Camera)

Figure 21. Two reconstructions of the Joint European Torus (JET) fusion device. This is a particularly challenging scenario, due to the many reflective surfaces inside the reactor vessel. The laser scans only covered about 75% of the torus, as well as part of the Tile Carrier Transfer Facility, shown on the far left in (a), used to gain access to the reactor vessel. In (b) we show a view inside JET from a laser and stereo camera reconstruction. Note that in the bottom-right of the image, the reconstruction does not have color information, since that area was never in view of the camera. Instead, laser intensity is used to color that portion of the reconstruction.

reconstruction (2.5 cm voxels), and a stereo camera used for visual odometry. We ran our pipeline on some of the data obtained, and Figure 21 shows the reconstruction. The recovered surface has a 287 m^2 area.

The inside of the JET reactor vessel contains many reflective surfaces, which induce considerable noise in the laser measurements. This kind of scenario is particularly challenging, and benefits from our fusion and regularization components.

8 Conclusion

We presented our state-of-the-art BOR^{2}G dense mapping system for large-scale dense reconstructions. When compared to object-centered reconstructions, mobile-robotics applications have 3,100 times fewer depth observations per square meter, thus we utilized regularizers in both 2D and 3D to serve as priors which improve the reconstruction quality. We overcame the primary technical challenge of regularizing 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 regularization and prevents the regularizer from erroneously extrapolating surface data.

We know of no other dense reconstruction system which 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 quality of our reconstructions, for different granularities, against 7.3 km of stereo-camera-only reconstructions. Our regularizer 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 which are both higher quality and more comprehensive than either sensor achieves independently. We demonstrated our system on a practical inspection task by producing a reconstruction of the EUROfusion Joint European Torus.

We showed how a CNN can be trained on historical high-fidelity data to learn a prior that can correct gross errors in meshes, such as holes in the road or cars. We overcome the challenge of processing meshes with CNNs by rasterizing appearance and geometric features and applying 2D techniques. The learnt CNN prior closes the gap in coverage by 18–36% without increasing the metric error.

Finally, we hope the release of our ground-truth KITTI pointclouds, ground-truth KITTI 3D models, and the comprehensive Oxford Broad Street dataset will become helpful tools of comparison for the community.

Supplemental material

- Oxford Broad Street dataset: http://ori.ox.ac.uk/dense-reconstruction-dataset/
- Stanford, Oxford, and KITTI reconstructions video: https://youtu.be/QHrQUIxPwOs
- Ground truth trajectories and Velodyne pointclouds
- Final colored dense reconstruction models

Notes

1. https://www.euro-fusion.org/

Acknowledgements

The authors would like to acknowledge the support of the UKs Engineering and Physical Sciences Research Council (EPSRC) through the Centre for Doctoral Training in Autonomous Intelligent Machines and Systems (AIMS) Programme Grant EP/L015897/1, Programme Grant EP/M019918/1, and the Doctoral Training Award (DTA). Additionally, the donation from Nvidia of a Titan Xp GPU used in this work is also gratefully acknowledged.

Part of this work has been carried out within the framework of the Contract for the Operation of the JET Facilities and has received funding from the European Union's Horizon 2020 research and

innovation programme. The views and opinions expressed herein do not necessarily reflect those of the European Commission.

References

Agarwal S, Snavely N, Simon I, Seitz SM and Szeliski R (2009) Building rome in a day. In: *Twelfth IEEE International Conference on Computer Vision (ICCV 2009)*. Kyoto, Japan: IEEE.

Amanatides J, Woo A and others (1987) A fast voxel traversal algorithm for ray tracing. In: *Eurographics*, volume 87. p. 10. 00652.

Anguelov D, Dulong C, Filip D, Frueh C, Lafon S, Lyon R, Ogale A, Vincent L and Weaver J (2010) Google street view: Capturing the world at street level. *Computer* 43(6): 32–38.

Bok Y, Choi DG and Kweon I (2014) Sensor Fusion of Cameras and a Laser for City-Scale3D Reconstruction. *Sensors* 14(11): 20882–20909. DOI:10.3390/s141120882.

Boyd S and Vandenberghe L (2004) *Convex optimization*. Cambridge university press.

Bredies K, Kunisch K and Pock T (2010) Total Generalized Variation. *SIAM Journal on Imaging Sciences* 3(3): 492–526. DOI:10.1137/090769521. 00428.

Canny JF (1986) A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 8: 679–698.

Chambolle A and Pock T (2011) A First-Order Primal-Dual Algorithm for Convex Problems with Applications to Imaging. *J. Math. Imaging Vis.* 40(1): 120–145.

Chen J, Bautembach D and Izadi S (2013) Scalable Real-time Volumetric Surface Reconstruction. *ACM Trans. Graph.* 32(4): 113:1–113:16. DOI:10.1145/2461912.2461940.

Choi S, Zhou QY and Koltun V (2015) Robust reconstruction of indoor scenes. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. pp. 5556–5565. DOI: 10.1109/CVPR.2015.7299195.

Curless B and Levoy M (1996) A volumetric method for building complex models from range images. In: *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. ACM, pp. 303–312. 02090.

Engel J, Stueckler J and Cremers D (2015) Large-scale direct slam with stereo cameras. In: *International Conference on Intelligent Robots and Systems (IROS)*.

Felzenszwalb P and Huttenlocher D (2004) Distance transforms of sampled functions. Technical report, Cornell University.

Furukawa Y, Curless B, Seitz SM and Szeliski R (2010) 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*. pp. 1434–1441.

Geiger A, Lenz P and Urtasun R (2012) Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite. In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 00627.

Godard C, Aodha OM and Brostow GJ (2017) Unsupervised monocular depth estimation with left-right consistency. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. pp. 6602–6611.

Goldluecke B, Strekalovskiy E and Cremers D (2012) The natural vectorial total variation which arises from geometric measure theory. *SIAM Journal on Imaging Sciences* 5(2): 537–563.

Güney F and Geiger A (2015) Displets: Resolving stereo ambiguities using object knowledge. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. pp. 4165–4175.

Handa A, Newcombe RA, Angeli A and Davison AJ (2011) Applications of Legendre-Fenchel transformation to computer vision problems. Technical Report DTR11-7, Imperial College - Department of Computing.

Handa A, Whelan T, McDonald JB and Davison AJ (2014) A Benchmark for RGB-D Visual Odometry, 3D Reconstruction and SLAM. In: *IEEE Intl. Conf. on Robotics and Automation, ICRA*. Hong Kong, China.

He K, Zhang X, Ren S and Sun J (2016) Learning for image recognition. In: *Computer Vision and Pattern Recognition (CVPR)*. ISBN 978-1-4673-8851-1. DOI:10.1109/CVPR.2016.90.

Heise P, Klose S, Jensen B and Knoll A (2013) PM-Huber: PatchMatch with Huber regularization for stereo matching. In: *IEEE International Conference on Computer Vision (ICCV)*. pp. 2360–2367.

Kingma D and Ba J (2015) Adam: A method for stochastic optimization. In: *International Conference on Learning Representations (ICLR)*.

Klingensmith M, Dryanovski I, Srinivasa S and Xiao J (2015) Chisel: Real time large scale 3d reconstruction onboard a mobile device. In: *Robotics Science and Systems 2015*.

Klingner B, Martin D and Roseborough J (2013) Street view motion-from-structure-from-motion. In: *Proceedings of the IEEE International Conference on Computer Vision*. pp. 953–960.

Laina I, Rupprecht C, Belagiannis V, Tombari F and Navab N (2016) Deeper depth prediction with fully convolutional residual networks. In: *International Conference on 3D Vision*. IEEE, pp. 239–248.

Lorensen WE and Cline HE (1987) 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. ISBN 0-89791-227-6, pp. 163–169.

Maddern W, Stewart A and Newman P (2014) LAPS-II: 6-DoF day and night visual localisation with prior 3D structure for autonomous road vehicles. In: *2014 IEEE Intelligent Vehicles Symposium Proceedings*. pp. 330–337. DOI:10.1109/IVS.2014.6856471.

Mahjourian R, Wicke M and Angelova A (2018) Unsupervised learning of depth and ego-motion from monocular video using 3d geometric constraints. ArXiv:1802.05522.

Mur-Artal MJMM Raúl and Tardós JD (2015) ORB-SLAM: a versatile and accurate monocular SLAM system. *IEEE Transactions on Robotics* 31(5): 1147–1163. DOI:10.1109/TRO.2015.2463671.

Newcombe RA, Davison AJ, Izadi S, Kohli P, Hilliges O, Shotton J, Molyneaux D, Hodges S, Kim D and Fitzgibbon A (2011a) KinectFusion: Real-time dense surface mapping and tracking. In: *International Symposium on Mixed and Augmented Reality (ISMAR)*. IEEE, pp. 127–136.

Newcombe RA, Lovegrove SJ and Davison AJ (2011b) DTAM: Dense tracking and mapping in real-time. In: *International Conference on Computer Vision (ICCV)*. IEEE, pp. 2320–2327.

Nießner M, Zollhöfer M, Izadi S and Stamminger M (2013) Real-time 3D Reconstruction at Scale Using Voxel Hashing. *ACM Trans. Graph.* 32(6): 169:1–169:11. 00050.

Nocedal J and Wright SJ (2006) *Numerical optimization*. Springer Series in Operations Research and Financial Engineering. Berlin: Springer.

Owen AB (2007) A robust hybrid of lasso and ridge regression. *Contemporary Mathematics* 443: 59–72.

Piniés P, Paz LM and Newman P (2015) 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)*. Seattle, WA, USA.

Pock T and Chambolle A (2011) Diagonal preconditioning for first order primal-dual algorithms in convex optimization. In: *Computer Vision (ICCV), 2011 IEEE International Conference On*. IEEE, pp. 1762–1769. 00142.

Pollefeys M, Nistér D, Frahm JM, Akbarzadeh A, Mordohai P, Clipp B, Engels C, Gallup D, Kim SJ, Merrell P, Salmi C, Sinha S, Talton B, Wang L, Yang Q, Stewénius H, Yang R, Welch G and Towles H (2008) Detailed real-time urban 3d reconstruction from video. *International Journal of Computer Vision* 78(2-3): 143–167.

Ranftl R, Gehrig S, Pock T and Bischof H (2012) Pushing the limits of stereo using variational stereo estimation. In: *Intelligent Vehicles Symposium (IV), 2012 IEEE*. IEEE, pp. 401–407.

Riegler G, Ulusoy AO and Geiger A (2017) Octnet: Learning deep 3d representations at high resolutions. In: *Conference on Computer Vision and Pattern Recognition (CVPR)*.

Rockafellar RT (1970) *Convex Analysis*. Princeton, New Jersey: Princeton University Press.

Ronneberger O, Fischer P and Brox T (2015) U-net: Convolutional networks for biomedical image segmentation. In: *Medical Image Computing and Computer Assisted Intervention (MICCAI)*.

Scharstein D and Szeliski R (2002) A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International journal of computer vision* 47(1-3): 7–42.

Schöps T, Sattler T, Häne C and Pollefeys M (2015) 3d modeling on the go: Interactive 3d reconstruction of large-scale scenes on mobile devices. In: *3D Vision (3DV), 2015 International Conference On*. IEEE, pp. 291–299.

Shang W, Sohn K, Almeida D and Lee H (2016) Understanding and Improving Convolutional Neural Networks via Concatenated Rectified Linear Units. In: *International Conference on Machine Learning (ICML)*. pp. 2217–2225.

Stewart A and Newman P (2012) 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)*. Minnesota, USA.

Tanner M, Piniés P, Paz LM and Newman P (2015) BOR2G: Building Optimal Regularised Reconstructions with GPUs (in cubes). In: *International Conference on Field and Service Robotics (FSR)*. Toronto, ON, Canada.

Tanner M, Piniés P, Paz LM and Newman P (2016) What Lies Behind: Recovering Hidden Shape in Dense Mapping. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. Stockholm, Sweden.

Tanner M, Săftescu Ș, Bewley A and Newman P (2018) Meshed up: Learnt error correction in 3D reconstructions. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. Brisbane, Australia.

Whelan T, Kaess M, Fallon MF, Johannsson H, Leonard JJ and McDonald JB (2012) Kintinuous: Spatially Extended KinectFusion. In: *RSS Workshop on RGB-D: Advanced Reasoning with Depth Cameras*. Sydney, Australia.

Whelan T, Kaess M, Johannsson H, Fallon M, Leonard JJ and McDonald J (2014) Real-time large-scale dense RGB-D SLAM with volumetric fusion. *The International Journal of Robotics Research* 34: 598–626.

Xiao J and Furukawa Y (2014) Reconstructing the world's museums. *International journal of computer vision* 110(3): 243–258.

Zabih R and Woodfill J (1994) Non-parametric local transforms for computing visual correspondence. In: *Computer Vision - ECCV'94, Third European Conference on Computer Vision, Stockholm, Sweden, May 2-6, 1994, Proceedings, Volume II*. pp. 151–158.

Zach C (2008) Fast and high quality fusion of depth maps. In: *Proceedings of the international symposium on 3D data processing, visualization and transmission (3DPVT)*, volume 1.

Zach C, Pock T and Bischof H (2007) A globally optimal algorithm for robust TV-L 1 range image integration. In: *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*. IEEE, pp. 1–8. 00142.

Zhou QY and Koltun V (2013) Dense Scene Reconstruction with Points of Interest. *ACM Trans. Graph.* 32(4): 112:1–112:8. DOI:10.1145/2461912.2461919.

Statement of Authorship for joint/multi-authored papers for PGR thesis

To appear at the end of each thesis chapter submitted as an article/paper

The statement shall describe the candidate's and co-authors' independent research contributions in the thesis publications. For each publication there should exist a complete statement that is to be filled out and signed by the candidate and supervisor **(only required where there isn't already a statement of contribution within the paper itself).**

Title of Paper	Large scale outdoor scene reconstruction and correction with vision
Publication Status	☒ Published ☐ Accepted for Publication ☐ Submitted for Publication ☐ Unpublished and unsubmitted work written in a manuscript style
Publication Details	Tanner, M., Piniés, P., Paz, L.M., **Săftescu, Ș.**, Bewley, A., Jonasson,E., & Newman, P. (2020). Large scale outdoor scene reconstruction and correction with vision. *The International Journal of Robotics Research (IJRR)*.

Student Confirmation

Student Name:	Ștefan Săftescu		

Contribution to the Paper	• Experiments and write-up regarding mesh *correction*
	  ○ Section 6
	  ○ Section 7.6
	• EUROfusion Joint European Torus Reconstruction (qualitative results, Section 7.7)
	• Editing, formatting, revising (corresponding author)		

Signature	Ștăftescu	Date	29/08/2020

Supervisor Confirmation

By signing the Statement of Authorship, you are certifying that the candidate made a substantial contribution to the publication, and that the description described above is accurate.

Supervisor name and title:	Professor Paul Newman		
Supervisor comments			
Signature	P. Newman	Date	10/09/2020

This completed form should be included in the thesis, at the end of the relevant chapter.

5.1 Discussion

At the beginning of this chapter, we sought to answer whether correcting inverse-depth views of a mesh results in improved meshes. We concluded that our method improves coverage, recovering up to 36% of the missing surface, without changing metric error. Since the meshes we correct are built from stereo-camera depth estimates, this is expected. Most of the error in depth estimation is due to poor texture or occlusion, and this results primarily in missing rather than incorrect surface.

A note on evaluation

As the article in this chapter is concerned with 3D reconstruction as a whole, rather than only mesh correction, the 3D metrics for the KITTI sequences are computed with respect to full lidar *pointclouds*. This differs from the evaluation proposed in Section 2.4.1, where the reference is lidar *meshes*. We take this approach here since lidar meshes are used only for supervising the CNN and not for other components of the reconstruction system. In practice, this results in the following distinctions:

1. Errors are computed from mesh vertices to the nearest neighbours in the pointcloud. This roughly corresponds to the correctness direction mentioned in Section 2.4.1.

2. The density of vertices used for comparison, as well as the density of points in the reference pointcloud are not uniform. Areas of the mesh where triangles are smaller will have more points and therefore weigh more.

A consequence of this difference is that the error metrics from this chapter are not directly comparable to the ones in Chapters 6 and 7. However, we will directly compare the CNN model employed in this chapter with our other contributions in Chapter 7.

5.2 Conclusion

This chapter and Chapter 4 have considered a mesh correction network that takes as input mesh features and matching RGB images taken with a real camera, and produces corrected inverse-depth images. When used as part of a 3D reconstruction pipeline, as a mesh post-processing step, this improves the completeness of the meshes.

Using real colour images as input can be useful, since they may contain information that has been discarded by the 3D reconstruction pipeline, and indeed colour images can be a simple source of additional information (e.g. boundaries in images often correspond to boundaries in depth, roads and buildings are less colourful than cars). As shown in the ablation study of Chapter 4, RGB images bring a small but noticeable contribution to the performance of our method.

However *real* colour images are limited in quantity which places an upper bound on how much data can be used for training. For example, the KITTI Odometry Dataset we use for our experiments provides stereo images at 10 Hz, or approximately 1 frame/m. In the next chapter, we will remove the dependency on real colour images, replacing them with synthetic ones rendered along with the other mesh features.

6

Learning Geometrically Consistent Mesh Correction

In Chapter 5, we introduced our method for correcting meshes by synthesising and correcting 2D views of a 3D reconstruction. In addition to the synthesised mesh features, we used corresponding real colour images captured along with the rest of the data. These images are limited in quantity which places an upper bound on how much data can be generated for training.

To enable synthesising arbitrarily many views – even entirely novel ones such as a top-down view – in this chapter we replace live colour images with synthetic ones rendered from the mesh. Since BOR²G, our 3D reconstruction system, does not currently implement texturing of meshes, and only stores colour information per vertex, the quality of the synthetic images is limited by the resolution of the mesh. Furthermore, errors in the reconstruction used to render the images will also be present, unlike in live images (see Figure 6.1). These differences, however, have a negligible effect on performance (less than 1% relative difference in iRMSE) and are mitigated by the ability to generate much more data.



(a) Real KITTI Image (b) Colour Rendering from a 3D Mesh

Figure 6.1: Comparison between an image captured by a camera, and a synthetic colour image from the same viewpoint, rendered from one of our 3D meshes. The quality of the rendered image is limited by the resolution of the mesh, and by the accuracy of the surface present in the mesh. For example, the pole supporting the sign on the right side of (a) is not present in (b). Nevertheless, we have found this to have minimal impact on our metrics, and the differences are offset by the ability to generate any number of synthetic views.

The following manuscript is a reproduction of the paper presented at the International Conference on Computer Vision Theory and Applications (VISAPP), 2020. The paper was nominated for the best student paper award.

Learning Geometrically Consistent Mesh Corrections

Ștefan Săftescu and Paul Newman
Mobile Robotics Group, Oxford Robotics Institute, University of Oxford
{*stefan,pnewman*}*@robots.ox.ac.uk*

Keywords: 3D reconstruction, depth refinement, deep learning

Abstract: Building good 3D maps is a challenging and expensive task, which requires high-quality sensors and careful, time-consuming scanning. We seek to reduce the cost of building good reconstructions by correcting views of existing low-quality ones in a post-hoc fashion using learnt priors over surfaces and appearance. We train a convolutional neural network model to predict the difference in inverse-depth from varying viewpoints of two meshes – one of low quality that we wish to correct, and one of high-quality that we use as a reference.
In contrast to previous work, we pay attention to the problem of excessive smoothing in corrected meshes. We address this with a suitable network architecture, and introduce a loss-weighting mechanism that emphasises edges in the prediction. Furthermore, smooth predictions result in geometrical inconsistencies. To deal with this issue, we present a loss function which penalises re-projection differences that are not due to occlusions. Our model reduces gross errors by 45.3%–77.5%, up to five times more than previous work.

1 Introduction

Dense 3D maps are a crucial component in many systems and better maps make robots easier to build and safer to operate. Despite recent progress in hardware such as the wide availability of GPUs, and algorithms that scale with the amount of data and the available hardware, high-quality maps, especially at large scales and outdoors, remain difficult to build cheaply, often requiring expensive sensors such as 3D lidars, and careful, dense scans.

The main motivation of our work is to reduce the cost of building good dense 3D maps. The reduction in cost can come from either: a) cheaper but nosier sensors, such as stereo cameras; b) less data, and therefore less time spent densely scanning an area. There are two ways in which we can produce better reconstructions with cheaper data. We can learn the kinds of errors a certain modality produces. For a stereo camera, for example, there will be missing data in areas without a lot of texture (walls, roads), and the ambiguity in depth is usually along the viewing rays. In addition, we can learn priors for a target environment: cars usually have known shapes, roads and buildings do not have holes in them, surfaces tend to be vertical or horizontal in an urban environment, etc.

We tackle the problem of correcting dense reconstructions with a convolutional neural network (CNN) that operates on rasterised views of a 3D mesh as a post-processing step, following a classical reconstruction pipeline. To train this model, we start with two meshes, a low-quality one and a reference high-quality

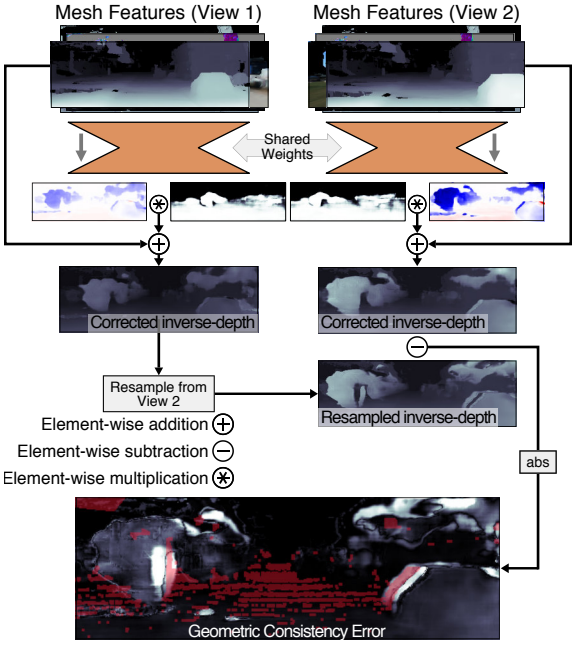


Figure 1: Illustration of geometric consistency for two views of the same scene a few meters apart. The predictions of our model can be used to compute corrected depth maps for a set of views. The corrected depth-maps should be consistent: they should be describing the same scene. To enforce this, we densely reproject inverse-depth from View 1 to View 2, using the corrected depth of View 2. The absolute difference (bottom) is the inconsistency of View 1 with respect to View 2. This is minimised during the training of our model, which enforces geometrically consistent predictions. The red overlay is the occlusion mask: the parts of the View 2 prediction that cannot be consistent with the View 1 prediction.

one. From each reconstruction, we render multiple types of images (such as inverse-depth, normals, etc.), referred to as *mesh features*, from multiple viewpoints. We then train the model on the mesh features to predict the difference in inverse-depth between the high-quality reconstruction and the low-quality one, thus enabling us to correct the low-quality mesh.

Previous work (Tanner et al., 2018) has demonstrated the idea of correcting meshes post-hoc via 2D rasterised views. We address its two main limitations. Firstly, we deal with the issue of overly smooth predictions. We propose changes to the architecture and training more suitable for our task: we add skip connections from the encoder to the decoder in our CNN, which are known to help in localising edges in predictions; we propose a loss-weighting method that penalises incorrect predictions more the closer they are to an edge. Secondly, predictions on nearby views are not always consistent, i.e. when applying the predicted corrections the geometry of the scene is not always the same. To improve consistency, we employ a view synthesis based loss, and show that this also improves the performance of the network. An illustration of this idea is shown in Figure 1.

Our contributions are as follows:

Error correction: We propose a CNN model that is able to correct 2D views of a dense 3D reconstruction. We propose a novel weighting mechanism to improve performance around edges in the prediction. We evaluate it against existing work and show that we outperform it, especially when images are from multiple viewpoints.

Geometric consistency: We adapt the photometric consistency loss that features in related tasks such as depth-from-mono (Godard et al., 2017) to the task of correcting reconstructions. This is a novel use of the loss, which has thus far only been employed on RGB images. We leverage the existing reconstruction to compute occlusion masks, and thus exclude from the loss areas where geometric consistency is impossible. We show that this use of geometric consistency as an auxiliary loss further improves our model.

2 Related Work

Several systems for building 3D reconstructions have been proposed, such as BOR^{2}G (Tanner et al., 2015) or KinectFusion (Newcombe et al., 2011). Our approach is to correct the *output* of such a system by looking at the meshes it builds. As we operate on inverse-depth images, our work is similar to depth refinement. Below, we review some of the literature on that topic, as well as some other methods that our system draws inspiration from.

Filtering and optimisation of depth

Improving depth or disparity images is a well-researched area, with various methods proposed, some of which are guided by an additional aligned colour image. The most straightforward unguided techniques are based on minimising either on a joint bilateral filter (JBF) (Tomasi and Manduchi, 1998) when filtering, or Total Generalised Variation (TGV) (Bredies et al., 2010) for optimisation. Guided techniques use another image aligned with the depth-map to inform the refinement. The guide image is used especially around edges, to modulate the smoothing effect of filtering or optimisation. For example, Kopf et al. (2007) or Matsuo et al. (2013) propose variants of a JBF for refining depth, while Tanner et al. (2016) demonstrates a TGV-based method that uses both magnitude and direction of colour images to regularise depth maps. An interesting combination of learning and optimisation is proposed by Riegler et al. (2016). The Primal-Dual algorithm (Chambolle and Pock, 2011) normally employed for TGV optimisation is unrolled and included into a CNN model that learns to upsample depth images. Our work is relies on an existing 3D reconstruction pipeline (Tanner et al., 2015), in which some of these depth refinement techniques have already been employed. We focus on post-processing meshes and fixing errors that cannot easily be fixed in single, live depth images.

Learnt depth refinement and completion

Some methods for *learning* depth map refinement and completion have been recently proposed. Eldesokey et al. (2018); Uhrig et al. (2017) propose CNN model for solving the KITTI Depth Completion Challenge, where sparse depth maps produced from laser data are densified. Hua and Gong (2018) use normalised convolutions (Knutsson and Westin, 1993) to predict dense depth maps that have been sparsely sampled. These methods all rely on having some sort of high-quality, usually laser, depth information (albeit sparse) at run-time, and are only designed to fill in the missing data. In contrast, we only require high-quality data during training, and our method aims to refine depth from a low-quality mesh in a more general sense – both filling in blanks, as well as refining existing surfaces.

A few other methods more closely related in spirit to ours aim to learn depth refinement using meshes as reference. Kwon et al. (2015) use dictionary learning to model the statistical relationships between raw RGB-D images, and high-quality depth data obtained by

fusing multiple depth maps with KinectFusion (Newcombe et al., 2011). Two other recent works use fused RGB-D reconstruction to obtain high-quality reference data and train CNNs to enhance depth. Zhang and Funkhouser (2018) uses a colour image to predict normals and occlusion boundaries, supervised by the 3D reconstruction, and then formulates an optimisation problem to fill in holes in an aligned depth image. Jeon and Lee (2018) introduce a 4000-image dataset of raw/clean depth image pairs, and train a CNN to enhance raw depth maps. Furthermore, they show that 3D reconstructions can be obtained with fewer data and quicker when using their depth-enhancing network. These methods all rely on live colour images to guide the refinement or completion of live depth. That means that only limited data is available for training. In contrast, our purely mesh-based formulation allows us to extract many more training pairs from any viewpoint, removing any viewpoint-specific bias that might otherwise surface while learning.

Learnt depth estimation

Another related line of research has been monocular depth estimation, which inspires our choices of network architecture and the use of geometry as a source of self-supervision. Dharmasiri et al. (2019); Godard et al. (2017); Klodt and Vedaldi (2018); Laina et al. (2016); Mahjourian et al. (2018); Ummenhofer et al. (2017); Zhou et al. (2017), etc. propose various CNN models for depth estimation from single colour images. When explicit depth ground-truth is unavailable, the training is self-supervised using multiview geometry: the predicted depths and relative poses should maximise the photometric consistency of nearby input images. In this work, we show how the photometric consistency loss can be adapted to ensure consistency between inverse-depth predictions directly, without need for colour images.

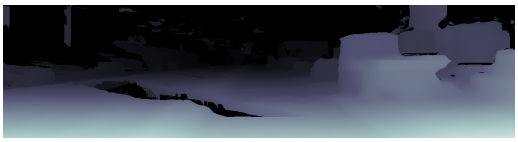
3 Method

3.1 Training data

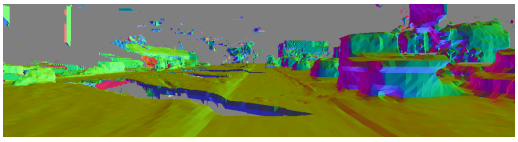
In contrast to many existing approaches, this work is about correcting *existing* reconstructions, and therefore we assume the existence of 3D meshes of a scene. In addition to the low-quality reconstruction we wish to correct, we also have a high-quality reconstruction of the same scene. For example, we learn to correct 3D reconstructions from depth-maps using laser reconstructions as a reference. To build the meshes we train on, we use the BOR^{2}G (Tanner et al., 2015) system.



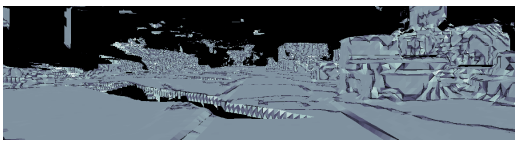
(a) Colour Reconstruction



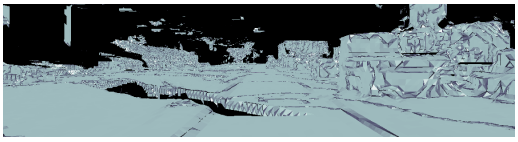
(b) Inverse-depth



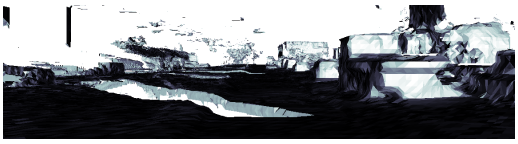
(c) Triangle Surface Normal



(d) Triangle Area



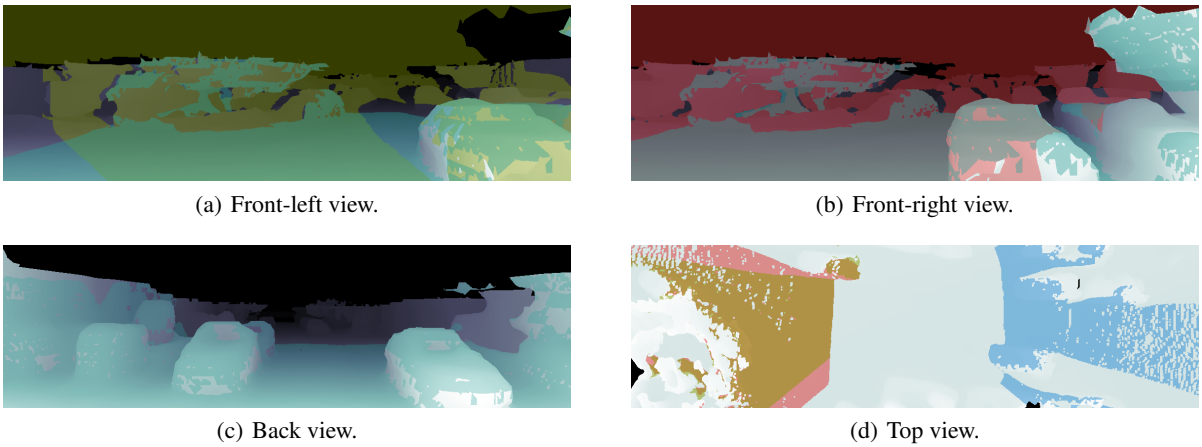
(e) Triangle Edge Length Ratios



(f) Surface to Camera Angle

Figure 2: Example mesh features. We fly a camera through an existing mesh and at each location produce mesh features. During training, inverse-depth images of a high-quality mesh are available, and our model learns a mapping from the mesh features pictured above to the high-quality inverse-depth.

The training data consists of 2D views of the meshes. We use the same virtual camera to generate aligned views of the low-quality mesh and the high-quality mesh for each scene. Because a mesh is available, we can generate multiple types of images at each viewpoint. In particular, we generate inverse-depth, normals, mesh triangle area, mesh triangle edge ratio (ratio between the shortest and the longest edge of each triangle), and surface-to-camera angle (Figure 2). We refer to these images as *mesh features*. The



(a) Front-left view.

(b) Front-right view.

(c) Back view.

(d) Top view.

Figure 3: Visibility masks for views generated at a location. The image regions highlighted in red, green, blue, and cyan are also visible in the (a) left, (b) right, (c) back, and (d) top views, respectively. (a): inverse-depth view from a dense 3D reconstruction; the regions visible in the left, and top views are respectively highlighted in green and cyan. (b): a view of the same scene as (a), 2 m to the right; the regions visible in the left, and top views are respectively highlighted in red and cyan. (c): a view of the same scene as (a), looking back; the region visible in top view is highlighted in cyan. (d): a view of the same scene as (a), looking down from 25 m above; the regions visible in the left, right, and back views are respectively highlighted in red, green, and blue. Areas outside the highlighted regions are occluded – they are not visible from the other view. For example, distant areas of (a)–(c), such as the sky, are not visible from the top view; the region behind the car on the right in (a) is not visible from (b). These occluded parts of the image are ignored when computing the geometric consistency loss (Section 3.3.2). For visualisation purposes, morphological closing has been applied to the visibility masks shown here, to remove some of the small-scale noise.

ground-truth labels (Δ) are computed as the difference in inverse-depth between high-quality and low-quality reconstructions:

$$\Delta(p) = d^{hq}(p) - d^{lq}(p) \tag{1}$$

where p is a pixel index, and d^{hq} and d^{lq} are inverse-depth images for the high-quality and low-quality reconstruction, respectively. For notational compactness, $\Delta(p)$ is referred to as Δ, and future definitions are over all values of p, unless otherwise mentioned.

Inverse-depth is used instead of depth for several reasons. Firstly, it emphasises surfaces close to the camera where more information is available per pixel. Secondly, background (non-surface) pixels are not processed separately – they are assigned a value of zero, corresponding to points infinitely far away from the camera. If we used depth, those pixels would either have to be assigned an arbitrary finite value, which would result in semantic discontinuities in the output, or learnt to be ignored by the network, since there is no standard way to deal with infinite values in a CNN. Finally, another advantage of inverse-depth is that resampling it, which is needed to compute geometric consistency, is simpler than resampling depth images.

3.2 Geometric consistency

Intuitively, since the reconstructions we wish to correct are static, the predictions made from overlapping views should be geometrically consistent. In other words, surfaces that appear in a certain location according to a prediction should appear in the same location in all predictions where they are in view.

We resample nearby predicted views according to the predicted geometry of the current view, and minimise the absolute difference in inverse-depth. This dense warping is similar to reprojecting nearby views into the current view, but has the advantage of being differentiable, and of generating dense images instead of sparse reprojected pointclouds. An illustration of this idea is shown in Figure 1.

Normally, dense warping is used in conjunction with colour images, where the values of the pixels are view-independent. Since we are warping inverse-depth images, where the pixel values depend on the viewpoint, we need to compute the absolute difference in the same camera frame.

Concretely, for a view t, a nearby view n, let Δ_t^* be a CNN prediction for the target view, let Δ_n^* be a prediction for the nearby view, and let $d_t^* = d_t^{lq} + \Delta_t^*$ and $d_n^* = d_n^{lq} + \Delta_n^*$ the corrected inverse-depth images for the two views. Furthermore, let $\mathbf{p}_t$ be pixel coordinates in the target view, K the intrinsic matrix of

the virtual camera, and $T_{n,t}$ the SE(3) transform from view t to view n. We can then define the geometric inconsistency $d^*_{n,t} - \widetilde{d}^*_{n,t}$ where $d^*_{n,t}$ is the predicted inverse-depth from view t in the frame of view n and $\widetilde{d}^*_{n,t}$ is the warped inverse-depth from view n.

They are defined as follows:

$$\widetilde{d}^*_{n,t}(\mathbf{p}_t) = d^*_n(\mathbf{p}_n), \tag{2}$$

$$d^*_{n,t}(\mathbf{p}_t) = \frac{\mathbf{x}^{(4)}_n}{\mathbf{x}^{(3)}_n + \varepsilon}, \tag{3}$$

where superscripts indicate vector elements, $\mathbf{x}_n$ is the 3D homogeneous point in view n corresponding to pixel $\mathbf{p}_t$, and $\mathbf{p}_n$ its projection:

$$\mathbf{p}_n = \frac{1}{\mathbf{x}^{(3)}_n + \varepsilon} \begin{pmatrix} \mathbf{x}^{(1)}_n & \mathbf{x}^{(2)}_n \end{pmatrix}^T, \tag{4}$$

$$\mathbf{x}_n = F_h \mathbf{x}_t, \tag{5}$$

$$\mathbf{x}_t = (\mathbf{p}_t \; 1 \; d^*_t(\mathbf{p}_t))^T, \tag{6}$$

$$F_h = K_h T_{n,t} K_h^{-1}, \tag{7}$$

$$K_h = \begin{pmatrix} K & \mathbf{0} \\ \mathbf{0}^T & 1 \end{pmatrix}. \tag{8}$$

The sample pixel $\mathbf{p}_n$ does not necessarily have integer coordinates, and thus may lie in-between pixels in the d^*_n grid. Note that we add a small value ε with the same sign as $\mathbf{x}^{(3)}_n$ in Equations 3 and 4 to avoid dividing by zero. Under the mild assumption that surfaces between pixels are planar, we can sample d^*_n by linearly interpolating the four pixels nearest to $\mathbf{p}_n$ – another advantage of the inverse-depth formulation.

Occlusion masks

A common problem with this approach is that views cannot be consistent in the presence of occlusions. In our setting, however, since the views are synthetic (and therefore the intrinsic and extrinsic parameters are perfectly known), we are able to create occlusion masks and only apply the geometric consistency loss where there are no occlusions, as noted in Equation 11.

We compute occlusion masks from the reference high-quality reconstruction. Each mesh triangle is assigned an index by hashing its world frame coordinates. Then, in addition to mesh features, an image with mesh triangle indices is generated at each location. For each pair of views, each pixel of the triangle index image is resampled from one view into the other in a similar fashion to the inverse-depth above. Instead of interpolating between the four nearest neighbours, these are returned as four separate samples. The pixels where the indices match at least one of the four samples are considered unoccluded, and the pixels where they do not are considered occluded (see Figure 3).

Small errors appear due to rasterisation when mesh triangles are too small, often those that are far away from the camera. We could avoid these errors by generating occlusion masks with the OpenGL rasterisation pipeline when the rest of the data is generated. However, this greatly increases the space required to store training data – the number of occlusion masks scales quadratically with the number of views we want to enforce consistency between. Generating the occlusion masks on the fly means that geometric consistency can be enforced between arbitrary views, so more settings can be explored without regenerating part of the training data.

3.3 Model

3.3.1 Network architecture

The model used is an encoder-decoder CNN similar to the one proposed in Tanner et al. (2018). The encoder is composed of residual blocks based on the ResNet-50 architecture (He et al., 2016), and the decoder uses up-convolutions proposed by Shi et al. (2016). U-Net Ronneberger et al. (2015) style skip connections are added between the encoder and the decoder to improve the sharpness of predictions. As a simple way to offer our model some introspective capabilities, we predict a soft attention mask (with values $\in [0,1]$) in addition to the error in inverse-depth. This mask is multiplied pixel-wise with the error prediction to modulate which parts are going to be used and does not require extra supervision. Table 1 provides an overview for each of the layers of the proposed CNN.

3.3.2 Loss

The objective function has several components, as follows. The first term is the data loss that minimises the error between the output and the label. To compute this loss, we use berHu norm (Owen, 2007). For large errors, this behaves in the same way as an L_2 norm. For small errors, where the gradients of L_2 become too small to drive the error completely to zero, L_1 norm is used instead. The advantages of this norm have also been observed in Laina et al. (2016); Ma and Karaman (2018). The data loss is defined as follows:

$$\mathcal{L}^{data} = \sum_{p \in V} \mathrm{W} \cdot \|\Delta^* - \Delta\|_{berHu}, \tag{9}$$

where p is the pixel index, V is the set of valid pixels (to account for missing data in the ground-truth), W is a per-pixel weight detailed in Section 3.3.3, Δ^* and Δ are the prediction and the target, respectively, and $\|\cdot\|_{berHu}$ is the berHu norm.

Table 1: Overview of the CNN architecture for error prediction

Block Type	Filter Size/Stride	Output Size
Input	-	$96 \times 288 \times F$
Convolution	$3 \times 3/1$	$96 \times 288 \times 64$
Residual	$3 \times 3/1$	$96 \times 288 \times 64$
Convolution	$5 \times 5/1$	$48 \times 144 \times 64$
Max Pool	$3 \times 3/2$	$24 \times 72 \times 64$
Residual$\times 2$	$3 \times 3/1$	$24 \times 72 \times 64$
Projection	$3 \times 3/2$	$12 \times 36 \times 256$
Residual$\times 2$	$3 \times 3/1$	$12 \times 36 \times 256$
Projection	$3 \times 3/2$	$6 \times 18 \times 512$
Residual$\times 2$	$3 \times 3/1$	$6 \times 18 \times 512$
Projection	$3 \times 3/2$	$3 \times 9 \times 1024$
Residual$\times 8$	$3 \times 3/2$	$3 \times 9 \times 2048$
Up-projection	$3 \times 3/\frac{1}{2}$	$6 \times 18 \times 1024$
Up-projection	$3 \times 3/\frac{1}{2}$	$12 \times 36 \times 512$
Up-projection	$3 \times 3/\frac{1}{2}$	$24 \times 72 \times 256$
Up-projection	$3 \times 3/\frac{1}{2}$	$48 \times 144 \times 128$
Up-projection	$3 \times 3/\frac{1}{2}$	$96 \times 288 \times 32$
Residual	$3 \times 3/1$	$96 \times 288 \times 32$
Convolution	$3 \times 3/1$	$96 \times 288 \times 2$

To improve small-scale details and prevent artefacts in the prediction, while also allowing for sharp discontinuities, we also apply a loss on the gradient of the predictions:

$$\mathcal{L}^{\nabla} = \frac{1}{2} \sum_{p \in V} \mathrm{W} \cdot \left(\left| \partial_x \Delta^* - \partial_x \Delta \right| + \left| \partial_y \Delta^* - \partial_y \Delta \right| \right). \quad (10)$$

We use the Sobel operator (Sobel and Feldman, 1968) to approximate the gradients in the equation above.

The geometric consistency loss guides nearby predictions to have the same 3D geometry, and relies on reprojected nearby views $\widetilde{d}^*_{n,t}$. For a target view t, a set of nearby views N, the set of pixels unoccluded in a nearby view U_n (see Figure 3), this loss is defined as:

$$\mathcal{L}^{gc} = \sum_{n \in N} \sum_{p \in U_n} \left| \widetilde{d}^*_{n,t} - d^*_{n,t} \right|. \quad (11)$$

Note that U_n has no relation to the set of valid pixels (V) from the previous losses, since this loss is only computed between predictions. This enables the network to make sensible predictions even in parts of the image which have no valid label.

Finally, we also include an L_2 weight regulariser, $\mathcal{L}^{reg}$, to reduce overfitting by keeping the weights small. The overall objective is thus defined as:

$$\mathcal{L} = \lambda^{data} \mathcal{L}^{data}_s + \lambda^{\nabla} \mathcal{L}^{\nabla} + \lambda^{gc} \mathcal{L}^{gc} + \lambda^{reg} \mathcal{L}^{reg}, \quad (12)$$

where s is the scale, and the λs are weights for each of the components (see Table 2 for values).

Table 2: Summary of Hyperparameters Used in System

Symbol	Value	Description
λ^{data}	1	Weight of the data loss.
λ^{∇}	0.1	Weight of the smoothness loss.
λ^{gc}	0.1	Weight of the geometric consistency loss.
λ^{reg}	10^{-6}	Weight of the L_2 variable regulariser.
w_{min}	0.1	Minimum per-pixel loss scaling.
w_{max}	5	Maximum per-pixel loss scaling.
η_{max}	10^{-4}	Initial learning rate.
η_{min}	$5 \cdot 10^{-6}$	Final learning rate.
T_{max}	$1.2 \cdot 10^5$	Learning rate decay steps.
β_1	0.9	Adam exponential decay rate for first moment estimates.
β_2	0.999	Adam exponential decay rate for second moment estimates.
	80	Norm at witch gradients are clipped during training.
	16	Batch size.
	$5 \cdot 10^5$	Number of training steps.

3.3.3 Loss weight

Inspired by the work of Ronneberger et al. (2015) on U-Nets, we use a loss-weighting mechanism based on the Euclidean Distance Transform (Felzenszwalb and Huttenlocher, 2012) to emphasise edge pixels when regressing to the error in depth. We first extract Canny edges (Canny, 1986) from the ground-truth labels. Based on these edges, we then compute the per-pixel weights as:

$$\begin{aligned} d(p) &= \ln(1 + \mathrm{EDT}(p)) \\ \mathrm{W}(p) &= (w_{max} - w_{min}) \left(1 - \frac{d(p)}{\max_p d(p)} \right) + w_{min} \end{aligned}, \quad (13)$$

where $\mathrm{W}(p)$ is the loss weight for pixel p, $\mathrm{EDT}(p)$ is the Euclidean Distance Transform at pixel p, and w_{min} and w_{max} are the desired range of the per-pixel weight. This will assign w_{max} weight to edge pixels, and lower weights the farther pixels are from an edge, down to w_{min} for the farthest pixels.

4 Experiments

4.1 Experimantal setup

Training and inference

The network is implemented in Python using TensorFlow v1.12. Each model is trained on an Nvidia Titan V GPU. The weights are optimised for 500 000

Table 3: Generalisation Capability of Depth Error Correction

Model	Train	Test	iMAE	iRMSE	$\delta < 1.05$	$\delta < 1.15$	$\delta < 1.25^1$	$\delta < 1.25^2$	$\delta < 1.25^3$
Uncorrected	–	00	$1.76 \cdot 10^{-2}$	$8.10 \cdot 10^{-2}$	52.71%	79.00%	84.02%	88.73%	90.71%
——"——	–	05	$1.57 \cdot 10^{-2}$	$8.15 \cdot 10^{-2}$	55.11%	80.61%	84.27%	88.30%	90.46%
——"——	–	06	$2.81 \cdot 10^{-2}$	$1.24 \cdot 10^{-1}$	49.47%	72.71%	75.07%	78.64%	80.49%
Ours	05; 06	00	$1.45 \cdot 10^{-2}$	$8.03 \cdot 10^{-2}$	66.14%	85.27%	89.93%	94.76%	96.82%
——"——	00; 06	05	$1.19 \cdot 10^{-2}$	$7.67 \cdot 10^{-2}$	70.38%	86.31%	90.25%	94.86%	96.99%
——"——	00; 05	06	$2.37 \cdot 10^{-2}$	$1.15 \cdot 10^{-1}$	62.27%	78.14%	81.72%	85.85%	88.20%

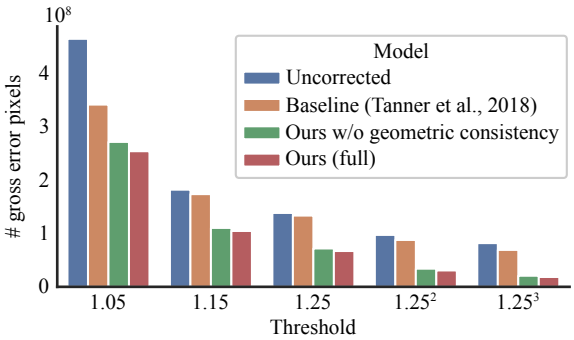


Figure 4: Gross error correction. For each threshold (Equation 14), we count how many pixels are incorrect in the predictions over the test set. Our full model removes 45.3% of the smaller errors and 77.5% of the gross errors. The baseline model is unable to effectively handle the multi-view setup, and fails to correct gross errors.

steps with a batch size of 16 using the Adam optimiser (Kingma and Ba, 2015). The learning rate (η_t) is decayed linearly for the first 120 000 steps. Generating the mesh features takes an average of 52 ms per view using OpenGL on an Nvidia GTX Titan Black, and inference takes an average of 12.5 ms on the Titan V. All the training hyper-parameters are defined in Table 2.

Dataset

Three sequences from the KITTI visual odometry (KITTI-VO) dataset were used as the input to the reconstruction pipeline. For each sequence, two reconstructions are built: one from the stereo camera depth-maps, and one from the laser data. Both the meshes are generated with a fixed voxel width of 0.2 m. In the experiments, we show how to learn a correction of the depth-map reconstruction using the laser reconstruction as reference. Using OpenGL Shading Language (GLSL), we create a virtual camera and project each dense reconstruction into mesh features. We sample locations along the original trajectory in each sequence every 0.3m, and at each location we generate mesh features from four different viewpoints. An illustration of the viewpoints is shown in Figure 3. For all experiments, the KITTI-VO sequences 00, 05, and 06 are used, from which a total of 96 728 training examples of size 96×288 are generated. We split each of the mesh feature sequences into three distinct parts: the first 80% we use as training data, the next 10% we use for validating hyperparameter choices, and the last 10% we use for evaluation. All three sequences are predominantly in urban environments with small amounts of visible vegetation.

Performance metrics

We use some metrics common in literature for assessing inverse-depth predictions.

Our first metric measures the accuracy of our network's ability to estimate errors under a given threshold, serving as an indication of how often our estimate is correct. The thresholded accuracy measure is essentially the expectation that a given pixel in V is within a threshold thr of the label:

$$\delta = \mathbb{E}_{p \in V}\left[\mathbb{I}\left(\max\left(\frac{d^{hq}}{d^*}, \frac{d^*}{d^{hq}}\right) < thr\right)\right], \quad (14)$$

where d^{hq} is the reference inverse-depth, d^* is the predicted inverse depth, V is the set of valid pixels, and n is the cardinality of V, and $\mathbb{I}(\cdot)$ represents the indicator function. For granularity, we use $thr \in \{1.05, 1.15, 1.25, 1.25^2, 1.25^3\}$.

In addition, the mean absolute error (MAE) and root mean square error (RMSE) metrics provide a quantitative measure of per pixel error and are computed as follows:

$$\text{iMAE} = \frac{1}{n}\sum_{p \in V}\left|d^* - d^{hq}\right|, \quad (15)$$

$$\text{iRMSE} = \sqrt{\frac{1}{n}\sum_{p \in V}(d^* - d^{hq})^2}, \quad (16)$$

where the 'i' indicates that the metrics are computed over inverse-depth images.

4.2 Gross error correction

We first look at how well our model corrects gross errors in inverse-depth. Equation 14 can be used to

classify pixels in an image as either correct or incorrect at a given threshold. Using this method, we count the number of incorrect pixels in our predictions, and compare it to the number of incorrect pixels in the input inverse-depth. As a baseline, we train the model proposed in Tanner et al. (2018) on our dataset, and compare it to our full model, trained with and without the geometric consistency loss (Figure 4). Our proposed model outperforms the baseline at correcting both small errors ($thr = 1.05$) as well as larger errors, reducing the number of errors at $thr = 1.25^3$ by 77.5%.

4.3 Generalisation capability

To be useful in practice, the model needs to be able to generalise to new data. For example, separate models could be trained for indoor scenes and outdoor scenes, or other different types of environments, but certainly within the same kind of environment, one model should work well for a variety of scenes.

To evaluate this, we train the full model on a subset of the available sequences, and test it on the rest (excluding the frames used for validation in Section 4.2). Sequences 00 and 05 are represent different suburban scenes of the same city, while the scene in sequence 06 is in an area where the road is much wider and the buildings are much farther apart. Table 3 shows the data splits and the performance of the models on the test sequences. Interestingly, the model trained only on the suburban scenes of sequences 00 and 05 performs particularly well on the visually distinct sequence 06. This highlights the ability of our model to reduce gross errors.

4.4 Ablation study

To better understand how different components of our model improve the learnt correction, we perform an ablation study. The results in Table 4 show that our proposed geometric consistency loss (rows with GC) improves performance at all error scales.

The proposed attention mask (rows with attn) improves the performance in the absence of geometric consistency, but slightly limits the performance especially with larger errors. However, qualitatively (Figure 5), the attention mask allows us to better handle missing training data (that is greyed out in the ground truth column). In particular, surfaces are not spuriously removed or added in those regions: the model learns to mask those regions out of the correction and keep them as they are.

5 Conclusion

In this paper we present a method for correcting gross errors in dense 3D meshes. We extracted paired 2D mesh features from two reconstructions and trained a neural network to predict the difference in inverse-depth between the two. We addressed the issue of overly-smooth predictions with a U-Net architecture and a loss-weighting mechanism that emphasises edges. The geometric consistency of our predictions is improved with a view-synthesis loss that targets inconsistencies. Our experiments show that the proposed method reduces gross errors in inverse-depth views of the mesh by up to 77.5%.

REFERENCES

Bredies, K., Kunisch, K., and Pock, T. (2010). Total generalized variation. *SIAM Journal of Imaging Sciences*, 3:4920–526.

Canny, J. F. (1986). A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8:679–698.

Chambolle, A. and Pock, T. (2011). A first-order primal-dual algorithm for convex problems with applications to imaging. *Journal of Mathematical Imaging and Vision*, 40(1):120–145.

Dharmasiri, T., Spek, A., and Drummond, T. (2019). ENG: End-to-end neural geometry for robust depth and pose estimation using CNNs. In *Proceedings of the Asian Conference on Computer Vision (*ACCV*)*, pages 625–642.

Eldesokey, A., Felsberg, M., and Khan, F. S. (2018). Propagating confidences through CNNs for sparse data regression. In *Proceedings of the British Machine Vision Conference (*BMVC*)*.

Felzenszwalb, P. F. and Huttenlocher, D. P. (2012). Distance transforms of sampled functions. *Theory of Computing*, 8:415–428.

Godard, C., Mac Aodha, O., and Brostow, G. J. (2017). Unsupervised monocular depth estimation with left-right consistency. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (*CVPR*)*.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (*CVPR*)*, pages 770–778.

Hua, J. and Gong, X. (2018). A normalized convolutional neural network for guided sparse depth upsampling. In *Proceedings of the International Joint Conference on Artificial Intelligence (*IJCAI*)*.

Jeon, J. and Lee, S. (2018). Reconstruction-based pairwise depth dataset for depth image enhancement using CNN. In *Proceedings of the European Conference on Computer Vision (*ECCV*)*.

Table 4: Depth Error Correction Ablation Study Results

Model	**iMAE**	**iRMSE**	$\delta < 1.05$	$\delta < 1.15$	$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$
Uncorrected	$1.85 \cdot 10^{-2}$	$8.73 \cdot 10^{-2}$	50.90%	80.77%	85.38%	89.72%	91.35%
Baseline	$1.47 \cdot 10^{-2}$	$6.25 \cdot 10^{-2}$	63.92%	81.65%	85.90%	90.72%	92.69%
w/ our losses; no GC	$1.28 \cdot 10^{-2}$	$6.26 \cdot 10^{-2}$	67.77%	84.35%	89.05%	94.16%	96.17%
w/ our losses; GC	$1.17 \cdot 10^{-2}$	$5.44 \cdot 10^{-2}$	68.66%	84.64%	89.45%	94.81%	96.65%
Ours (no attn; no GC)	$1.12 \cdot 10^{-2}$	$5.49 \cdot 10^{-2}$	70.47%	87.68%	92.08%	96.35%	97.72%
Ours (no GC)	$1.12 \cdot 10^{-2}$	$5.63 \cdot 10^{-2}$	71.31%	88.34%	92.41%	96.41%	97.81%
Ours (no attn)	$\mathbf{1.02 \cdot 10^{-2}}$	$\mathbf{5.31 \cdot 10^{-2}}$	*72.93%*	**89.27%**	**93.20%**	**96.87%**	**98.10%**
Ours (full)	*1.06 · 10^-2*	*5.37 · 10^-2*	**73.16%**	*88.94%*	*92.91%*	*96.76%*	*98.05%*

Kingma, D. and Ba, J. (2015). Adam: A method for stochastic optimization. In *Proceedings of the International Conference on Learning Representations (*ICLR*)*.

Klodt, M. and Vedaldi, A. (2018). Supervising the new with the old: Learning SfM from SfM. In *Proceedings of the European Conference on Computer Vision (*ECCV*)*.

Knutsson, H. and Westin, C.-F. (1993). Normalized and differential convolution: Methods for interpolation and filtering of incomplete and uncertain data. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (*CVPR*)*.

Kopf, J., Cohen, M. F., Lischinski, D., and Uyttendaele, M. (2007). Joint bilateral upsampling. *ACM Transactions on Graphics*, 26:96.

Kwon, H., Tai, Y.-W., and Lin, S. (2015). Data-driven depth map refinement via multi-scale sparse representation. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (*CVPR*)*, pages 159–167.

Laina, I., Rupprecht, C., Belagiannis, V., Tombari, F., and Navab, N. (2016). Deeper depth prediction with fully convolutional residual networks. In *Proceedings of the IEEE International Conference on 3D Vision (*3DV*)*, pages 239–248.

Ma, F. and Karaman, S. (2018). Sparse-to-dense: Depth prediction from sparse depth samples and a single image. In *Proceedings of the IEEE International Conference on Robotics and Automation (*ICRA*)*.

Mahjourian, R., Wicke, M., and Angelova, A. (2018). Unsupervised learning of depth and ego-motion from monocular video using 3D geometric constraints. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (*CVPR*)*, pages 5667–5675.

Matsuo, T., Fukushima, N., and Ishibashi, Y. (2013). Weighted joint bilateral filter with slope depth compensation filter for depth map refinement. In *Proceedings of the International Conference on Computer Vision Theory and Applications (*VISAPP*)*.

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

Owen, A. B. (2007). A robust hybrid of lasso and ridge regression. *Contemporary Mathematics*, 443:59–72.

Riegler, G., Rüther, M., and Bischof, H. (2016). ATGV-Net: Accurate depth super-resolution. In *Proceedings of the European Conference on Computer Vision (*ECCV*)*.

Ronneberger, O., Fischer, P., and Brox, T. (2015). U-Net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer Assisted Intervention (*MICCAI*)*.

Shi, W., Caballero, J., Huszár, F., Totz, J., Aitken, A. P., Bishop, R., Rueckert, D., and Wang, Z. (2016). Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (*CVPR*)*, pages 1874–1883.

Sobel, I. and Feldman, G. (1968). A 3×3 isotropic gradient operator for image processing. presented at the Stanford Artificial Intelligence Project (SAIL).

Tanner, M., Piniés, P., Paz, L. M., and Newman, P. (2015). BOR^{2}G: Building optimal regularised reconstructions with GPUs (in cubes). In *Proceedings of the International Conference on Field and Service Robotics (*FSR*)*, Toronto, Canada.

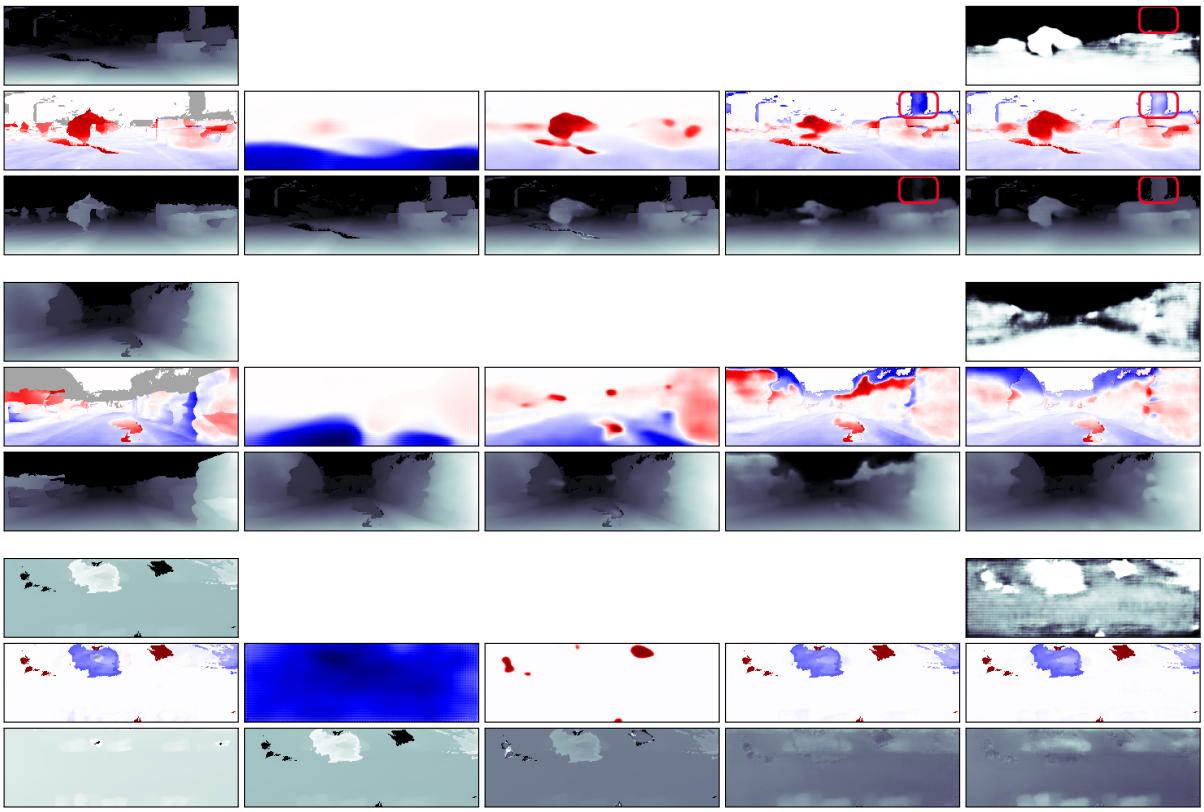
Tanner, M., Piniés, P., Paz, L. M., and Newman, P. (2016). Keep geometry in context: Using contextual priors for very-large-scale 3d dense reconstructions. In *Robotics: Science and Systems (*RSS*). Workshop on Geometry and Beyond: Representations, Physics, and Scene Understanding for Robotics*.

Tanner, M., Săftescu, Ș., Bewley, A., and Newman, P. (2018). Meshed up: Learnt error correction in 3D reconstructions. In *Proceedings of the IEEE International Conference on Robotics and Automation (*ICRA*)*, Brisbane, Australia.

Tomasi, C. and Manduchi, R. (1998). Bilateral filtering for gray and color images. In *Proceedings of the International Conference on Computer Vision (*ICCV*)*, pages 839–846.

Uhrig, J., Schneider, N., Schneider, L., Franke, U., Brox, T., and Geiger, A. (2017). Sparsity invariant CNNs. In *Proceedings of the IEEE International Conference on 3D Vision (*3DV*)*.

Ummenhofer, B., Zhou, H., Uhrig, J., Mayer, N., Ilg, E., Dosovitskiy, A., and Brox, T. (2017). DeMoN: Depth and motion network for learning monocular stereo.



(a) Ground truth (b) Baseline (c) Baseline + our losses (d) Ours (no attn; no GC) (e) Ours (full)

Figure 5: Illustration of how the quality of the predictions changes as different components are added to the system. Each three rows show an example, as follows: (a) Input inverse-depth and ground-truth error, and ground-truth inverse-depth. The shaded areas in the ground-truth error represent missing data in the reference mesh. (b)–(d) Prediction and corrected inverse-depth. (e): Attention mask, prediction, and corrected inverse-depth. The baseline model (b) only learns very rough predictions and is unable to generalise well to the top viewpoint (last example). Our proposed losses help with generalisation across viewpoints, but without skip connections in the network predictions are not very well localised (c). Our model (d); (e) makes well-localised predictions. The attention mask removes some of the spurious predictions where there is no reference data (e). For example, the highlighted background structure in the first example is removed when the attention mask is not used (d), while the attention mask disables edits to that region of the input image (e).

In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition* (CVPR), pages 5622–5631.

Zhang, Y. and Funkhouser, T. A. (2018). Deep depth completion of a single RGB-D image. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition* (CVPR), pages 175–185.

Zhou, T., Brown, M., Snavely, N., and Lowe, D. G. (2017). Unsupervised learning of depth and ego-motion from video. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition* (CVPR).

Statement of Authorship for joint/multi-authored papers for PGR thesis

To appear at the end of each thesis chapter submitted as an article/paper

The statement shall describe the candidate's and co-authors' independent research contributions in the thesis publications. For each publication there should exist a complete statement that is to be filled out and signed by the candidate and supervisor **(only required where there isn't already a statement of contribution within the paper itself).**

Title of Paper	Learning geometrically consistent mesh corrections
Publication Status	☒ Published ☐ Accepted for Publication ☐ Submitted for Publication ☐ Unpublished and unsubmitted work written in a manuscript style
Publication Details	**Săftescu, Ș.**, & Newman, P. (2020). Learning geometrically consistent mesh corrections. *Proceedings of The International Conference on Computer Vision Theory and Applications (VISAPP)*, Valletta, Malta.

Student Confirmation

Student Name:	Ștefan Săftescu		
Contribution to the Paper	All work except editorial changes and advice, including: • Generation of ideas • Software development • Designed and performed experiments • Wrote the paper		
Signature	Ș. Săftescu	Date	29/08/2020

Supervisor Confirmation

By signing the Statement of Authorship, you are certifying that the candidate made a substantial contribution to the publication, and that the description described above is accurate.

Supervisor name and title:	Professor Paul Newman		
Supervisor comments			
Signature	P. Newman	Date	10/09/2020

This completed form should be included in the thesis, at the end of the relevant chapter.

6.1 Further Results

In this section, we will present two further results that provide more insight into our work. First, we look at how well our method works in correcting each of the four different viewpoints in our dataset. Then, we analyse the corrected 3D meshes that result from the application of our method.

6.1.1 Per-View Breakdown

In the article presented in this chapter, we use four different synthetic viewpoints when correcting meshes: two corresponding to the stereo setup of the data collection vehicle (left and right), a back view facing the opposite direction, and a top-down view from 28 m above the road surface. We have shown that our method successfully corrects inverse-depth from these viewpoints and, for additional insight, we include in Figure 6.2 a per-view breakdown of the number of incorrect pixels that our method corrects.

The first three synthetic views we use – left, right, and back, are all at the road level. The number of incorrect pixels is similar between them, with the left view having slightly fewer errors than the other two. This is because the left view corresponds to the camera with respect to which the uncorrected reconstruction has been built, so the unobserved regions play no role in this viewpoint. The right and back views, on the other hand, contain some of the unobserved regions, which will not have correct surface.

The top view largely contains road surface, which is easier to reconstruct than the rest of the scene. This explains why this viewpoint has so many fewer errors – even before correction. Additionally, more than 40% of the incorrect pixels are at the largest threshold value (more than 95% error), corresponding to holes in the road or unobserved regions, and are easier to correct.

What we see in Figure 6.2 is consistent with the aggregated results shown in the paper. The baseline method presented in Chapter 4 corrects small errors in the road level views, but fails to correct larger errors effectively, or to generalise to the top view, which is different in appearance. The extensions presented in this chapter help correct errors at all magnitudes from all four views. Furthermore, note that in the "easy" top view we manage to correct almost all of the errors bigger than 15%, and more than three quarters of the errors bigger than 5%.

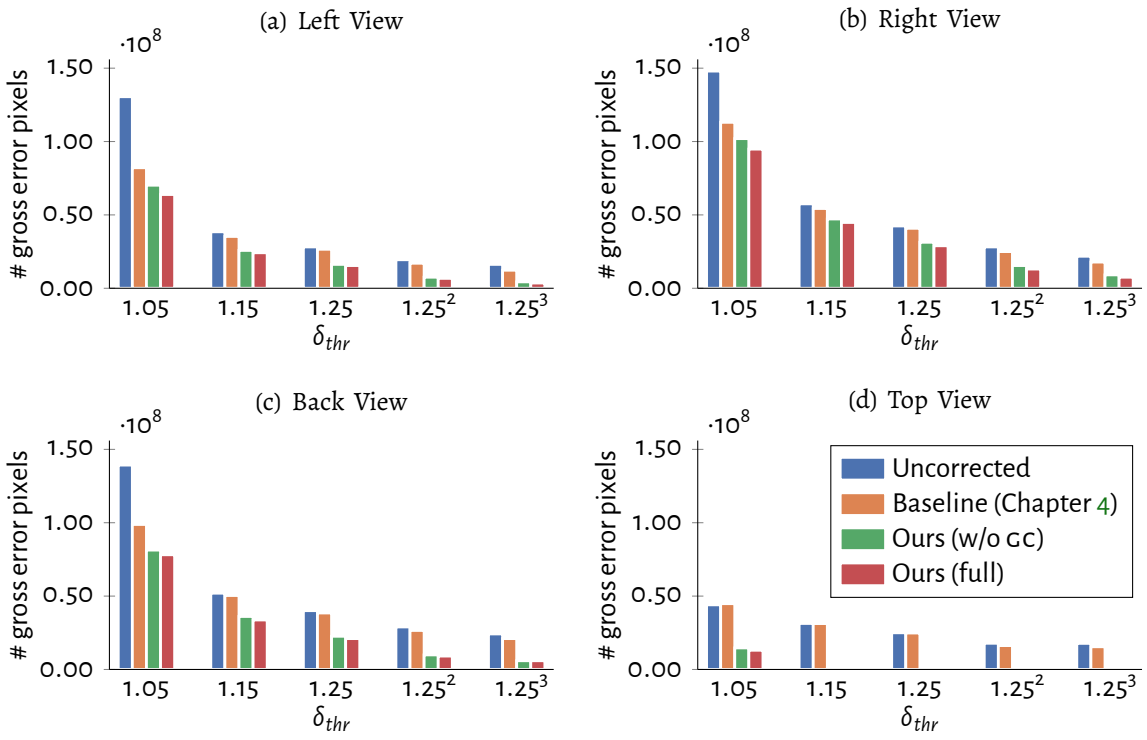


Figure 6.2: Per-view gross error correction. For each error threshold, we count how many pixels are incorrect in the predictions over the test set. The additions presented in this chapter allow our method to correct errors of all magnitudes and to generalise to all viewpoints considered. Note that in the easier top view our method corrects almost all the errors.

6.1.2 *Post-Correction 3D Reconstructions*

Thus far in this chapter, we have looked at how our models perform when correcting *2D inverse-depth images* rendered from meshes. This is a proxy for fixing *3D reconstructions*, and also serves as the learning objective when training our CNN. However, what we are eventually interested in is how good the resulting, corrected reconstruction is – the mesh built from corrected inverse-depth images. In this section, we look at how different components proposed in this chapter affect the quality of the corrected reconstruction.

Evaluation

To evaluate our 3D reconstructions, we use the method described in Section 2.4.1, and we report both the median and the mean for surface metrics.

We divide the available data as in the paper: we use models trained on two out of the three KITTI sequences to correct a reconstruction of the third. In the results (Tables 6.1

Table 6.1: 3D Reconstruction Error Results for Models Proposed in this Chapter (Correction 6.1)

Test Seq.	Model	Correctness (cm) Median	Correctness (cm) Mean	Completeness (cm) Median	Completeness (cm) Mean	Total (cm) Median	Total (cm) Mean
KITTI 00	Uncorrected (camera)	16.35	**29.57**	44.62	118.72	47.52	122.35
	w/o GC; w/o SM	**15.82**	*46.20*	32.18	85.05	*35.86*	**96.79**
	with GC; w/o SM	*16.25*	47.49	*32.01*	*84.75*	35.90	*97.15*
	with GC; with SM	17.28	51.37	**31.18**	**82.64**	**35.65**	97.30
KITTI 05	Uncorrected (camera)	12.84	**24.93**	51.75	119.84	53.32	122.41
	w/o GC; w/o SM	12.06	34.22	**33.66**	*86.11*	**35.76**	*92.66*
	with GC; w/o SM	**11.71**	34.23	35.31	88.40	37.20	94.80
	with GC; with SM	*12.01*	*32.85*	*33.89*	**84.05**	*35.96*	**90.24**
KITTI 06	Uncorrected (camera)	10.57	*24.16*	19.55	74.02	22.22	77.86
	w/o GC; w/o SM	7.39	27.26	11.06	48.78	13.30	55.88
	with GC; w/o SM	**6.90**	24.63	*10.97*	*48.64*	*12.96*	*54.52*
	with GC; with SM	*6.91*	**20.75**	**10.48**	**47.33**	**12.55**	**51.68**

to 6.4), we consider how the two new components introduced in this chapter, geometric consistency loss (GC), and the soft mask (SM) improve the effectiveness of our method.

The lidar sensor in the KITTI Odometry Dataset only returns measurements of up to 2.5 m above the road, while the depth-maps estimated from the camera contain measurements up to 5.6 m above the road. This means that for the top of buildings, the models we train are not supervised, and cannot be evaluated. As such, we remove those parts of the reconstructions when computing the error statistics presented in this section.

Overall, our method greatly increases completeness of reconstructions, with small effects on correctness, as also observed in Chapter 5. Unlike the 2D metrics, the contribution of different components of our method is less pronounced. In Table 6.1 we see that our proposed geometric consistency (GC) has only a small effect – sometimes even negative – while the proposed soft mask (SM) has a small positive effect on completeness. In Table 6.2 we see that our method recovers 44.5–62.7% of the surface missing between the input camera reconstruction and the reference lidar one.

Part of the reason this happens is that most of the error can be easily corrected by the base model, and the additional features bring little benefit without more data. Indeed, we think this is why our models behave differently between the three KITTI sequences considered here. As seen in Table 6.2, Sequences 00 and 05 are much larger than Sequence 06. The models trained on the first two sequences and tested on the last use a lot more

Table 6.2: 3D Surface Area for Models Proposed in this Chapter (Correction 6.2)

Test Sequence	Model	Area (m^2)
KITTI 00	Uncorrected (camera)	45 523
	w/o GC; w/o SM	76 644
	with GC; w/o SM	77 154
	with GC; with SM	81 537
	Reference (lidar)	117 086
KITTI 05	Uncorrected (camera)	28 331
	w/o GC; w/o SM	50 128
	with GC; w/o SM	49 030
	with GC; with SM	50 127
	Reference (lidar)	77 292
KITTI 06	Uncorrected (camera)	14 125
	w/o GC; w/o SM	23 837
	with GC; w/o SM	23 363
	with GC; with SM	23 548
	Reference (lidar)	29 617

data, so our geometric consistency loss and soft mask bring improvements that cannot be captured by the simpler model. We analyse this further in Section 7.2.2.

Behaviour on unknown regions

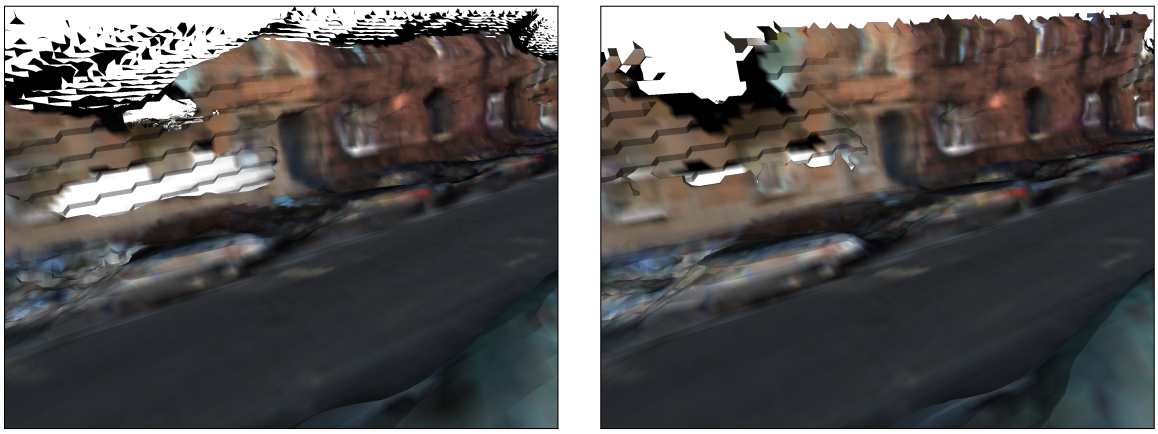
For the regions where there is no laser data to supervise our model, and which cannot be quantitatively evaluated, we ask what sensible predictions might look like qualitatively. One possible prediction for such a region would be "no change" – since no better information is available during training, it would be sensible to retain the low-quality data unaltered for these areas. To measure this, we compare the parts of reconstructions for which there is no lidar reconstruction with the camera reconstruction and show the results in Table 6.3.

We see from Table 6.4 that the models trained with geometric consistency loss create a lot more surface in areas that were not supervised. Without geometric consistency, predictions between different views in these areas do not match, and therefore far less surface is created when fusing predictions into a 3D reconstruction. However, even though training with geometric consistency yields more surface, that surface will not necessarily match the one in the input reconstruction – it will simply be consistent between predictions. The models trained to predict a soft mask learn to keep these regions unchanged. The soft

Table 6.3: 3D Reconstruction Error Results for Unlabelled Input Regions (Correction 6.3)

Test Seq.	Model	Correctness (cm) Median	Correctness (cm) Mean	Completeness (cm) Median	Completeness (cm) Mean	Total (cm) Median	Total (cm) Mean
KITTI 00	w/o GC; w/o SM	*69.56*	*107.40*	91.54	127.24	114.97	166.51
	with GC; w/o SM	71.84	107.93	*55.88*	*98.25*	*91.01*	*145.95*
	with GC; with SM	**41.86**	**72.63**	**14.93**	**48.95**	**44.44**	**87.59**
KITTI 05	w/o GC; w/o SM	*56.39*	*93.87*	67.63	105.42	88.05	141.16
	with GC; w/o SM	62.07	102.30	*48.00*	*82.90*	*78.46*	*131.67*
	with GC; with SM	**31.37**	**66.47**	**9.88**	**31.20**	**32.89**	**73.43**
KITTI 06	w/o GC; w/o SM	*39.57*	*74.29*	48.52	103.89	62.61	127.72
	with GC; w/o SM	43.12	82.49	*18.00*	*35.95*	*46.73*	*89.98*
	with GC; with SM	**32.21**	**67.15**	**12.10**	**26.99**	**34.41**	**72.37**

mask effectively provides some introspective capabilities to our model, allowing it to decide which areas it has learnt to correct and which it has not. These models thus leave a lot more of the unknown surfaces unchanged – arguably a sensible behaviour, illustrated in Figure 6.3.



(a) Reconstruction without soft mask

(b) Reconstruction with soft mask

Figure 6.3: Comparison between models reconstructions without and with soft mask. The soft mask modulates the amount of correction applied, and can learn to ignore corrections in regions where the model is uncertain. In this figure, the model is uncertain about the top of buildings, where there is no matching lidar data. When the soft mask is not present, the model distorts such regions (a), while otherwise the model learns to keep these regions unchanged (b).

Table 6.4: 3D Surface Area for Unlabelled Input Regions (Correction 6.4)

Test Sequence	Model	Area (m^2)
KITTI 00	Uncorrected (camera)	59 536
	w/o GC; w/o SM	27 916
	with GC; w/o SM	37 020
	with GC; with SM	*54 995*
KITTI 05	Uncorrected (camera)	33 896
	w/o GC; w/o SM	14 912
	with GC; w/o SM	19 483
	with GC; with SM	*30 764*
KITTI 06	Uncorrected (camera)	11 784
	w/o GC; w/o SM	3 774
	with GC; w/o SM	7 301
	with GC; with SM	*7 544*

6.2 Conclusion and Extensions

In this chapter we have presented a mesh correction method that noticeably improves on our previous approach. By removing the requirement for real RGB images, we were able to generate vastly more data, from four different viewpoints to train our mesh correction network, and have shown that promoting consistent predictions between views improves 2D metrics. When building the corrected reconstructions we have seen that consistent 2D predictions do not directly translate into improved reconstructions, but have shown that predicting a soft mask to modulate our corrections leads to sensible behaviour in unlabelled regions due to added introspective capabilities. To better understand our contributions, in the next chapter we will provide large-scale evaluation of our models in a shared framework.

One limitation of the approach discussed in this chapter is that, while geometric consistency is encouraged during training between viewpoints, at inference time multi-view information is still unavailable. In Chapter 8, we will present a novel method for aggregating arbitrary views both during training and during inference.

7

Large Scale Results

In the previous chapters, we have introduced and developed the idea of correcting meshes by rendering 2D views of them and producing corrected inverse-depth images that can be used to build new, corrected meshes. We have explored how to ensure consistent predictions between multiple viewpoints and have evaluated our contributions on a small subset of the KITTI Odometry Dataset. In this chapter, we asses the practical applicability of our method by turning to the full KITTI Odometry Dataset, evaluating it on 21.8 km of driving data.

We start by detailing the reconstructions and data we obtained from the KITTI Odometry Dataset. We then evaluate our contributions both in their ability to improve inverse-depth images (2D), as well as by the quality of the corrected meshes (3D). We then conclude this chapter by proposing some directions for future work.

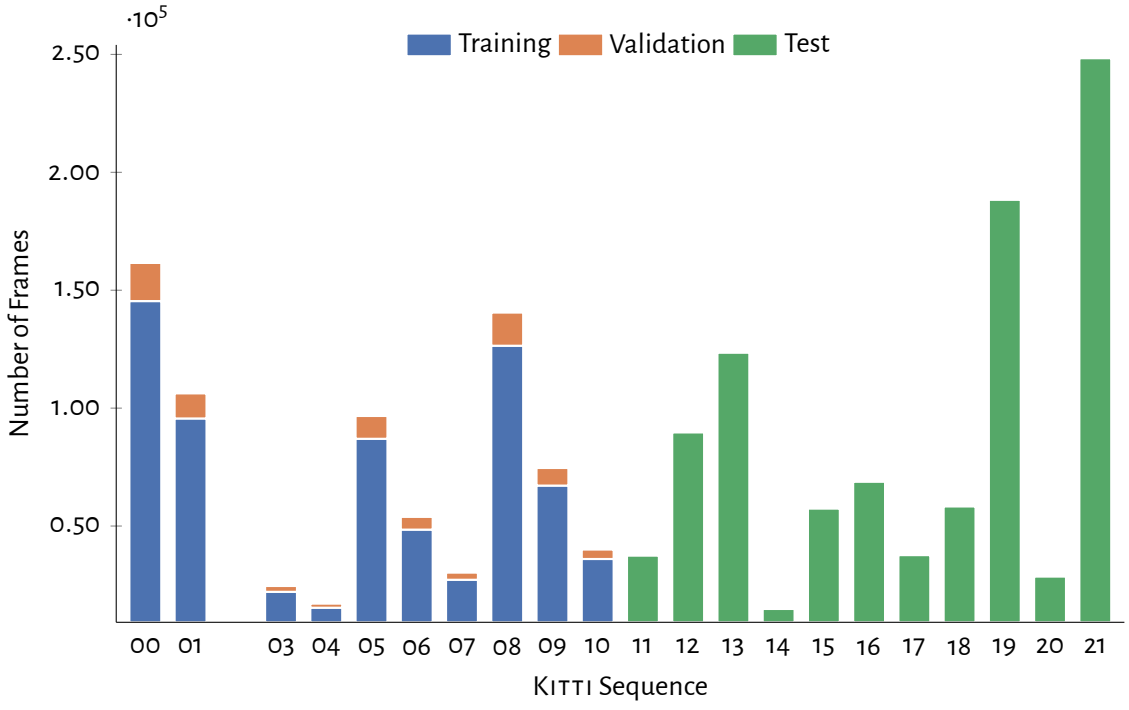


Figure 7.1: Number of mesh feature frames rendered from the KITTI Odometry dataset. The dataset is split into 11 training sequences and 11 test sequences. For each training sequence, we set 10% of the frames aside for validation. In total, there are 745 440 training frames and 952 800 test frames of size 288×96. Sequence 02 is not included as it was too large to reconstruct at the same quality as the rest of the data.

7.1 The KITTI Odometry Dataset

The KITTI Odometry Dataset (Andreas Geiger, Philip Lenz, et al., 2012) is composed of stereo images and lidar data collected over 39.2 km of driving, grouped into 22 sequences showing a variety settings, and it was originally intended for developing visual or lidar odometry algorithms. It includes good camera-to-lidar calibration and we use this to build well-aligned meshes from the two modalities. The first half of the sequences is meant for training and validation and includes ground-truth trajectory data, while the second half is used for benchmarking, so trajectories are not provided.

To keep the same 3D reconstruction pipeline across all sequences, we use ORB-SLAM2 (Mur-Artal and Tardós, 2017) for computing trajectories. Incidentally, as noted in Chapter 5, this is preferable to using the provided Global Navigation Satellite System (GNSS) reference trajectories for our application, since it is more accurate when revisiting locations. Despite not being constrained by the knowledge of reference trajectories, we keep the same splits between training and testing, for consistency. However, we exclude Se-

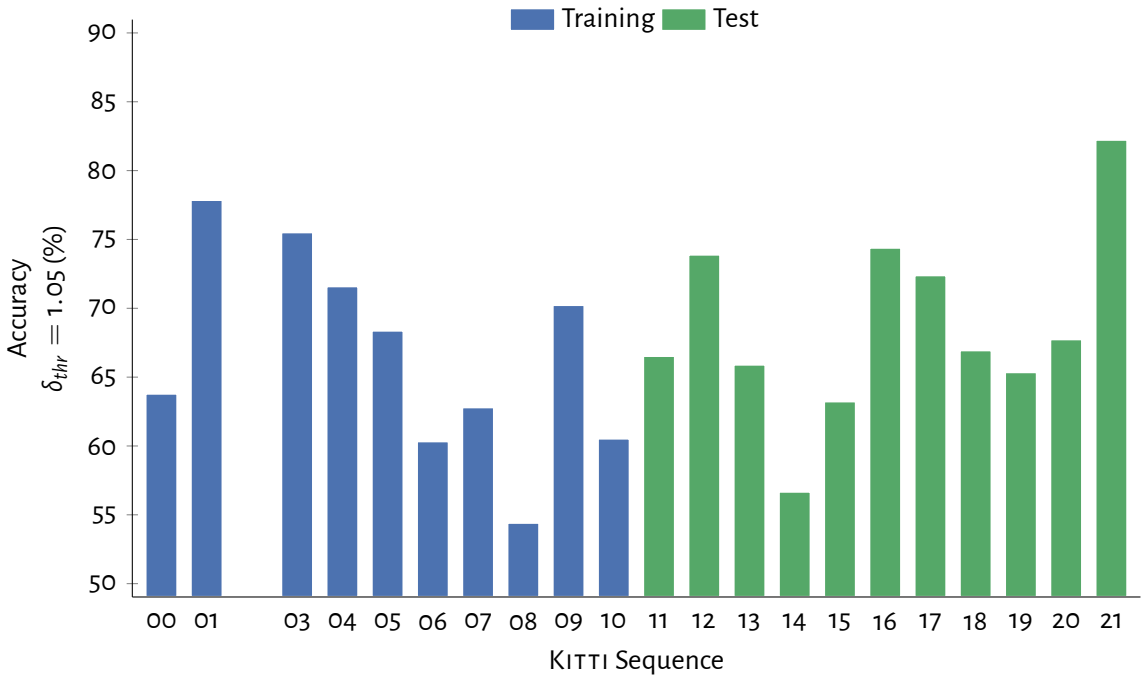


Figure 7.2: Accuracy (at $\delta_{thr} = 1.05$) *before* correction for the KITTI Odometry Dataset. The amount of errors present in the meshes vary widely between sequences and depend on several factors such as textures or presence of dynamic objects. Sequence 02 is not included as it was too large to reconstruct at the same quality as the rest of the data.

quence 02 from our data since it is too large to reconstruct at the same quality as the rest of the sequences.

As detailed in Chapter 3, we use BOR²G to produce 3D reconstructions from the stereo-camera and the lidar data. For each sequence in the dataset we obtain two meshes, one for each modality, from which we extract mesh features along the original trajectory, from four viewpoints of size 288×96. In total, we obtain about 1.7 million viewpoints across 21 sequences, 44% of which we use for training. A per-sequence breakdown of the number of viewpoints is shown in Figure 7.1. Due to the different conditions and locations in which the dataset was collected, the stereo-camera reconstructions have varying degrees of error with respect to the lidar reconstructions. However, both the training and the test sequences have similar amounts of low- and high-error data, as illustrated in Figure 7.2.

7.2 Experiments

For the experiments in this section, we employ the following methodology. We train several models corresponding to our most important contributions and show results on each of the 11 test sequences. Importantly, and unlike in the previous chapters, for every contribution we train a single model. The models we consider are the following:

1. The architecture we proposed in Chapter 4, which we shall refer to as *CH4* in orange (■);
2. The architecture we proposed in Chapter 5 and also used in Chapter 6, which we shall refer to as *CH5* in green (♠);
3. The same architecture as above, but trained with the geometric consistency loss proposed in Chapter 6, which we shall refer to as *CH6-GC* in pink (♦);
4. The full model proposed in Chapter 6, including soft mask prediction and training with geometric consistency loss, which we shall refer to as *CH6-GC-SM* in red (●).

To ensure results are comparable, we use the same encoder architecture for each of the models, and the decoders only differ where strictly necessary (e.g. skip connections, soft mask prediction). Furthermore, the models presented in Chapters 4 and 5 were trained on a combination of mesh features and real colour images, which significantly reduced

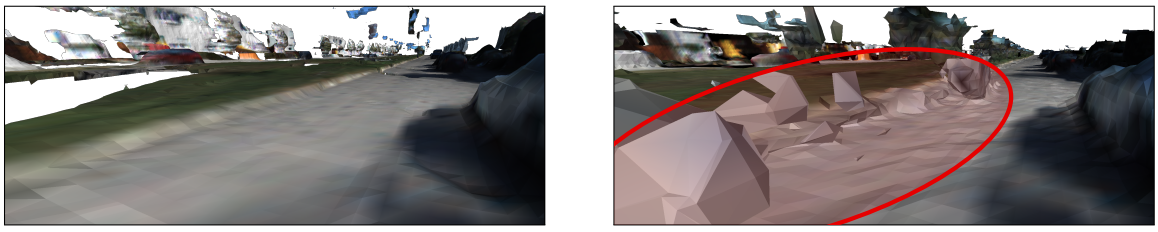
the amount of training data and viewpoints available. Here, we use the same dataset configuration as in Chapter 6 and all the models are trained on the same data, for a million steps.

As seen in Figure 7.2, the quality of stereo-camera reconstructions varies considerably between sequences. For this reason, we present our results for each sequence, relative to the uncorrected meshes, instead of showing aggregate absolute results.

7.2.1 *Inverse-Depth Refinement (2D Results)*

As our method uses a CNN trained to refine inverse-depth views rendered from meshes, we first evaluate the performance at this 2D task. Our evaluation proceeds as follows. For each model, we pick the top-10 checkpoints according to validation iRMSE, and compute thresholded accuracy for each of them on all sequences. We then report means over each of the 10 samples in Figure 7.4. We make two observations on these results as we increase the threshold (and therefore the amount of error we consider).

First, for all but small errors, the amount of correction is inverse-proportional to uncorrected accuracy. This suggests that there are some errors, especially large ones, that cannot be removed with our proposed method. One source of large errors in our data is due to limitations in the stereo-camera modality (e.g. limited viewpoint, lack of textures). The other source of large errors comes from reconstruction artefacts of our static-world assumption. Dynamic objects in the scene, especially moving cars, sometimes result in misplaced surface, as can be seen in Figure 7.3. The former type of errors can be corrected with prior knowledge (roads and buildings are flat, cars have known shapes, etc.) and is



(a) Stereo-Camera Reconstruction (b) Lidar Reconstruction

Figure 7.3: Due to dynamic objects, artefacts may appear in the 3D reconstructions. Highlighted above in the lidar reconstruction (b) is unnecessary surface that appears in the middle of the road due to a moving car behind the data collection vehicle. This kind of noise in the labels cannot be easily accounted for and will appear as error in the metrics.

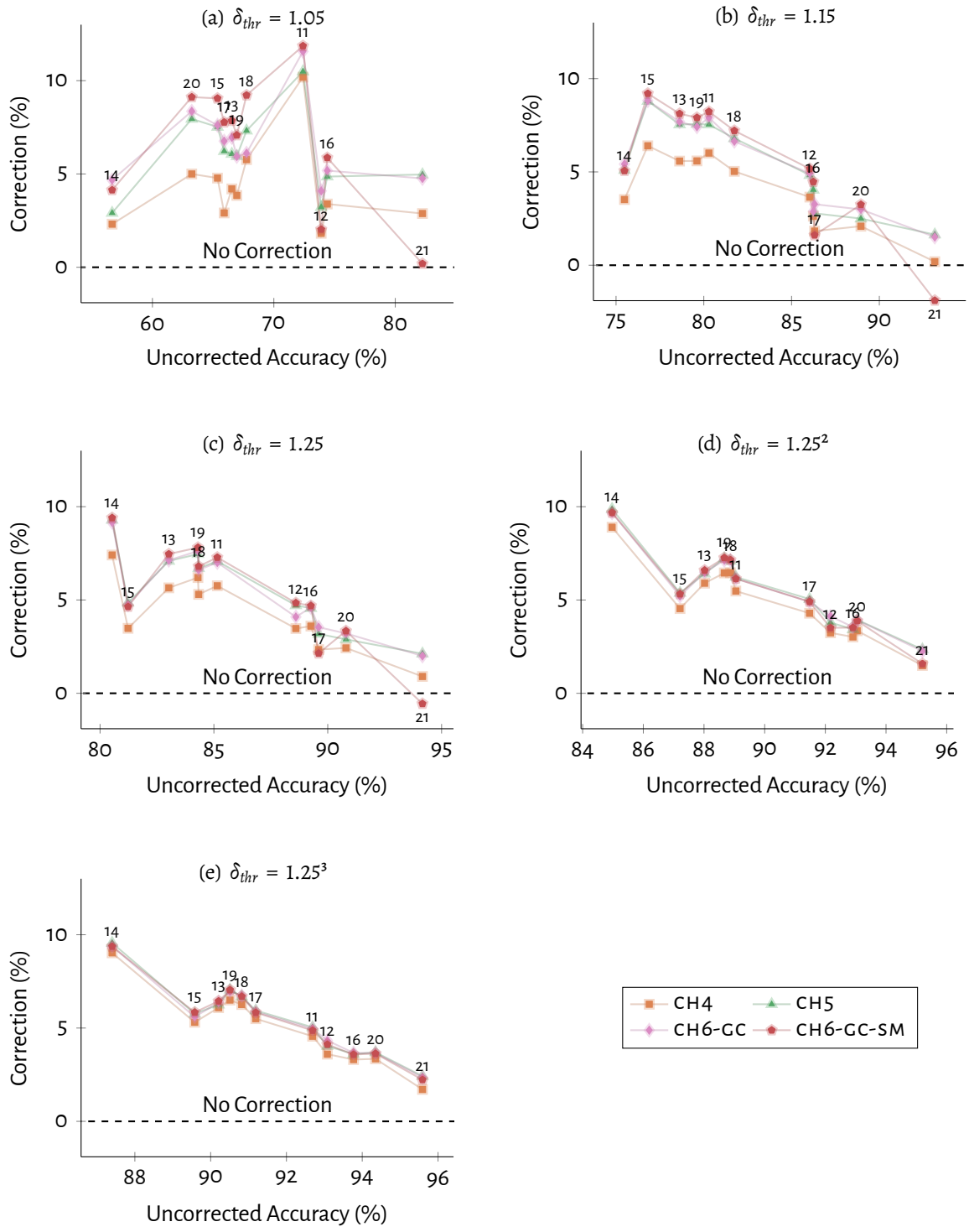


Figure 7.4: Improvement in thresholded accuracy metrics for different thresholds. We make two important observations about these results. First, we notice that the amount of correction is inversely proportional to the accuracy before correction for all but the smallest errors. Second, while all models fix the largest errors similarly, for other errors CH4 (—■—) performs noticeably worse than the other models, due to lack of skip connections, and that the contributions we propose in Chapter 6 bring cumulative improvements to CH5 (—▲—).

the kind we learn to correct, while the latter type appears non-deterministic and cannot effectively be captured by our model.

Second, while all of our models capture large errors with similar accuracy, for small and medium scale errors CH4 (—■—) performs significantly worse (Figures 7.4(a) to 7.4(c)). Furthermore, the geometric consistency loss (CH6-GC, —♦—) and the soft mask (CH6-GC-SM, —●—) proposed in Chapter 6 bring small cumulative improvements to CH5 (—▲—). A notable exception is Sequence 21, where CH6-GC-SM fails.

7.2.2 *Fusing Refined Inverse-Depth (3D Results)*

As described in Chapter 3, after refining inverse-depth images, we fuse them to build corrected meshes. For the results in this section, we use the same models as before, but use only the best validation checkpoint, since fusion is time-consuming. Following the evaluation methodology described in Section 2.4.1, we compute errors with respect to the lidar mesh for each sequence.

We first look in Figure 7.5 at the trade-off between correctness and completeness errors of models across all the test sequences.

The first observation we make is that CH4 (■) greatly reduces completeness error, but at great cost to correctness. Due to lack of skip connections between encoder and decoder, this model cannot predict fine structure, so corrections will not be very detailed and well-localised. As a result, meshes corrected with this model have a great deal of additional surface (much more than necessary, as Figure 7.7 shows), but much of the extra surface will be incorrect. CH5 (▲) improves completeness less than CH4, but does so without affecting correctness much – thanks to skip connections that allow for more precise corrections, this model learns to fill in holes without producing unnecessary surface. Geometric consistency (CH6-GC, ♦) and the soft mask (CH6-GC-SM, ●) further improve completeness on 9 out of 11 test sequences, with only a small cost in correctness.

To better understand the relative performance of different models, Figure 7.6 shows the change in total median error after applying each model to the uncorrected stereo-camera mesh. Due to high correctness error, CH4 (—■—) performs poorly in this large-scale scenario, often *increasing* total error. In contrast, CH5 (—▲—) and subsequent improvements always reduce total error. Using the soft mask model (CH6-GC-SM, —●—) comes with a small

penalty when compared to CH6-GC (), but as discussed in Section 6.1.2 this enables the model to deal with regions unobserved during training.

Finally, Figure 7.7 shows the increase in surface area after correcting meshes. We can see that this increase tends to be proportional to the surface area of the uncorrected mesh. This is to be expected, since the faults we are trying to correct, such as missing surfaces due to lack of texture or poor visibility are all local. Unlike the later models, CH4 creates much more surface than present in the reference lidar reconstructions, indicating that a lot of the surface is unnecessary. This explains the very low correctness of this model, as seen in Figure 7.5.

While many of the differences between our models can be evaluated quantitatively, there are also qualitative differences, as illustrated in Figure 7.8. Our initial model, CH4, makes predictions that are not sufficiently well localised to match the existing geometry, while later models that include skip connections do not suffer from this issue. The pres-

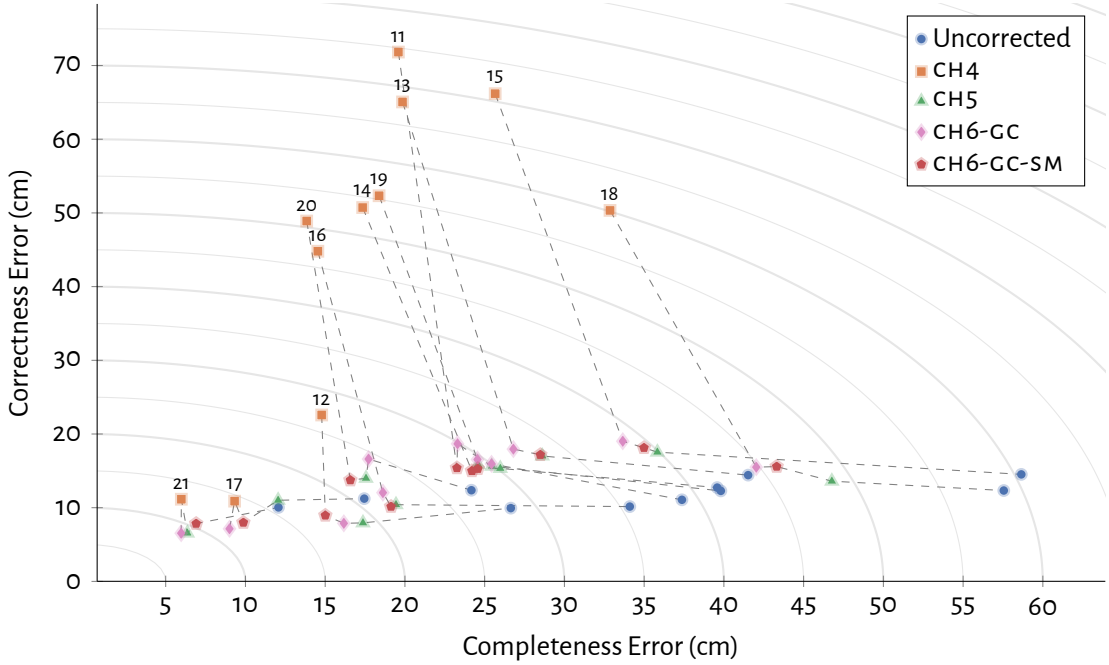


Figure 7.5: 3D metrics for the KITTI Odometry Dataset. Each point represents a model evaluated on a given KITTI sequence and we plot median errors. The points belonging to the same sequence are connected by dashed lines. The curves in the background represent contour lines of the total error (Euclidean distance from origin). We notice that CH4 () improves completeness, but at great cost to correctness, CH5 () brings a considerable improvement in completeness with modest changes in correctness (both positive and negative), while CH6-GC () and CH6-GC-SM () bring further improvements in completeness at a small cost in correctness.

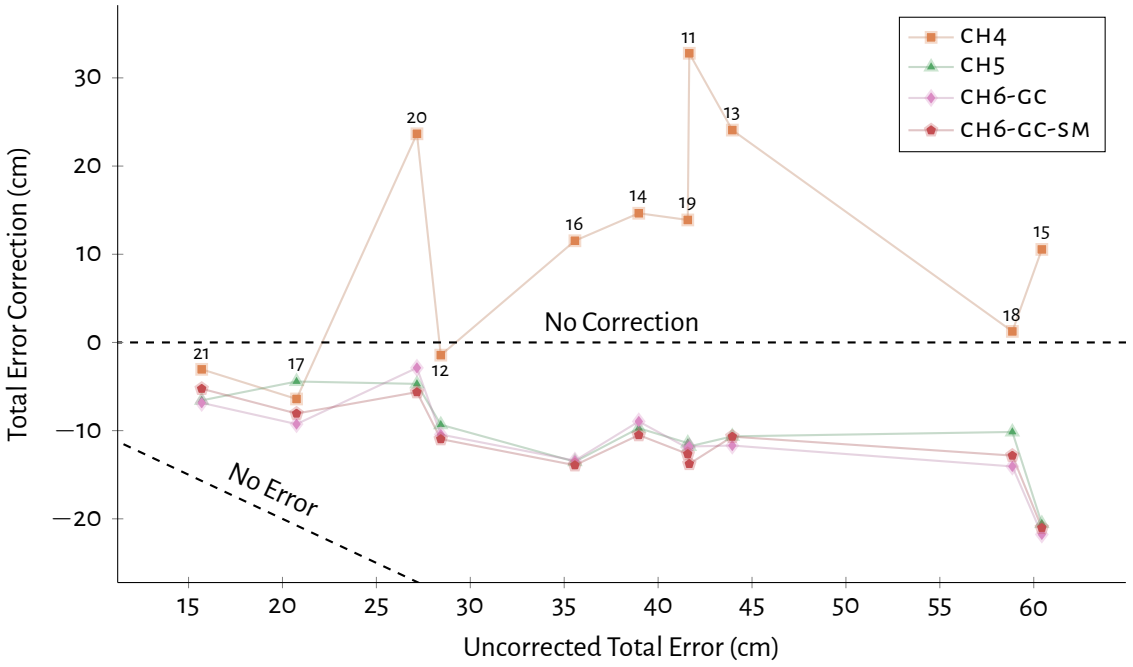


Figure 7.6: Change in total 3D error after mesh correction. While our initial model (—■—) fails to improve the total error in most cases (due to the significat increase in correctness error seen in Figure 7.5), the rest of the models reduce median total error by up to 21.8 cm.

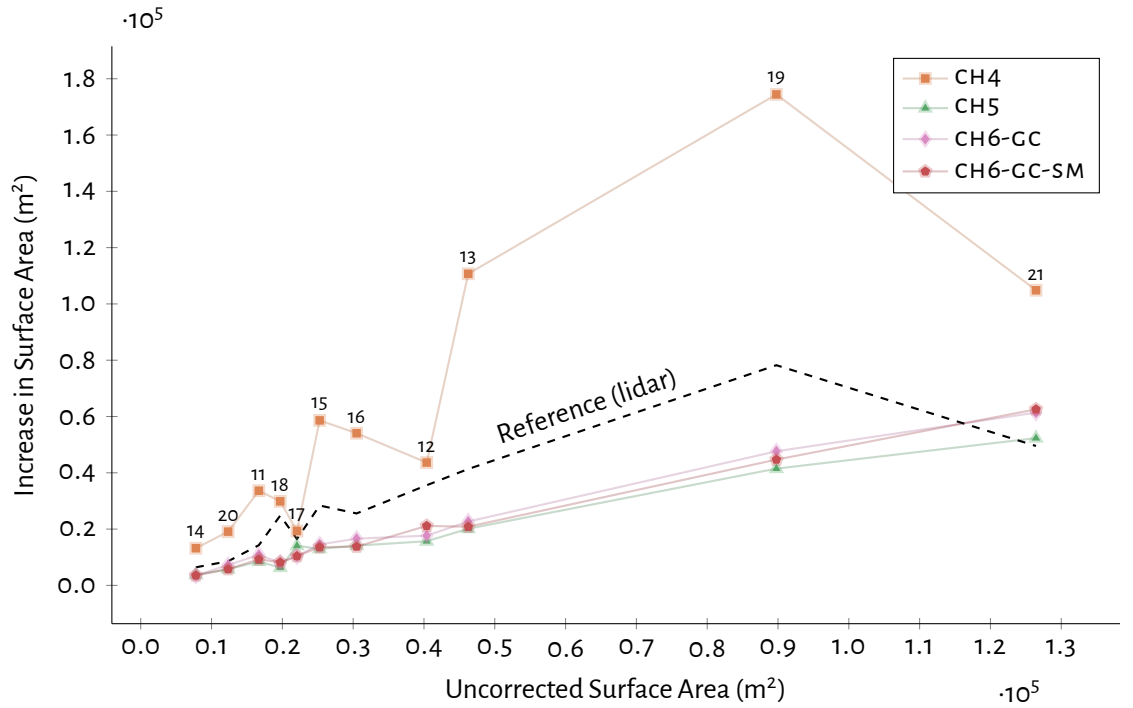
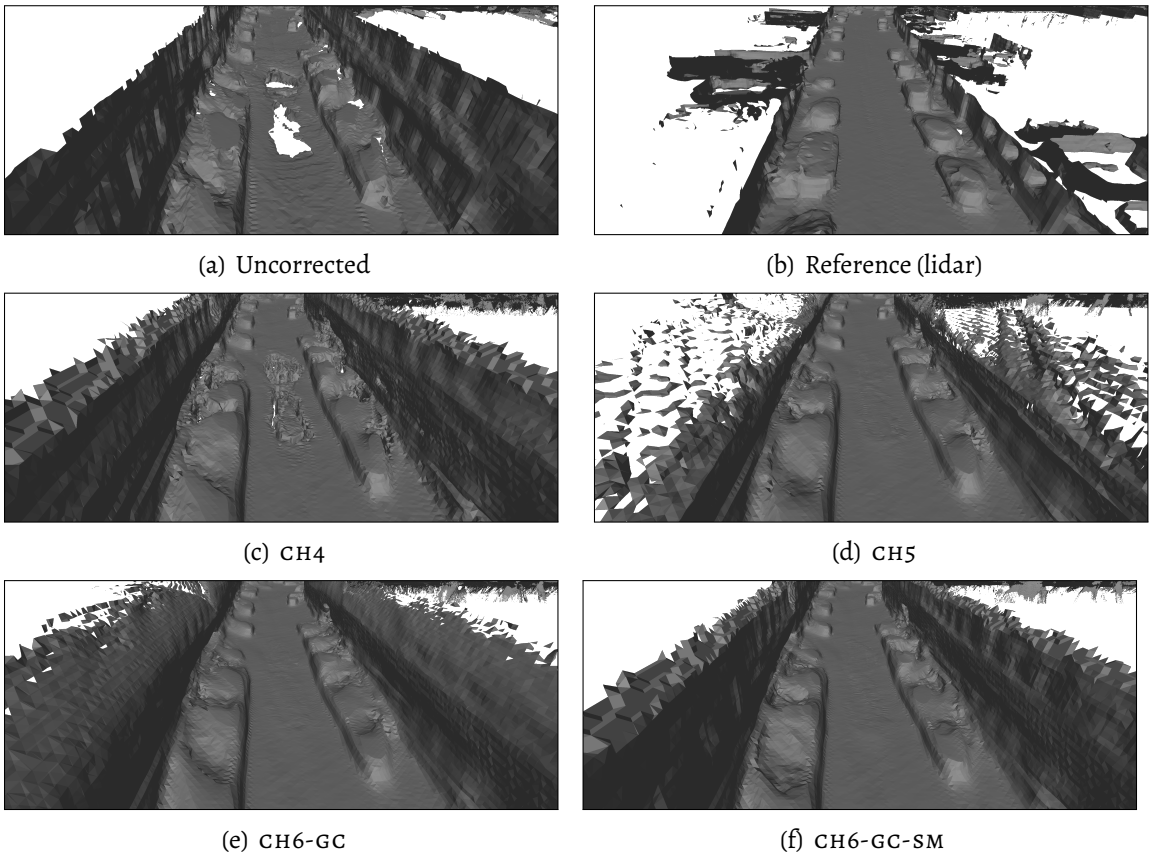


Figure 7.7: Change in surface area after meshes correction. The dashed line represents the difference in surface area between the reference lidar meshes and the uncorrected stereo-camera meshes. Points above this line indicate that the corrected meshes have more surface than the corresponding reference meshes, not all of which is necessary or useful.



(a) Uncorrected (b) Reference (lidar)

(c) CH4 (d) CH5

(e) CH6-GC (f) CH6-GC-SM

Figure 7.8: Qualitative results for models evaluated in this chapter. Our initial model (CH4, c) is unable to localise corrections well, so the holes seen in the uncorrected mesh (a) are not completely filled. The model trained with skip connections (CH5, d) correctly fills in these holes. The benefit of the geometric consistency loss that model CH6-GC (e) is trained with is largely quantitative, but we observe much more surface in the upper part of the mesh where the reference lidar reconstruction (b) used during training has no data. Finally, predicting a soft mask (CH6-GC-SM, f) allows for a degree of introspection, such that regions that were not supervised during training (the top of buildings) remain largely unchanged.

ence of the geometric consistency loss and of the predicted soft mask changes the predictions especially in areas where supervision is lacking, such as the top of the walls.

7.3 Future Work

While our mesh correction pipeline manages to reduce total error in meshes at large scale, it does so predominantly by recovering missing surface. However, this is done without much change in the correctness of generated meshes – for some sequences it is slightly improved, while for others there is a slight decrease.

Our future work will focus on improving correctness at the same time as completeness. One possible source of improvement comes from the clear dependence between geometry and semantics (e.g. roads are flat and horizontal, walls are flat and vertical). Currently, our model operates without any explicit notion of scene semantics, but could easily be extended to either use semantic maps as an input or as a second prediction task. Indeed, multi-task learning has been shown to improve performance (Baxter, 2000; Kokkinos, 2017), especially when combining semantics and geometry (Kendall, Gal, and Cipolla, 2018; S. Liu, Johns, et al., 2019).

7.4 Conclusion

In this chapter, we have demonstrated that our mesh correction method can be successfully used at a large scale by evaluating it on meshes constructed from 21.8 km of data. Our method reduces total error by 20.7–39.1% primarily by filling in missing surface in meshes built from stereo-camera data. This improves the completeness of our meshes but sometimes slightly decreases correctness. This observation revealed that one direction for future research is to focus more on improving both measures simultaneously by making the relationship between semantics and geometry explicit.

So far in this thesis we have been making predictions one view at a time – though during training we used a loss to promote geometrically consistent predictions in our best-preforming models. In the next chapter, we will explore how information can be effectively shared between views when making predictions. To that end, we propose a novel method for aggregating any number of views irrespective of viewpoint-dependent inputs.

8

Learning to Correct 3D Reconstructions from Multiple Views

In the previous chapters we have introduced and developed a system for correcting 3D meshes, and have demonstrated it in practice on a large scale scenario. We correct 3D meshes via multiple 2D mesh features rendered from the same mesh (as discussed in Section 3.3), so we expect predictions to be geometrically consistent – predictions on nearby views should result in the same geometry. We promote this by introducing a geometric consistency loss during training in Chapter 6, and show in Chapter 7 that such a loss benefits large scale scenarios.

However, when making predictions, views are still processed independently, with no information being shared between nearby views. Intuitively, sharing information should benefit predictions, ensuring consistency, so in this chapter we explore how multiple views can be aggregated effectively during inference.

We make the observation that processing view-dependent inputs, such as depth or normals, generally results in view-dependent representations within the network. Such dependence makes aggregation difficult between vastly different viewpoints. In this chapter, we address this by proposing a novel method of transforming view-dependent representations between views.

The following manuscript is a reproduction of the paper presented at the International Conference on Computer Vision Theory and Applications (VISAPP), 2021.

Learning to Correct 3D Reconstructions from Multiple Views

Keywords: Deep Learning, Multi-View Fusion, 3D Reconstruction, Mesh Correction

Abstract: This paper is about reducing the cost of building good large-scale 3D reconstructions post-hoc. This is an important consideration for survey vehicles which are equipped with sensors which offer mixed fidelity or are restricted by road rules to high-speed traversals. We render 2D views of an existing, lower-quality, reconstruction and train a convolutional neural network (CNN) that refines inverse-depth to match to a higher-quality reconstruction. Since the views that we correct are rendered from the same reconstruction, they share the same geometry, so overlapping views complement each other. We take advantage of that in two ways. Firstly, we impose a loss during training which guides predictions on neighbouring views to have the same geometry and has been shown to improve performance. Secondly, in contrast to our previous work, which corrects each view independently, we also make predictions on sets of neighbouring views jointly. This is achieved by warping feature maps between views and thus bypassing memory-intensive 3D computation. We make the observation that features in the feature maps are viewpoint-dependent, and propose a method for transforming features with dynamic filters generated by a multi-layer perceptron from the relative poses between views. In our experiments we show that this last step is necessary for successfully fusing feature maps between views.

1 INTRODUCTION

Building good dense 3D reconstructions is essential in many robotics tasks, such as surveying, localisation, or planning. Despite numerous advancements in both hardware and techniques, large-scale reconstructions remain costly to build.

We approach this issue by trying to reduce the data acquisition cost either through the use of cheaper sensors, or by collecting less data. To make up for the cheaper but lower quality data, we have to turn to prior information from the operational environment (e.g. roads and buildings are flat, cars and trees have specific shapes, etc). To learn these priors, we train a CNN over 2D views of 3D reconstructions, and predict refined inverse-depth maps that can be fused back into an improved 3D reconstruction. We take this detour through two dimensions in order to avoid the high memory requirements that a volumetric approach over large-scale reconstructions would impose.

While operating in 2D, neighbouring views are related by the underlying geometry. Previous work (Săftescu & Newman, 2020) has leveraged this relation during training, where a geometric consistency loss is imposed between neighbouring views that penalises mismatched geometry. Here, we improve the adeptness of this approach by showing how neighbouring views can be used together when *predicting* refined depth, and to that end introduce a method for aggregating feature maps in the CNN.

To fuse feature maps from multiple views, we

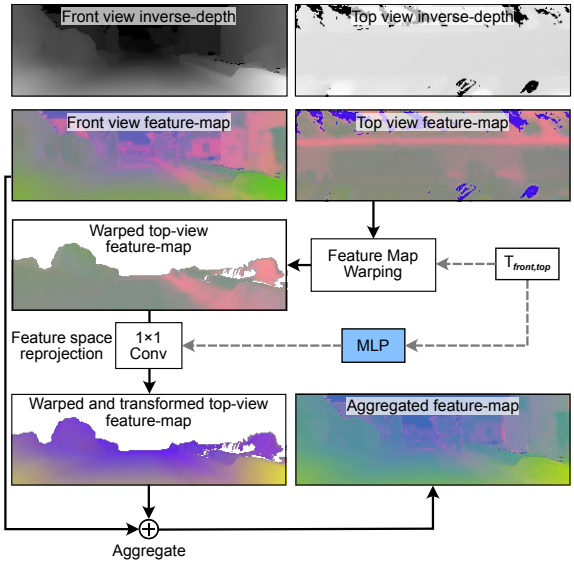


Figure 1: Feature map aggregation. In the top two rows we show inverse-depth images for a front view and a top view, along with two feature maps. To aggregate the feature map of the top view with the front view, we first warp the top feature map into the front view. The relative transform between the top and the front view, $\mathbf{T}_{front,top}$, is processed by a multi-layer perceptron (MLP) to generate the weights for a linear transform that maps the *features* from the top view to the front view. Finally, the resulting feature map can be aggregated with the front view feature map. Note that in the front feature map the features fade from green to violet towards the horizon, while in the warping of the top view, the features do not change with depth. Only after transforming the *features* do we see them fading towards the horizon. Analogously, we aggregate the front view feature map with the top view one. For visualisation, the above feature maps are projected to three channels using the same random projection.

could either "un-project" them into a common 3D volume or "collect" them into a common target view through reprojection, as proposed by Donne and Geiger (2019). As un-projecting into 3D re-introduces the limitation we wished to avoid, we take the latter approach.

Directly aggregating feature maps between views – either in a 3D volume or in a common target view – implies features are somewhat independent of viewpoint. To lift this restriction, this paper proposes a method for transforming *features* between views, enabling us to more easily aggregate feature maps from arbitrary viewpoints. Concretely, we use the relative pose between views to generate a projection matrix in feature space that can be used to transform feature maps, as illustrated in Figure 1.

The contributions of this paper are as follows:

1. We introduce a method for fusing multi-view data that decouples much of the multi-view geometry from model parameters. Not only do we warp feature maps between views, but we make the key observation that features *themselves* can be viewpoint dependent, and show how to transform the *feature space* between views.
2. We apply this method to the problem of correcting dense 3D meshes. We render 2D views from reconstructions and learn how to refine inverse-depth, while making use of multi-view information.

In our experiments we look at two ways of aggregating feature maps and conclude that the feature space transformation is necessary to benefit from the use of multiple views when correcting reconstructions.

2 RELATED WORK

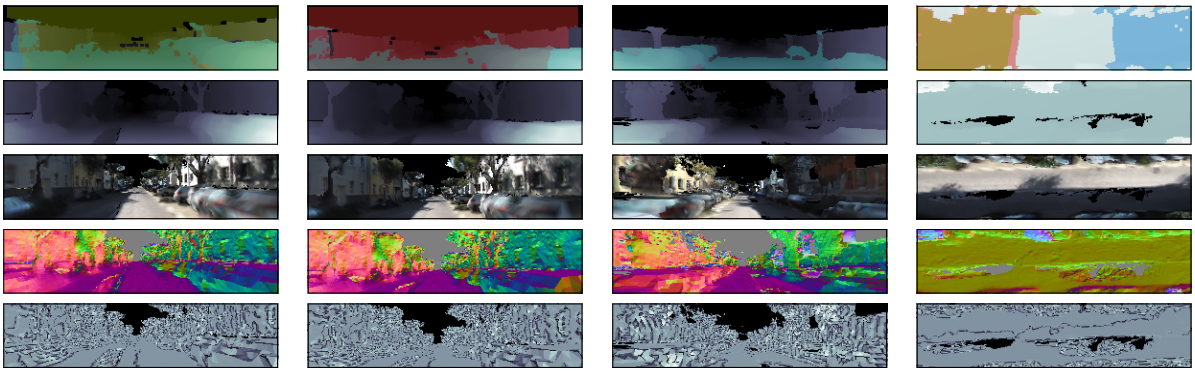
Our work focuses on refining the *output* of an existing 3D reconstruction system such as BOR^{2}G (Tanner et al., 2015) or KinectFusion (Newcombe et al., 2011), thus producing higher-quality reconstructions. Since we achieve this by operating on 2D projections and refining inverse-depth, our work is related to depth refinement. In the following we summarise some of the related literature and methods used in this work.

Mesh correction Tanner et al. (2018) first propose fixing 3D reconstructions by refining 2D projections of them with a CNN, one at a time. The geometrical relation between neighbouring views is leveraged by Săftescu and Newman (2020) during training, by the addition of a geometric consistency loss that penalises differences in geometry. In this work, we process neighbouring views jointly not only while training, but also when making predictions.

Learnt depth refinement and completion There are several depth refinement methods similar to our approach. Multiple depth maps are fused by Kwon et al. (2015) with KinectFusion (Newcombe et al., 2011) to obtain a high-quality reference mesh and use dictionary learning to refine raw RGB-D images. Using a CNN on the colour channels of an RGB-D image, Zhang and Funkhouser (2018) predict normals and occlusion boundaries and use them to optimise the depth component, filling in holes. In the method proposed by Jeon and Lee (2018), depth images are rendered from a reconstruction at the same locations as the raw depth, obtaining a 4000-image dataset of raw/clean depth image pairs. The authors train a CNN to refine the raw depth maps, and show that using it reduces the amount of data and time needed to build 3D reconstructions. All of these methods require a colour image in order to refine depth, and operate on live data, which limits the amount of training data available. In contrast, our method is designed to operate post-hoc, on existing meshes. We can therefore generate an arbitrary number of training pairs from any viewpoint, removing any viewpoint-specific bias that might otherwise surface while learning.

Another recent approach proposes depth refinement by fusing feature maps of neighbouring views through warping (Donne & Geiger, 2019). While this is similar to our approach, we take the additional step of transforming the *features* between views, and consider two feature aggregation methods.

Dynamic filter networks Generating filters for convolutions dynamically conditioned on network inputs is presented by Brabandere et al. (2016), where filters are predicted for local spatial transforms that help in video prediction tasks. Our feature transformation is also based on this framework: given a relative transform between two views, we predict the weights that would transform features from one view to another. A key distinction is that, while the filters in the original work are demonstrated over the spatial domain, we operate solely on the channels of the feature map with a 1×1 convolution.



(a) Left (b) Right (c) Back (d) Top

Figure 2: Example of training data generated. Each column represents a different view rendered around the same location. The top row shows the inverse-depth images rendered from the lidar reconstruction, with areas visible in the other views shaded: red for left, green for right, blue for back, and cyan for top. The next four rows show the mesh features we render from the stereo camera reconstruction: inverse-depth, colour, normals, and triangle surface area. Our proposed model learns to refine the low-quality inverse-depth (second row), using the rendered mesh features (rows 2–5) as input, processing all four views jointly, and supervised by the high-quality inverse-depth label (first row).

3 METHOD

3.1 Training Data

Our main goal is to correct *existing* dense 3D reconstructions. To bypass the need for expensive 3D computation, we operate on 2D projections of a mesh from multiple viewpoints. As we want to capture as much of the 3D geometry as possible in our projections, we render several mesh features for each viewpoint: inverse-depth, colour, normals, and triangle surface area (see Figure 2).

During training, we have access to two reconstructions of the same scene: a low-quality one that we learn to correct, and a high-quality one that we use for supervision. In particular, we learn to correct stereo-camera reconstructions using lidar reconstructions as supervision. Figure 3 shows an overview of our method.

The ground-truth labels, Δ, are computed as the difference in inverse-depth between high-quality and low-quality reconstructions:

$$\Delta(\mathbf{p}) = d^{hq}(\mathbf{p}) - d^{lq}(\mathbf{p}) \tag{1}$$

where $\mathbf{p}$ is a pixel index, and d^{hq} and d^{lq} are inverse-depth images for the high-quality and low-quality reconstruction, respectively. For notational compactness, $\Delta(\mathbf{p})$ is referred to as Δ, and future definitions are over all values of $\mathbf{p}$, unless otherwise specified.

There are several advantages to using inverse-depth. Firstly, geometry closer to the camera will have higher values and therefore these areas will be emphasised during training. Secondly, inverse-depth smoothly fades away from the camera, such that the background – which has no geometry and is infinitely far away – has a value of zero. If we were to use depth, we would have to treat background as a special case, since neural networks are not equipped to deal with infinite values out-of-the-box. Finally, when warping images from one viewpoint from another, as described in the following sections, we are in essence re-sampling. To correctly interpolate depth values, we would have to use harmonic mean, which is less numerically stable, whereas interpolating inverse-depth can be done linearly.

3.2 Image Warping

During both training and prediction, we need to fuse information from neighbouring views. While training, we want to penalise the network for making predictions that are geometrically inconsistent between views, using the geometric consistency loss from Săftescu and Newman (2020), described in Section 3.3.4. When making predictions, we want to be able to aggregate information from multiple views. To enable this, we need to warp images between viewpoints, such that corresponding pixels are aligned.

Consider a view t, an inverse-depth image d_t, and a pixel location within the image, $\mathbf{p}_t = [u\ v\ 1]^T$. The homogeneous 3D point corresponding to $\mathbf{p}_t$ is:

$$\mathbf{x}_t = \begin{bmatrix} \mathbf{K}^{-1}\mathbf{p}_t \\ d_t(\mathbf{p}_t) \end{bmatrix}, \tag{2}$$

where $\mathbf{K}$ is the camera intrinsic matrix. Consider further a nearby view n, an image I_n, and the relative transform $\mathbf{T}_{n,t} \in \mathrm{SE}(3)$ from t to n that maps 3D points between views: $\mathbf{x}_n = \mathbf{T}_{n,t}\mathbf{x}_t$. The pixel in view n corre-

sponding to $\mathbf{p}_t$ is:

$$\mathbf{p}_n = \mathbf{K} \frac{\mathbf{x}_n^{(1:3)}}{\mathbf{x}_n^{(3)}}, \tag{3}$$

where a superscript indexes into the vector $\mathbf{x}_n$.

We can now *warp* image I_n into view t:

$$I_{t \leftarrow n}(\mathbf{p}_t) = I_n(\mathbf{p}_n). \tag{4}$$

Note here that $\mathbf{p}_n$ might not have integer values, and therefore might not lie exactly on the image grid of I_n. In that case, we linearly interpolate the nearest pixels.

Since the value of inverse-depth is view-dependent, when warping inverse-depth images we make the following additional definition:

$$\widetilde{d_{t;n}}(\mathbf{p}_t) = \frac{\mathbf{x}_n^{(4)}}{\mathbf{x}_n^{(3)}}, \tag{5}$$

which represents an image aligned with view t, with inverse-depth values in the frame of n.

Occlusions Pixel correspondences computed through warping are only valid where there are no occlusions. We therefore have need of a mask to only take into account unoccluded regions. Therefore, when rendering mesh features, we also render an additional image where every pixel is assigned the ID of the visible mesh triangle at that location. The triangle ID is computed by hashing the global coordinates of its vertices. We can then warp this image of triangle IDs from the source to the target view. If the ID of a pixel matches between the warped source and the target image, we know that the same surface is in view in both images – and thus that pixel is unoccluded.

3.3 Network architecture

3.3.1 Model

We use an encoder-decoder architecture, with asymmetric ResNet (He et al., 2016) blocks, the sub-pixel convolutional layers proposed by Shi et al. (2016) for upsampling in the decoder, and skip connections between the encoder and the decoder to improve sharpness, as introduced by the U-Net architecture (Ronneberger et al., 2015). Throughout the network, we use ELU (Clevert et al., 2015) activations and group normalisation (Wu & He, 2018). Table 1 details the blocks used in our network. We use F = 8 input channels: 3 for colour, 3 for normals, 1 for inverse-depth, and 1 for triangle area.

Since a fair portion of the input low-quality reconstruction is already correct, we train our model to predict the error in the input inverse-depth, Δ^*. We then compute the refined inverse depth as the output of our network:

$$d^* = \max(d^{lq} + \Delta^*, 0). \tag{6}$$

Clipping is required here because inverse-depth cannot be negative. However, since we are supervising the predicted error, the network can learn even when the predicted inverse-depth is clipped and would therefore lack a gradient. To ensure our network can deal with any range of inverse-depth, we offset the input such that it has zero mean, scale it to have standard deviation of 1, and undo the scaling on the predicted error, Δ^*.

Table 1: Overview of the CNN architecture for error prediction

Block Type	**Filter Size / Stride**	**Output Size**
Input	-	$96 \times 288 \times$ F
Projection	$7 \times 7/1$	$96 \times 288 \times$ 16
Residual	$3 \times 3/1$	$96 \times 288 \times$ 16
Projection, Residual$\times 2$	$3 \times 3/2$	$48 \times 144 \times$ 32
Projection, Residual$\times 2$	$3 \times 3/2$	$24 \times 72 \times$ 64
Projection, Residual$\times 2$	$3 \times 3/2$	$12 \times 36 \times 128$
Projection, Residual$\times 5$	$3 \times 3/2$	$6 \times 18 \times 256$
Up-projection, Skip	$3 \times 3/\frac{1}{2}$	$12 \times 36 \times 384$
Up-projection, Skip	$3 \times 3/\frac{1}{2}$	$24 \times 72 \times 192$
Up-projection, Skip	$3 \times 3/\frac{1}{2}$	$48 \times 144 \times$ 96
Up-projection, Skip	$3 \times 3/\frac{1}{2}$	$96 \times 288 \times$ 48
Residual$\times 2$	$3 \times 3/1$	$96 \times 288 \times$ 48
Convolution	$3 \times 3/1$	$96 \times 288 \times$ 1

3.3.2 Feature Map Warping and Aggregation

As our predictions are related by the 3D geometry of a scene, we must would like to ensure predictions are consistent between views. This is taken into account during training by using the geometric consistency loss from Săftescu and Newman (2020), as described in Section 3.3.4.

However, during inference we would like to aggregate information from multiple views to improve predictions. Take, for example, two views, t and n, and feature maps in the network, F_t and F_n, after a certain number of layers, corresponding to each of the views. We would like to aggregate them such that $F_t \oplus F_n$ is a feature map containing information from both views. Since the feature maps are aligned with the input views, we cannot do that in a pixel-wise fashion. Using the input depth, we warp the feature map of one view into the frame of the other, such that the input geometry is aligned. For the two input views, we can thus warp F_n to the viewpoint of t (according to Equation 4) and then aggregate the two feature maps:

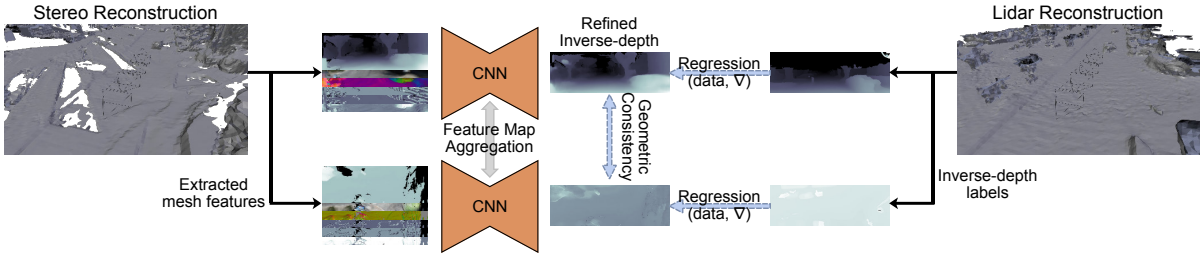


Figure 3: Illustration of our training set-up. Starting with a low-quality reconstruction (stereo camera in this instance), we extract mesh features from several viewpoints. Our network learns to refine the input inverse-depth with supervision from a high-quality (lidar) reconstruction. The blue arrows above indicate the losses used during training: for each view, we regress to the high-quality inverse-depth, as well as to its gradient; between nearby prediction, we apply a geometric consistency loss to encourage predictions with the same geometry. Within the network, feature maps are aggregated between views so information can propagate within a neighbourhood of views. Adapted with permission from Săftescu and Newman (2020).

$F_{t,n} = F_t \oplus F_{t \leftarrow n}$, obtaining a feature map aligned with t that combines information from both views.

This aggregation step is necessary (instead of simply concatenating aligned feature maps) to allow for an arbitrary number of views. For the same reason, the aggregation function needs to be invariant to permutations of views. We consider two such aggregation functions: averaging, and attention. For a target view t with neighbourhood $N = \{t\} \cup \{n_1, n_2, ...\}$, averaging is defined as $\sum_{n \in N} F_{t \leftarrow n} / |N|$, where we consider $F_{t \leftarrow t} \equiv F_t$.

For the attention-based aggregation, we use the attention method proposed by Bahdanau et al. (2014). A per-pixel score $E_{t,n} = a(F_t, F_{t \leftarrow n})$ is computed using a small 3-layer convolutional sub-network. The per-pixel weight of each view is obtained by applying softmax to the scores: $A_{t,n} = \exp(E_{t,n}) / \sum_{m \in N} \exp(E_{t,m})$. Finally, the aggregation function becomes $\sum_{n \in N} A_{t,n} \cdot F_{t \leftarrow n}$. In both cases, pixels deemed occluded are masked out.

In our model, we apply this warping and aggregation to every skip connection and to the encoder output, thus mixing information across views and scales. For every view in a batch, we aggregate all the other overlapping views within the batch.

3.3.3 Feature Space Reprojection

When the network is trained to make predictions one input view at a time, the feature maps at intermediate layers within the network contain viewpoint-dependent features, as illustrated in Figure 1. Warping a feature map from one view to another only aligns features using the scene geometry, but does not change their dependence on viewpoint.

Imagine, for example, two RGB-D images in an urban scene – one from above looking down, and one at the road level. The surface of the road will have the same colour in both views. The depth of the road, however, will be different: in the top view, it will be mostly uniform, while in the road-level view it will increase towards the horizon. We can easily establish correspondences between pixels in the two images if we know their relative pose. When not occluded, corresponding pixels will have the same colour, but not necessarily the same depth. This is because colour is view-point independent, while depth depends on the viewpoint. In other words, we can resample the colour from one view to obtain the other view by directly indexing and interpolating (as in Equation 4), while for depth we also need to transform the *values* of the first view to match the reference frame of the second view (as in Equation 5). Armed with this observation we propose a way to learn a mapping of *features* between viewpoints.

Consider again two views t and n, and a warped feature map $F_{t \leftarrow n}$ from the vantage point n. Given a spatial location $\mathbf{p}$, we have a feature vector $\mathbf{f}_n = F_{t \leftarrow n}(\mathbf{p}) \in \mathbb{R}^{\mathrm{D}}$, its corresponding location in space $\mathbf{x}_n = (x, y, z, w) \in \mathbb{P}^3$, and a transform $\mathbf{T}_{t,n} \in \mathrm{SE}(3)$. We compute a matrix $\mathbf{W} = g(\mathbf{T}_{t,n}) \in \mathscr{M}_{\mathrm{D,D+4}}$, using a small multi-layer perceptron (MLP) to model g. This allows us to learn a linear transform in *feature space* between view n and view t:

$$\mathbf{f}_t = \mathbf{W} \begin{bmatrix} \mathbf{f}_n \\ \mathbf{x}_n \end{bmatrix}. \tag{7}$$

Without this mechanism, the network would have to learn to extract viewpoint-independent features to allow for feature aggregation between views.

Concretely, we implement this as a dynamic filter network (DFN), with a 4-layer MLP generating filters for a 1×1 linear convolution of the warped feature map $F_{t \leftarrow n}$, which is equivalent to applying the multiplication in Equation 7 at every spatial location in the feature map. To keep the MLP small, we first project the input feature maps to $\mathrm{D} = 32$ dimensions, apply the dynamically generated filters, and then project back to the desired number of features. Both projections are implemented using 1×1 convolutions. We use the



(a) Sequence 00 (b) Sequence 05 (c) Sequence 06

Figure 4: Illustration of the training (blue), validation (orange), and test (green) splits on the three KITTI-VO sequences we are using. Map data copyrighted OpenStreetMap contributors (2017) and available from https://www.openstreetmap.org.

same transformation $\mathbf{W}$ across each of the the scales that we aggregate, and all the weights associated with feature transform operations are learnt jointly with the rest of the weights of the network.

In the experiments, we show that this mechanism is essential for enabling effective multi-view aggregation.

3.3.4 Losses

We supervise our training with labels from a high-quality reconstruction, as sown in Figure 3. The labels provide two per-pixel supervision signals, one for direct regression, $\mathcal{L}^{data}$, and one for prediction gradients, $\mathcal{L}^{\nabla}$:

$$\mathcal{L}^{data} = \sum_{\mathbf{p}\in V} \|\Delta^* - \Delta\|_{berHu}; \tag{8}$$

$$\mathcal{L}^{\nabla} = \frac{1}{2} \sum_{\mathbf{p}\in V} \left(|\partial_x \Delta^* - \partial_x \Delta| + |\partial_y \Delta^* - \partial_y \Delta| \right), \tag{9}$$

where V is the set of valid pixels (to account for missing data in the ground-truth), Δ^* and Δ are the prediction and the target, respectively, and $\|\cdot\|_{berHu}$ is the berHu norm (Owen, 2007), whose advantages for depth prediction have been explored by Laina et al. (2016), Ma and Karaman (2018). We use the Sobel operator (Sobel & Feldman, 1968) to approximate the gradients in Equation 9.

The geometric consistency loss guides nearby predictions to have the same 3D geometry, and relies on warped nearby views $d^*_{t\leftarrow n}$. For a target view t, a set of nearby views N, the set of pixels unoccluded in a nearby view U_n (see Figure 2, top row), this loss is defined as:

$$\mathcal{L}^{gc} = \sum_{n\in N} \sum_{p\in U_n} \left| d^*_{t\leftarrow n} - \widetilde{d^*_{t;n}} \right|. \tag{10}$$

Both $d^*_{t\leftarrow n}$ and $\widetilde{d^*_{t;n}}$ are aligned with view t and contain inverse-depth values in frame n, as per Equation 4 and Equation 5. Note that U_n has no relation to the set of valid pixels (V) from the previous losses, since this loss is only computed between predictions. This enables the network to make sensible predictions even in parts of the image which have no valid label.

Finally, we also include an L_2 weight regulariser, $\mathcal{L}^{reg}$, to reduce overfitting by keeping the weights small. The overall objective is thus defined as:

$$\mathcal{L} = \lambda^{data}\mathcal{L}^{data} + \lambda^{\nabla}\mathcal{L}^{\nabla} + \lambda^{gc}\mathcal{L}^{gc} + \lambda^{reg}\mathcal{L}^{reg}, \tag{11}$$

where the λs are weights for each of the components. We use $\lambda^{data} = 1$, $\lambda^{\nabla} = 0.1$, $\lambda^{gc} = 0.1$, and $\lambda^{reg} = 10^{-6}$.

4 EXPERIMENTS

4.1 Experimental Setup

4.1.1 Dataset

For the experiments, we use sequences "00", "05", and "06" from the KITTI visual odometry (KITTI-VO) dataset (Geiger et al., 2013). Using the BOR^{2}G reconstruction system (Tanner et al., 2015), we create pairs of low/high quality reconstructions (meshes) from the stereo camera, and lidar, respectively. Following the same trajectory used when collecting data (as it is collision-free), every 0.65 m we render mesh features from four views (left, right, back, top), illustrated in Figure 2. For each view, we render a further 3 samples with small pose perturbations for data augmentation. In total, we obtain 178 544 distinct views of size 96×288 over 7.2 km.

4.1.2 Training and inference

We train all our models on Nvidia Titan V GPUs, using the Adam optimiser (Kingma & Ba, 2015), with $\beta_1 = 0.9$, $\beta_2 = 0.999$, and a learning rate that decays linearly from 10^{-4} to $5 \cdot 10^{-6}$ over 120 000 training steps. We clip the gradient norm to 80. Each training batch contains 4 different examples, and each example is composed of the four views rendered around a single location. Unless otherwise mentioned, we train our

Table 2: Depth Error Correction Results

Model	iMAE	iRMSE	$\delta < 1.05$	$\delta < 1.15$	$\delta < 1.25$	$\delta < 1.56$	$\delta < 1.95$
Uncorrected	$1.21 \cdot 10^{-2}$	$3.80 \cdot 10^{-2}$	51.92%	82.44%	86.66%	90.54%	91.97%
Baseline	$7.09 \cdot 10^{-3}$	$2.47 \cdot 10^{-2}$	73.81%	89.33%	93.08%	96.71%	97.98%
Ours (no feat. tf.; avg.)	$7.11 \cdot 10^{-3}$	$2.45 \cdot 10^{-2}$	72.70%	89.08%	92.87%	96.53%	97.86%
Ours (no feat. tf.; attn.)	$7.22 \cdot 10^{-3}$	$2.52 \cdot 10^{-2}$	73.60%	89.18%	92.90%	96.57%	97.87%
Ours (w/ feat. tf.; avg.)	$\mathbf{6.82 \cdot 10^{-3}}$	$\mathbf{2.38 \cdot 10^{-2}}$	*74.22%*	**90.01%**	**93.51%**	*96.79%*	*98.01%*
Ours (w/ feat. tf.; attn.)	*$6.88 \cdot 10^{-3}$*	*$2.40 \cdot 10^{-2}$*	**74.34%**	*89.85%*	*93.47%*	**96.89%**	**98.10%**

Table 3: Generalisation Capability of Depth Error Correction

Model	Train	Test	iMAE	iRMSE	$\delta < 1.05$	$\delta < 1.15$	$\delta < 1.25$	$\delta < 1.56$	$\delta < 1.95$
Uncorrected	–	06	$2.43 \cdot 10^{-2}$	$8.39 \cdot 10^{-2}$	49.63%	72.89%	75.07%	78.48%	80.23%
Baseline	00; 05	06	$2.09 \cdot 10^{-2}$	$8.21 \cdot 10^{-2}$	59.99%	79.16%	82.62%	86.33%	88.54%
Ours (no feat. tf.)	00; 05	06	$2.06 \cdot 10^{-2}$	$8.11 \cdot 10^{-2}$	61.57%	77.95%	81.46%	85.45%	87.88%
Ours (w/ feat. tf.)	00; 05	06	$1.94 \cdot 10^{-2}$	$6.80 \cdot 10^{-2}$	63.90%	78.80%	82.00%	85.80%	88.04%
Uncorrected	–	05	$1.15 \cdot 10^{-2}$	$3.59 \cdot 10^{-2}$	55.85%	81.60%	85.07%	88.86%	90.90%
Baseline	00; 06	05	$7.76 \cdot 10^{-3}$	$2.98 \cdot 10^{-2}$	72.23%	87.54%	91.23%	95.46%	97.41%
Ours (no feat. tf.)	00; 06	05	$7.79 \cdot 10^{-3}$	$2.97 \cdot 10^{-2}$	71.57%	87.20%	90.94%	95.31%	97.34%
Ours (w/ feat. tf.)	00; 06	05	$7.65 \cdot 10^{-3}$	$2.96 \cdot 10^{-2}$	73.24%	87.61%	91.16%	95.34%	97.29%
Uncorrected	–	00	$1.19 \cdot 10^{-2}$	$3.24 \cdot 10^{-2}$	53.53%	80.46%	85.18%	89.55%	91.38%
Baseline	05; 06	00	$8.40 \cdot 10^{-3}$	$2.50 \cdot 10^{-2}$	67.52%	86.83%	91.28%	95.85%	97.70%
Ours (no feat. tf.)	05; 06	00	$8.32 \cdot 10^{-3}$	$2.49 \cdot 10^{-2}$	66.04%	86.65%	91.34%	95.80%	97.64%
Ours (w/ feat. tf.)	05; 06	00	$8.40 \cdot 10^{-3}$	$2.53 \cdot 10^{-2}$	67.90%	87.02%	91.39%	95.83%	97.64%

models for 500 000 steps. During inference, our full model runs at 11.3 Hz when aggregating 4 input views, compared to the baseline that runs at 12.6 Hz, so our method comes with little computational overhead.

4.1.3 Metrics

As our method operates on 2D views extracted from the mesh we are correcting, we measure how well our network predicts inverse-depth images, with the idea that better inverse-depth images result in better reconstructions. We employ several metrics common in the related tasks of depth prediction and refinement.

One way to quantify performance is to see how often the error in prediction is small enough to be correct. The thresholded accuracy measure is essentially the expectation that a given pixel is within a fraction *thr* of the label:

$$\delta = \mathbb{E}_{p \in V}\left[\mathbb{I}\left(\max\left(\frac{d^{hq}}{d^*}, \frac{d^*}{d^{hq}}\right) < thr\right)\right], \quad (12)$$

where d^{hq} is the reference inverse-depth, d^* is the predicted inverse depth, V is the set of valid pixels, n is the cardinality of V, and $\mathbb{I}(\cdot)$ represents the indicator function. For granularity, we use $thr \in \{1.05, 1.15, 1.25, 1.25^2, 1.25^3\}$.

In addition, we also compute the mean absolute error (MAE) and root mean square error (RMSE) metrics to quantify per pixel error:

$$\text{iMAE} = \frac{1}{n}\sum_{p \in V}\left|d^* - d^{hq}\right|, \quad (13)$$

$$\text{iRMSE} = \sqrt{\frac{1}{n}\sum_{p \in V}(d^* - d^{hq})^2}, \quad (14)$$

where the 'i' indicates that the metrics are computed over inverse-depth images.

4.2 Gross Error Correction

For the first set of experiments, we take the first 80% of the views from each sequence as training data, the next 10% for validation, and show our results on the last 10%. An illustration of the KITTI sequences and splits is shown in Figure 4.

As baseline, we train our model with geometric consistency loss but without any feature aggregation. During inference, this model makes predictions one view at a time.

To illustrate our method, we train a further two models for each aggregation method (averaging and attention): one with the feature transform disabled, and one with it enabled.

As it can be seen in Table 2, the baseline already refines inverse-depth significantly. Without our feature

transformation, the models are unable to use multi-view information because of the vastly different viewpoints, and indeed this slightly hurts performance. Only when transforming the features between viewpoints does the performance increase over the baseline, highlighting the importance of our method for successfully aggregating multiple views.

4.3 Generalisation

To asses the ability of our method to generalise on unseen reconstructions, we divide our training data by sequence: we use two of the sequences for training, and the third for testing. Sequences 00 and 05 are recorded in a suburban area with narrow roads, while sequence 06 is a loop on a divided road with a median strip, a much wider and visually distinct space. We train models for 200 000 steps and aggregate feature maps by averaging. The results in Table 3 show that our method successfully uses information from multiple views, even in areas of a city different from the ones it was trained on. Furthermore, they reaffirm the need for our feature transformation method in addition to warping.

5 CONCLUSION AND FUTURE WORK

In conclusion, we have presented a new method for correcting dense 3D reconstructions via 2D *mesh feature* renderings. In contrast to previous work, we make predictions on multiple views at the same time by warping and aggregating feature maps inside a CNN. In addition to warping the feature maps, we also transform the *features* between views and show that this is necessary for using arbitrary viewpoints.

The method presented here aggregates feature maps between every pair of overlapping input views. This scales quadratically with the number of views and thus limits the size of the neighbourhood we can reasonably process. Future work will consider aggregation into a shared 2D spatial representation, such as a 360° view, which would scale linearly with the input neighbourhood size.

While in this paper we have applied our method to correct stereo reconstructions using lidar as high-quality supervision, our approach operates strictly on meshes, so it is agnostic to the types of sensors used to produce the low- and high-quality reconstructions, so long as it is trained accordingly.

References

Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate, In *Proceedings of The International Conference on Learning Representations (*ICLR*)*.

Brabandere, B. D., Jia, X., Tuytelaars, T., & Gool, L. V. (2016). Dynamic filter networks, In *Proceedings of The Conference on Neural Information Processing Systems (*NIPS*)*.

Clevert, D.-A., Unterthiner, T., & Hochreiter, S. (2015). Fast and accurate deep network learning by exponential linear units (ELUs), In *Proceedings of The International Conference on Learning Representations (*ICLR*)*.

Donne, S., & Geiger, A. (2019). Defusr: Learning non-volumetric depth fusion using successive reprojections, In *Proceedings of The IEEE/CVF International Conference on Computer Vision and Pattern Recognition (*CVPR*)*.

Geiger, A., Lenz, P., Stiller, C., & Urtasun, R. (2013). Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, *32*(11), 1231–1237.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition, In *Proceedings of The IEEE/CVF International Conference on Computer Vision and Pattern Recognition (*CVPR*)*.

Jeon, J., & Lee, S. (2018). Reconstruction-based pairwise depth dataset for depth image enhancement using CNN, In *Proceedings of The European Conference on Computer Vision (*ECCV*)*.

Kingma, D., & Ba, J. (2015). Adam: A method for stochastic optimization, In *Proceedings of The International Conference on Learning Representations (*ICLR*)*.

Kwon, H., Tai, Y.-W., & Lin, S. (2015). Data-driven depth map refinement via multi-scale sparse representation, In *Proceedings of The IEEE/CVF International Conference on Computer Vision and Pattern Recognition (*CVPR*)*.

Laina, I., Rupprecht, C., Belagiannis, V., Tombari, F., & Navab, N. (2016). Deeper depth prediction with fully convolutional residual networks, In *Proceedings of The IEEE International Conference on 3d Vision (*3DV*)*.

Ma, F., & Karaman, S. (2018). Sparse-to-dense: Depth prediction from sparse depth samples and a single image, In *Proceedings of The IEEE International Conference on Robotics and Automation (*ICRA*)*.

Newcombe, R. A., Izadi, S., Hilliges, O., Molyneaux, D., Kim, D., Davison, A. J., Kohli, P., Shotton, J., Hodges, S., & Fitzgibbon, A. W. (2011). KinectFusion: Real-time dense surface mapping and tracking, In *Proceedings of The IEEE International Symposium on Mixed and Augmented Reality (*ISMAR*)*.

OpenStreetMap contributors. (2017). Planet dump retrieved from https://planet.osm.org.

Owen, A. B. (2007). A robust hybrid of lasso and ridge regression. *Contemporary Mathematics*, *443*, 59–72.

Ronneberger, O., Fischer, P., & Brox, T. (2015). U-Net: Convolutional networks for biomedical image segmentation, In *Medical Image Computing and Computer Assisted Intervention (*MICCAI*)*.

Săftescu, S., & Newman, P. (2020). Learning geometrically consistent mesh corrections, In *Proceedings of The International Conference on Computer Vision Theory and Applications (*VISAPP*)*.

Shi, W., Caballero, J., Huszár, F., Totz, J., Aitken, A. P., Bishop, R., Rueckert, D., & Wang, Z. (2016). Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network, In *Proceedings of The IEEE/CVF International Conference on Computer Vision and Pattern Recognition (*CVPR*)*.

Sobel, I., & Feldman, G. (1968). A 3×3 isotropic gradient operator for image processing.

Tanner, M., Piniés, P., Paz, L. M., & Newman, P. (2015). BOR^{2}G: Building optimal regularised reconstructions with GPUs (in cubes), In *Proceedings of The International Conference on Field and Service Robotics (*FSR*)*, Toronto, Canada.

Tanner, M., Săftescu, S., Bewley, A., & Newman, P. (2018). Meshed up: Learnt error correction in 3d reconstructions, In *Proceedings of The IEEE International Conference on Robotics and Automation (*ICRA*)*, Brisbane, Australia.

Wu, Y., & He, K. (2018). Group normalization, In *Proceedings of The European Conference on Computer Vision (*ECCV*)*.

Zhang, Y., & Funkhouser, T. A. (2018). Deep depth completion of a single RGB-D image, In *Proceedings of The IEEE/CVF International Conference on Computer Vision and Pattern Recognition (*CVPR*)*.

Statement of Authorship for joint/multi-authored papers for PGR thesis

To appear at the end of each thesis chapter submitted as an article/paper

The statement shall describe the candidate's and co-authors' independent research contributions in the thesis publications. For each publication there should exist a complete statement that is to be filled out and signed by the candidate and supervisor **(only required where there isn't already a statement of contribution within the paper itself).**

Title of Paper	Learning to correct 3D reconstructions from multiple views
Publication Status	☐ Published ☐ Accepted for Publication ☒ Submitted for Publication ☐ Unpublished and unsubmitted work written in a manuscript style
Publication Details	**Săftescu, Ș.**, & Newman, P. (2020). Learning to correct 3D reconstructions from multiple views. arXiv: 2001.08098 [cs.CV].

Student Confirmation

Student Name:	Ștefan Săftescu		
Contribution to the Paper	All work except editorial changes and advice, including: • Generation of ideas • Software development • Designed and performed experiments • Wrote the paper		
Signature	Ș. Săftescu	Date	29/08/2020

Supervisor Confirmation

By signing the Statement of Authorship, you are certifying that the candidate made a substantial contribution to the publication, and that the description described above is accurate.

Supervisor name and title:	Professor Paul Newman		
Supervisor comments			
Signature	P. Newman	Date	10/09/2020

This completed form should be included in the thesis, at the end of the relevant chapter.

8.1 Extensions and Future Work

In in this chapter we introduce a transform of network features that is predicted from the relative pose between views. In the article, there are no constraints imposed on the transform and no structure beyond dimensionality. A natural direction to extend this work in is to impose (by design) or promote (through losses, during training) additional structure for the learnt transform.

To guide the choice of structure we might want the transform to have, we must look at how we interpret it. Consider, again, our definition of the feature transform, $\mathbf{W}_{a,b} = f(\mathbf{T}_{a,b})$, and the interpretation we gave to it that it maps view-dependent features between viewpoints – acting like an adaptor between the feature maps of different viewpoints. Under such an interpretation, we would like our transform to have at least the following two properties:

1. $\mathbf{W}_{a,b} \cdot \mathbf{W}_{b,c} = \mathbf{W}_{a,c}$: Transforming features from view c to view b and then from view b to a is the same as applying the transform directly from view c to a;

2. $\mathbf{W}_{a,a} = \mathbf{I}_D$: If the source and target viewpoints are the same then the features do not need to be "adapted". This naturally follows from the first property.

If f is a group homomorphism, then these two desired properties of the feature transform are implied by the structure of the rigid transform space, $\mathrm{SE}(3)$. We would therefore like f to have the following property:

$$f(\mathbf{T}_a) \cdot f(\mathbf{T}_b) = f(\mathbf{T}_a \cdot \mathbf{T}_b), \qquad \forall\, \mathbf{T}_a, \mathbf{T}_b \in \mathrm{SE}(3). \tag{8.1}$$

Concretely, this could be implement by applying a loss that guides the homomorphism structure of f (an MLP) during training, as follows. For an input training example consisting of multiple views and relative transforms between them, $\mathbf{T}_i$, we randomly sample N pairs $(\mathbf{T}_{a_n}, \mathbf{T}_{b_n})$. Following Equation (8.1), we compute a loss for the pairs of transforms. Finally, for additional supervision, we encourage the image of the identity element of $\mathrm{SE}(3)$ under f to be $\mathbf{I}_D$:

$$\mathcal{L}^{morph} = \frac{1}{2N} \sum_{n=1}^{N} \left| f(\mathbf{T}_{a_n}) \cdot f(\mathbf{T}_{b_n}) - f(\mathbf{T}_{a_n} \cdot \mathbf{T}_{b_n}) \right|_F + \frac{1}{2} \left| f(\mathbf{I}) - \mathbf{I}_D \right|_F , \tag{8.2}$$

where $|\cdot|_F$ is the Frobenius norm. The two components of the loss correspond to the two properties desired for the feature transform.

If only the homomorphism loss were considered when training f, it would quickly lead to the trivial solution $f(\mathbf{T}) = \mathbf{I}_D$, which lies at one of the global minima of the loss function. However, since we would train f at the same time as the rest of the network, this trivial solution is counterbalanced by the main training objective, and the homomorphism loss can be seen as a way of reducing overfitting by imposing additional structure to the predicted feature transform.

Currently, the feature transform is only trained on a small number of input views as well as small random perturbations thereof, with no guarantee that it will generalise to views not encountered during training. One way this might be overcome is to train f additionally on random samples from $\mathrm{SE}(3)$. To avoid the trivial solution of $f(\mathbf{T}) = \mathbf{I}_D$ for views not seen during training, we would ensure the injectivity of f by adding another trainable function $g = f^{-1}$ in an auto-encoder setup.

While we presented the feature transform as a method to improve mesh correction with CNNs, we believe this novel approach is applicable in many other tasks as a general-purpose method for aggregating multiview data in CNNs. In future work, we will explore further applications of our method, such as depth estimation, 3D reconstruction, or localisation.

8.2 Conclusion

In this chapter we have presented a novel neural network architecture for aggregating information from multiple views. We achieved this by transforming the feature maps in a CNN between viewpoints with a learnt linear transform derived from the relative pose of the viewpoints. We have proposed an extension to the training method that promotes additional structure for this transform and we believe will improve generalisation. In our future work we will investigate to what degree the proposed extension benefits our method and to what other applications the proposed architecture can be applied.

In the next chapter we will conclude this thesis with a summary of the key ideas, contributions and future research directions.

9

Summary

This thesis has presented a system that reduces the cost of building good 3D reconstructions by correcting meshes. In this chapter we summarise the key ideas developed throughout the thesis and present several avenues for future research.

As discussed in Chapter 1, building high-quality reconstructions is important to several fields. Good 3D reconstructions are essential for digital preservation of cultural heritage sites and artefacts, yet current methods are very costly. In mobile robotics, dense maps are used for a variety of tasks, and better maps lead to better robots. This, in particular, has been the primary motivation in our work.

In Chapter 2, we provided a summary of the methods and theory necessary to develop our mesh correction system, whose components we detailed in Chapter 3.

Chapter 4 introduced our core idea, a CNN model that learns a mapping from rasterised geometric and appearance features to the error in a low-quality mesh. The method presented in this chapter improved rasterised depth images by up to 10% RMSE. Chapter 5 addresses several limitations of the method proposed in Chapter 4. We demonstrated how our mesh correction method can be used in practice in a 3D reconstruction system and showed that it recovered up to 36% of the missing surface. To achieve this, we handled overly smooth predictions by adding U-Net-style (Ronneberger, Fischer, et al., 2015) skip connections to our model, thus allowing predictions to be better localised.

One of the major challenges in supervised machine learning is the limited availability of data. In our case, the method presented in Chapters 4 and 5 required real colour images as input in addition to mesh features – thus limiting the amount of available training data. In Chapter 6 we replaced real colour images with synthetic ones rendered in the same way as other mesh features. Without impact to our method, this enabled us to generate vastly more data and to render entirely novel views that had not been observed during data collection. With arbitrary viewpoints available, we ensured that predictions on neighbouring viewpoints result in the same geometry by introducing a geometric consistency loss during training, calculated over sets of views. Furthermore, to allow for a degree of introspection and to account for missing labels in the training data, we modified our model to predict a soft mask in addition to the corrected inverse-depth, which resulted in better performance in unknown regions.

In Chapter 7 we evaluated our mesh correction system on 21.8 km of driving data, comparing our different contributions in a shared framework, and showed that our best models reduced total error by up to 39.1%. This improvement was primarily the result of recovering and filling in missing surface.

The models presented in Chapters 4 to 7 predicted corrections one view at a time. However, as the views our model processes arise from the same underlying scene, it may be useful to process them together and share information between viewpoints. To this end, in Chapter 8 we explored a novel method for aggregating information from arbitrary views when making predictions. We observed that features inside the CNN are usually viewpoint-dependent – which makes it difficult to aggregate disparate viewpoints –, and proposed an architecture that can transform features between viewpoints. This step was shown to be necessary for effective aggregation.

9.1 Future Work

Throughout this thesis, we have identified several avenues for future research, which we summarise here:

1. *Interpretable uncertainty:* The soft mask we used in Chapter 6 is an architecture that allows the CNN to learn how to modulate or discard its outputs. While we have shown that the model learns to discard poor predictions due to a lack of labels, formulations such as Kendall and Gal (2017) and Barron (2019) deal with uncertainty and noisy labels explicitly and result in interpretable uncertainty masks. Our future work will explore the improvements such formulations can bring to our mesh correction system.

2. *Combining semantics and geometry:* As identified in Chapter 7, most of the improvement comes from recovering missing surface in the reconstructions. A clear direction for future work is to improve the correctness at the same time as the completeness of reconstructions. One way we propose to achieve this is by making the relation between semantics (e.g. car, road, building, etc.) and geometry (e.g. flat, curved, horizontal, etc.) explicit. For example, adding semantics as further prediction task has been shown to improve performance in other geometry prediction tasks, such as Kendall, Gal, and Cipolla (2018) or S. Liu, Johns, et al. (2019).

3. *Promoting group structure for feature transform:* In Chapter 8 we proposed a novel architecture for handling multi-view data, by transforming features inside the network according to the relative pose between views. In our experiments, we imposed no constraints on the feature transform. However, for our future work we proposed a loss that can be used to promote the group structure such a transform should intuitively have. This research direction will explore ways in which such a loss can be used in order to improve the quality and generalisation capability of the proposed architecture.

4. *Using feature transform for other tasks:* In Chapter 8 we used the proposed multi-view architecture with our mesh correction network to process several views at the same time when predicting corrections. However, the feature transform architecture is not restricted to this task. Instead, it can be used as a general way to aggregate information from multiple views, irrespective of task or representation. In our future work, we will explore how our method can be applied to other tasks such as depth refinement, 3D reconstruction, or localisation. For the former application, representations are usually 2D (depth maps), like in our case, while for the latter two many representations can be used, such as voxel grids or implicit representations (as discussed in Section 2.3).

5. *Transfer learning to depth refinement:* Since our mesh correction network operates on 2D representations of meshes, the same architecture could be used for refining depth images. Future work will investigate how to transfer the knowledge learnt from the mesh correction task to the task of depth refinement. Such an approach would maintain the advantage our method has – access to vast amounts of generated training data – while being able to operate online on live depth images.

References

Ahmadi, A., & Patras, I. (2016). Unsupervised convolutional neural networks for motion estimation, In *Proceedings of The IEEE International Conference on Image Processing (ICIP)*. (Cit. on p. 10).

Attene, M., Campen, M., & Kobbelt, L. (2013). Polygon mesh repairing: An application perspective. *ACM Computing Surveys*, *45*(2) (cit. on p. 18).

Attene, M., Falcidieno, B., Rossignac, J., & Spagnuolo, M. (2005). Sharpen & bend: Recovering curved sharp edges in triangle meshes produced by feature-insensitive sampling. *IEEE Transactions on Visualization and Computer Graphics*, *11*, 181–192 (cit. on p. 18).

Barequet, G., & Kumar, S. (1997). Repairing cad models, In *Proceedings of Visualization*. (Cit. on p. 18).

Barequet, G., & Sharir, M. (1995). Filling gaps in the boundary of a polyhedron. *Computer Aided Geometric Design*, *12*, 207–229 (cit. on p. 18).

Barron, J. T. (2019). A general and adaptive robust loss function, In *Proceedings of The IEEE/CVF International Conference on Computer Vision and Pattern Recognition (CVPR)*. (Cit. on p. 118).

Baxter, J. (2000). A model of inductive bias learning. *The Journal of Artificial Intelligence Research (JAIR)*, *12*, 149–198 (cit. on p. 103).

Centin, M., Pezzotti, N., & Signoroni, A. (2015). Poisson-driven seamless completion of triangular meshes. *Computer Aided Geometric Design*, *35–36*, 42–55 (cit. on p. 19).

Chang, J.-R., & Chen, Y.-S. (2018). Pyramid stereo matching network, In *Proceedings of The IEEE/CVF International Conference on Computer Vision and Pattern Recognition (CVPR)*. (Cit. on p. 16).

Chen, C.-Y., & Cheng, K.-Y. (2008). A sharpness-dependent filter for recovering sharp features in repaired 3D mesh models. *IEEE Transactions on Visualization and Computer Graphics*, *14*(1), 200–212 (cit. on p. 18).

Cherabier, I., Schönberger, J. L., Oswald, M. R., Pollefeys, M., & Geiger, A. (2018). Learning priors for semantic 3D reconstruction, In *Proceedings of The European Conference on Computer Vision (ECCV)*. (Cit. on p. 17).

Choi, D., Shallue, C. J., Nado, Z., Lee, J., Maddison, C. J., & Dahl, G. E. (2019). On empirical comparisons of optimizers for deep learning. (Cit. on p. 8).

Choi, S., Zhou, Q.-Y., & Koltun, V. (2015). Robust reconstruction of indoor scenes, In *Proceedings of The IEEE/CVF International Conference on Computer Vision and Pattern Recognition (CVPR)*. (Cit. on p. 16).

Cireșan, D. C., Meier, U., Gambardella, L. M., & Schmidhuber, J. (2010). Deep, big, simple neural nets for handwritten digit recognition. *Neural Computation*, *22*, 3207–3220 (cit. on p. 8).

Cireșan, D. C., Meier, U., Masci, J., Gambardella, L. M., & Schmidhuber, J. (2011). Flexible, high performance convolutional neural networks for image classification, In *Proceedings of The International Joint Conference on Artificial Intelligence (IJCAI)*. (Cit. on p. 8).

Curless, B., & Levoy, M. (1996). A volumetric method for building complex models from range images, In *Proceedings of The ACM Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*, ACM. (Cit. on p. 16).

Dai, A., Nießner, M., Zollhöfer, M., Izadi, S., & Theobalt, C. (2017). BundleFusion: Real-time globally consistent 3D reconstruction using on-the-fly surface reintegration. *ACM Transactions on Graphics*, *36*, 1–18 (cit. on p. 16).

Dumoulin, V., & Visin, F. (2016). A guide to convolution arithmetic for deep learning. (Cit. on p. 9).

Fan, H., Yu, Y., & Peng, Q. (2010). Robust feature-preserving mesh denoising based on consistent subneighborhoods. *IEEE Transactions on Visualization and Computer Graphics*, *16*, 312–324 (cit. on p. 19).

Felzenszwalb, P. F., & Huttenlocher, D. P. (2004). Efficient belief propagation for early vision, In *Proceedings of The IEEE/CVF International Conference on Computer Vision and Pattern Recognition (CVPR)*. (Cit. on p. 15).

Fleishman, S., Drori, I., & Cohen-Or, D. (2003). Bilateral mesh denoising. *ACM Transactions on Graphics*, *22*, 950–953 (cit. on p. 19).

Geiger, A., Lenz, P., Stiller, C., & Urtasun, R. (2013). Vision meets robotics: The KITTI dataset. *The International Journal of Robotics Research (IJRR)*, *32*(11), 1231–1237 (cit. on p. 22).

Geiger, A. [Andreas], Lenz, P. [Philip], & Urtasun, R. (2012). Are we ready for autonomous driving? The KITTI vision benchmark suite, In *Proceedings of The IEEE/CVF International Conference on Computer Vision and Pattern Recognition (CVPR)*. (Cit. on pp. 3, 95).

Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks, In *Proceedings of The International Conference on Artificial Intelligence and Statistics (AISTATS)*. (Cit. on p. 8).

Godard, C., Mac Aodha, O., & Brostow, G. J. (2017). Unsupervised monocular depth estimation with left-right consistency, In *Proceedings of The IEEE/CVF International Conference on Computer Vision and Pattern Recognition (CVPR)*. (Cit. on p. 10).

Hartley, R. I., & Zisserman, A. (2004). *Multiple view geometry in computer vision* (second ed.). Cambridge University Press. (Cit. on p. 10).

He, K., Zhang, X., Ren, S., & Sun, J. (2016a). Deep residual learning for image recognition, In *Proceedings of The IEEE/CVF International Conference on Computer Vision and Pattern Recognition (CVPR)*. (Cit. on p. 9).

He, K., Zhang, X., Ren, S., & Sun, J. (2016b). Identity mappings in deep residual networks, In *Proceedings of The European Conference on Computer Vision (ECCV)*. (Cit. on p. 9).

Henriques, J. F., Ehrhardt, S., Albanie, S., & Vedaldi, A. (2019). Small steps and giant leaps: Minimal newton solvers for deep learning, In *Proceedings of The International Conference on Computer Vision (ICCV)*. (Cit. on p. 8).

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift, In *Proceedings of The International Conference on Machine Learning (ICML)*. (Cit. on p. 8).

Jaderberg, M., Simonyan, K., Zisserman, A., & Kavukcuoglu, K. (2015). Spatial transformer networks, In *Proceedings of The Conference on Neural Information Processing Systems (NIPS)*. (Cit. on p. 10).

Jeon, J., & Lee, S. (2018). Reconstruction-based pairwise depth dataset for depth image enhancement using CNN, In *Proceedings of The European Conference on Computer Vision (ECCV)*. (Cit. on p. 35).

Jouppi, N. P., Young, C., Patil, N., Patterson, D., Agrawal, G., Bajwa, R., Bates, S., Bhatia, S., Boden, N., Borchers, A., Boyle, R., Cantin, P.-l., Chao, C., Clark, C., Coriell, J., Daley, M., Dau, M., Dean, J., Gelb, B., … Yoon, D. H. (2017). In-datacenter performance analysis of a tensor processing unit, In *Proceedings of The Annual International Symposium on Computer Architecture (ISCA)*, Toronto, ON, Canada, Association for Computing Machinery. (Cit. on p. 8).

Ju, T. (2004). Robust repair of polygonal models. *ACM Transactions on Graphics*, *23*, 888–895 (cit. on p. 19).

Kar, A., Häne, C., & Malik, J. (2017). Learning a multi-view stereo machine, In *Proceedings of The Conference on Neural Information Processing Systems (NIPS)*. (Cit. on p. 17).

Kazhdan, M. M., Bolitho, M., & Hoppe, H. (2006). Poisson surface reconstruction, In *Proceedings of The Eurographics Symposium on Geometry Processing (SGP)*. (Cit. on p. 19).

Kendall, A., & Gal, Y. (2017). What uncertainties do we need in bayesian deep learning for computer vision?, In *Proceedings of The Conference on Neural Information Processing Systems (NIPS)*. (Cit. on p. 118).

Kendall, A., Gal, Y., & Cipolla, R. (2018). Multi-task learning using uncertainty to weigh losses for scene geometry and semantics, In *Proceedings of The IEEE/CVF International Conference on Computer Vision and Pattern Recognition (CVPR)*. (Cit. on pp. 103, 118).

Kendall, A., Martirosyan, H., Dasgupta, S., & Henry, P. (2017). End-to-end learning of geometry and context for deep stereo regression, In *Proceedings of The International Conference on Computer Vision (ICCV)*. (Cit. on p. 16).

Khamis, S., Fanello, S. R., Rhemann, C., Kowdle, A., Valentin, J. P. C., & Izadi, S. (2018). StereoNet: Guided hierarchical refinement for real-time edge-aware depth prediction, In *Proceedings of The European Conference on Computer Vision (ECCV)*. (Cit. on p. 16).

Kingma, D., & Ba, J. (2015). Adam: A method for stochastic optimization, In *Proceedings of The International Conference on Learning Representations (ICLR)*. (Cit. on p. 8).

Kokkinos, I. (2017). UberNet: Training a 'universal' convolutional neural network for low-, mid-, and high-level vision using diverse datasets and limited memory, In *Proceedings of The IEEE/CVF International Conference on Computer Vision and Pattern Recognition (CVPR)*. (Cit. on p. 103).

Kolmogorov, V., & Zabih, R. (2001). Computing visual correspondence with occlusions using graph cuts, In *Proceedings of The International Conference on Computer Vision (ICCV)*. (Cit. on p. 15).

Kraevoy, V., & Sheffer, A. (2005). Template-based mesh completion, In *Proceedings of The Eurographics Symposium on Geometry Processing (SGP)*. (Cit. on p. 19).

Laina, I., Rupprecht, C., Belagiannis, V., Tombari, F., & Navab, N. (2016). Deeper depth prediction with fully convolutional residual networks, In *Proceedings of The IEEE International Conference on 3D Vision (3DV)*. (Cit. on pp. 9, 45).

LeCun, Y. (1985). Une procédure d'apprentissage pour réseau a seuil asymmetrique, In *Proceedings of Cognitiva 85*, Paris, France. (Cit. on p. 7).

LeCun, Y. (1986). Learning processes in an asymmetric threshold network (E. Bienenstock, F. Fogelman-Soulié, & G. Weisbuch, Eds.). In E. Bienenstock, F. Fogelman-Soulié, & G. Weisbuch (Eds.), *Disordered systems and biological organization*, Les Houches, France, Springer-Verlag. (Cit. on p. 7).

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*(7553), 436–444 (cit. on p. 8).

LeCun, Y., Boser, B. E., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W. E., & Jackel, L. D. (1989). Backpropagation applied to handwritten zip code recognition. *Neural Computation*, *1*, 541–551 (cit. on pp. 8, 9).

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition, In *Proceedings of the IEEE*. (Cit. on p. 8).

Liu, S., Johns, E., & Davison, A. J. (2019). End-to-end multi-task learning with attention, In *Proceedings of The IEEE/CVF International Conference on Computer Vision and Pattern Recognition (CVPR)*. (Cit. on pp. 103, 118).

Long, J., Shelhamer, E., & Darrell, T. (2015). Fully convolutional networks for semantic segmentation, In *Proceedings of The IEEE/CVF International Conference on Computer Vision and Pattern Recognition (CVPR)*. (Cit. on p. 8).

Lorensen, W. E., & Cline, H. E. (1987). Marching cubes: A high resolution 3D surface construction algorithm, In *Proceedings of The ACM Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*. (Cit. on pp. 16, 24).

Luo, W., Schwing, A. G., & Urtasun, R. (2016). Efficient deep learning for stereo matching, In *Proceedings of The IEEE/CVF International Conference on Computer Vision and Pattern Recognition (CVPR)*. (Cit. on p. 15).

Mahjourian, R., Wicke, M., & Angelova, A. (2018). Unsupervised learning of depth and ego-motion from monocular video using 3D geometric constraints, In *Proceedings of The IEEE/CVF International Conference on Computer Vision and Pattern Recognition (CVPR)*. (Cit. on p. 10).

Mayer, N., Ilg, E., Häusser, P., Fischer, P., Cremers, D., Dosovitskiy, A., & Brox, T. (2016). A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation, In *Proceedings of The IEEE/CVF International Conference on Computer Vision and Pattern Recognition (CVPR)*. (Cit. on p. 16).

McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, *5*, 115–133 (cit. on p. 7).

Mescheder, L., Oechsle, M., Niemeyer, M., Nowozin, S., & Geiger, A. (2019). Occupancy networks: Learning 3D reconstruction in function space, In *Proceedings of The IEEE/CVF International Conference on Computer Vision and Pattern Recognition (CVPR)*. (Cit. on p. 17).

Mildenhall, B., Srinivasan, P. P., Tancik, M., Barron, J. T., Ramamoorthi, R., & Ng, R. (2020). NeRF: Representing scenes as neural radiance fields for view synthesis, In *Proceedings of The European Conference on Computer Vision (ECCV)*. (Cit. on p. 17).

Murali, T. M., & Funkhouser, T. A. (1997). Consistent solid and boundary representations from arbitrary polygonal data, In *Proceedings of The Symposium on Interactive 3DGraphics (I3D)*. (Cit. on p. 19).

Mur-Artal, R., & Tardós, J. D. (2017). ORB-SLAM2: An open-source SLAM system for monocular, stereo and RGB-D cameras. *IEEE Transactions on Robotics (T-RO)*, *33*(5), 1255–1262 (cit. on pp. 22, 23, 95).

Nair, V., & Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines, In *Proceedings of The International Conference on Machine Learning (ICML)*, Haifa, Israel. (Cit. on p. 8).

Newcombe, R. A., Izadi, S., Hilliges, O., Molyneaux, D., Kim, D., Davison, A. J., Kohli, P., Shotton, J., Hodges, S., & Fitzgibbon, A. W. (2011). KinectFusion: Real-time dense surface mapping and tracking, In *Proceedings of The IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*. (Cit. on p. 16).

Nguyen, M. X., Yuan, X., & Chen, B. (2005). Geometry completion and detail generation by texture synthesis. *The Visual Computer*, *21*, 669–678 (cit. on p. 18).

Niemeyer, M., Mescheder, L., Oechsle, M., & Geiger, A. (2020). Differentiable volumetric rendering: Learning implicit 3D representations without 3D supervision, In *Proceedings of The IEEE/CVF International Conference on Computer Vision and Pattern Recognition (CVPR)*. (Cit. on p. 17).

Nießner, M., Zollhöfer, M., Izadi, S., & Stamminger, M. (2013). Real-time 3D reconstruction at scale using voxel hashing. *ACM Transactions on Graphics*, *32*, 1–11 (cit. on pp. 16, 17).

Noh, H., Hong, S., & Han, B. (2015). Learning deconvolution network for semantic segmentation, In *Proceedings of The International Conference on Computer Vision (ICCV)*. (Cit. on p. 9).

Oh, K.-S., & Jung, K. (2004). GPU implementation of neural networks. *Pattern Recognition*, *37*(6), 1311–1314 (cit. on p. 8).

Panchetti, M., Pernot, J.-P., & Véron, P. (2010). Towards recovery of complex shapes in meshes using digital images for reverse engineering applications. *Computer-Aided Design*, *42*, 693–707 (cit. on p. 18).

Parker, D. B. (1985). *Learning logic* (tech. rep. No. 47). Center for Computational Research in Economics and Management Science, MIT. (Cit. on p. 7).

Peyravian, M., Roginsky, A., & Kshemkalyani, A. D. (1998). On probabilities of hash value matches. *Computers & Security*, *17*, 171–176 (cit. on p. 28).

Polyak, B. T. (1964). Some methods of speeding up the convergence of iteration methods (H. Cleaves, Trans.). *USSR Computational Mathematics and Mathematical Physics*, *4*(5), 1–17 (cit. on p. 7).

Ranftl, R., Gehrig, S. K., Pock, T., & Bischof, H. (2012). Pushing the limits of stereo using variational stereo estimation, In *IEEE Intelligent Vehicles Symposium (IV)*. (Cit. on pp. 15, 23, 24).

Riegler, G., Ulusoy, A. O., Bischof, H., & Geiger, A. (2017). OctNetFusion: Learning depth fusion from data, In *Proceedings of The IEEE International Conference on 3D Vision (3DV)*. (Cit. on p. 17).

Riegler, G., Ulusoy, A. O., & Geiger, A. [Andreas]. (2017). OctNet: Learning deep 3D representations at high resolutions, In *Proceedings of The IEEE/CVF International Conference on Computer Vision and Pattern Recognition (CVPR)*. (Cit. on p. 17).

Robbins, H., & Monro, S. (1951). A stochastic approximation method. *The Annals of Mathematical Statistics*, *22*(3), 400–407 (cit. on p. 7).

Ronneberger, O., Fischer, P., & Brox, T. (2015). U-Net: Convolutional networks for biomedical image segmentation, In *Medical image computing and computer assisted intervention (MICCAI)*. (Cit. on pp. 9, 117).

Rosenblatt, F. (1957). *The perceptron: A perciving and recognizing automation* (tech. rep. No. 85-460-1). Cornell Aeronautical Laboratory, Inc. (Cit. on p. 7).
Rosenblatt, F. (1960). *On the convergence of reinforcement procedures in simple perceptrons* (tech. rep. VG-1196-G-4). Cornell Aeronautical Laboratory, Inc. (Cit. on p. 7).
Rublee, E., Rabaud, V., Konolige, K., & Bradski, G. (2011). ORB: An efficient alternative to SIFT or SURF, In *Proceedings of The International Conference on Computer Vision (ICCV)*. (Cit. on p. 23).
Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986a). Learning internal representations by error propagation. In *Parallel distributed processing: Explorations in the microstructure of cognition* (pp. 318–362). Cambridge, MA, USA, MIT Press. (Cit. on p. 7).
Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986b). Learning representations by back-propagating errors. *Nature*, *323*(6088), 533–536 (cit. on p. 7).
Săftescu, Ș., Gadd, M., & Newman, P. (n.d.). Look here: Learning geometrically consistent refinement of inverse-depth images for 3D reconstruction [Under review.]. *The International Journal of Pattern Recognition and Artificial Intelligence* (cit. on p. 6).
Săftescu, Ș., & Newman, P. (2020). Learning geometrically consistent mesh corrections, In *Proceedings of The International Conference on Computer Vision Theory and Applications (VISAPP)*, Valletta, Malta. (Cit. on p. 6).
Săftescu, Ș., & Newman, P. (2021). Learning to correct 3D reconstructions from multiple views, In *Proceedings of The International Conference on Computer Vision Theory and Applications (VISAPP)*. (Cit. on p. 6).
Sarkar, K., Varanasi, K., & Stricker, D. (2017). Learning quadrangulated patches for 3D shape parametrization and completion, In *Proceedings of The IEEE International Conference on 3D Vision (3DV)*. (Cit. on p. 18).
Sarkar, K., Varanasi, K., & Stricker, D. (2018). 3D shape processing by convolutional denoising autoencoders on local patches, In *Proceedings of The IEEE Winter Conference on Applications of Computer Vision (WACV)*. (Cit. on p. 18).
Scharstein, D., & Szeliski, R. (2002). A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International Journal of Computer Vision*, *47*, 7–42 (cit. on p. 15).
Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural Networks*, *61*, 85–117 (cit. on p. 8).
Schmidt, R. M., Schneider, F., & Hennig, P. (2020). Descending through a crowded valley—benchmarking deep learning optimizers. (Cit. on p. 8).
Shen, C., O'Brien, J. F., & Shewchuk, J. R. (2004). Interpolating and approximating implicit surfaces from polygon soup. *ACM Transactions on Graphics*, *23*, 896–904 (cit. on p. 19).
Simonyan, K., & Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition, In *Proceedings of The International Conference on Learning Representations (ICLR)*. (Cit. on p. 9).
Sitzmann, V., Thies, J., Heide, F., Nießner, M., Wetzstein, G., & Zollhöfer, M. (2019). DeepVoxels: Learning persistent 3D feature embeddings, In *Proceedings of The IEEE/CVF International Conference on Computer Vision and Pattern Recognition (CVPR)*. (Cit. on p. 17).

Sterzentsenko, V., Saroglou, L., Chatzitofis, A., Thermos, S., Zioulis, N., Doumanoglou, A., Zarpalas, D., & Daras, P. (2019). Self-supervised deep depth denoising, In *Proceedings of The International Conference on Computer Vision (ICCV)*. (Cit. on p. 35).

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., & Rabinovich, A. (2015). Going deeper with convolutions, In *Proceedings of The IEEE/CVF International Conference on Computer Vision and Pattern Recognition (CVPR)*. (Cit. on p. 9).

Tanner, M., Piniés, P., Paz, L. M., & Newman, P. (2015). BOR^{2}G: Building optimal regularised reconstructions with GPUs (in cubes), In *Proceedings of The International Conference on Field and Service Robotics (FSR)*, Toronto, Canada. (Cit. on pp. 16, 23, 24, 35).

Tanner, M., Piniés, P., Paz, L. M., Săftescu, Ș., Bewley, A., Jonasson, E., & Newman, P. (2020). Large scale outdoor scene reconstruction and correction with vision. *The International Journal of Robotics Research (IJRR)* (cit. on p. 6).

Tanner, M., Săftescu, Ș., Bewley, A., & Newman, P. (2018). Meshed up: Learnt error correction in 3D reconstructions, In *Proceedings of The IEEE International Conference on Robotics and Automation (ICRA)*, Brisbane, Australia. (Cit. on p. 6).

Taubin, G. (1995). A signal processing approach to fair surface design, In *Proceedings of The ACM Conference on Computer Graphics and Interactive Techniques (SIGGRAPH)*. (Cit. on p. 19).

Tulyakov, S., Ivanov, A., & Fleuret, F. (2018). Practical Deep Stereo (PDS): Toward applications-friendly deep stereo matching, In *Proceedings of The Conference on Neural Information Processing Systems (NeurIPS)*. (Cit. on p. 16).

Wang, X., Hu, J., Zhang, D., Guo, L., Qin, H., & Hao, A. (2018). Multi-scale geometry detail recovery on surfaces via empirical mode decomposition. *Computers & Graphics*, *70*, 118–127 (cit. on p. 18).

Wang, Y., Yang, Y., Yang, Z., Zhao, L., & Xu, W. (2018). Occlusion aware unsupervised learning of optical flow, In *Proceedings of The IEEE/CVF International Conference on Computer Vision and Pattern Recognition (CVPR)*. (Cit. on p. 10).

Weber, M., Humenberger, M., & Kubinger, W. (2009). A very fast census-based stereo matching implementation on a graphics processing unit, In *International Conference on Computer Vision (ICCV) Workshops*. (Cit. on p. 15).

Weder, S., Schönberger, J. L., Pollefeys, M., & Oswald, M. R. (2020). RoutedFusion: Learning real-time depth map fusion, In *Proceedings of The IEEE/CVF International Conference on Computer Vision and Pattern Recognition (CVPR)*. (Cit. on p. 17).

Werbos, P. J. (1974). *Beyond regression: New tools for prediction and analysis in the behavioral sciences* (Doctoral dissertation). Harvard University. (Cit. on p. 7).

Wiles, O., & Zisserman, A. (2017). SilNet: Single- and multi-view reconstruction by learning from silhouettes, In *Proceedings of The British Machine Vision Conference (BMVC)*, BMVA Press. (Cit. on p. 17).

Woodford, O. J., Torr, P. H. S., Reid, I. D., & Fitzgibbon, A. W. (2008). Global stereo reconstruction under second order smoothness priors. *Proceedings of The IEEE/CVF International Conference on Computer Vision and Pattern Recognition (CVPR)*, 1–8 (cit. on p. 15).

Wu, Y., & He, K. (2018). Group normalization, In *Proceedings of The European Conference on Computer Vision (ECCV)*. (Cit. on p. 8).

Wu, Z., Song, S., Khosla, A., Yu, F., Zhang, L., Tang, X., & Xiao, J. (2015). 3DShapeNets: A deep representation for volumetric shapes, In *Proceedings of The IEEE/CVF International Conference on Computer Vision and Pattern Recognition (CVPR)*. (Cit. on p. 17).

Yagi, Y., & Kawato, S. (1990). Panorama scene analysis with conic projection, In *Proceedings of The EEE International Workshop on Intelligent Robots and Systems (IROS)*. (Cit. on p. 11).

Yee, K., & Chakrabarti, A. (2020). Fast deep stereo with 2D convolutional processing of cost signatures, In *Proceedings of The IEEE Winter Conference on Applications of Computer Vision (WACV)*. (Cit. on p. 16).

Yoon, K.-J., & Kweon, I. S. (2006). Adaptive support-weight approach for correspondence search. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, *28*(4), 650–656 (cit. on p. 15).

Žbontar, J., & LeCun, Y. (2016). Stereo matching by training a convolutional neural network to compare image patches. *The Journal of Machine Learning Research (JMLR)*, *17*, 1–32 (cit. on p. 15).

Zeng, M., Zhao, F., Zheng, J., & Liu, X. (2012). A memory-efficient KinectFusion using octree (S.-M. Hu & R. R. Martin, Eds.). In S.-M. Hu & R. R. Martin (Eds.), *Computational visual media*, Berlin, Heidelberg, Springer Berlin Heidelberg. (Cit. on p. 16).

Zeng, M., Zhao, F., Zheng, J., & Liu, X. (2013). Octree-based fusion for realtime 3D reconstruction. *Graphical Modelodels*, *75*(3), 126–136 (cit. on p. 16).

Zhong, M., & Qin, H. (2016). Surface inpainting with sparsity constraints. *Computer Aided Geometric Design*, *41*, 23–35 (cit. on p. 19).

Zhou, T., Brown, M., Snavely, N., & Lowe, D. G. (2017). Unsupervised learning of depth and ego-motion from video, In *Proceedings of The IEEE/CVF International Conference on Computer Vision and Pattern Recognition (CVPR)*. (Cit. on p. 10).