AI-enabled flaw detection using multi-sensory data fusion

A Thesis Submitted for the Degree of Doctor of Philosophy

By

Benedict Marsh

Department of Electronic and Electrical Engineering, Brunel University London

2025

Abstract

This thesis focuses on the challenge of automated flaw detection by developing a method for surface crack detection that uses a data fusion approach. Flaw detection is needed to detect damage that could compromise structural integrity, leading to further consequences. Identification of flaws is important to analyse the severity and then take action to rectify any issues. Automated approaches using AI are needed to reduce cost as well as to speed up identification and increase accuracy. The presented research developed a method that followed a multi-stage approach, where data from multiple sensors are fused into a 3D representation with the use of AI models. Then, detection is done on that representation to identify the cracks so that further analysis can be done to determine the crack severity. Research contributions are from both stages. First, data fusion improvements for RGB images were worked on, and a novel method for fusing depth data from RGB stereo and LiDAR data was developed. Then, a method for crack identification from RGB-D data using a novel synthetic data generation method was developed. Evaluation of the contributions was carried out to demonstrate the improvements due to data fusion with comparisons to other methods. The key findings included: metric evaluations showing the developed enhanced image fusion technique improved RGB image quality. The developed novel stereo and LiDAR data fusion method showed lower error than either input method alone. The evaluation of the developed novel crack segmentation method using synthetic data showed that models can be effectively trained in the absence of extensive real-world data.

Code and Data Availability

Relevant source code and data developed for this thesis will be made publicly available and added to a centralised repository. The codebase is hosted on GitHub.

- **Repository:** https://github.com/benedictmarsh2/AI-enabled-flaw-detection-using-multi-sensory-data-fusion

Acknowledgements

I would like to express my gratitude to everyone who supported me throughout the course of this work. Your guidance, encouragement, and assistance have been invaluable, and I deeply appreciate your contributions.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The aim of this research is to develop methods to detect and analyse flaws such as cracks or defects by using a multi-sensor fusion approach. The benefit of using multiple sensor types is that it allows higher accuracy in scenarios where one of the sensors would have lower accuracy by using the data from the other sensors, so the fused data is higher accuracy than any one of the individual sensors on its own [1]. Flaw detection is essential in a range of industries, such as manufacturing processes and infrastructure maintenance.

1.1 Motivation

This research is motivated by the need for automated approaches to identifying flaws, as previous manual inspection approaches can miss up to 40% of defects [2], while automated inspection systems can achieve detection accuracy as high as 99% [3]. In manufacturing, flaws such as manufacturing defects can cause disruptions to the production line, and the end product can have impaired structural integrity. Detecting these flaws at an early stage in the production line allows for automatic decision-making to rectify the errors and ensure more efficient production with no breakdowns and high-quality products. Similarly, with infrastructure such as roads and pavements, it is important to detect cracks and potholes in the surfaces to ensure efficient movement along these roads and avoid further potential degradation that could result in damage to vehicles and present a safety risk. Manual inspection for flaws is costly due to the time and labour needed to perform these checks, and it can be more error-prone, where it has been found that human inspectors failed to catch 15% of defects on average [4]. Automated approaches

such as AI-driven systems can achieve detection accuracy of 97-99% [3]. A specific example of this is shown in a technical textile plant, which achieved a reduction in defect rates by 30% and increased inspection speeds by 25% [5]. Automated methods require sensors and computer vision algorithms to identify the flaws or to do non-destructive testing. However, they usually only use a single sensor modality, such as ultrasonic or visible RGB.[6, 7, 8]

1.2 Contributions

The core objective of this research project is to develop robust flaw detection methods that are capable of analysing the data representation of the scene captured from the sensors. This representation is obtained by combining data from multiple sensor modalities and then identifying specific visual cues that indicate if it contains a flaw. This will build on existing techniques and result in improvements in detection and analysis with higher performance evaluated with objective metrics. The aim is to develop new algorithms by using deep learning approaches to accurately characterise flaws, including cracks. The detection will be performed by training a deep neural network on data to improve accuracy with data fusion and detection. This enables automated analysis and classification of the detected structural flaws. Once flaws have been effectively identified and characterised, analysis can be performed on the potential consequences and implications for the object's structural integrity. To do this, the aim is to achieve improvements with techniques for data fusion, demonstrating the improvements made by using multiple sensors compared to a single sensor. An additional aim is to connect these data fusion techniques to detection and identification techniques to demonstrate a complete method that would use data fusion to identify flaws in a real-world application.

This follows a detection process composed of the following steps:

- The data from multiple sensors is fused into a single representation of the object that potentially contains a flaw. The representation would contain more information than any individual sensor, such as capturing a 3D representation from multiple sensors, each capturing only 2D information.

- The fused representation is then analysed with a detection method to determine the presence of a flaw and to analyse the severity of the flaw.

This modular approach is followed since it can build on the wealth of existing techniques and access a larger amount

of data used with machine learning-based methods. The data fusion methods that perform the detection or analysis directly from data sources are not explored since validating with real-world data would be more difficult due to the lack of data availability from multiple sensors. In order to do data fusion and detection in a single method, it would require data captured with multiple modalities as well as labelled flaws in the data; however, these datasets are limited. So, the approach followed develops a data fusion model that is not specific to any flaws, and can use a wider range of data, but it can then be combined with a detection model for specific flaws. The current trends with data fusion and flaw direction have pushed towards deep learning-based methods, so the research will focus on this area.

1.3 Multi-sensor fusion

The use of multi-sensor fusion is a fundamental aspect of this research. It focuses on developing methods that utilise data acquired by sensors of various modalities, including visible light red-green-blue (RGB) cameras, Time-of-Flight (ToF) sensors, and Light Detection and Ranging (LiDAR) sensors. These methods combine the sensor data to generate a unified three-dimensional (3D) image representation of the observed scene, leveraging the strengths of each sensor. For instance, when regions lack texture, RGB cameras will struggle to detect depth accurately. In this situation, having an additional sensor like ToF can provide precise depth information for those specific regions. Conversely, in areas with fine and complex structures, the higher resolution of RGB cameras enables more accurate depth determination. This fusion approach enables improved reconstruction of the objects of interest that potentially contain flaws, enhancing the effectiveness of flaw detection methods.

For RGB cameras, these will produce a 2-dimensional colour image from which 3D information cannot be directly computed. For the process of 3D reconstruction with RGB cameras, multiple images are captured at different viewing positions so corresponding information can be found between images, and depth can be computed from the disparity by using the camera calibration parameters. ToF sensors will produce a depth map of the scene where each pixel contains a depth measurement. These depth measurements are used to map to a 3D point if the sensor's calibration parameters are known. LiDAR sensors will produce 3D points of the scene that will form a point cloud. Other sensor modalities, such as ultrasonic, were not considered due to the lack of data availability for these modalities.

Data from multiple sensors can be easily fused into a single 3D representation by using each sensor's pose to transform the data into the same global coordinates. They can then be fused into 3D representations such as voxels, point clouds and meshes. Voxels are a 3D grid of values that can be used to represent a scene, with the location of the value in the grid representing a 3D coordinate, and the value gives information about the scene at that location, such as whether an object in the scene is occupying that space. Voxels have a fixed size, so large amounts of data can be fused without the scene growing in memory. However, a voxel grid will need lots of memory to represent high amounts of detail. Voxels can also be converted into a mesh with algorithms such as the marching cubes algorithm [9]. Point clouds can be used to represent highly detailed scenes without using the same amount of memory that a voxel grid would need. Since point clouds are sets of 3D points, when fusing large amounts of data, the number of points will also increase. Point clouds can also be converted to a polygon mesh, which has vertices (points) that are connected by edges. These point clouds are converted to meshes with algorithms such as Screened Poisson Surface Reconstruction [10].

The fused representation produced from the data fusion process is useful for the analysis of the flaws. Machine learning and numerical computation can be used to model the data from the fused representations, which can then be used for specific flaws such as cracks. This can then be used to train machine learning models to make inferences about the data to enable automated decision-making depending on the flaws. The focus is on data fusion representations for crack scenes and then using machine learning models to analyse the severity of the cracks by using properties such as the length of the crack.

1.4 Thesis outline

The following chapters are structured as follows:

- **Chapter 2: Background**

  This chapter covers the background information that is relevant to this research. It first covers the background for data fusion and flaw detection. For data fusion, there are detailed explanations for key data fusion methods for 3D reconstruction, which inform the methods developed. It focuses on methods that use LiDAR sensors and RGB stereo sensors, which are the sensor modalities used in the research. For flaw detection, a high-level overview of the key trends and methods used for flaw detection is presented. In the following sections, further

details about the application area of focus are given, which is crack detection on roads. Background for more specific areas of research focus, such as crack segmentation and synthetic data generation, is covered.

- **Chapter 3: Image fusion**

  This chapter covers Image fusion, which describes the methods implemented, which are "Image Fusion with Guided Filtering" [11] and "Infrared and Visible Image Fusion using a Deep Learning Framework" [12]. For these methods, evaluations are performed, explaining the metrics implemented to get the objective evaluations. Modifications made are described, and the results and comparisons with the original method are showcased. An image fusion method that uses 3D data is shown. Work done with the implementation and the improvements made with evaluations compared to the original is covered.

- **Chapter 4: LiDAR and stereo data fusion**

  This chapter covers the LiDAR and stereo data fusion. Work on producing a review paper that describes and compares state-of-the-art methods that use these sensor modalities, by looking at the accuracy metrics and techniques used, is presented first. These methods include RGB stereo methods, depth completion methods and LiDAR Stereo fusion methods. Work done in developing a novel method, which builds on the techniques from the state-of-the-art methods, is covered, and an evaluation of the method is provided.

- **Chapter 5: Crack detection**

  This chapter covers crack detection. The first section is the work done to implement the method "Downstream Semantic Segmentation Model for Low-Level Surface Crack Detection" for the detection and analysis of cracks and determining significance. The following section covers a novel method for segmentation of road cracks, describing the process used for synthetic data generation, training the deep learning model with that data, and finally, the results of the method with comparisons to other methods.

- **Chapter 6: Final conclusions**

  This chapter covers the conclusions of each chapter, explaining the results obtained from this research. The limitations of the research are discussed, and future directions for the work are noted.

Publications

These papers have been accepted and published:

Benedict Marsh, Abdul Hamid Sadka, and Hamid Bahai. A critical review of deep learning-based multi-sensor fusion techniques. *Sensors*, 22(23):9364, 2022.

Benedict Marsh and Ruiheng Wu. Crack segmentation in roads using synthetic data and RGB-D data fusion. *Computer Vision and Image Understanding*, 2025 Jul 25:104452, 2025.

The first paper, "A critical review of deep learning-based multi-sensor fusion" relates to the work done in Chapter 4. The second paper, "Crack segmentation in roads using synthetic data and RGB-D data fusion" relates to the work done in Chapter 5.

Chapter 2

Background

This chapter provides the research context, including relevant literature and background information that informed the research described in this thesis. The first section covers 3D reconstruction from data, which is needed to analyse the object that could contain a flaw. In order to perform an analysis of a flaw, the geometry of the object under inspection needs to be accurately captured. Therefore, this research focuses on 3D reconstruction methods using data fusion to construct the 3D representation since existing automated inspection systems are limited by relying on a single sensor modality.

The following sections cover LiDAR and RGB stereo methods. The RGB cameras can provide accurate geometry estimates where there is rich texture detail, but are less effective at estimating geometry for surfaces that lack texture, whereas active sensors like LiDAR can capture precise depth measurements in these regions but are limited by only providing sparse measurement points, so the research focuses on these two sensor modalities and how they can be fused to provide more robust geometry estimates.

In the following sections, the background of flaw detection, crack segmentation, and synthetic data generation is covered. This covers the primary application of this research, which is flaw detection, where it focuses on cracks, which are a class of defects that are very common and also significant for predicting the structural integrity in manufacturing and infrastructure. To determine the severity of a crack, the crack geometry needs to be determined, which is why crack segmentation is covered. Deep learning-based methods are the leading approaches in the research

for crack detection. However, these methods are limited by data availability, so synthetic data generation is covered.

2.1 Data fusion for 3D reconstruction

Multiple depth maps can be fused directly into point clouds or voxels [13] using sensor poses calibration parameters. Another approach is to obtain directly overlaid depth maps where the depth pixels across the depth map will be measuring the same points instead of fusing different points captured from different sensors into a single 3D representation, such as with point clouds or voxels, where data from one sensor would fill in the gaps of the other sensors. This way, the depth measurements in the depth map are measuring the same positions in the scene, so image fusion can be carried out to get a fused depth map. This has been done with multiple sensor modalities, such as an RGB camera pair and a ToF sensor to get depth maps from stereo matching, where the pair of RGB images captured by the camera are searched for corresponding points to generate a depth map, the ToF depth map is reprojected over the RGB camera's depth map for fusion [14, 15, 16, 17].

2.1.1 Multi-view Image and ToF Sensor Fusion for Dense 3D Reconstruction

For the method "Multi-view Image and ToF Sensor Fusion for Dense 3D Reconstruction" [18], the input data used is 5 RGB cameras and 3 ToF depth sensors, each capturing a single image or depth map, so there will be 5 colour images $\mathbf{I} = \{I_1, I_2, ..., I_5\}$ each with a resolution 1024×768 and 3 depth maps $\mathbf{D} = \{D_A, D_B, D_C\}$. The 3 ToF sensors are each paired to an RGB camera with an approximately identical viewing position, so the depth maps can be each paired with a colour image $(I_1, D_A), (I_3, D_B), (I_5, D_C)$.

Calibration using silhouettes

To best align the depth maps, the error from the true measurements needs to be minimised. This error is the result of some unavoidable random noise and some systematic bias, so this error can be modelled with the following probability distribution model:

$$p(z|x) \sim \mathcal{N}(x + b(x), \sigma^2(x)) \tag{2.1}$$

Where the depth measurement z follows a normal distribution $\mathcal{N}(\mu, \sigma^2)$ where the mean μ is $x + b(x)$, x is the true depth, and $b(x)$ is the bias, which is a function of the true depth. Then the variance σ^2 is $\sigma^2(x)$, which is a function

of the true depth. The bias is reduced by finding a lookup table to approximate $b(x)$:

$$b(x) = \begin{cases} b_1, & \text{if } x = x_1 \\ b_2, & \text{if } x = x_2 \\ \vdots & \vdots \\ b_n, & \text{if } x = x_n \end{cases} \tag{2.2}$$

This is assumed to be the same function for all depth pixels. The bias function is found by reprojecting the set of depth maps $\mathbf{D}$ onto the paired colour image to get a set of depth map colour image pairs. Silhouette edges are found in each pair by extracting the edges in the images where they are close to discontinuities in the depth map. The depth values are shifted by some small value to find the bias values at different depth intervals that will align the depth discontinuities best with the silhouette edges. The best values are stored as a lookup table, which will be used as the bias function and will be different for each ToF sensor.

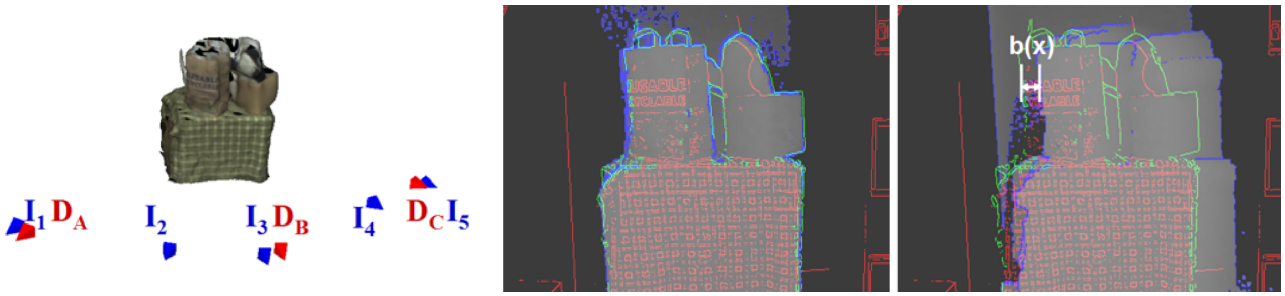

Figure 2.1: This figure from "Multi-view Image and ToF Sensor Fusion for Dense 3D Reconstruction"[18]. The left image shows the views of the 5 intensity images $I = \{I_1, I_2, ..., I_5\}$ and the 3 ToF depth maps $D = \{D_A, D_B, D_C\}$ capturing an object. The middle image shows the view from I_3 of the aligned geometry after the bias function has been applied. The right image shows the view from I_3 without any bias correction. The depth edges are shown as blue, the silhouette edges are shown as green, and the image edges are shown as red.

Initial surface reconstruction

The 3 depth maps D_A, D_B, D_C are combined into a voxel grid, which stores an occupancy probability in each voxel. These occupancy probabilities are set for each measured depth value z from a depth map by projecting the ray through the voxel grid and then setting the occupancy probabilities $p(m_x|z)$ of the voxels along the ray such that $p(m_x|z) = 0$ where the voxel is not occupied which will be the empty space between the sensor and the surface and it will be $p(m_x|z) = 1$ where the voxel is occupied. This is done by designing a heuristic function that will set

$p(m_x|z) = 0.5$ when the expected true distance $E(x)$ is measuring the distance to voxel m_x and the expected true distance is $E(x) = z - b(x)$. The heuristic function gradually decreases the probability for voxels closer to the sensor and increases for voxels further from the sensor. The gradient will be proportional to the standard deviation σ of the sensor noise to reflect the uncertainty of the measurement. For voxels behind the measured surface, the heuristic function will also gradually decrease the voxel occupancy probability down to 0.5 as those voxels are occluded, and 0.5 is the probability of an unobserved voxel being occupied.

The 3 depth maps will generate 3 occupancy probabilities for the voxels. To combine them, a log-odds occupancy voxel grid is used. Each voxel will store the value of $log(\frac{p(m_x|z)}{1-p(m_x|z)})$. Here when $\lim_{p(m_x|z)\to 0} \log\left(\frac{p(m_x|z)}{1-p(m_x|z)}\right) = -\infty$ and $\lim_{p(m_x|z)\to 1} \log\left(\frac{p(m_x|z)}{1-p(m_x|z)}\right) = \infty$ so the heuristic function sets the probabilites to be between 0.4 and 0.6. Then, by assuming the sensors are independent, they can be merged into a joint occupancy voxel that stores the log-odds as follows:

$$log(\frac{p(m_x|z_1, z_2, z_3)}{1 - p(m_x|z_1, z_2, z_3)}) = \sum_{i=1}^{3} log(\frac{p(m_x|z_i)}{1 - p(m_x|z_i)}) \tag{2.3}$$

This will give a joint occupancy voxel grid where each voxel represents the joint log-odds as $log(\frac{p(m_x|z_1,z_2,z_3)}{1-p(m_x|z_1,z_2,z_3)})$. Then a surface X is extracted as a triangle mesh from this voxel grid using the marching cubes algorithm [9], which will generate a small triangular mesh for each surface voxel depending on the pattern of neighbouring voxels with 14 different possible patterns and then fuse them all into a single triangular mesh.

Surface refinement

Next, the mesh surface is refined by minimising an objective function which contains a measurement term E_Z, a photo consistency term E_C, a silhouette term E_S and a prior term E_X:

$$E(X) = E_Z + E_C + E_S + E_X \tag{2.4}$$

This objective function is then optimised using a gradient-based optimiser to get a mesh surface of the scene $\hat{X}$, which is more accurate where it is closer to the real-world ground truth. The mesh will get more accurate the closer

it is to 0, so $\hat{X}$ is found as follows:

$$\hat{X} = \arg\min_{X} E(X) \tag{2.5}$$

The measurement term minimises the amount the mesh will move from its initial position:

$$E_Z = \sum_{i=1}^{N} ||r_i||^2 \tag{2.6}$$

$$r_i = \begin{cases} 0 & \text{if } r_i < W \\ s_i & \text{otherwise} \end{cases} \tag{2.7}$$

Where s_i is the distance vertex i has moved from its initial position, and W is the width of a voxel, so r_i is the distance vertex i has moved from its initial position, unless the distance is smaller than a voxel, then $r_i = 0$.

The photo consistency term moves the vertices closer to photo-consistent points in the images:

$$E_C = \sum_{(\mathbf{v}_i, p_k) \in C} \mathcal{H}(||\mathbf{v}_i - p_k||) \tag{2.8}$$

Where $\mathcal{H}(\cdot)$ is the Huber loss and C contains vertices $\mathbf{v}_i$ and points p_k found by projecting a vertex into the colour images and calculating the photo consistency of the patch centred at a point close to the vertex and if the photo consistency is a local maximum larger than a threshold then $\mathbf{v}_i, p_k$ will be in C.

The silhouette term moves the vertices that are on an occlusion boundary towards an intensity boundary in the images:

$$E_S = \sum_{(\mathbf{v}_i, p_u) \in S} \mathcal{H}(||\mathbf{v}_i - p_u||) \tag{2.9}$$

Where S contains points p_u that are the closest on an intensity boundary when $\mathbf{v}_i$ is projected onto the colour images, and if $\mathbf{v}_i$ is on an occlusion boundary, then $\mathbf{v}_i, p_u$ will be in S.

The prior term uses a Laplacian prior [19] to smooth the 3D mesh:

$$E_X = \sum_{i}^{N} ||\rho \Delta \mathbf{v}_i + (1+\rho)(-\Delta^2 \mathbf{v}_i)||^2 \tag{2.10}$$

Where $\rho = 0.6$, $\Delta \mathbf{v}_i$ is the discrete Laplacian operator at vertex $\mathbf{v}_i$ and $\Delta^2 \mathbf{v}_i$ is the Bi-Laplacian. The objective function is minimised with the L-BFGS-B algorithm [20], which is a quasi-Newton optimisation method that is more memory efficient than the BFGS algorithm.

2.1.2 Fusing Structure from Motion and LiDAR for Dense Accurate Depth Map Estimation

The method "Fusing Structure from Motion and LiDAR for Dense Accurate Depth Map Estimation" [21] uses a point cloud generated from a LiDAR scan of a scene and a set of images of the same scene as the input data.

Fusing structure from motion (SfM) and LiDAR point clouds

First, an estimation for the initial camera parameters is found for the following camera model

$$\tilde{p}_i^n = K_n[R_n|t_n]\tilde{P}_i^W \tag{2.11}$$

where $\tilde{P}_i^W = (X_i^W, Y_i^W, Z_i^W, 1)^T$ is the homogeneous point world coordinates in the world space W.For input image n: K_n is the intrinsic matrix, R_n is the rotation matrix, t_n is the translation vector, and the camera model computes $\tilde{p}_{n,i} = (u_{n,i}, v_{n,i}, w_{n,i})^T$ which is the homogeneous coordinates on the image plane and gives the final projected 2D coordinates as $(u'_{n,i}, v'_{n,i}) = (u_{n,i}/w_{n,i}, v_{n,i}/w_{n,i})$ which is the coordinates of the point projected to in the input image n.

The parameters of the camera model K_n, R_n, t_n are found by using incremental SfM [22], which finds the 3D reconstruction and camera parameters by sequentially adding the images with their associated camera parameters to be optimised to improve and expand the reconstruction.

This will generate a point cloud computed from the SfM algorithm, so a transformation between the LiDAR point cloud and the SfM point cloud is computed. The coherent point drift algorithm [23] is used to find a transformation

between the point clouds represented by a rotation matrix, a translation vector and a scaling factor.

Fusing RGB images and LiDAR depth maps

The transformation between the LiDAR and SfM point clouds is used to compute transformations for the LiDAR point to each input image as follows:

$$R_n^l = sR_n^s R^M, \tag{2.12}$$

$$t_n^l = sR_n^s t^M + t_n^s \tag{2.13}$$

where the mapping rotation matrix R^M, mapping translation vector t^M and mapping scaling factor s give the transformation from the LiDAR point cloud coordinate space to the SfM point cloud coordinate space. The LiDAR rotation matrix R_n^l and LiDAR translation vector t_n^l for input image n project a point in the LiDAR point cloud onto the input image using the intrinsic matrix K_n. The SfM rotation matrix R_n^s and SfM translation vector t_n^s project from the SfM coordinate space to the input image n.

Next, the computed camera model $K_n[R_n^l|t_n^l]$ is used to synthesise an image from the LiDAR point cloud. To do this, a hidden point removal operator is used to remove the occluded points in the LiDAR point cloud for a given camera position, so the visible points can be projected onto the image. A transformation is found between the synthesised image and the corresponding input image by using the generalised dual bootstrap-ICP algorithm [24], which extracts feature points and iteratively aligns them. The transformation will model homography and radial distortion. The LiDAR depths are fused with the input images, and then a bilateral filter is applied to the depth map to improve the accuracy.

2.1.3 Noise-aware Unsupervised Deep LiDAR-Stereo Fusion

The method "Noise-aware Unsupervised Deep LiDAR-Stereo Fusion" [25] uses a point cloud from a LiDAR sensor and a pair of RGB images from a stereo camera setup. This method is relevant to the research as it uses data fusion, which is one of the aims of this research. This informs how a data fusion stage can be integrated into a flaw detection method. The data fusion stage reduces the error due to noise or data sparsity in the input data. This then results in reduced error for the flaw detection method. This method informs the research by demonstrating

how data fusion can be applied to the sensor modalities of LiDAR and RGB stereo images. It also shows how a deep learning approach can be used to achieve this.

Deep convolutional neural network

First, a pair of disparity maps for both input images is computed from the LiDAR points by projecting the point cloud onto the input images to get depth maps. Depth can be converted to disparity using $D = (bf)/d$ where d is depth, b is the distance between the 2 cameras, and f is the focal length.

The disparity maps and input images are fused into a pair of depth maps using a deep convolutional neural network trained using a self-supervising loss function [26]. The network is trained to remove erroneous disparity values from the LiDAR data by forward-passing the data through the network 2 times before updating the weights. The first is a verification pass to remove noisy LiDAR disparities, and the second is the update pass to output the fused disparities and compute the loss for backpropagation. In the verification pass, the network will take the 2 input images and the 2 disparity maps from the LiDAR data and will output 2 initial fused disparity maps. Then, these outputs are compared to the LiDAR disparity maps and are checked for consistency to mask the noisy LiDAR disparities, so there are sparse LiDAR disparities for the inputs of the update phase. Next, in the update pass, the input image pair is used again with the new sparse disparities from the verification phase input into the network to output the final fused disparity maps. The loss function is computed from only the final fused disparity maps, so backpropagation can be done only during the update stage.

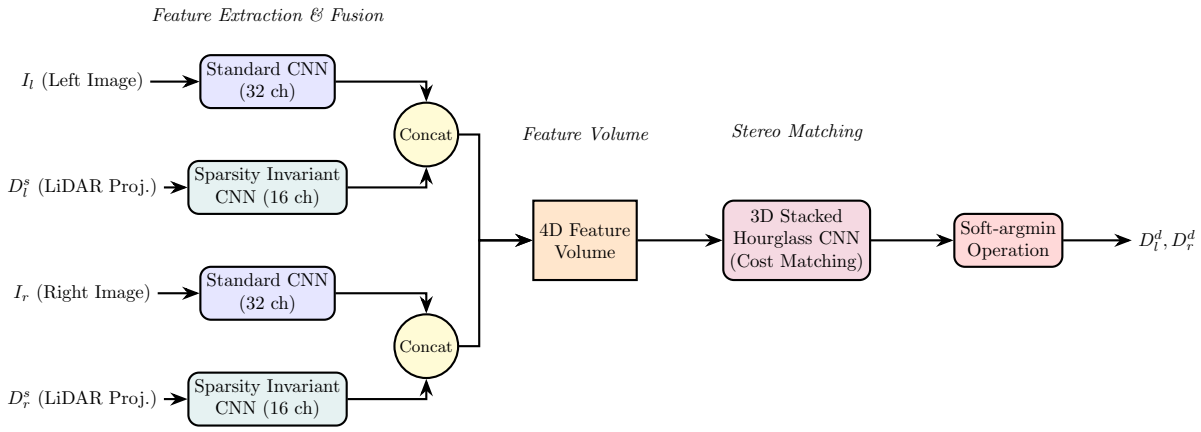

Figure 2.2: The core network architecture of the LiDAR-Stereo Fusion model. The left and right images I_l, I_r are used to extract features concatenated with the corresponding extracted features from the left and right input disparity maps D_l^s, D_r^s. The core network architecture will output a left-right disparity map estimate pair D_l^d, D_r^d.

First features are extracted from both the stereo image pairs and disparity map pairs, with two separate convolutional neural networks for each of the input modalities, where the feature extraction network for the disparity map uses sparsity invariant convolutional layers and the input images are passed through a feature extraction network from a Pyramid Stereo Matching Network [27]. The feature maps output by the network from the disparity maps are concatenated with the feature maps from the input images and stacked to form a volume. The feature volume is passed through 3D convolution layers to get a matching cost volume from which the disparity map output is computed using the soft-arg min operation [28], which is differentiable and is defined mathematically as:

$$\text{soft-}\arg\min = \sum_{d=0}^{D_{\max}} d \times \sigma(-c_d) \tag{2.14}$$

where c_d are the costs outputted by the model, d are the corresponging disparities and σ is the softmax function.

Loss function

The loss function is applied to the data output in the second pass through the network and is defined as follows:

$$\mathcal{L} = \mathcal{L}_{LiDAR} + \lambda_{warp}\mathcal{L}_{warp} + \lambda_{smooth}\mathcal{L}_{smooth} + \lambda_{plane}\mathcal{L}_{plane} \tag{2.15}$$

where $\mathcal{L}_{LiDAR}$ is the LiDAR loss, $\mathcal{L}_{warp}$ is the image warping loss, $\mathcal{L}_{smooth}$ is the image smoothing loss, $\mathcal{L}_{plane}$ is the plane fitting loss and $\lambda_{warp}, \lambda_{smooth}$, and λ_{plane} are defined constants.

The LiDAR loss minimises the difference between the output disparity maps and the masked LiDAR disparity maps, so it is defined as follows:

$$\mathcal{L}_{LiDAR} = ||M(\hat{D} - D_c)||_\tau \tag{2.16}$$

where $||\cdot||_\tau$ is the truncated mean squared loss clamped by a maximum error value ϵ, for which the value was omitted by the authors. $\hat{D}$ is the final outputted fused disparity maps, D_c is the LiDAR disparity maps that were masked for consistency in the verification pass, and M is the mask used for D_c in the verification pass.

The image warping loss uses the final fused disparity maps to warp the right input image onto the left input image

and compute the loss from these images, which is defined as follows:

$$\mathcal{L}_{warp} = \mathcal{L}_{photo} + \lambda_{census}\mathcal{L}_{census} + \lambda_{grad}\mathcal{L}_{grad} \tag{2.17}$$

where $\mathcal{L}_{photo}$ is the photometric loss, $\mathcal{L}_{census}$ is the census loss, $\mathcal{L}_{grad}$ is the image gradient loss and $\lambda_{census}, \lambda_{grad}$ are user-defined constants.

The photometric loss is defined as follows:

$$\mathcal{L}_{photo} = [\sum_i \varphi(\hat{I}_i - I_i) * O_i] / \sum_i O_i \tag{2.18}$$

where $\hat{I}_i$ and I_i are the warped and input images at pixel i, $\varphi(x) = \sqrt{x^2 + 0.001^2}$, and O is the occlusion mask, which is the pixels occluded due to stereo vision and computed using a consistency check.

The census loss is defined as follows:

$$\mathcal{L}_c = [\sum_i \varphi(\hat{C}_i - C_i) * O_i] / \sum_i O_i \tag{2.19}$$

where $\hat{C}$ and C_i are the warped and input images with the census transform applied to them at pixel i.

The image gradient loss is defined as follows:

$$\mathcal{L}_{grad} = [\sum_i \varphi(\nabla\hat{I}_i - \nabla I_i) * O_i] / \sum_i O_i \tag{2.20}$$

where $\nabla\hat{I}$ and ∇I_i are the gradient of the warped and input images at pixel i.

The image smoothing loss is defined as follows:

$$\mathcal{L}_{smooth} = \sum (e^{-\alpha_1|\nabla I|}|\nabla d| + e^{-\alpha_1|\nabla^2 I|}|\nabla^2 d|) \tag{2.21}$$

where d is a disparity value in the final fused disparity map, I is the pixel colour in the same location, ∇ is the gradient operator, and α_1 and α_2 are user-defined constants.

The plane fitting loss is defined as follows:

$$\mathcal{L}_{plane} = ||\hat{d} - \tilde{d}|| \tag{2.22}$$

where $\hat{d}$ is the disparity value in the final fused disparity map and $\tilde{d}$ is the disparity value for $\hat{d}$ when it is fitted to the plane of the SLIC (simple linear iterative clustering) superpixel [29] it is within.

The network is trained using the Adam optimiser, and the data is augmented during training using a random crop. RGB-D video is captured using an RGB camera paired with a depth sensor such as ToF. The sensors will have close to identical poses, so they can be easily calibrated, and a single pose can be used to represent both sensors. The sensor pair can record a video while being moved around the scene to capture large amounts of data and allow highly detailed reconstructions.

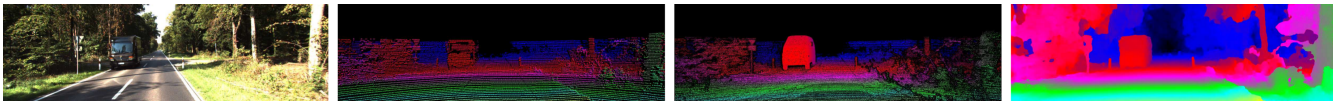

Figure 2.3: Image of the output from the method "Noise-aware Unsupervised Deep LiDAR-Stereo Fusion" [25]. From left to right, the images show. The input image, the input LiDAR disparities, the ground truth LiDAR disparities and the disparities outputted by the model.

2.1.4 Robust Reconstruction of Indoor Scenes

The method "Robust Reconstruction of Indoor Scenes" [30] uses an RGB-D video with unknown camera poses for each frame as input data. Here, pose refers to the 3D position (x, y, z) and orientation (yaw, pitch, roll) of the camera. The poses are estimated for each frame with a hierarchical approach that partitions the sequence of frames into smaller segments of k frames.

Initial pose estimation

For each segment, a trajectory of poses is estimated by using RGB-D odometry [31], which minimises a photometric error between a pair of consecutive frames to find a transformation from the pose of the first frame to the pose of the second frame. These transformations are found for every consecutive pair in the segment, so each frame has a pose relative to the first frame in the segment. From the segment, a mesh is extracted [13] by using the poses to reproject the depth maps as rays into a voxel grid, which will store the distances of each voxel along the reprojected

ray from the surface in the depth map, with the value being negative if the voxel is inside the surface. From this signed distance function, a mesh P_i can be extracted using marching cubes [9]. For each pair of meshes P_i, P_j from segments i, j, a geometric registration algorithm [32] is run to find the best transformation T_{ij} to align the overlapping areas of the 2 meshes. A transformation will only be found if there is significant overlap. Otherwise, a transformation will not be stored for that pair of segments.

Pose refinement

Next, a pose for each mesh $\mathbb{T} = \{T_i\}$ is found with pose graph optimisation by minimising the following objective function:

$$E(\mathbb{T}, \mathbb{L}) = \sum_i f(T_i, T_{i+1}, R_i) + \sum_i l_{ij} f(T_i, T_j, T_{ij}) + \mu \sum \Psi(l_{ij}) \tag{2.23}$$

where T_i and T_j are the poses of meshes i and j. T_{ij} is the transformation from mesh i to mesh j, R_i is the transformation of mesh i to mesh $i+1$ and $\mathbb{L} = \{l_{ij}\}$ is a parameter modelling the validity of the transformation T_{ij} and $\Psi(l_{ij}) = (\sqrt{l_{ij}} - 1)^2$. The transformation R_i is found by using the trajectory of the frames in segment i and the transformation between the last frame in segment i and the first frame in segment $i+1$ computed with RGB-D odometry.

The alignment function is defined as follows:

$$f(T_i, T_j, X) = \sum_{(p,q) \in \mathcal{K}_{ij}} ||X^{-1} T_j^{-1} T_i p - p||^2 \tag{2.24}$$

where $\mathcal{K}_{ij}$ is the set of vertices in mesh i and mesh j that are within distance ϵ when mesh i is transformed by X so $(p, q) \in \mathcal{K} \implies ||Xp - q|| < \epsilon$. The parameters $\mathbb{T}, \mathbb{L}$ are jointly optimised by minimising the objective function to find the final set of poses $\{T_i\}$ which can be used to reconstruct the whole sequence of frames by computing a global pose for each frame then computing a signed distance field from the posed depth maps to extract a final mesh of the whole scene.

2.1.5 Intrinsic3D: High-Quality 3D Reconstruction by Joint Appearance and Geometry Optimisation with Spatially-Varying Lighting

The method "Intrinsic3D: High-Quality 3D Reconstruction by Joint Appearance and Geometry Optimisation with Spatially-Varying Lighting" [33] uses an RGB-D video for its input data, which consists of colour frames and depth frames. The method produces information about the geometry by producing a Signed Distance Field (SDF), which gives the distance to the closest surface, with positive values outside the surface and negative values within the surface. The method also requires a set of initial pose estimates for the video frames, which it obtains from the method VoxelHashing [34], but other methods that can produce pose estimates with low error can also be used, such as BundleFusion [35] or the method from Robust reconstruction of indoor scenes [30].

Voxel grid representation

First, the initial poses are used to construct a truncated signed distance function (TSDF). The TSDF is a voxel grid that has the distance $D(v)$ to the closest surface at each voxel. The distance will be negative if the voxel is inside the surface, and the distance is clamped between a maximum and minimum distance. The voxel grid will also store the colour $C(v)$ of the voxel, the illumination albedo $A(v)$ of the voxel and the integration weight $W(v)$ of the voxel. The depth maps are integrated into the TSDF by using the initial pose estimates to project the depth values into the voxels that lie along the viewing rays, which are updated using the integration weights as follows:

$$D(v) = \frac{\sum_{i=1}^{M} w_i(v) d_i(v)}{W(v)}, W(v) = \sum_{i=1}^{M} w_i(v) \tag{2.25}$$

where M is the number of RGB-D input frames, $d_i(v)$ is the distance between voxel v and depth frame i. The integration weight is computed as $w_i(v) = cos(\theta)$ where θ is the angle between the depth ray and the surface normal computed from the depth map i, so when the ray is normal to the surface the weight will be 1, and when the ray is at a grazing angle to the surface, the weight will be close to 0. Since the distances are truncated, most of the voxels will contain the minimum or maximum distances, so the memory usage for the voxel grid is reduced by using a hash table to store the voxels [34], which each contain the distance, colour, albedo and weight.

The hash entries are accessed using the following hash function for a x, y, z location in the voxel grid:

$$H(x, y, z) = (x \cdot p_1 \oplus y \cdot p_2 \oplus z \cdot p_3) mod\, n \tag{2.26}$$

where p_1, p_2, p_3 are large prime numbers and n is the hash table size.

The voxel colours are set by selecting the RGB frames with low blur and then projecting the views into the voxel grid and averaging channel values separately. The albedo values in the voxel grid are initialised to be uniform.

Estimating lighting

Next, the shading is used to refine the initial 3D reconstruction [36]. The lighting is modelled with spherical harmonics, so the shading of a voxel $B(v)$ can be computed by:

$$B(v) = A(v) \sum_{m=1}^{b^2} l_m H_m(n(v)) \tag{2.27}$$

where l_m are the scene lighting parameters and H_m are the spherical harmonics basis functions with up to $b = 3$ being used. $A(v)$ is the voxel albedo, and $n(v)$ is the normal voxel surface unit computed from the TSDF. The reconstruction volume is partitioned into subvolumes, each with a set of lighting parameters.

The lighting parameters are jointly optimised by minimising the following objective function:

$$E_{lighting}(L_1, L_2, ..., L_K) = E_{appearance} + \lambda_{diffuse} E_{diffuse} \tag{2.28}$$

where $L_k = \{l_m\}$ is the set of lighting parameters for subvolume k and $\lambda_{diffuse}$ is a user-defined constant.

The appearance term is minimised by reducing the difference between the average intensity of a voxel and the computed shading as follows:

$$E_{appearance} = \sum_{v \in V_0} (B(v) - I(v))^2 \tag{2.29}$$

where $I(v)$ is the average intensity of a voxel computed for all voxels v in the voxel grid V_0 that are close to the surface.

The regularisation term minimises the difference between the lighting parameters of each subvolume and its neighbouring subvolumes, which is computed as follows:

$$E_{diffuse} = \sum_{s \in \mathcal{S}} \sum n \in \mathcal{N}_{\int}(L_s, L_n)^2 \tag{2.30}$$

where $\mathcal{S}$ is the set of subvolumes, $\mathcal{N}_{\int}$ is the set of neighbour subvolumes of s and L_s, L_n are the lighting parameters for subvolumes s, n.

Voxel grid refinement

Next, the TSDF, albedo values, camera poses, and camera intrinsics are jointly optimised by minimising an objective function that has 1 shading term and 4 regularisation terms defined as follows:

$$E_{scene}(\mathcal{X}) = \sum_{v \in V_0} \lambda_g E_g + \lambda_v E_v + \lambda_s E_s + \lambda_a E_a \tag{2.31}$$

where $\mathcal{X}$ is the set of parameters to be optimised, V_0 is set of voxels close to the surface and $\lambda_g, \lambda_v, \lambda_s, \lambda_a$ are user-defined constants.

The gradient-based shading term minimises the difference between the change in intensity computed by $B(v)$ and the change in intensity computed from the RGB frames:

$$E_g(v) = \sum_{\mathcal{I}_i \in V_{best}} w_i^v ||\nabla B(v) - \nabla \mathcal{I}_i(\pi(v_i))||_2^2 \tag{2.32}$$

where $\mathcal{I}_i$ is an intensity image computed from the input images. V_{best} contains the top intensity images for voxel V sorted by $w_i^v = \frac{cos(\theta)}{d^2}$ where d is the distance from the camera to v and θ is the angle between the normal and $\mathcal{I}_i$ projection in v. v_i is the 3D position of voxel v transformed in the intensity image $\mathcal{I}_i$ coordinates then π is the camera model to transform the 3D coordinates to a 2D coordinates on the intensity image.

The Laplacian smoothness term smooths the signed distance values between neighbouring voxels:

$$E_v(v) = (\Delta D(v))^2 \tag{2.33}$$

The surface stabilisation term keeps the refined TSDF close to the initial TSDF constructed using the initial poses:

$$E_s = (D(v) - D_{initial}(v))^2 \tag{2.34}$$

where $D_{initial}$ is the TSDF before optimisation.

The albedo regularisation term minimises the difference between neighbouring albedo values:

$$E_a(v) \sum_{u \in \mathcal{N}_v} \phi(\Gamma(v) - \Gamma(u)) \cdot (A(v) - A(u))^2 \tag{2.35}$$

where $\mathcal{N}_v$ is the set of neighbouring voxels to v and Γ computes the chromaticity of a voxel and the ϕ is a robust kernel defined as $\phi(x) = 1/(1 + t_{rob} \cdot x)^3$ with a user-defined parameter t_{rob}.

The objective function $E_{scene}(\mathcal{X})$ is minimised using the Levenberg-Marquardt algorithm, and then a mesh is extracted from the final TSDF using marching cubes [9].

2.1.6 BAD SLAM: Bundle Adjusted Direct RGB-D SLAM

The method "BAD SLAM: Bundle Adjusted Direct RGB-D SLAM" uses an RGB and depth sensor pair capturing a video for input data and can run in real-time while capturing video.

SLAM

Simultaneous localisation and mapping (SLAM) methods aim to solve the task of building a 3D model of the scene while also determining the pose of the camera in the scene. The method will take the input frames, which consist of RGB images with corresponding depth frames, and will then estimate the camera pose used to capture each frame while simultaneously estimating the 3D geometry.

Modern SLAM methods use a two-part approach [37]:

- front-end part: This part of the method will determine estimates of the camera's movements between frames, working in real time.
- back-end part: This part of the method refines the camera poses that were determined by the front-end part

and determines the 3D geometry using Bundle Adjustment [38] to optimise these parameters.

Bundle Adjustment [38] in the back-end part of the method works by projecting the current estimates for the 3D geometry onto the input images using the current estimates for the camera poses, then the method computes the error between the reprojection and the input images to optimise the parameters, which are the camera poses and the 3D geometry.

The BAD SLAM algorithm uses RBG-D video as the input data that is preprocessed by using a bilateral filter on the depth map in each input frame, to reduce noise. From the sequence of input frames, keyframes are selected where every 10th frame of the RGB-D video stream is a keyframe to be used for later optimisation of the 3D reconstructed geometry. The keyframes are used as input data for the back-end Bundle Adjustment.

Initial poses are first found for the keyframes using RGB-D odometry to track each transformation between frames, so a trajectory can be found between consecutive keyframes. Then, as the video is being processed, the current keyframe k is matched with each previous keyframe to find the most similar keyframe m so a transformation can be found. The transformation between keyframes k and $m+1$ and between keyframes k and $m-1$ are then selected, and if these transformations are sufficiently consistent, then a pose graph optimisation step will be run to correct the current keyframe trajectory. The pose graph optimisation step is used for loop closure, where a camera pose is detected as having revisited a previously mapped area. For this, the average pose is used to correct the initial pose estimate before Bundle Adjustment is done.

3D representation using surfels

The keyframes are used to update a surfel representation, which is a cloud of surfaces, each defined by a centre 3D point $\mathbf{p}_s$ of the surface, a normal vector to the surface $\mathbf{n}_s$, a surface radius $\mathbf{r}_s$ and a scalar visual descriptor $\mathbf{d}_s$. Surfels are created from each new keyframe by using the initial pose estimates. The keyframe is partitioned into a grid of 4×4 pixels, which are each projected into the surfel cloud to check if there is already a surfel at that location. For the 4×4 pixels without a surfel, a new one is created, and the centre point is computed by projecting the depth map as follows:

$$\mathbf{p}_s = T_k^G \pi_{D,k}^{-1}(d_p) \tag{2.36}$$

where d_p is the depth value of the pixel the surfel is created from, $\pi^{-1}_{D,k}(x)$ is the inverse projection function to compute the 3D point in the local coordinate system of keyframe k computed with depth value at the pixel in the depth map D of that keyframe and T^G_k is the transformation for the local coordinate system of keyframe k to the global coordinate system G of the scene. The normal of the surfel $\mathbf{n}_s$ is computed using the differences between neighbouring depth values in the depth map. The surface radius of the surfel $\mathbf{r}_s$ is computed as the minimum distance between the surfel 3D point $\mathbf{p}_s$ and the 3D points of the neighbouring pixels in the keyframe. The scalar visual descriptor $\mathbf{d}_s$ is computed as the intensity gradient magnitude of the pixel computed by taking the differences between neighbouring pixels.

3D refinement

Next, an optimisation loop is run on the surfel cloud by minimising an objective function that computes the differences between the geometry and colour of the projected surfels and the pixel values in the keyframes.

This objective function is defined as follows:

$$E(K,S) = \sum_{k \in K} \sum_{s \in S_k} (\rho_{Tukey}(\sigma_D^{-1} E_{geom}(k,s)) + \lambda_{photo} \rho_{Huber}(\sigma_p^{-1} E_{photo}(k,s))) \tag{2.37}$$

where K is the set of keyframes, each with an estimation of its pose, S is the set of surfels in the surfel cloud, so S_k is the set of surfels in keyframe K, E_{geom}, E_{photo} are the geometric and photometric objective functions, ρ_{Tukey} is the Tukey loss function, ρ_{Huber} is the Huber loss function, σ_D, σ_p are estimates for the standard deviations of the depth and photometric measurements and λ_{photo} is a user-defined constant. The robust Tukey loss function and Huber loss function are both robust loss functions which are used to learn models in the presence of outliers, resulting in less extreme outputs for large errors compared to other loss functions such as the MSE.

The geometric objective function computes the point-to-plane distance between the surfel and its projection from the keyframe and is defined as follows:

$$E_{geom}(s,k) = (T^k_G n_s)^T (\pi^{-1}_{D,k}(\hat{\pi}_{D,k}(T^k_G \mathbf{p}_s)) - T^k_G \mathbf{p}_s) \tag{2.38}$$

where $\mathbf{p}_s$ is the 3D point of the surfel s, $\hat{\pi}_{D,k}(x)$ is the function that projects a 3D point of a surfel $\mathbf{p}_s$ to the depth

map D in keyframe k and outputs the value in that depth map, $\pi_{D,k}^{-1}(x)$ is the inverse to project depth vales to 3D points so the difference is between $\mathbf{p}_s$ and its corresponding depth value in keyframe k. Both $\hat{\pi}_{D,k}(x)$ and $\pi_{D,k}^{-1}(x)$ are for the local coordinates of keyframe k, so they are transformed to the global coordinates of the surfel point cloud with T_G^k. So the distance between $\mathbf{p}_s$ and its corresponding depth value is computed in the normal direction with the surface normal n_s of the surfel to get the point to plane distance.

The photometric objective function computes the difference between the surfel scalar visual descriptor d_s and its projected gradient magnitude and is defined as follows:

$$E_{photo}(s,k) = \left|\left| \begin{pmatrix} I(\pi_{I,k}(\mathbf{s}_1)) & I(\pi_{I,k}(\mathbf{p}_s)) \\ I(\pi_{I,k}(\mathbf{s}_2)) & I(\pi_{I,k}(\mathbf{p}_s)) \end{pmatrix} \right|\right|_2 - d_s \tag{2.39}$$

where $\pi_{I,k}(x)$ is the function to project a 3D point to the RGB image in the keyframe k and outputs the pixel colour, $I(x)$ is the function to compute the image intensity of a pixel and $\mathbf{s}_1, \mathbf{s}_2$ are points on the surfel's boundary defined by its surface radius $\mathbf{r}_s$ with $\mathbf{s}_1 - \mathbf{p}_s$ and $\mathbf{s}_2 - \mathbf{p}_s$ being orthogonal.

The optimisation stage of this algorithm is done after a new keyframe is added to the input sequence. The objective function is minimised with alternating optimisation to speed up computation, where different parameters are updated separately with each iteration in the optimisation algorithm. In a single iteration, first the 3D positions and scalar visual descriptors of the surfels are updated. When optimising the 3D position of a surfel, it is only moved along the normal direction to prevent holes in the surfel cloud. This is done by parameterising the position as $\mathbf{P}_s + t \cdot n_s$ with only t being optimised. Next, surfels are merged, which contain similar normal vectors and 3D positions. This merging step is only done on the first iteration after position optimisation to remove the surfels added due to noise in the keyframe. Then, the poses for each keyframe and the camera intrinsics are updated. The Gauss-Newton method is a method used to minimise the objective function (Equation 2.37) by alternating which variables are updated for each optimisation stage. This is done for the surfel parameters, the camera poses, and the camera intrinsics as the 3 sets of variables that are updated alternately. The Gauss-Newton method uses gradient-based optimisation to minimise the objective function. More details about the Gauss-Newton method can be found in "Numerical Optimisation" [39, pp. 254–255].

2.1.7 DROID-SLAM: Deep Visual SLAM for Monocular, Stereo, and RGB-D Cameras

The method "DROID-SLAM: Deep Visual SLAM for Monocular, Stereo, and RGB-D Cameras" [40] uses RGB-D video as its input with unknown poses. This method differs from the method "BAD SLAM: Bundle Adjusted Direct RGB-D SLAM", which uses mathematical and physics-based rules to project and optimise the 3D geometry and camera poses, where DROID-SLAM uses an end-to-end Deep Learning approach to determine the camera poses and 3D geometry.

It uses the RGB images as inputs to a neural network that computes optical flow [41] between pairs of images. The optical flow is used to optimise an objective function along with the input depth maps, which will refine the estimated poses and depth maps. The refined depth maps and poses can then be used to construct a 3D representation of the scene.

Optical flow estimation

First, a feature network shown in Figure 2.4 is used to extract feature maps for a pair of input images. The feature network is a convolutional neural network that contains 3 downsampling layers to output a feature map, which is 1/8 the size of the input image.

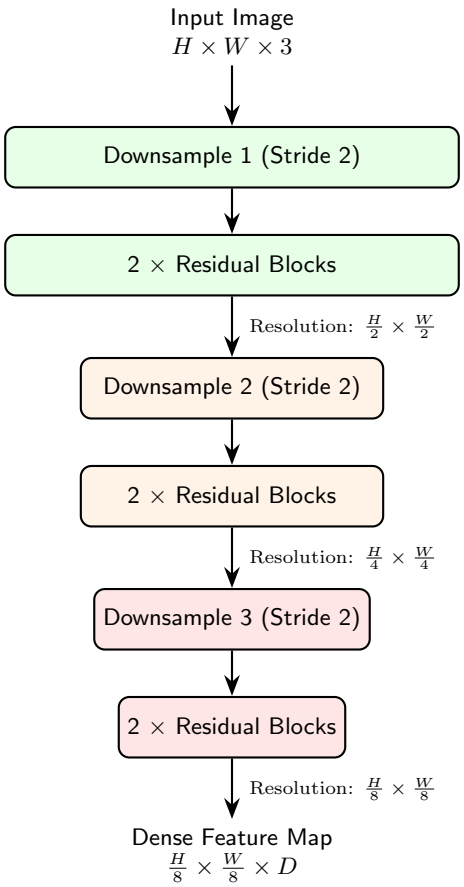

Figure 2.4: The architecture of the DROID-SLAM Feature Extraction Network. The spatial resolution reduces as it goes down through the layers, as the feature map is downsampled. For an input image of height H pixels and width W pixels with 3 colour channels, a $\frac{H}{8} \times \frac{W}{8}$ feature map is outputted with D feature channels.

A context network that has an identical architecture to the feature network is used to extract a context feature map. For a pair of RGB input images, the optical flow between them is computed by first creating a correlation pyramid. The feature maps ϕ^i, ϕ^j for images i and j are used to create a 4D correlation volume by computing the dot product of each feature pair as follows:

$$C^{ij}_{x_i,y_i,x_j,y_j} = \langle \phi^i_{x_i,y_i}, \phi^j_{x_j,y_j} \rangle \tag{2.40}$$

where $\phi^i_{x_i,y_i}$ is the feature vector at pixel x_i, y_i in the feature map for input image i and $\phi^j_{x_j,y_j}$ is the feature vector at pixel x_j, y_j in the feature map for input image j. The correlation pyramid is created with correlation volumes at 4 different sizes for the image pair. This is done by downsampling the last 2 dimensions for the correlation volume with average pooling.

Next, a correspondence field is initialised with an optical flow of 0 so each pixel coordinate in i maps to the same coordinate in j. Then, once pose and depth map estimates are found, these are used to compute an initial correspondence field and optical flow. The correspondence field p_{ij}, where the set of all the 2D image coordinates for each pixel in image i is mapped to pixel coordinates in image j. This mapping is computed as follows:

$$p_{ij} = \Pi_c(T_{ij}\Pi_c^{-1}(p_i, d_i)) \tag{2.41}$$

where Π_c^{-1} is inverse projection function to map the pixel coordinates p_i of image i and the corresponding depth map d_i to 3D points. T_{ij} is the transformation to map the 3D points in the local coordinates of image i to image j. Π_c is the projection function to map the 3D points onto the image. From this correspondence field, the optical flow is computed as the coordinate change for each pixel $p_{ij} - p_i$.

Next, the correspondence field is used to index each correlation volume in the correlation pyramid to get correlation values in the neighbourhood of the correspondences. This indexing operation will select the correlations between the pixel in image i and the pixels in a window with radius $r = 3$ around the pixel found with the correspondence field in image j. This is done for each volume in the pyramid and concatenated together into a grid of vectors, which is then concatenated with the optical flow.

Next, the correspondence field is updated with a recurrent neural network that will output updates to the correspondence field estimates so the network can be iteratively applied to improve the estimates over many steps. The grid of concatenated correlation values and optical flow vectors is input into the network, which has 2 convolutional layers, and then the outputs are added to the feature map output from the context network. The context feature map is extracted from image i. The network next has a convolutional gated recurrent unit that will output its hidden state as a grid of vectors. These hidden states are fed through 2 convolutional layers to output the final grid of correction terms r_{ij} as well as a confidence map. The grid of correction terms is added to the correspondence field $p_{ij}^* = p_{ij} + r_{ij}$ to get a corrected correspondence field p_{ij}^*.

Refining pose and depth estimates

Next, an objective function is minimised to refine pose and depth map estimates. For the input sequence of RGB and depth frames, the optical flow between consecutive frames is found. The objective function is minimised

after each frame is processed to get depth map and pose estimates that are used to compute optical flow between non-consecutive frames when the current frame is viewing the same area of the scene.

The objective function is computed as follows:

$$E(\mathbb{T}, \mathbb{D}) = \sum_{(i,j)\in\mathcal{E}} ||p^*_{i,j} - \Pi_c(T_{ij}\Pi_c^{-1}(p_i, d_i)||^2_{\Sigma i,j} + \lambda \sum_{i\in\mathcal{E}} ||d_i - m_i||^2_2 \tag{2.42}$$

where $\mathcal{E}$ is the set of input frames to be optimised for, $||\cdot||_\Sigma$ is the Mahalanobis distance, which uses the confidence weights outputted by the network, and m_i is the measured depth map in the input frame.

This objective function is minimised with the Gauss-Newton algorithm, which will then be backpropagated to train the network. Finally, the depth maps and camera poses are used to reconstruct a final 3D point cloud representing the scene.

Training the network

The network is trained with the synthetic TartanAir dataset [42]. This dataset consists of 1,037 long motion sequences of RGB stereo and depth frames as well as ground truth labels. These data samples were generated in Unreal Engine [43] using the AirSim plugin [44]. To use this dataset for training the network, it is split into examples of small 7 frame sequences with optical flow and pose ground truths for each frame. The training is done by backpropagating through the objective function minimisation algorithm, with a range of iterations done with the optical flow estimation network. This is done by using the network to update the optical flow estimations up to N iterations, collecting the estimations after each update step, then performing the objective function minimisation on optical flow estimations with the same number of update steps separately. These optical flow estimations are generated by collecting the iterative estimations to get a set of poses $\{T^1, T^2, ..., T^N\}$ and depth map estimations $\{d^1, d^2, ..., d^N\}$. Then these outputs are used to generate the set of optical flow estimations $\{O^1, O^2, ..., O^N\}$ using 3D projection. The loss function is computed by using an exponentially increasing weight to sum the losses for each update step estimation as follows:

$$\mathcal{L} = \sum_{n}^{N} \gamma^{n-N} \mathcal{L}_n \tag{2.43}$$

where γ is a user-defined constant between 0 and 1. $\mathcal{L}_n$ is the loss of the estimations with n update steps being used:

$$\mathcal{L}_n = \sum_i ||O_i^n - F_i||_1 + \sum_i || \log((T_i^n)^{-1} G_i)||_2 \tag{2.44}$$

where F_i is the optical flow ground truth for input frame i and G_i is the pose ground truth. O_i^n, T_i^n are the predicted optical flow and pose for input frame i with n update steps being used.

2.2 LiDAR and RGB stereo data fusion methods

In this research, the techniques that are focused on use RGB cameras and a LiDAR sensor. In LiDAR and RGB stereo data fusion, these sensors are used to capture multiple images of the same scene and are fused into a single dense depth image representation. This approach to data fusion is the core focus of 4 and is chosen as an area of focus in this research, as there exists a large amount of real-world data that is open and available to use to train data fusion models and evaluate their performance.

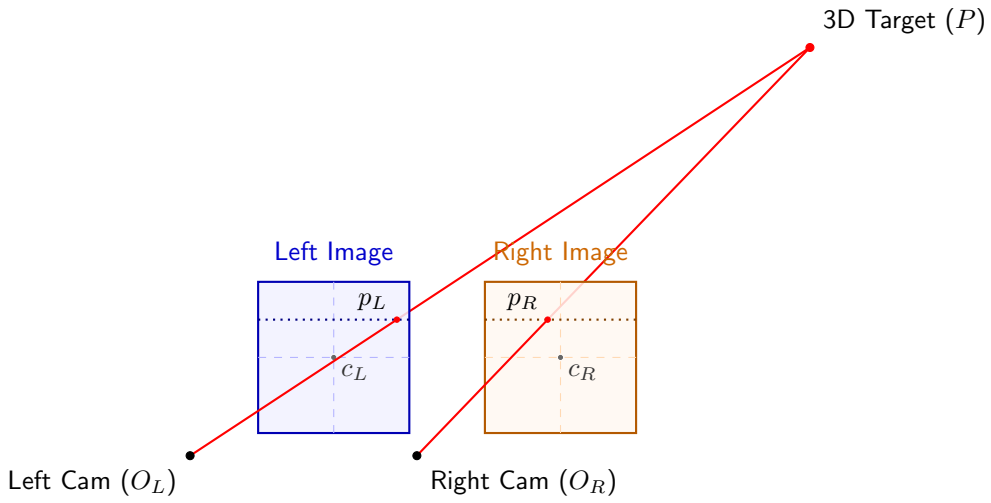

Figure 2.5: The setup for capturing stereo images, shown capturing the same 3D point at different points along a horizontal line in both images. This offset is computed to get a disparity map, which can then be converted into a depth map using the camera parameters.

For LiDAR and RGB stereo data fusion, the RGB cameras will capture two stereo images visualised in Figure 2.5, and the LiDAR sensor will capture a 3D point cloud. The point cloud will be projected onto one of the RGB images, and the fusion method will be used to combine this data into a single dense depth map.

Depth computed from stereo will be less reliable in environments with high occlusion and a lack of textured surfaces. LiDAR sensors will capture point clouds, which will result in sparse depth maps when projected. These modalities can be combined into a depth map, which is dense and can be more accurate in an environment with a lack of occlusion and less texture, which is useful for autonomous driving applications.

The depth map can be computed using the 2 RGB cameras with stereo matching to get a disparity map, which can be converted into a depth map using the camera parameters. Recently, the most predominant techniques for computing disparity from rectified stereo images have been deep learning-based methods [27, 45, 28, 46, 47, 48, 26, 49, 50, 51, 52] started by MC-CNN [53] and then extended to use end-to-end trainable networks starting with DispNet [54].

The depth maps produced directly from LiDAR sensors are sparse, so the task of depth completion addresses this by estimating depths for the missing values as well as reducing the error in the existing sparse depth values. These methods are often data-fusion methods that use an RGB image as an additional input with the sparse depth map. Recent methods for depth completion from sparse depth maps are primarily deep learning-based methods. The depth completion methods CSPN [55] and CSPN++ [56] use a Convolutional Spatial Propagation Network, which propagates neighbouring depth information with a recurrent operation of the convolutional neural network (CNN). The method FCFR-Net [57] uses a two-stage deep learning approach where it first uses a simple CNN to get a coarse depth estimate from a sparse depth input, then it estimates a fine depth map with a Coarse-to-Fine CNN model.

The method RigNet [58] uses a two-branch deep learning network architecture, where it uses one branch for the estimation of the depth features with an RGB guidance image. The other branch uses the sparse input depth and the features from the guidance image branch to make the final depth estimation. The method DeepLiDAR [59] uses a similar two-branch approach, but it estimates the surface normals as an intermediate stage of the depth branch and fuses the depth estimations of the two branches to get the final depth estimate. The Joint 2D-3D method [60] uses both 2D and 3D convolutions using a two-branch approach with a 2D convolution branch and a 3D convolution branch, then fusing them to get the final depth map estimate.

The method ACMNet [61] uses a graph-based model to handle the input depth information as a graph representation. The Plane-Residual method [62] uses a Plane-Residual representation, which the CNN is trained to estimate, so that from this estimated representation, the final depth map output can be obtained. The Multi-task GANs method

[63] uses a Generative Adversarial Network (GAN) to perform the generation of both semantic and depth maps using a sparse depth map and an RGB image.

The depth map can be computed using both the RBG stereo images and the sparse LiDAR depth map. The most recent methods are deep learning-based [64, 65, 66] and use similar network architectures to deep learning-based stereo methods.

Other methods work by computing the depth map from a single RGB image, or another way is by using only the sparse LiDAR depth map as the input. However, this research only focuses on the methods that use data from multiple sensors for data fusion. To determine if data fusion is better than non-data fusion approaches, comparisons will be made to existing single modality state-of-the-art methods, as well as comparing to the input data individually to determine if the fused result is more accurate than each input data source alone.

In summary, the current state-of-the-art methods for data fusion of LiDAR and RGB stereo data are primarily end-to-end trained deep learning-based methods. These methods are limited by data availability due to the necessity to have a multi-sensor setup to capture data that can be used for data fusion. This research aims to solve the limitations of data availability in data fusion so it can be applied to flaw detection.

2.3 Flaw detection

In recent years, deep learning methods have emerged as effective tools for flaw detection, leveraging the power of data-driven models. This section provides an overview of the literature on flaw detection, discussing the approaches and techniques used in this field.

Deep learning methods have been successfully applied to flaw detection tasks. These models are trained on labelled data to predict flaws accurately. For instance, in additive manufacturing, researchers have used a fully connected Neural Network (NN) architecture and data fusion techniques to predict defects from multiple sensors [67] using multiple modalities such as electro-optical (EO) imagery and multi-spectral (MS) emissions.

However, this approach is limited by the network architecture, which is unable to scale to higher-dimensional input data. To overcome this, convolutional neural network (CNN) architectures are used, where in the manufacturing industry, fault detection in machinery has been achieved by using sensor fusion and a CNN model [68], where

high-dimensional timeseries data is processed by a CNN model. Although visual image data was not used in this approach, it demonstrates the success of CNN models in being applied to diverse sensor modalities, including time series sensor data, as well as the application areas where flaw detection using deep-learning-based sensor fusion approaches has been applied.

In addition to additive manufacturing and manufacturing machinery, flaw detection using deep learning has also been explored in other domains. Ultrasound sensors, for example, have been utilised to predict flaws in specimen objects, offering a non-invasive and efficient approach [69, 70]. Here, the specimen object means the object that is being examined to determine if it has a flaw or not. For flaw detection tasks, CNN models have been widely adopted in detection methods that involve image data. Popular CNN architectures such as YOLO (You Only Look Once) [71, 72] and SSD (Single Shot MultiBox Detector) [73] have been used to enhance the accuracy and efficiency of flaw detection systems [74].

Beyond CNNs, other deep learning approaches have been investigated for flaw detection. Recurrent models, which excel at capturing temporal dependencies, have shown promise in this domain [75]. By leveraging the sequential nature of data, recurrent models can effectively identify flaws in dynamic processes. Additionally, unsupervised training methods, such as those based on autoencoders, have been explored for flaw detection [76]. These techniques aim to uncover hidden patterns and anomalies in the data, enabling the detection of flaws without relying on labelled examples.

The data fusion approach for flaw detection has similar techniques to those used to detect medical issues from data, such as brain diseases, which include Alzheimer's disease, Parkinson's disease, multiple sclerosis, schizophrenia and brain tumors[77]. It has been successfully applied in medical datasets, where multiple sources of data are combined to enhance the accuracy and reliability of detection [78]. These papers inform this research by presenting novel deep learning architectures and training techniques that can be adapted to other tasks, such as the U-Net architecture [79], which was designed for medical image segmentation but has since been widely used for other segmentation tasks [80].

2.4 Crack detection in roads

Roads are a vital part of modern infrastructure and support economic growth [81] by improving productivity, trade, investment, and labour supply, as well as improving overall quality of life. Well-maintained roads ensure efficient traffic flow, reduce travel time and lower vehicle operating costs, whereas deterioration increases fuel use, wear and tear, and journey times [82]. However, over time, roads and pavements are likely to develop cracks and other forms of damage due to heavy traffic loads and weather conditions. Early cracks in roads can lead to further issues that can deteriorate the underlying structure and can result in more severe damage, posing risks to vehicles and increasing maintenance costs. Manual inspection of roads for signs of damage is labour-intensive, time-consuming, and subject to human error [83]. These issues can be addressed by automated methods, specifically computer vision methods that use sensors such as RGB to identify cracks, reducing the cost and time of the process.

The earlier approaches to automated detection of cracks using RGB images were using traditional image processing methods such as edge detection [84] and thresholding [85]. These techniques relied heavily on manual feature extraction and required precise tuning of parameters to accurately identify cracks. Later, to address these limitations, classical machine learning methods were applied to this problem, using algorithms such as Support Vector Machines (SVM) [86] and Random Forests [87]. However, these methods still relied on hand-crafted features, which are not robust to varying lighting conditions or complex surface textures.

In recent years, deep learning methods, which do not use any hand-crafted features, have been applied to the task of crack detection [80, 88, 89] following their success in detection tasks within the medical field and quickly showed superior results [90, 91, 92] at crack detection compared to traditional image processing methods and classical machine learning methods.

Deep learning crack detection can be done by image classification [93], object recognition [94], or image segmentation [95]. Image classification is the estimation at the image or patch level of whether it contains a crack. Object recognition is estimating the location of the crack by outputting bounding boxes that draw a box around the cracks in the image. Image segmentation will do classification at the pixel level, predicting if each pixel is within a crack or not.

Crack segmentation stands out among these methods for crack detection because it can give complete information

about the exact shape of the crack. This is needed for further analysis when determining the severity of a crack. As a result, crack segmentation has become the most widely used approach for crack detection in recent years [80].

The use of RGB-D data is an extension to RGB segmentation [96], where there is an additional data modality with a depth map. RGB-D data includes colour information from the RGB image and depth information from the depth map, which adds additional information to be used by the model, where the information is fused within the neural network. The model is trained to predict the final segmentation map using this additional information, which results in a more accurate segmentation result than with the RGB image alone. This shows a promising direction for crack detection research since it has been successfully applied to the general segmentation of objects in indoor scenes [97, 98]. This could be a promising direction for crack detection research. However, RGB-D data and image fusion have not yet had much research within crack detection [99, 100] due to issues with the lack of data availability, which is essential for deep learning-based methods.

2.5 Crack segmentation

Crack segmentation is a vital aspect of flaw detection, particularly when it comes to surfaces. Deep learning methods have gained popularity as effective approaches for crack detection. Within crack detection, there are three main techniques: classification, bounding box detection, and segmentation [88, 80]. However, crack segmentation stands out as it enables precise shape estimation at the pixel level, leading to extensive research in this area [89].

In the area of pavement crack estimation, CNN models have been widely used [101]. Notably, the encoder-decoder [102, 103, 104] approach and specifically the UNet architecture have gained significant traction [105, 106, 107]. UNet architectures incorporate skip connections between the encoder and decoder layers at the same resolution, facilitating more accurate estimations. Researchers have also explored alternative training approaches for crack estimation, such as transfer learning [108] and generative adversarial training [109]. These techniques aim to improve the performance and generalisation of crack segmentation models.

Attention mechanisms have emerged as a more recent method for crack estimation [110]. By emphasising relevant features and suppressing noise, attention mechanisms enhance the accuracy and robustness of crack segmentation models.

In the broader field of segmentation, state-of-the-art methods commonly use foundation models [111, 112]. These large-scale models are trained on numerous segmentation tasks, leading to improved accuracy for any specific task compared to training solely on that individual task [113]. Additionally, multi-model training approaches have demonstrated enhanced segmentation performance by combining vision and language modelling [114].

Vision transformers (ViTs) [115] have become the prevailing architecture for state-of-the-art segmentation methods [116, 117, 118, 119]. These models, originally developed for image recognition, have also been adapted for crack segmentation tasks [120, 121]. By leveraging the transformer architecture's attention mechanisms, ViTs offer powerful capabilities for capturing intricate details and accurately segmenting cracks. However, ViTs lack the inductive bias which the CNN architecture has with spatial locality and translation invariance, which results in CNNs having better efficiency for limited training data and compute [122].

2.6 Data generation

In most scenarios of flaw detection using deep learning methods, the availability of labelled training data can be limited. To address this challenge, synthetic data generation techniques have emerged as a valuable approach to augment the training dataset.

One application of synthetic data generation involves combining crack images with road scenes to generate realistic crack instances in roads, along with ground truth annotations from different viewpoints [123]. By synthesising diverse crack patterns within road environments, researchers can expand the training dataset and improve the generalisation of deep learning models for road crack detection. However, the data generation process requires careful design to ensure it correctly models the real-world data. Otherwise, the models trained on this data will not be able to generalise when tested on real-world data.

Another method for data generation involves creating 2D crack images from scratch using random noise to generate the background texture and define the crack shape [124]. By controlling the parameters of the generated noise and crack shape, researchers can simulate a wide range of crack patterns, thus enriching the training data and enhancing the model's ability to generalise to real-world crack scenarios.

In addition to road crack detection, data generation techniques have also been explored for other domains. For

instance, in the context of flaw detection in steel objects [125, 126], researchers have used data generation using tools such as Blender [127]. By creating 3D models of steel objects and overlaying crack textures onto them, realistic synthetic images can be rendered. This process enables the generation of large quantities of labelled crack images, facilitating the training of deep learning models for steel flaw detection. Cracks present in steel, asphalt and concrete are important to identify to maintain civil infrastructure. These different materials create cracks with different characteristics; however, these data generation methods can be adapted to produce cracks with the specific characteristics needed to match a material.

2.7 Conclusions

This research aims to apply a data fusion approach to the detection of cracks, to show an improvement from using multiple sensors that have lower error than each sensor individually. This approach is needed to overcome the limitations of only using a single sensor, where certain environments or scenarios can have significant error for that sensor. Developing this detection approach would have applications in manufacturing or infrastructure by improving automated detection of flaws. This assessment of the current knowledge base relevant to this research shows that the main deficiencies are the application of deep learning-based data fusion methods to the task of crack detection. Deep learning-based approaches to crack detection and data fusion are the current state-of-the-art. However, they are highly dependent on the training data used. The deficiency in the knowledge base for deep-learning data fusion approaches is due to the limitations of real-world data availability, which is essential for these methods. This informs the research that data-efficient methods should be explored, as well as synthetic data methods.

Chapter 3

Image Fusion

This chapter covers the work done on image fusion. The research aims to find how data fusion can be applied to crack detection to reduce errors. Image fusion looks at performing data fusion on data that is arranged spatially in a 2D grid and fusing multiple of these images. The images that are fused are captured with the same view of the scene so that corresponding values in the 2D grid, which have the same coordinates, are measuring the same point in the scene being captured. These images that are fused can capture visual information, such as images from RGB cameras, or capture geometric information, such as depth map images captured from depth sensors.

The work in this chapter covers the RGB data image fusion methods, which were "Image Fusion with Guided Filtering" [11] and "Infrared and Visible Image Fusion using a Deep Learning Framework" [12]. These methods were implemented, and novel modifications were made to the methods to find improvements. These improvements were assessed with RGB image fusion quality metrics to show the improvements over the original methods. The depth image fusion method "Stereo and ToF Data Fusion by Learning from Synthetic Data" [128] was implemented, and novel modifications were made to find improvements, which can be shown in the results by computing depth error metrics against the ground truth depth data.

The deficiencies shown in the knowledge base were shown to be data efficiency for the task of crack detection using a data fusion approach. This work addresses this difficulty by taking image fusion methods which do not have this constraint of requiring task-specific data since they are general task methods. Novel improvements were made to

these methods so that they can be incorporated into a full data fusion crack detection pipeline, resulting in lower error for detection.

3.1 Image Fusion with Guided Filtering

The first fusion algorithm implemented is "Image Fusion with Guided Filtering" [11], which has been shown to work well on a wide variety of image fusion tasks. These image fusion tasks are all fusing multiple images into a single, higher-quality image. The images that are fused are captured with the same view, where each image contains some differences, which include:

- Multi-spectral image fusion: This task takes images taken with different wavelength ranges.
- Multi-focus image fusion: This task takes images taken with different focal distances.
- Multi-modal image fusion: This task takes images taken with different sensor modalities, such as infrared and visible.
- Multi-exposure image fusion: This task takes images taken with different exposure settings.

3.1.1 Guided image filtering

Guided image filtering [129] is an optimisation-based image filtering method. It computes an output image Q given a guidance image I and an input image P. The output Q is filtered based on the contents of I to allow guidance of the filtering. The guidance image I can be chosen as the input image P itself or another different user-determined image. The method applies a linear transform on the guidance image I within a window ω_k where k is the pixel at the centre of the window to output each pixel Q_i in the window w_k:

$$Q_i = a_k I_i + b_k, \forall i \in \omega_k \tag{3.1}$$

For each pixel in the image, this transformation is computed using all the pixels from the corresponding window to construct a full output image. The pixels near the edges are computed by padding the pixels around the image with a reflection of the pixels on the edge of the image, so k can be any pixel in I. The window size is a user-determined parameter. For the window size used in the implementation, see Section 3.1.3.

The filtered output will have multiple values for each pixel from overlapping windows, so the pixel values are averaged into a single pixel value. The coefficients a_k, b_k are constant in the window ω_k and are computed by minimising the difference between the output filtered image from the linear transformation and the input image P and reducing ϵa_k^2 to prevent a_k from being too large with regularisation parameter ϵ:

$$(a_k, b_k) = \sum_{i \in \omega_k} ((a_k I_i + b_k - P_i)^2 + \epsilon a_k^2) \tag{3.2}$$

This cost function is solved with linear regression as follows:

$$a_k = \frac{\frac{1}{|\omega|} \sum_{i \in \omega_k} I_i P_i - \mu_k \bar{P}_k}{\sigma_k^2 + \epsilon} \tag{3.3a}$$

$$b_k = \bar{P}_k - a_k \mu_k \tag{3.3b}$$

where $|\omega|$ is the number of pixels in the window ω_k, μ_k is the mean of the guidance image I in the window ω_k, σ^2 is the variance of the guidance image I in the window ω_k and $\bar{P}_k$ is the mean of the input image P in the window ω_k. Since the final output is the average of the overlapping windows, the coefficients can be first averaged where they overlap to calculate coefficients $\bar{a}_i, \bar{b}_i$ for each pixel in the guidance image I, so each pixel can be computed directly:

$$\bar{a}_i = \frac{1}{|\omega|} \sum_{k \in \omega_i} a_k \tag{3.4a}$$

$$\bar{b}_i = \frac{1}{|\omega|} \sum_{k \in \omega_i} b_k \tag{3.4b}$$

$$q_i = \bar{a}_i I_i \bar{b}_i \tag{3.4c}$$

For RGB images with 3 channels, each channel in the input image P can be computed separately with a 3-channel guidance image I as follows:

$$q_i = a_k^T I_i b_k, \forall i \in \omega_k \tag{3.5}$$

where the user determined RGB guidance image is I and I_i is a 3×1 colour vector at pixel i in the guidance image and a_k is a 3×1 coefficient vector, so b_k is a scalar to output a scalar Q_i. This is computed for each colour

channel in the input image p, which then generates each colour channel in the output image q, so the coefficients are computed as follows:

$$a_i = (\sum_k + \epsilon U)^{-1}(\frac{1}{|\omega|}\sum_{i \in \omega_l} I_i P_i - \mu \bar{P}_k) \tag{3.6a}$$

$$b_i = \bar{P}_k - a_k^T \mu_k \tag{3.6b}$$

where $\sum_k$ is the 3×3 covariance matrix of guidance image I in window w_k and U is the 3×3 identity matrix.

Figure 3.1: Illustration of before and after a guided filter is applied. The left image shows the image that was used as both the guidance image and input image. The right shows the output of the guided filter, where the image has been smoothed, but the edges are preserved.

Shown in Figure 3.1, this filter has an edge-preserving and smoothing property that works by outputting a pixel that is closer to the average of its neighbouring pixels if it is in an area of the image with a low variance of pixel values. If it is in an area of the image where there is a high variance in the pixel values, the output pixel value will be closer to its input pixel value. An example of this code can be seen in the linked Codebase [130].

3.1.2 Image fusion

Next, the implementation of guided filtering for image fusion [11] was used. This method takes a set of images and applies guided filtering to compute a weight map for each image. The set of input images can be of any size. The

fusion is done at 2 levels by first decomposing each source image I into a base layer B and detail layer D so the set of detail layers and set of base layers can be fused separately and then finally combined into the fused image.

For each source image, the base layer is computed by applying an average filter to the source image. Then, the detail layer is computed by subtracting the base layer from the source image $D = I - B$, so the source image is reconstructed by adding the detail and base layer together.

This decomposition process is so that the base layer has a large-scale variation with low-frequency changes, and the detail layer contains a smaller-scale variation with high-frequency changes. For fusion, the weight maps need to be computed, with each source image having weight maps for both the base and detail layers, which contain weights for each pixel. Each weight map is computed by first applying a 3×3 Laplacian filter to the source image before decomposition, then taking the absolute value of each pixel and applying a Gaussian filter to get the local average.

Figure 3.2: Illustration of the saliency map output. The left image shows the image that was used as the input image. The right image shows the saliency map output. The high values shown in white correspond to the important information in that image.

This output will be a saliency map shown in Figure 3.2, where it will have high values where its source image pixel contains important information, so comparing across each saliency map at the same pixel, the source image with

the most information can be determined for each pixel as follows:

$$P_n^k = \begin{cases} 1 & \text{if} \quad S_n^k = \max(S_1^k, S_2^k, ..., S_N^k) \\ 0 & \text{otherwise} \end{cases} \tag{3.7}$$

where N is the number of source images, and S_n^k is the value at pixel k of the saliency map. S_n^k is computed from the source image n, so for P_n^k to be 1, S_n^k has to have the largest pixel value across the set of all saliency maps. This computation will give the initial weight map P. Next, to generate the weight maps for both the base layer and detail layer, guided filtering is used to filter the initial weight map P by using it as the input image, and the source image is used as the guidance image with different window radius for the base and detail layers as follows:

$$W_n^B = G_{r_1,\epsilon_1}(P_n, I_n) \tag{3.8a}$$

$$W_n^D = G_{r_2,\epsilon_2}(P_n, I_n) \tag{3.8b}$$

with r_1 and r_2 being the window radius for guided filtering, so the window width is $2 \times r + 1$. ϵ_1 and ϵ_2 are the regularisation parameters. These filtered weight maps W^B and W^D are normalised by dividing by the pixel sum across all weight maps, so the weight maps will sum to 1 at each pixel for both sets of weight maps for the base and detail layers. To fuse the sets of base and detail layers, they are multiplied by corresponding weight maps and then summed to get a single fused base and detail layer as follows:

$$\bar{B} = \sum_{n=1}^{N} W_n^B B_n \tag{3.9a}$$

$$\bar{D} = \sum_{n=1}^{N} W_n^D D_n \tag{3.9b}$$

where B_n and D_n are the base and detail layers from source image I_n. Then, the fused base layer $\bar{B}$ and the fused detail layer $\bar{D}$ are summed together to get the final fused image.

3.1.3 Experiment with multi-layer fusion

To improve this method, fusion at more layers than just a base layer and a single detail layer was tested. The source images were decomposed into 2 detail layers as follows:

$$B_n = Z_{w_1}(I_n) \tag{3.10a}$$

$$D_n^1 = Z_{w_2}(I_n) - B_n \tag{3.10b}$$

$$D_n^2 = I_n - Z_{w_2}(I_n) \tag{3.10c}$$

where Z is an average filter and w_1, w_2 are the widths of the windows in which the pixels are averaged. For source image I_n, a base layer B_n and 2 detail layers D_n^1, D_n^2 are generated. The parameters used for testing were $w_1 = 31$ and $w_2 = 15$ with $r = 45, \epsilon = 0.3$ for the base layer, $r = 20, \epsilon = 0.1$ for detail layer 1 and $r = 7, \epsilon = 1 \times 10^{-6}$ for detail layer 2. Where r is the radius of the square window used in the Guided image filtering for a window size of $(2r + 1, 2r + 1)$. These values were chosen to follow the parameters used in "Image Fusion with Guided Filtering" [11].

Results for multi-layer fusion

This section investigates the results from modifying the method to apply fusion at more layers than just a base layer and a single detail layer. The research aims to apply data fusion to the task of crack detection to find improvements in reducing errors in detection. RGB image data is needed in the detection task, so the fusion of RGB data can be used to improve detection by producing higher-quality images on which detection is performed. Finding improvements to existing image fusion methods can then be used in a full crack detection pipeline, resulting in reduced error in detection. These results investigate whether the proposed modification will result in improved image fusion.

The method just uses a single base layer and a single detail layer. This limits the method, as using multiple detail layers could allow for improved fusion by separating the visual information of the images into these detail layers to be fused separately. To get results, the method Image Fusion with Guided Filtering [11] is implemented and changed to use 2 detail layers instead of only 1 detail layer.

The implementation was evaluated for multiple sets of images that have some variation between them. The method was first tested on a dataset with variation in focal length [131] to see through visual inspection and with image quality metrics if the fused image is of higher visual quality, containing more information than either of the input images alone. Only two source images were used for fusion, so the visual results are easier to interpret.

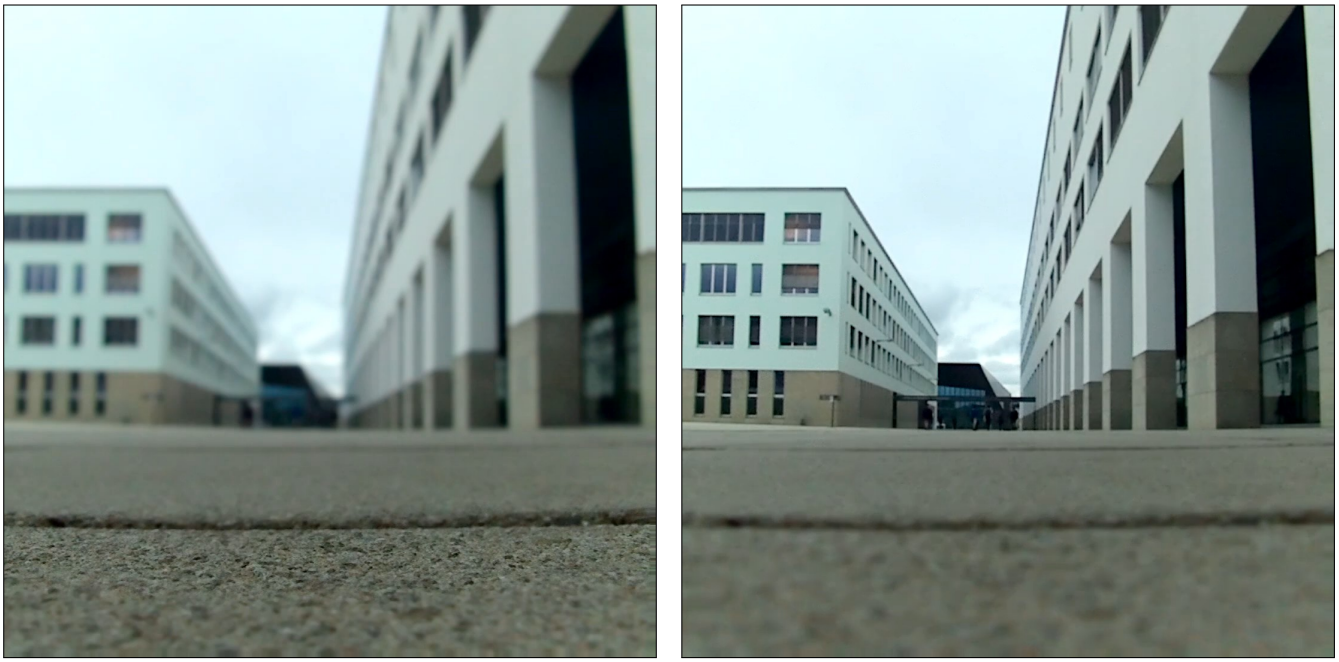

Figure 3.3: The two source images from the multi-focus dataset. The left image has a short focal distance where the buildings in the background are blurred, and the ground close to the camera is in focus. The right image has a long focal distance where the buildings in the background are in focus, and the ground close to the camera is blurred.

Figure 3.4: The two fused images using the original (left), modified method (middle) and the difference between them (right). Both the fused images show successful data fusion as they show both the background and foreground in focus. The right image of the differences created by subtracting the left image from the middle image shows that the modified method is only producing differences in a small number of places, shown in white, while the rest is shown to have no differences in black.

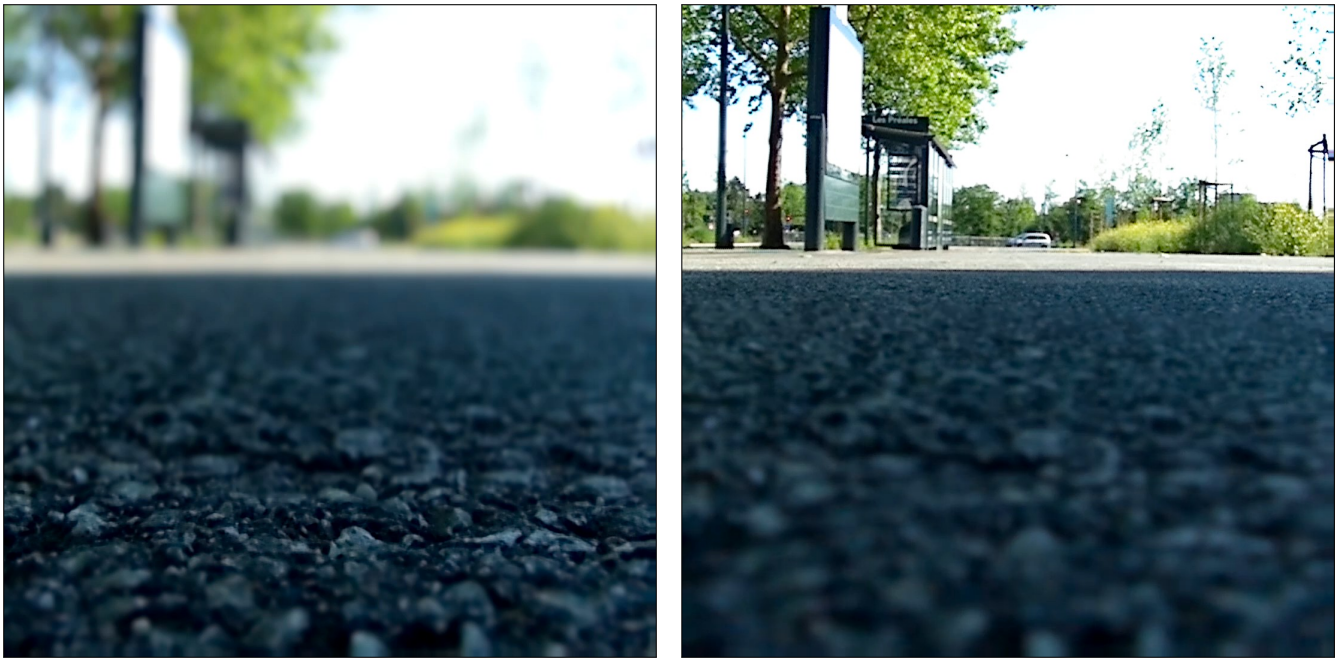

Figure 3.5: The two source images from another scene in the multi-focus dataset. The left image has a short focal distance where the trees and objects in the background are blurred, and the ground close to the camera is in focus. The right image has a long focal distance where the trees and objects in the background are in focus, and the ground close to the camera is blurred.

Figure 3.6: The two fused images using the original (left), modified method (middle) and the difference between them (right). The right image of the differences created by subtracting the left image from the middle image also shows that the modified method is only producing differences in a small number of places, shown in white, while the rest is shown to have no differences in black.

From visual inspection, these results (Figures 3.3 to 3.6) show that the fusion method is outputting an image with more information than from a single source image (Figures 3.3, 3.5) as the fused images (Figures 3.4, 3.6) combine the sharp regions from both souce images so both the foregrounds and backgrounds are in focus. The left source images have a short focal distance, so the image is sharper where it is closer to the camera, and the right images have a longer focal distance, so they are sharper where they are farther from the camera. The method sets the

correct weights for these pixels to compute the final image that is clear in both these areas by setting higher weights for those sharper regions and lower weights where it is blurred. From the fused images, the difference in the quality of image fusion is not distinguishable between the original method and the multi-layer decomposition method with multiple detail layers that were implemented. The difference image shows the areas where it is different in white; however, these are only in a few areas, and from visual inspection, it is challenging to see what has changed.

To objectively evaluate the fusion accuracy of the modified implementation, metrics to compute image fusion quality were used. The metrics take the 2 source images and the fused image as inputs and output a score that indicates how similar the fused image is to both of the source images. These metrics were used to evaluate the implemented method that uses 2 detail layers, and then the original method was evaluated that uses a single detail layer, and the change in the score between the 2 methods was recorded.

The first metric described by Yang et al. is Q_Y [132], which uses the structural similarity index measure (SSIM) [133] to compare the fused image with both of the source images as follows:

$$Q_Y = \begin{cases} \lambda_w \text{SSIM}(A_w, F_w) + (1 - \lambda_w)\text{SSIM}(B_w, F_w), \\ \qquad \text{if SSIM}(A_w, B_w|w) \leq 0.75 \\ \max\{\text{SSIM}(A_w, F_w), \text{SSIM}(B_w, F_w)\} \text{ otherwise} \end{cases} \tag{3.11a}$$

$$\lambda_w = \frac{var(A_w)}{var(A_w) + var(B_w)} \tag{3.11b}$$

where w is a window of width 7 and the source images are A, B and the fused image is F. The local weight λ_w is calculated using the $var(A_w), var(B_w)$, which are the variances of the source images A, B in the window w.

The metric Q_C by Cvejic et al. [134] uses the universal image quality index (UIQI) [135] to compare the fused image to the source images as follows:

$$Q_C = \mu(A_w, B_w, F_w)\text{UIQI}(A_w, F_W) + (1 - \mu(A_w, B_w, F_w))\text{UIQI}(B_w, F_W) \tag{3.12a}$$

$$\mu(A_w, B_w, F_w) = \begin{cases} 0 & \text{if} \quad \frac{\sigma_{AF}}{\sigma_{AF}+\sigma_{BF}} < 0 \\ \frac{\sigma_{AF}}{\sigma_{AF}+\sigma_{BF}} & \text{if} \quad 0 \leq \frac{\sigma_{AF}}{\sigma_{AF}+\sigma_{BF}} < 1 \\ 1 & \text{otherwise} \end{cases} \tag{3.12b}$$

where σ_{AF} is the covariance between source image A and the fused image F and σ_{BF} is the covariance between source image B and fused image F.

The metric Q_G by Xydeas et al. [136] uses a measurement for edge information preservation to compare the source images to the fused image as follows:

$$Q_G = \frac{\sum_{i=1}^{N}\sum_{j=1}^{M}(Q^{AF}(i,j) + Q^{BF}(i,j)\tau^B(i,j))}{\sum_{i=1}^{N}\sum_{j=1}^{M}(\tau^A(i,j) + \tau^B(i,j))} \tag{3.13}$$

where Q^{AF} and Q^{BF} are edge information preservation measures that use the edge strength and edge orientation to compare the similarity between A, F and B, F at pixel i, j. N, M are the width and height of the source image, which are the same for both A and B.

The metric Q_{MI} by Hossny et al. [137] uses mutual information to compare the source images to the fused images:

$$Q_{MI} = 2[\frac{MI(A,F)}{H(A)+H(F)} + \frac{MI(B,F)}{H(B)+H(F)}] \tag{3.14}$$

where $MI(A, F)$ is the mutual information of source image A and fused image F and $MI(B, F)$ is the mutual information of source image B and fused image F. $H(A), H(B), H(F)$ are the marginal entropy of source images A, B and fused image F.

The modified fusion method using 2 detail layers was compared to the original paper's method, which used a 31×31 averaging filter to generate the base layer, and the parameters were set to $r = 45$ and $\epsilon = 0.3$ for the guided filtering for the base layer and $r = 7$ and $\epsilon = 1 \times 10^{-6}$ for the detail layer.

Results evaluated on three types of publicly available image datasets were obtained. A multi-exposure dataset [138] containing 36 stacks, a multi-spectral dataset [139] containing 32 stacks and a multi-focus dataset [131] containing 32 stacks.

- For the multi-focus dataset [131], each stack contains multiple images with a range of different focal distances.
- Each stack in the multi-spectral dataset [139] contains images with variation in wavelengths.
- Each stack in the multi-exposure dataset [138] contains images with variation in exposures.

Each dataset contains multiple stacks. These image stacks are just sets of images that all capture the same scene from the same view, but with differences between them depending on the dataset type, such as different focal lengths. Each of these stacks contains the same number of images. For each stack, the images contained are taken in the same scene with the camera in the same pose.

For an extensive evaluation of testing the image fusion of 2 input images, all combinations of 2 images are used for evaluation, which is 979 pairs for multi-focus, 1600 pairs for multi-spectral and 360 pairs for multi-exposure. Below are the results showing the percentage change of decomposing into a base layer and 2 detail layers from the source images when compared to only decomposing into a base layer and a single detail layer, so a positive change would show an improvement in accuracy from using the modified method.

Multi-Focus	Percentage improvement
Q_Y	0.2906%
Q_C	1.3709%
Q_G	1.0370%
Q_{MI}	0.8357%

Table 3.1: Results on the multi-focus dataset. The percentage increase is shown for each metric compared using 1 detail layer and 2 detail layers when tested on the multi-focus dataset.

Multi-Spectral	Percentage improvement
Q_Y	0.0921%
Q_C	0.5197%
Q_G	0.2400%
Q_{MI}	0.8065%

Table 3.2: Results on the multi-spectral dataset. The percentage increase is shown for each metric compared using 1 detail layer and 2 detail layers when tested on the multi-spectral dataset.

Multi-Exposure	Percentage improvement
Q_Y	0.2495%
Q_C	0.8984%
Q_G	0.2093%
Q_{MI}	0.4696%

Table 3.3: Results on the multi-exposure dataset. The percentage increase is shown for each metric compared using 1 detail layer and 2 detail layers when tested on the multi-exposure dataset.

The results (Tables 3.1 to 3.3) show that using 2 detail layers has a small improvement that is consistent for all the datasets. This shows that the modification made to the method with the use of 2 detail layers results in improved fusion quality over the original method's use of a single detail layer. The use of a single detail layer limits the image fusion, whereas using the 2 detail layers allows the method to separate information across these layers so they can be fused separately.

Next, the method was evaluated on depth data to see if this method is suitable for improving 3D reconstruction. To evaluate the fusion accuracy with depth data, the previous metrics were not suitable because a ground truth of the true depth values at each pixel was needed, so the accuracy of the fusion algorithm could be measured.

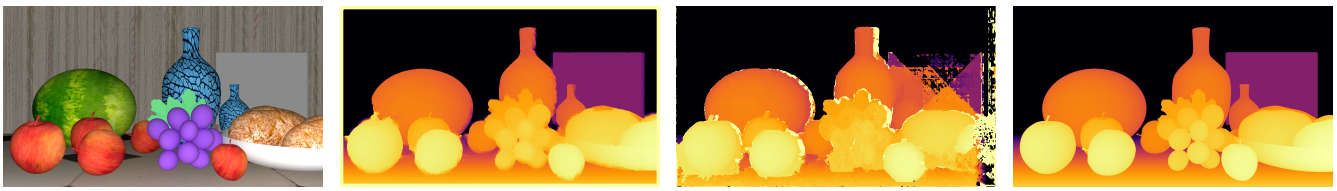

Figure 3.7: Example from the SYNTH3 dataset. From left to right, the first image shows the RGB left stereo rendered image. The second image shows the ToF depth map. The third shows the stereo depth map. The fourth image shows the ground truth depth map.

The dataset that was chosen to be used for testing is the SYNTH3 [128, 140] dataset, which is publicly available for use. SYNTH3 is a synthetic dataset that contains 15 3D scenes that have been created using Blender [127]. Figure 3.7 shows an example scene from the dataset. The 3D scenes are used to simulate using a time-of-flight camera and a stereo camera pair to generate 2 separate depth maps that have been obtained from different sensors. The scenes are generated to test a wide variety of scenarios that will be challenging for each of the sensors, including reflections, repetitive patterns, and global illumination. The time-of-flight and stereo camera pair were simulated using the 3D scene.

The chosen dataset contains disparity maps that have been computed with the Semi-Global matching stereo algorithm [141] from the pair of rendered stereo camera images. The Semi-Global matching stereo algorithm was not covered in Chapter 2 since it is a stereo-matching method aimed at real-time computation using traditional non-deep-learning based methods for computing the disparity. However, the Semi-Global matching stereo algorithm was used in this dataset to simulate the stereo depth map that would be captured by a stereo depth cameras which compute the disparity in real time, commonly using algorithms such as Semi-Global matching.

The simulated depth map from the ToF camera has been re-projected over the poses of the stereo cameras to get a disparity map, so there are 2 disparity maps from the stereo cameras and the ToF camera, which can be easily converted into a depth map. The ground truth disparity map is directly computed from the 3D scene. To evaluate, the mean squared error (MSE) and the mean absolute error (MAE) were used. These were calculated between the ground truth and the fused disparity map and averaged across all the scenes. Then, this was compared with the errors of the stereo and ToF disparity map to see if the error was reduced after fusion.

Disparity map	MSE	MAE
Stereo	17.2960	0.9225
ToF	8.1430	0.6756
Fused	14.6503	1.1138

Table 3.4: Results on the SYNTH3 dataset. The errors are shown compared to the ground truth when the implementation with 2 detail layers is tested on the SYNTH3 dataset.

These results (Table 3.4) show that although the MSE error for the fused disparity map is reduced compared to the stereo disparity map, it is still a greater error compared to the ToF disparity map. Also, for MAE, the fused disparity map was unable to reduce the error compared to either of the input disparity maps. For this research project, it means that the RGB image fusion method had limited success when applied to depth image fusion tasks. To overcome this, a separate image fusion method should be used for depth map images, which is more general without being specific to RGB image data.

3.2 Infrared and Visible Image Fusion using a Deep Learning Framework

Next, another method was implemented for image fusion that was then tested on the disparity map dataset. This method [12] uses VGG-19 [142], a pre-trained deep convolutional neural network for image recognition. It uses the VGG-19 network to detect features at multiple layers in the network with a range of receptive field sizes. The method also decomposes each of the source images into a base layer and a detail layer so the different layers can be fused separately.

3.2.1 Fusion of base layers

For the source image to be decomposed into a base layer and a detail layer, the base layer is first computed from the source image by using a low-pass filter, so the output will contain low-frequency changes but at a larger scale. The filter used is equivalent to Tikhonov regularisation with the regularisation parameter λ, and the regularisation term is a discrete gradient operator G:

$$B_n = \arg\min_{B_n} ||B_n - I_n||_2^2 + \lambda \sum_i ||G_i B_n||_2^2 \tag{3.15}$$

where I_n is source image n and G_i computes the discrete gradient along axis i. To compute the detail layer, the base layer is subtracted from the source image $D_n = I_n - B_n$. The fused base layer can be directly computed from the base layers by averaging them as follows:

$$\bar{B} = \frac{1}{N} \sum_n^N B_n \tag{3.16}$$

where N is the number of source images to be fused.

3.2.2 Fusion of detail layers

To fuse the detail layers, features are extracted by using the VGG-19 convolutional neural network. This is done by first extracting the network outputs at multiple layers to get features at different depths in the network

$$\phi_n^{i,m} = \Phi_i(D_n) \tag{3.17}$$

where Φ_i the features from VGG-19 computed up to chosen layer i, 4 layers are chosen so $i \in \{1, 2, 3, 4\}$ corresponding to the layers chosen $[1, 6, 11, 20]$ so the output will be the feature maps at layer i in VGG-19 so $m \in \{1, 2, ..., M\}$ where M is the number of channels at layer i which is $M = 64 \times 2^{i-1}$ for VGG-19. D_n is the detail layer from the source image n. Next, the feature maps are reduced to a single activity map C_n^i by using the l_1 norm and an average filter:

$$C_n^i = Z_w * ||\phi_n^{i,1:M}||_1 \tag{3.18}$$

where Z_w is a mean filter with window width w. Here, a width of 3 is used. This is done to make it more robust to misaligned features, and the input is reflection-padded to conserve the input size. The l_1 norm takes the absolute values and sums the M channels, so large negative and positive values in the feature maps will result in larger values in the activity maps. From these activity maps, weight maps can be computed using the soft-max operator, which divides by the sum of all the activity maps computed from N source images, where a weight map is computed W_n^i for each VGG-19 feature map i extracted from the detail layer from source image n:

$$\bar{C}^i = \sum_n^N C_n^i \tag{3.19a}$$

$$W_n^i = \frac{C_n^i}{\bar{C}^i} \tag{3.19b}$$

The VGG-19 network contains max-pooling layers, which reduce the size, resulting in different-sized feature maps. The feature map layers from the network are chosen as the output of the layer following a max-pool operation. The max-pool operations have stride 2, resulting in the feature map size reduced by $1/2^{i-1}$ where $i \in \{1, 2, 3, 4\}$. To get feature maps that are all the same size as the detail layer, they are upsampled using nearest neighbour interpolation, resulting in 4 weight maps with the same size as the detail layer and normalised to sum to 1. They are each multiplied by the N detail layers and summed to generate 4 fused detail layers, where pixel k of the fused detail layer is computed as follows:

$$\bar{D}^i = \sum_{n=1}^N W_n^i D_n \tag{3.20}$$

The final fused detail layer is the maximum pixel values from the 4 fused detail maps as follows:

$$\bar{D}_k = \max(\bar{D}_k^1, \bar{D}_k^2, \bar{D}_k^3, \bar{D}_k^4) \tag{3.21}$$

3.2.3 Results

The method "Infrared and Visible Image Fusion using a Deep Learning Framework" [12] was implemented. Next, results were obtained from testing the implemented method on the disparity maps from the SYNTH3 dataset of 15

scenes from the stereo camera pair and the time of flight camera (ToF).

Disparity map	MSE	MAE
Stereo	17.2960	0.9225
ToF	8.1430	0.6756
Fused	7.5064	0.7539

Table 3.5: Results on the SYNTH3 dataset. The errors shown are compared to the ground truth when the implemented method of [12] is tested on the SYNTH3 dataset.

The results (Table 3.5) show that this method has significantly better results than the previous method (Section 3.1) due to the reduced error of the fused disparity map. The MSE is shown to be lower than both the stereo and ToF disparity maps. The MAE for the fused disparity map is shown to be lower than that of the stereo disparity map. However, the result also shows that the ToF disparity map has a lower error than the fused disparity map with the MAE. The result indicates that a more general method is needed to show a reduction in error in both the MSE and MAE compared to both of the input disparity maps used. The method used features computed with a pretrained RGB image model, which can limit the application to depth image data. To overcome this, a separate method will be used for the fusion of the depth image data to ensure a reduction in error for both the MSE and MAE compared with the input disparity maps.

3.3 Stereo and ToF Data Fusion by Learning from Synthetic Data

The method "Stereo and ToF Data Fusion by Learning from Synthetic Data" [128] uses a convolutional neural network to output confidence estimates for each disparity value, which is used to determine the best disparity value to use from the stereo or ToF disparity map at each pixel location as the one with the greatest confidence value. This method is specifically for the fusion of depth data and has been shown to improve the accuracy of the fused result. The convolutional neural network is trained on a synthetic dataset, which is described in Section 3.3.2. Training on the dataset allows it to accurately predict the areas in the input images that will cause the ToF camera or the stereo-matching algorithm to have disparity values that are far from the ground truth.

3.3.1 Convolutional neural network

The inputs used for the network are the disparity map from the stereo-matched camera pair and the disparity map from the ToF camera, as well as the difference between the left and right stereo image pair as a greyscale and the

amplitudes from the ToF camera. In total, 4 greyscale images are input into the network as a single 4 channel image to output the 2 confidence maps.

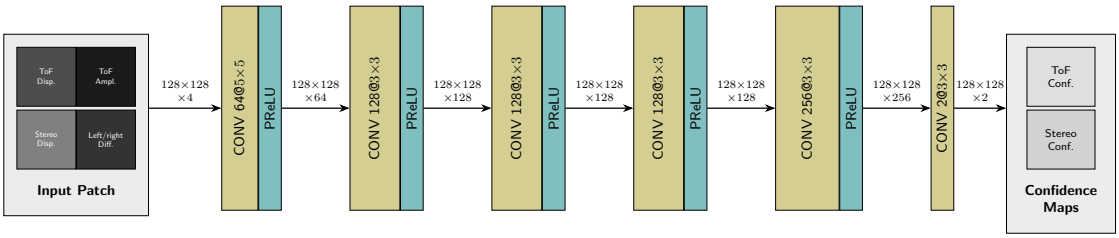

Figure 3.8: Network architecture of the model used in "Stereo and ToF Data Fusion by Learning from Synthetic Data".

The network architecture shown in Figure 3.8 has 6 convolutional layers. The first layer has a kernel size of 5×5, and the following layers all have 3×3 kernels. The number of channels outputted from the first is 64, then 128 channels from all the rest, apart from 256 for the second-to-last layer, and the final layer will output 2 channels for the 2 confidence maps for the stereo and ToF disparity maps. The input images are padded with border values to output images that are the same size. The activation function used is the Parametric Rectified Linear Unit (PReLU), defined as follows:

$$\text{PReLU}(x) = \begin{cases} x, & \text{if } x \geq 0 \\ ax, & \text{otherwise} \end{cases} \tag{3.22}$$

where a is a learnable parameter that determines the slope when negative. a is initially set to 0.02. This activation is used over the simpler ReLU to allow small gradients when the input is less than 0.

3.3.2 Training the network

The training dataset is a synthetic dataset which is publicly available [128]. This dataset consists of 40 scenes, which are 3D and rendered in Blender. 15 test scenes are also used for the evaluation of the trained model. For each scene data sample, there are 4 input images with a ground truth disparity map. The 55 scenes are used to generate larger amounts of data by extracting random patches with a width of 128 pixels and doing random vertical and horizontal flips and random rotations of $\pm 5°$. The ground truth disparity maps are used to compute the loss function by calculating a target confidence map, which is computed by taking the negative exponential of the absolute difference between the stereo disparity map and the ground truth disparity map and the absolute

difference between the ToF disparity map and the ground truth disparity as follows:

$$C_T^* = e^{-|D_T - D_{GT}|} \tag{3.23a}$$

$$C_S^* = e^{-|D_S - D_{GT}|} \tag{3.23b}$$

where D_T is the ToF disparity map, D_S is the stereo disparity map and D_{GT} is the ground truth disparity map. C_T^* is the target confidence map for the ToF disparity and C_S^* is the target confidence map for the stereo disparity. C_T^* and C_S^* are both used for computing the loss, which minimises the mean squared error of the network outputs with these computed target confidence maps as follows:

$$Loss = \sum (C_T - C_T^*) + \sum (C_S - C_S^*) \tag{3.24}$$

where C_T and C_S are the confidence estimates for the stereo and ToF disparity maps by the same network, which will be output in separate channels. The Adadelta algorithm is used for optimisation, with its initial learning rate set to 0.01, and the training is done with a batch size of 32.

These confidence maps can be used to find the best disparity values for each pixel by taking the maximum as follows:

$$D_F(i,j) = \begin{cases} D_T(i,j), & \text{if } C_T(i,j) \geq C_S(i,j) \\ D_S(i,j), & \text{otherwise} \end{cases} \tag{3.25}$$

where $D_T(i,j)$, $D_S(i,j)$ are the disparity values at pixel i,j from the stereo and ToF disparity maps.
$C_T(i,j), C_S(i,j)$ are the confidence values at pixel i,j for the stereo and ToF confidence maps outputted by the network.

3.3.3 Filtering

This method is limited by the final fused depth values only being computed from the corresponding pixels across the depth maps without using the neighbouring pixels for that computation. This method could be improved by using the neighbouring pixels to improve the accuracy of the pixel value by applying filtering methods to the image. These

methods allow the leveraging of priors about the image to reduce noisy depth values and remove outliers based on the value of the neighbouring pixels. To improve the method, filtering algorithms were tested to reduce the error metrics of the fused depth map. The method Contrast Limited Adaptive Histogram Equalisation (CLAHE) [143] and block match 3D(BM3D) [144] were used on the fused data and the input data to determine which approach best improves the result.

3.3.4 Results

The method Stereo and ToF Data Fusion by Learning from Synthetic Data [128] is implemented to evaluate for depth map fusion. The results here are obtained from testing the implemented method on the SYNTH3 dataset.

Disparity map	MSE	MAE
Stereo	17.2960	0.9225
ToF	8.1430	0.6756
Fused from Section 3.1	14.6503	1.1138
Fused from Section 3.2	7.5064	0.7539
Fused	**6.6931**	**0.4809**

Table 3.6: Results on the SYNTH3 dataset. The errors shown are compared to the ground truth when the implementation of [128] is tested on the SYNTH3 dataset. The lowest error in each column is shown in bold.

The results (Table 3.6) here show the method has shown a significant improvement in reducing the error in both the mean squared error and mean absolute error when used to fuse the disparity maps. The fused disparity maps achieve an MAE of **0.4809**, which is a reduction in the MAE of **28.82%** compared to the ToF MAE of **0.6756**. Percentage reduction is calculated as $\frac{(V_2 - V_1)}{|V_1|}$ where V_2 is the original value and V_2 is the reduced value.

For filtering, two strategies were tested. The first strategy is to apply the filtering before the data fusion to both inputs. The second strategy is to perform the data fusion first, then apply the filtering to the fused output. Both strategies are tested to determine which strategy results in the lowest error.

For the first strategy, filtering with CLAHE and BM3D is tested. Filtering is applied to both of the disparity maps before fusion, then the fusion method is used to compute the fused disparity map and record the accuracy at all stages to determine how much the error is reduced from the filtering for both the ToF and Stereo disparity maps, as well as in the final fused disparity map. For these results, the filtering methods are not applied to the disparity map after data fusion.

Disparity map	MSE	MAE
Stereo	17.2960	0.9225
Stereo-CLAHE	17.3475	1.2128
Stereo-BM3D	17.1839	0.8872
ToF	8.1430	0.6756
ToF-CLAHE	8.1740	0.7233
ToF-BM3D	8.1328	0.6731
Fused	6.6931	0.4809
Fused-CLAHE	6.8006	0.6791
Fused-BM3D	**6.6597**	**0.4579**

Table 3.7: Results on the SYNTH3 dataset. The errors shown are compared to the ground truth when the implementation of [128] is tested on the SYNTH3 dataset, along with using CLAHE and BM3D on the input disparity maps only. The lowest error in each column is shown in bold.

These results (Table 3.7) show that filtering with CLAHE increased the error before fusion for both Stereo and ToF disparity maps. This results in an increased error of the final fused disparity map when CLAHE is used. The filtering results from BM3D show that the stereo and ToF disparity maps both reduce the error. This results in a further reduction in error in the final fused disparity map. The Fused-BM3D disparity map achieves an MAE of **0.4579**, which is a reduction in the MAE of **4.783%** compared to the MAE of the fused disparity map without filtering, which is **0.4809**.

For the second strategy, filtering after using the fusion method is tested with CLAHE and BM3D. Filtering is applied to the disparity map after fusion, with the input ToF and Stereo disparity maps not being filtered before data fusion is done.

Disparity map	MSE	MAE
Stereo	17.2960	0.9225
ToF	8.1430	0.6756
Fused	6.6931	0.4809
Fused-CLAHE	6.8776	0.7551
Fused-BM3D	**6.5239**	**0.4431**

Table 3.8: Results on the SYNTH3 dataset. The errors shown are compared to the ground truth when the implementation of [128] is tested on the SYNTH3 dataset, along with using CLAHE and BM3D on the fused disparity maps only. The lowest error in each column is shown in bold.

These results (Table 3.8) show that filtering with CLAHE had an increase in error, and BM3D reduced the error. The Fused-BM3D disparity map achieves an MAE of **0.4431**, which is a reduction in the MAE of **7.860%** compared to the MAE of the fused disparity map without filtering, which is **0.4809**. This shows that the best results shown here are with the second fusion strategy when using BM3D after fusing the depth maps, which gives a reduction

of **7.860%** in the MAE, whereas the first fusion strategy, which uses BM3D before fusion, gives a reduction of **4.783%** in the MAE. These percentage reductions in MAE are calculated as the percentage change of the MAE of the Fused disparity map output with filtering applied compared to without filtering applied.

Results were collected for each scene in the dataset separately so that the error reduction could be individually inspected to see if the results were consistent.

Scene	Disparity map	MSE	MAE	Change
1	Fused	12.0265	0.6179	
1	Fused-BM3D	11.8108	0.5898	-4.5489%
2	Fused	0.1649	0.1497	
2	Fused-BM3D	0.1247	0.1191	-20.4864%
3	Fused	0.9902	0.4085	
3	Fused-BM3D	0.8985	0.3528	-13.6203%
4	Fused	47.2189	1.0317	
4	Fused-BM3D	47.1273	1.0038	-2.6951%
5	Fused	3.9348	0.4338	
5	Fused-BM3D	3.7148	0.3922	-9.5978%
6	Fused	0.6514	0.2235	
6	Fused-BM3D	0.5883	0.2032	-9.0887%
7	Fused	6.9951	0.6186	
7	Fused-BM3D	6.7049	0.5785	-6.4728%
8	Fused	6.8952	0.8318	
8	Fused-BM3D	6.6995	0.7900	-5.0330%
9	Fused	1.9569	0.5989	
9	Fused-BM3D	1.5482	0.5258	-12.2069%
10	Fused	3.2670	0.3401	
10	Fused-BM3D	3.1688	0.3230	-5.0343%
11	Fused	4.2768	0.5733	
11	Fused-BM3D	3.9003	0.5374	-6.2505%
12	Fused	0.4255	0.2346	
12	Fused-BM3D	0.3667	0.1883	-19.7364%
13	Fused	7.6537	0.5785	
13	Fused-BM3D	7.4859	0.5555	-3.9783%
14	Fused	0.2317	0.2100	
14	Fused-BM3D	0.1789	0.1695	-19.2833%
15	Fused	3.7078	0.3622	
15	Fused-BM3D	3.5402	0.3171	-12.4534%

Table 3.9: Results on the SYNTH3 dataset evaluated individually. The errors shown are compared to the ground truth when the implementation of [128] is tested on the individual SYNTH3 dataset, along with the percentage changed when using BM3D on the fused disparity maps only.

These results (Table 3.9) show that the reduction in error from using BM3D compared to the fused disparity map without using filtering is consistent across all scenes in the dataset. The dataset shows significant variation in error across the scenes, such as scene **2** shows a low error of **0.1247** for the Fused-BM3D, and scene **4** shows a high error of **47.1273** for the Fused-BM3D. The reason for this variation is due to the dataset being diverse and including

both easy scenes for the ToF and Stereo sensors, such as flat textured surfaces in scene **2**, as well as challenging scenes that include reflective and glossy surfaces in scene **4**.

3.4 Conclusions

In this chapter, image fusion methods were investigated for application to the fusion of RGB image data as well as depth image data. It shows novel improvements made to RGB image fusion methods by using multiple details layers for computing the fused output image, evaluated with image fusion quality metrics to show an increase in the fused image quality. The use of RGB image fusion methods on the task of depth image fusion showed limited success, leading to the use of depth fusion methods being explored. A deep learning depth fusion model was trained, and novel improvements were made by using filtering methods to further reduce the error. Different filtering strategies were tested to determine which resulted in the lowest error. These results show how RGB image fusion methods and Depth image fusion methods can be improved so they can produce higher visual quality images and images with reduced error, which is essential for use in a detection method. For the detection of cracks, these methods can be used to reduce the error in the input data, resulting in crack detection with reduced error.

Chapter 4

LiDAR and Stereo Data Fusion

This Chapter covers the work done in reviewing and comparing the state-of-the-art methods used for computing disparity or depth maps using RGB and LiDAR, then using this review to produce a data-fusion method built on what was learned from this review. The work done in this Chapter resulted in producing a review paper [145].

The review aims to find what the state-of-the-art data fusion methods were, how they performed on fusion tasks compared to each other and what the strengths and limitations of each method were. To determine this, the performance of each method was evaluated using the metrics reported and the visual results from the depth or disparity maps computed using colour to visualise the depth or disparity values. The methods are grouped by their input data type, covering RGB stereo, LiDAR and LiDAR stereo. The work done here is producing independent evaluations by using the methods which have trained models available. These are used to produce visual results and resulting error metrics when tested; other information is sourced from what is provided by the authors.

This Chapter also covers the work done in developing a novel LiDAR stereo data fusion pipeline in Section 4.6, which builds on the work done in Chapter 3 applied to the task of LiDAR stereo data fusion. This uses state-of-the-art methods that were reviewed in the first stage of this Chapter as part of the fusion pipeline, then after fusion produces an output lower in error than the methods used in the pipeline.

4.1 Datasets

To evaluate and compare methods for LiDAR and RGB stereo data, the KITTI Vision Benchmark Suite is used, which provides publicly available datasets. These datasets are developed for applications in mobile robotics and autonomous driving research. The data was recorded using a sensor platform attached to a car, which was driven in and around Karlsruhe, Germany, to capture a diverse dataset of road scenes, including both urban and rural environments. The sensor platform setup contained a Velodyne HDL-64E LiDAR scanner and two RGB colour cameras set up to capture stereo images. The setup captures a video from the colour cameras, which is synced with the LiDAR scans to capture each video frame as a separate scene in the dataset.

The dataset has been made suitable for training and testing models by generating a ground truth for each scene frame captured by the sensors. The ground truth is generated by a manual annotation process, which starts by masking out the non-rigid moving bodies, such as pedestrians. Then, separating the static environment from the rigid moving bodies, which are the moving vehicles, which are a fixed geometry that is moving across frames. The static environment is where these objects are not moving between captured scene frames, such as the building and the ground. The rigid moving bodies are labelled as foreground objects (fg), and the static environment is labelled as the background region (bg). Then, the ground truth is generated from this by accumulating multiple LiDAR scan frames into each scene's common coordinate system to get the ground truth for the background region. Then, the ground truth is generated for foreground objects by masking them out and reinserting 3D CAD models to match the position, then sampling points from these 3D models to use as the ground truth LiDAR scan for foreground objects.

4.1.1 KITTI-2015 stereo test dataset

The KITTI-2015 stereo test dataset [146, 147, 148] contains 200 scenes captured from the stereo camera pair. The stereo ground truth is obtained by projecting the LiDAR ground truth onto the left stereo image view to give a disparity map ground truth. For each pixel in the disparity map that has a value, it represents the horizontal pixel distance to map that corresponding pixel in the left image to the pixel in the right image viewing the same point.

The dataset can be used to evaluate a method by computing error metrics between the ground truth disparity map and the estimated disparity map from that method. The percentage of erroneous pixels averaged over all 200 scenes

is used as an error metric, where erroneous is defined as greater than a given pixel threshold, which is 3 pixels. This is computed separately for the pixels of foreground objects and the pixels of the background region. This pixel threshold error metric is provided by the KITTI Vision Benchmark Suite.

4.1.2 KITTI depth completion validation dataset

The KITTI depth completion validation dataset [149] contains 200 scenes captured from the LiDAR sensor and the stereo camera pair. The depth completion ground truth is obtained by also projecting the LiDAR ground truth onto the left stereo image, but keeping it in depth format, where the pixel represents the depth to the surface it is viewing, measured in meters.

The evaluation is done with the mean absolute error (MAE) reported in millimetres (mm), and the root mean squared error (RMSE) also in mm. The inverse depth is also used by converting the estimated and ground truth depths to be 1/depth. Evaluation is done with the inverse mean absolute error (iMAE) reported in units of 1/kilometres (1/km) and the inverse root mean squared error (iRMSE) in 1/km. These error metrics are provided by the KITTI Vision Benchmark Suite.

4.1.3 KITTI-141 subset

The KITTI-141 subset [66] is a subset of 141 scenes selected from the KITTI-2015 stereo test dataset. These 141 are selected due to having associated raw Velodyne scans for use in testing methods that require both the stereo and LiDAR data to estimate the disparity. The evaluation is done with the absolute relative error (Abs rel), the maximum ratio threshold error ($\delta < 1.25$) and the percentage of erroneous pixels with pixel thresholds of 2,3 and 5 ($> 2px, > 3px, > 5px$). The $> 3px$ matches the error metric used with the KITTI-2015 stereo test dataset, as well as providing additional error metrics.

4.2 Methods for depth estimation using RGB stereo

RGB stereo methods only use 2 RGB images as inputs, and these are taken with a stereo camera setup where two calibrated cameras next to each other take images, so that there is some disparity between the two images, and depth can then be detected.

4.2.1 RAFT-Stereo: Multilevel Recurrent Field Transforms for Stereo Matching

The method "RAFT-Stereo: Multilevel Recurrent Field Transforms for Stereo Matching" [150] computes the disparity from only RGB stereo images using an approach directly derived from RAFT (Recurrent All-Pairs Field Transforms for Optical Flow) [41]. RAFT-Stereo applies the approach used in RAFT to disparity estimation from stereo images.

RAFT

RAFT solves the task of Optical Flow, which produces an optical flow map that estimates per-pixel motion between video frames. This takes a pair of input images, where they are two consecutive video frames, and will output a two-channel output optical flow map, which represents a displacement vector to map from the first image to the second image. RAFT-Stereo is the method which is relevant to this Chapter, so the network is described in full below.

The network architecture for RAFT is almost identical, with only a few differences, which are that it uses a 4D correlation and does not include the multi-scale gated recurrent unit (GRU). RAFT uses a 4D correlation volume to represent the two spatial dimensions of the first image and the two dimensions for the optical flow estimation vector. RAFT-Stereo is estimating the disparity, which is just a single value. This can be seen as a fixed vector direction along the horizontal direction, where only the pixel distance needs to be estimated, so only a 3D correlation volume is needed for the two spatial dimensions, image dimensions, and the single disparity dimension. RAFT only uses a single-scale GRU, which is just used at the largest scale. RAFT-Stereo is developed from this to introduce the multi-scale GRU. Details of how the correlation volume is computed and used, as well as the details on the multi-scale GRU, are given in the description of RAFT-Stereo below.

RAFT-Stereo

In RAFT-Stereo, the task is disparity estimation from stereo images. This takes a pair of input stereo images, where they are captured from side-by-side cameras, and will output a single-channel output disparity map, which represents offset values to map from the left image to the right image.

To compute the output with the network first, feature maps are extracted from the left and right images using a

feature network, which is a convolutional network architecture. Then, feature maps are combined into a correlation volume by taking the dot product of each possible feature pair corresponding to a possible disparity.

The correlation volume is then downsampled in the dimension corresponding to the disparity. These downsampled correlation volumes are collected to get a set of multiple correlation volumes that are at different levels, where the more downsampled volumes have an increased receptive field. This set of different levels of correlation volumes is referred to as a correlation pyramid. In computer vision, image pyramids [151] are a type of representation where the 2D image is repeatedly downsampled to get multiple levels of images, decreasing the resolution of the higher level it goes, which forms a pyramid visually if stacked on top of each other. Here, this is referred to as a correlation pyramid because it does the same process, just with 3D volumes instead of a 2D image and downsampling in the single disparity dimension instead of downsampling in the height and width dimensions.

An initial estimation of 0 disparity for all pixels is used as a starting point for the disparity map. The initial disparity map is updated by using a recurrent neural network. To get the inputs for the recurrent network, the correlation pyramid is indexed by selecting windows in each volume around the current disparity estimates. The windows are concatenated together into a single feature map, which is then fed through convolutional layers to get correlation features. The current disparity estimation is also fed through convolutional layers to get disparity features.

The left image of the image pair is fed through a context network to extract context features. The right image is not used in the context network because the disparity map is defined for the left image's view, giving disparity values that map to the right image, so for the coordinate system to match with the other features, it needs to use the left image's view. This allows the context network to provide structural information, in that the left image view is used to compute the disparity. The right image is only used in producing the correlation volume, which uses both the left and right image features to allow the network to match features.

The correlation, disparity and context features are concatenated into a single feature map and used as the input for a multi-scale gated recurrent unit (GRU).

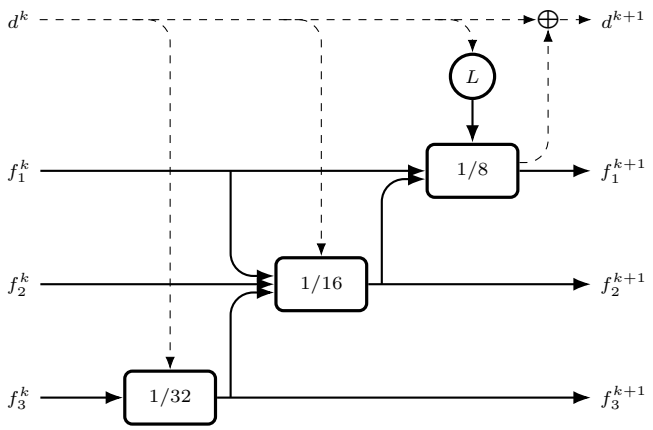

Figure 4.1: The multi-scale GRU shown with 3 feature maps f_1, f_2, f_3 at different scales $1/8, 1/16, 1/32$. The disparity map d^k shown is updated by adding the output of the highest resolution scale GRU 1/8, the correlation pyramid lookup operation is shown as L, and the result is input to the highest resolution scale GRU 1/8. Downsampling and upsampling are used to change the scales of the feature maps or disparity map when being passed between the GRUs.

The multi-scale GRU (Figure 4.1) is composed of 3 GRUs at multiple feature map resolution scales.

The first GRU and lowest resolution scale is at 1/32 the resolution of the input image, so the feature maps have the height and width both at 1/32 the size of the input image's height and width. This scale of 1/32 is needed to give information from a receptive field to be used for updating the disparity map. A lower resolution is not used since the feature map is too small; therefore, it lacks useful spatial information.

The output of the first GRU at the 1/32 scale is upsampled to 1/16 resolution scale and concatenated with the input feature map for the second GRU at the higher resolution of 1/16. Then the output of the second GRU is upsampled and concatenated with the input feature map for the third GRU, which is at the highest resolution of 1/8. To output the update to the disparity map estimate to get the new disparity map update with element-wise addition. The correlation pyramid lookup is used to input into the highest resolution 1/8 GRU, and the downsampled current disparity map estimate is used as an input for the other 2 GRUs. The second GRU of resolution 1/16 also uses the highest resolution 1/8 feature map as an input.

For training the network, the loss is computed after each update to the disparity map estimation, with disparity estimations that took more update steps having a greater weighting.

Performance Evaluation

This method was tested on the KITTI-2015 stereo test dataset described in Section 4.1.1 using the percentage of erroneous pixels with a 3-pixel threshold. The method resulted in 1.96% for all pixels, 2.89% for foreground pixels and 1.75% for background pixels.

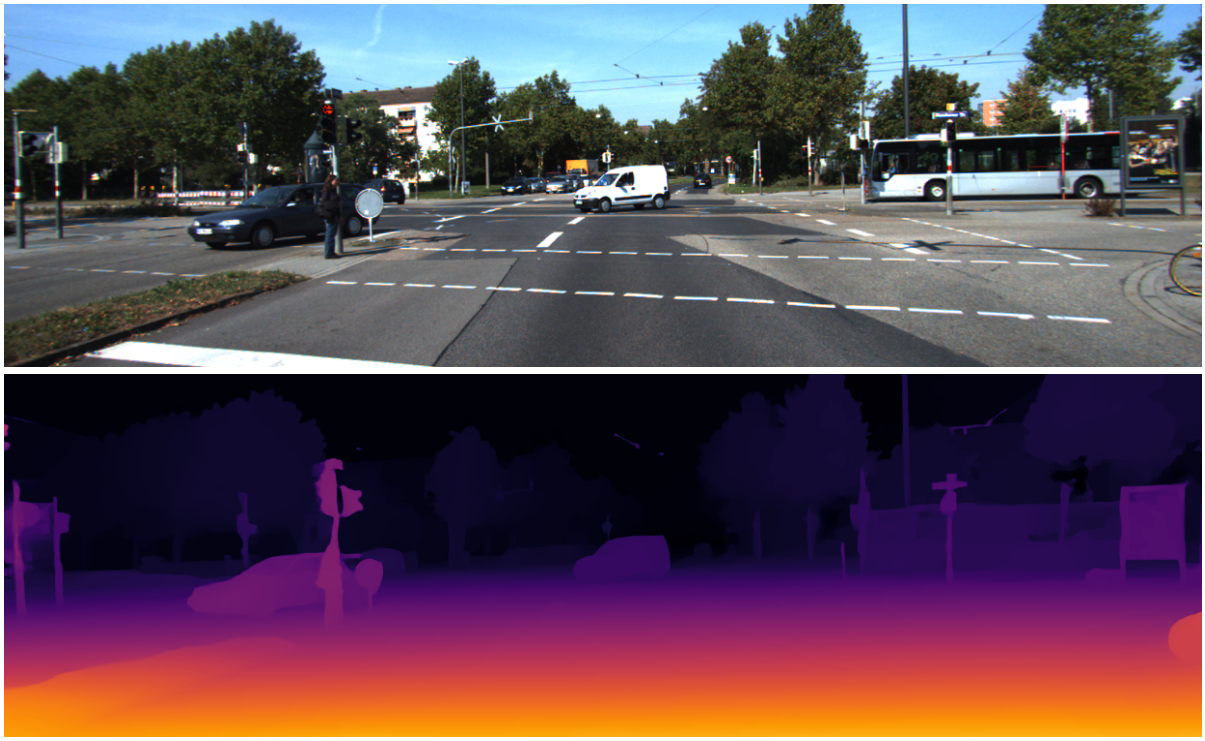

Figure 4.2: The qualitative results from [150] show the top image as the left RGB image from the first example scene from the KITTI-2015 stereo dataset. The corresponding disparity map estimation is shown in the bottom image.

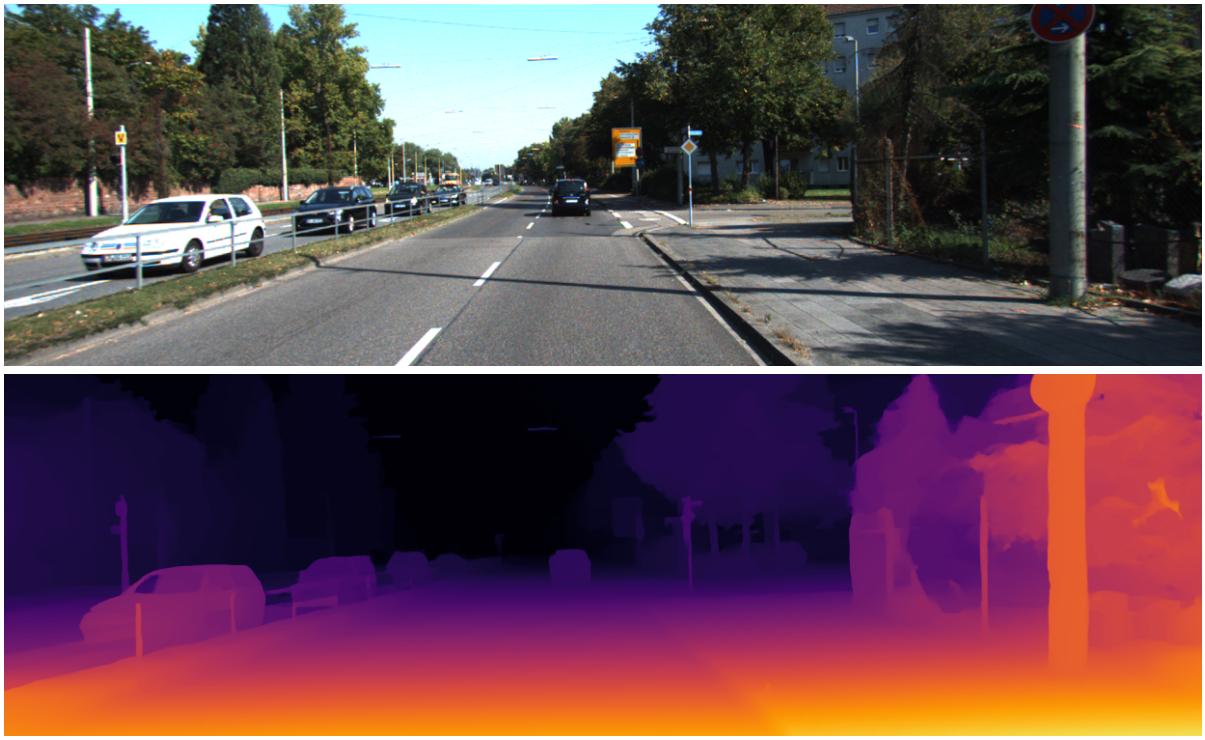

Figure 4.3: The qualitative results of RAFT-Stereo on the second example scene.

Figure 4.4: The qualitative results of RAFT-Stereo on the third example scene.

The qualitative results in Figures 4.2, 4.3, 4.4 show fine-depth detail with depth estimated in the fence and sign

posts on the left side of Figure 4.3. This is because it puts the image at full resolution without the need to upsample from a lower-resolution estimation.

4.2.2 Practical Stereo Matching via Cascaded Recurrent Network with Adaptive Correlation

The method "Practical Stereo Matching via Cascaded Recurrent Network with Adaptive Correlation" (CREStereo) [152] computes a disparity map from a pair of stereo images using a deep convolutional neural network. The disparity map is computed using a recurrent-based architecture to refine the disparity estimation starting from a zero-initialised disparity map.

First, a feature network is used on both left-right stereo images to extract feature maps, which are used to construct 3-level feature pyramids for both of the input images. Next, a positional encoding is added to the feature pyramid, and then it is input through a self-attention layer to get the final feature pyramid, which is used as the input for the recurrent update module.

Next, the recurrent update module uses an adaptive group correlation layer similar to [45]. This layer outputs the inputs for the recurrent neural network from the feature pyramids. This is done by first using a cross-attention layer to get grouped features, which are then sampled using the current predicted disparity map. The sampling is done using a search window with fixed offsets, which will alternate between 1D and 2D searches to allow for stereo matching that does not lie on the epipolar line. The feature pyramid is also used to compute learned offsets to allow for a deformable search window to be used by extending the method for deformable convolutions [153]. The pyramid sampling will collect features in a window to be used to construct a correlation volume of size $H \times W \times D$ where H and W are the input image height and width, and D is the number of correlation pairs, which is smaller than W. The correlation at x, y, d in the correlation volume is computed as follows:

$$corr(x, y, d) = \frac{1}{c} \sum_{i=1}^{C} F_1(i, x, y) F_2(i, x'', y'') \tag{4.1}$$

where $x'' = x + f(d) + dx$, $y'' = y + g(d) + dy$. F_1, F_2 are the feature maps, $f(d), g(d)$ are the fixed offsets which will alternate between 1D when $g(d) = 1$ and 2D when $g(d) \in [-4, 4]$ with $f(d) \in [-4, 4]$ for both and dx, dy are the learned offset which is computed from the feature pyramids.

The correlation volume is used as an input to the recurrent neural network, which is formed of multiple GRUs, each at a different scale. The final disparity map is then computed using cascaded refinement, where the output disparity map from a lower-level GRU is upsampled and then used as the initial disparity map for the next GRU level to finally get the highest-level GRU output, which is used as the final disparity map.

Performance Evaluation

This method was tested on the KITTI-2015 stereo test dataset described in Section 4.1.1 using the percentage of erroneous pixels with a 3-pixel threshold. The method resulted in 1.69% for all pixels, 2.86% for foreground pixels and 1.45% for background pixels.

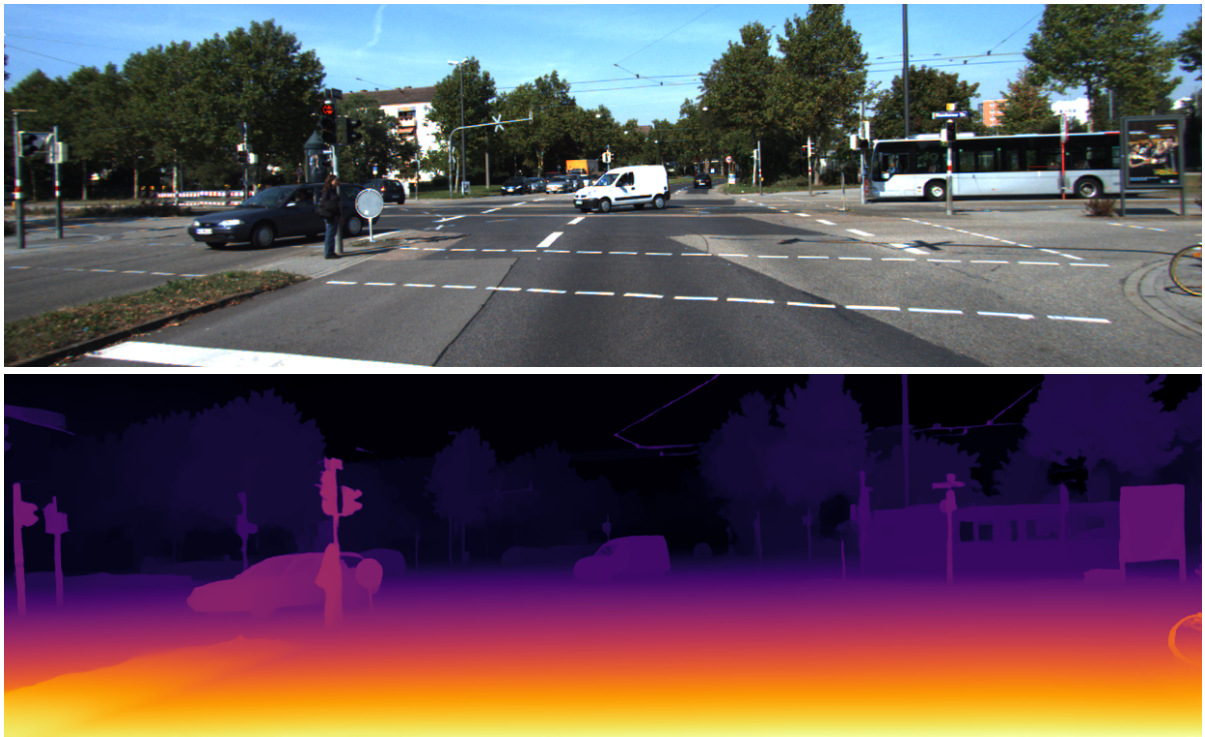

Figure 4.5: The qualitative results from CREStereo [152] show the top image as the left RGB image from the first example scene from the KITTI-2015 stereo dataset. The corresponding disparity map estimation is shown in the bottom image.

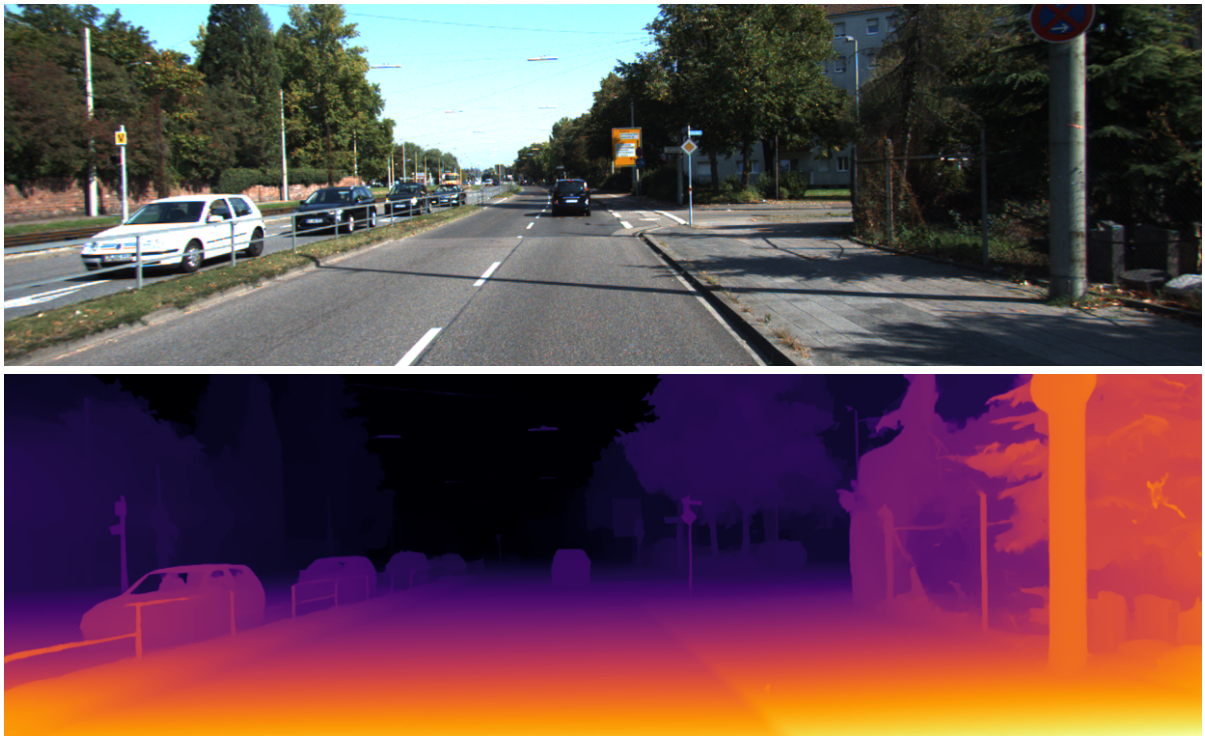

Figure 4.6: The qualitative results of CREStereo on the second example scene.

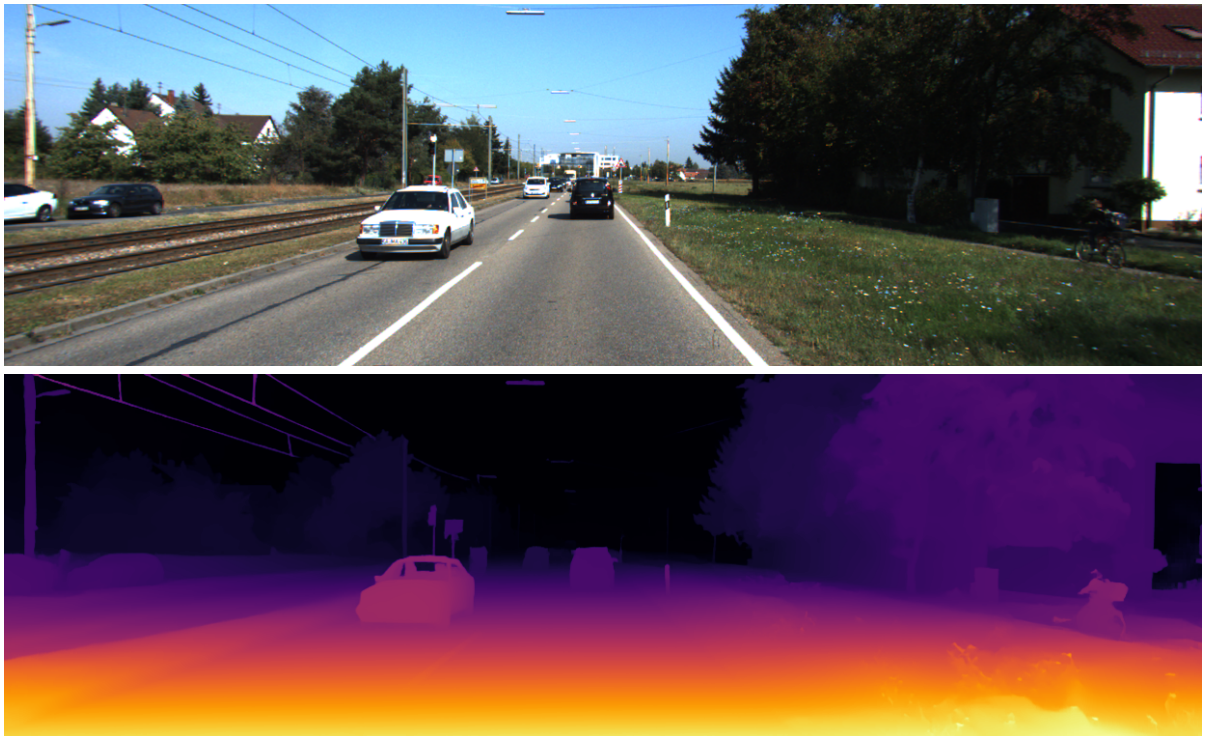

Figure 4.7: The qualitative results of CREStereo on the third example scene.

The qualitative results in Figures 4.5, 4.6, 4.7 show fine detail visible in the left of Figure 4.6 with the fence and

sign posts similar to [150] since it also can output the full resolution.

4.2.3 Hierarchical Neural Architecture Search for Deep Stereo Matching

This method, LEAStereo (Learning Effective Architecture Stereo) [154], computes a disparity map from only 2 RGB stereo images using a deep convolutional neural network. The network architecture is found using NAS (Neural Architecture Search), and then the found network architecture is trained to get the final model.

The neural network has four stages to compute the disparity map from the stereo image pair. The first stage uses a 2D feature network to extract a pair of feature maps from the stereo images. The second stage combines the feature maps into a 4D feature volume. The third stage is a 3D matching network that will compute a matching cost for each possible disparity using the 4D feature volume as the input. The final stage will project the 3D cost volume to a 2D disparity map using a $soft - argmin$ layer. Only the feature network and the matching network have trainable parameters, so NAS is used to find the best architecture for these networks.

The architecture search is done using differentiable architecture search [155], where there is continuous relaxation of discrete architectures, so the best architecture can be found using gradient descent. A two-level hierarchical search [156] is used to find the best cell-level and network-level structures.

Cell-level search space

A computational cell is the building block which is used to create the feature network and the matching network. The cell defines a sequence of operations to apply to the input feature maps and resulting intermediate feature maps to compute the output feature maps of the cell. The cells are represented as directed acyclic graphs that consist of an ordered sequence of n nodes, where the nodes are the feature maps and the edges are operations that transform the feature maps, so a node is computed by applying the operations and then combining the feature maps with element-wise addition. The feature network and the matching network are both deep convolutional neural networks that are constructed with computational cells, which are stacked in layers and share the same structure.

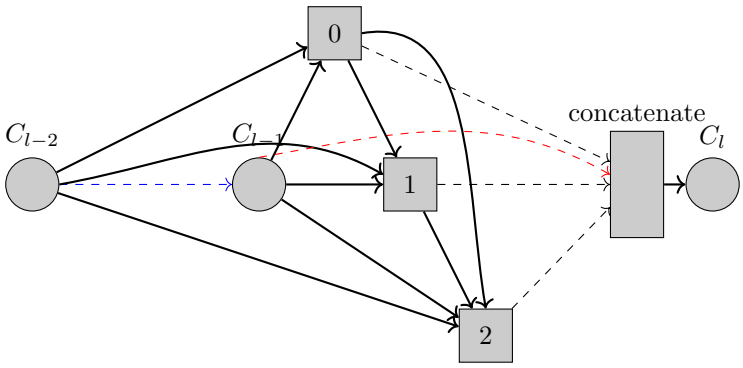

Figure 4.8: Cell-level search space. The cell output C_l is computed using the previous 2 cell outputs C_{l-2}, C_{l-1}. The intermediate nodes are shown as $0, 1, 2$, and for each input connection to these nodes, an operation will be searched for. The red dashed arrow shows the residual connection, and the blue dashed arrow shows the previous cell that uses C_{l-2} as one of its inputs to compute C_{l-1}. The intermediate nodes are concatenated, shown with black dashed arrows, to get the final output C_l.

The cell search space (Figure 4.8) has the first two nodes used as inputs, so for cell C_l in layer l, the two input nodes are outputs from the two preceding cells (C_{l-2}, C_{l-1}). The cell output is computed by concatenating nodes that are not input nodes to the cell. The cells in the final model that will be found will only have two input connections for each node. The architecture search will find these connections and the operation.

To make the architecture continuous in the search phase, the nodes have all possible connections, so node i is connected to nodes $\{1, 2, ..., i-1\}$, and the edges have all possible operations from the set of candidate operations. In the search phase, the output of a node is computed as follows:

$$s^{(j)} = \sum_{i \rightsquigarrow j} o^{(i,j)}(s^{(i)}). \tag{4.2}$$

where $\rightsquigarrow$ denotes node i is connected to node j. $o^{(i,j)}$ is defined as follows:

$$o^{(i,j)}(x) = \sum_{r=1}^{v} \frac{\exp(\alpha_r^{(i,j)})}{\sum_{s=1}^{v} \exp(\alpha_s^{(i,j)})} o_r^{(i,j)}(x) \tag{4.3}$$

where $o_r^{(i,j)}(x)$ is the r-th operation between nodes i and j. $\alpha^{(i,j)} = (\alpha_1^{i,j}, \alpha_2^{i,j}, ..., \alpha_v^{i,j})$ is a weight mixing vector that is used in the softmax function to weight the operations. After the search phase, the operation to be used $O_{r^*}^{(i,j)}$ is selected as the operation with the highest corresponding weight after the search phase $r^* = \arg\max_r \alpha_r^{(i,j)}$. The set of candidate operations for the feature net is $\mathcal{O}^F = \{$"3×3 convolution", "skip connection"$\}$. The set of candidate operations for the matching network is $\mathcal{O}^M = \{$"$3 \times 3 \times 3$ convolution", "skip connection"$\}$. The top two

highest operations are selected for each node, so each node has two input connections. Following ResNet[157], the cells are modified to include a residual connection between the output of the previous cell and the cell's output.

Network-level search space

The network-level search space will have a fixed number of layers L that will allow the network to downsample, upsample or keep the same spatial resolution of the feature maps at each layer. The search space has the maximum and minimum resolution the feature map can be, and the network will downsample by a scale of 1/2 and upsample by a scale of 2.

To make the architecture continuous at the network level, all possible configurations are computed with weighting parameters. In the search phase the output $h_{(s)}^{(l)}$ of layer l at spatial resolution s is computed as follows:

$$\begin{aligned} H_{(s)}^{(l)} = \bar{\beta}_{(s/2)} Cell(h_{(s/2)}^{(l-1)}, h_{(s)}^{(l-2)}) \\ + \bar{\beta}_{(s)} Cell(h_{(s)}^{(l-1)}, h_{(s)}^{(l-2)}) \\ + \bar{\beta}_{(2s)} Cell(h_{(2s)}^{(l-1)}, h_{(s)}^{(l-2)}) \end{aligned} \tag{4.4}$$

where $Cell(h^{(l-1)}, h^{(l-2)})$ is the cell computation which takes the outputs from the previous two layers $(l-1, l-2)$ as inputs. $\bar{\beta}_s$ is the weight parameter after the softmax function has been applied. The best network architecture is selected as the one with the highest weights for each layer.

The network used 6 layers for the feature network and 12 layers for the matching network. A maximum resolution of 1/3 of the input size and a minimum of 1/24 were used for the search space. The cells used five nodes.

Optimisation

The network is searched and trained in an end-to-end manner, so the loss function uses the final disparity output computed by the model and the ground truth disparity. The loss function is defined as follows:

$$\mathcal{L} = \ell(\mathbf{d}_{pred} - \mathbf{d}_{gt}), \text{where} \ell(x) = \begin{cases} 0.5x^2, & \text{if } |x| < 0 \\ |x| - 0.5, & \text{otherwise} \end{cases} \tag{4.5}$$

where $\mathbf{d}_{pred}$ is the predicted disparity map by the network and $\mathbf{d}_{gt}$ is the ground truth disparity map.

The training is done with two disjoint training sets. One optimises the network parameters, and the other optimises the architecture search parameters for the cell-level α and network-level β. The optimisation is done by alternating the update of the network weights and the network search parameters. The network has been trained on the SceneFlow dataset [54].

Performance Evaluation

This method was tested on the KITTI-2015 stereo test dataset described in Section 4.1.1 using the percentage of erroneous pixels with a 3-pixel threshold. The method resulted in 1.65% for all pixels, 2.91% for foreground pixels and 1.40% for background pixels.

Figure 4.9: The qualitative results from LEAStereo [154] show the top image as the left RGB image from the first example scene from the KITTI-2015 stereo dataset. The corresponding disparity map estimation is shown in the bottom image.

Figure 4.10: The qualitative results of LEAStereo on the second example scene.

Figure 4.11: The qualitative results of LEAStereo on the third example scene.

The qualitative results in Figures 4.9, 4.10, 4.11 shows less fine detail with smoother edges. This is due to it

outputting the disparity map at a lower resolution and then upsampling it. However, in Figure 4.10 on the left side, it is able to estimate parts of the fence that the previous method missed, although they are estimated as thicker with smoother edges than they appear in the RGB image.

4.3 Methods for LiDAR depth completion

LiDAR methods use depth data as the input that has been obtained by a LiDAR scanner. The LiDAR scanner computes a point cloud, and then the point cloud is projected onto a view to produce a depth map. However, the depth map is sparse, meaning not all pixels have a depth value, so these methods estimate the missing depth values in the depth map. An RGB image is also used, which captures the same view as the depth maps, so it can guide the depth completion methods.

4.3.1 PENet: Towards Precise and Efficient Image Guided Depth Completion

Precise and Efficient Image Guided Depth Completion (PENet) [158], computes a depth map from a sparse LiDAR depth map and a single RGB image. The method uses a deep convolutional neural network to compute an initial depth map, which is then refined with a convolutional spatial propagation network.

The neural network for computing the initial depth map uses a two-branch backbone. Both branches will output a dense depth map estimation and a confidence weight map, which are used to fuse into a single depth map estimation. The first branch is the colour-dominant branch, which uses both the RGB image and the sparse depth map as inputs. The network uses an encoder-decoder architecture with symmetric skip connections. This network will output a colour-dominant depth map estimation and a confidence map, which has confidence values for each pixel. The second branch is the depth-dominant branch, which uses the output depth map estimation of the colour-dominant branch and the sparse depth map for the network inputs [159]. This branch has a similar architecture to the colour-dominant branch, and it also takes the features in the colour-dominant decoder as extra inputs in the corresponding layers for the depth-dominant encoder.

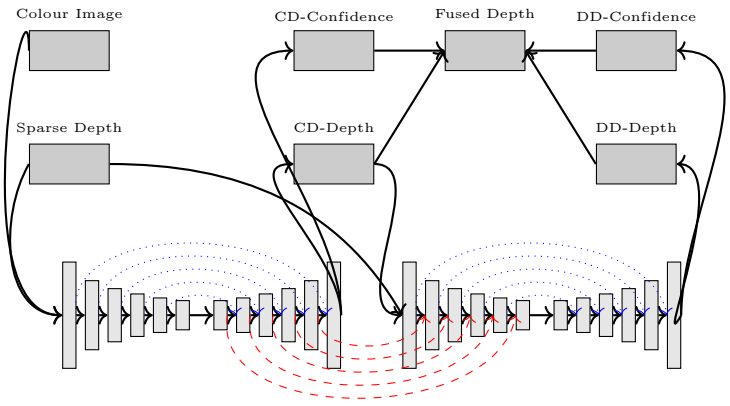

Figure 4.12: The model architecture for PENet. The inputs shown are the colour image and the sparse depth, which are input into the network to compute the fused depth map, which is then further refined. The blue lines show the skip connections within the colour-dominant branch and depth-dominant branch. The red lines show the skip connections from the colour-dominant branch to the depth-dominant branch.

In both the network branches, a geometric convolutional layer is used, which is similar to a standard convolutional layer, but it uses additional 3D position maps, which are then concatenated to the feature inputs to form the input to the geometric convolutional layer. The position map is derived from the sparse LiDAR depth map D as follows:

$$Z = D, X = \frac{(u - u_0)Z}{f_x}, Y = \frac{(v - v_0)Z}{f_y} \tag{4.6}$$

where (u, v) are the coordinates of a pixel and u_0, v_0, f_x, f_y are the intrinsic parameters of the camera.

The initial depth map estimation is then refined using a convolutional spatial propagation network [56] with dilated convolutions [160]. The depth refinement is done in iterations using affinity weight maps learned by the network. For each pixel, it aggregates information propagated from pixels within the neighbourhood. For the initial depth map D_0 that has been produced by the network, the depth refinement is as follows:

$$D_i^{t+1} = W_{ii} D_i^0 + \sum_{j \in \mathcal{N}(i)} W_{ji} D_j^t \tag{4.7}$$

where W_{ji} is the affinity between pixel i and pixel j, and W_{ii} is the affinity between pixel i and itself, and these affinity maps are learned by the network.

The loss $L(\hat{D})$ is used for training, which is an ℓ_2 loss, which means it is of the form $\ell_2 = (y - \bar{y})^2$ for a variable y

and a target $\bar{y}$. $L(\hat{D})$ is defined as follows:

$$L(\hat{D}) = ||(\hat{D} - D_{gt}) \odot \mathbb{1}(D_{gt} > 0)||^2 \tag{4.8}$$

where $\hat{D}$ is the final predicted depth map, D_{gt} is the ground truth, $\mathbb{1}$ is an indicator and $\odot$ is an element-wise multiplication. This is so that it only uses the valid pixels in the ground truth for training. For early epochs, a loss is applied to the intermediate depth estimations from the colour-dominant branch and the depth-dominant branch as follows:

$$L = L(\hat{D}) + \lambda_{cd} L(\hat{D}_{cd}) + \lambda_{dd} L(\hat{D}_{dd}) \tag{4.9}$$

where λ_{cd} and λ_{dd} are hyperparameters.

Performance Evaluation

This method was tested on the KITTI depth completion validation dataset described in Section 4.1.2. The method resulted in 209.00 for the MAE, 757.20 for the RMSE, 0.92 for the iMAE and 2.22 for the iRMSE.

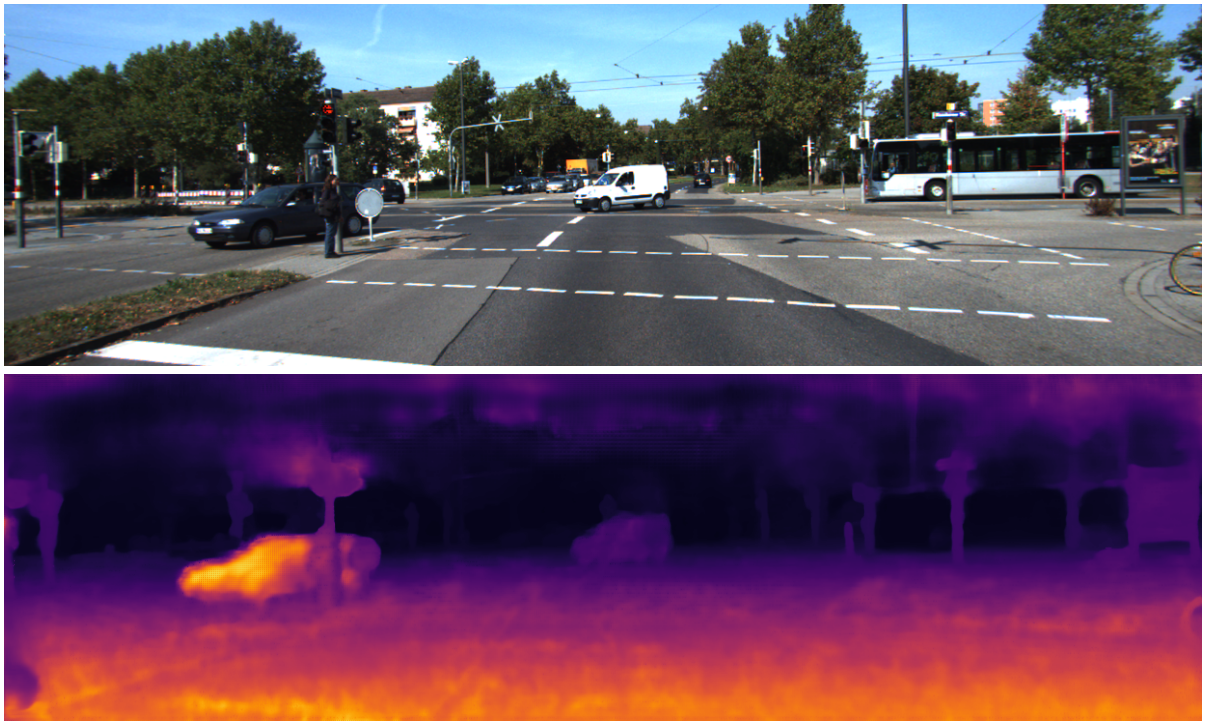

Figure 4.13: The qualitative results from PENet [158] show the top image as the left RGB image from the first example scene from the KITTI-2015 stereo dataset. The corresponding disparity map estimation is shown in the bottom image. The raw dataset was used for the LiDAR data mapped onto the scenes in the stereo dataset, and then the results were converted from depth values to disparity using the calibration parameters.

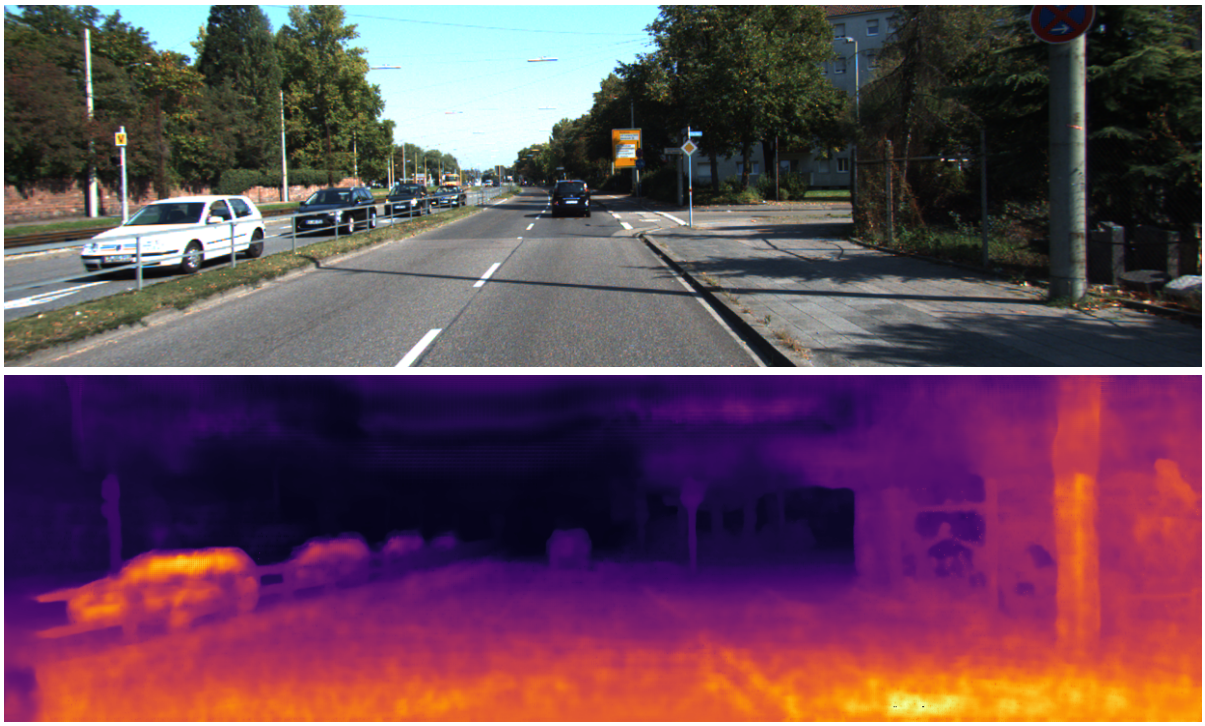

Figure 4.14: The qualitative results of PENet on the second example scene.

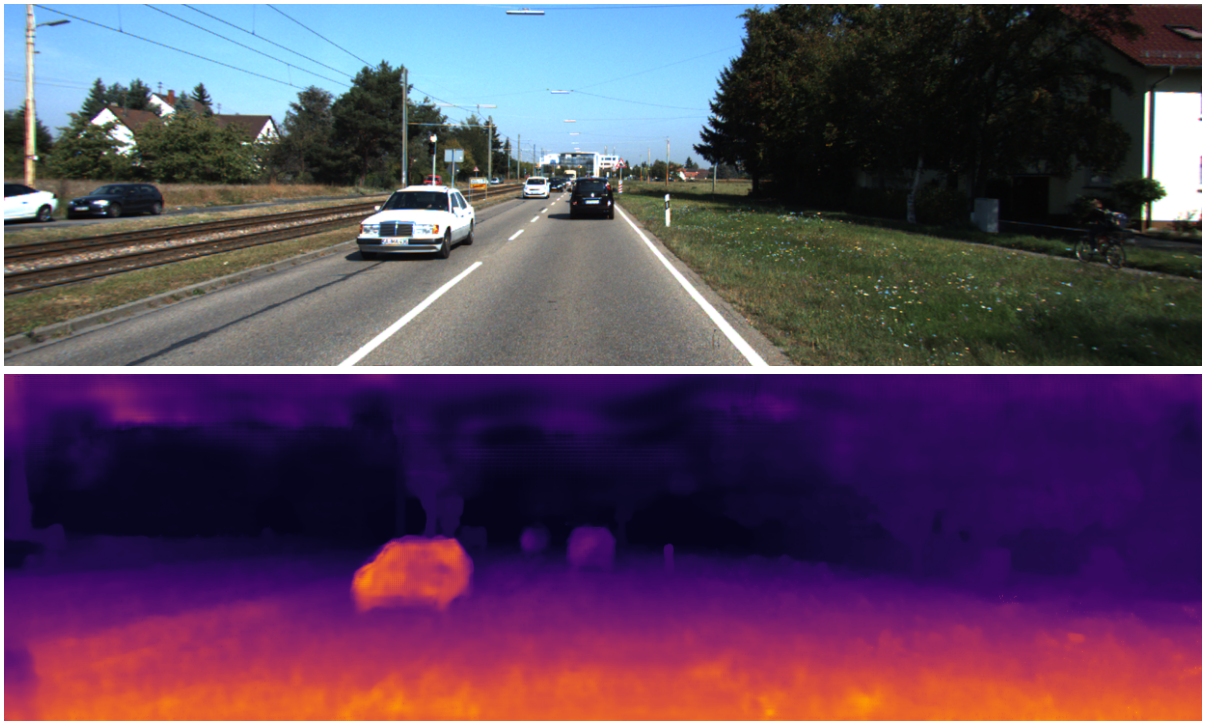

Figure 4.15: The qualitative results of PENet on the third example scene.

The qualitative results in Figures 4.13, 4.14, 4.15 show noise in the disparity maps visible in the estimates in disparity for the road, where it does not show a smooth gradient change in the disparity colour scale and instead displays a noise pattern. Since the LiDAR data is sparse, there are regions of pixels without any depth information, so there will be high uncertainty in those regions.

4.3.2 SemAttNet: Towards Attention-based Semantic Aware Guided Depth Completion

Attention-based Semantic Aware Guided Depth Completion (SemAttNet) [161], computes a depth map using a sparse LiDAR depth map and a single RGB image as inputs (and a semantic map). This method first uses a convolutional neural network to output an initial coarse depth map, which is refined with a spatial propagation network. The network used for computing the initial depth map uses a three-branch backbone, which consists of a colour branch, a depth branch and a semantic branch. The depth branch takes the output depth map estimations computed by the colour and semantic branches, as well as the sparse LiDAR depth map, as inputs. The network uses a semantic-aware multi-modal attention-based fusion block in the initial network [162]. It is first used to fuse feature maps from the colour and semantic branches, and then it is used in the depth branch to fuse features from

the other two branches.

The colour branch uses an encoder-decoder network with skip connections, which takes the colour image and the sparse depth map as inputs and will output a dense depth map and a confidence map. The semantic branch takes the depth map computed with the colour-guided branch as an input, as well as the sparse depth map and the semantic image [159, 158]. The network will output a dense depth map and a confidence map. The network uses a similar encoder-decoder architecture, uses the decoder features from the colour-guided branch, and fuses them with the features in the encoder network. The semantic image is computed from the RGB image with a pre-trained WideResNet38 [163] model. The depth branch uses the output depth maps from the colour branch and the semantic branch as inputs, as well as the sparse depth map. The branch uses an encoder-decoder network architecture similar to the other branches, with the decoder features from both branches being fused with the features in the encoder network.

The decoder features are fused with encoder features using a semantic-aware multi-modal attention-based fusion block. This uses channel-wise attention and then spatial-wise attention to output a single feature map.

The final depth maps output from each branch are fused using the corresponding learned confidence maps output from each branch [158, 164].

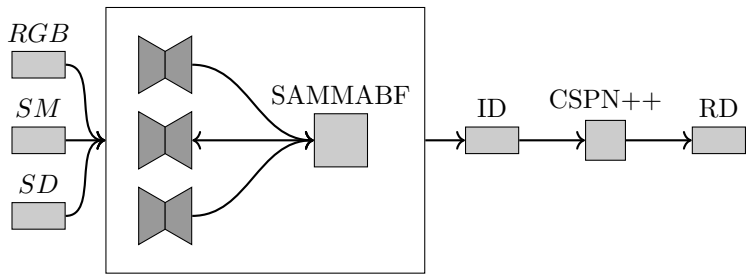

Figure 4.16: The model architecture for SemAttNet. The inputs shown on the left are the RGB image, the semantic map SM computed from the RGB image and the sparse depth map SD captured by the LiDAR sensor. The features from the 3-branch backbone are fused using the semantic-aware multi-modal attention-based fusion block (SAMMABF) and will output the initial depth map shown as ID. This is refined with the CSPN++ [56] with dilated convolutions [160] to output the refined depth map shown as RD.

The training loss computes the ℓ_2 loss for each branch's output depth map and the final fused depth map using the ground truth depth map. Then, the fused depth map from the three-branch network is refined using CSPN++ [56] with dilated convolutions.

Performance Evaluation

This method was tested on the KITTI depth completion validation dataset described in Section 4.1.2. The method resulted in 205.49 for the MAE, 709.41 for the RMSE, 0.90 for the iMAE and 2.03 for the iRMSE.

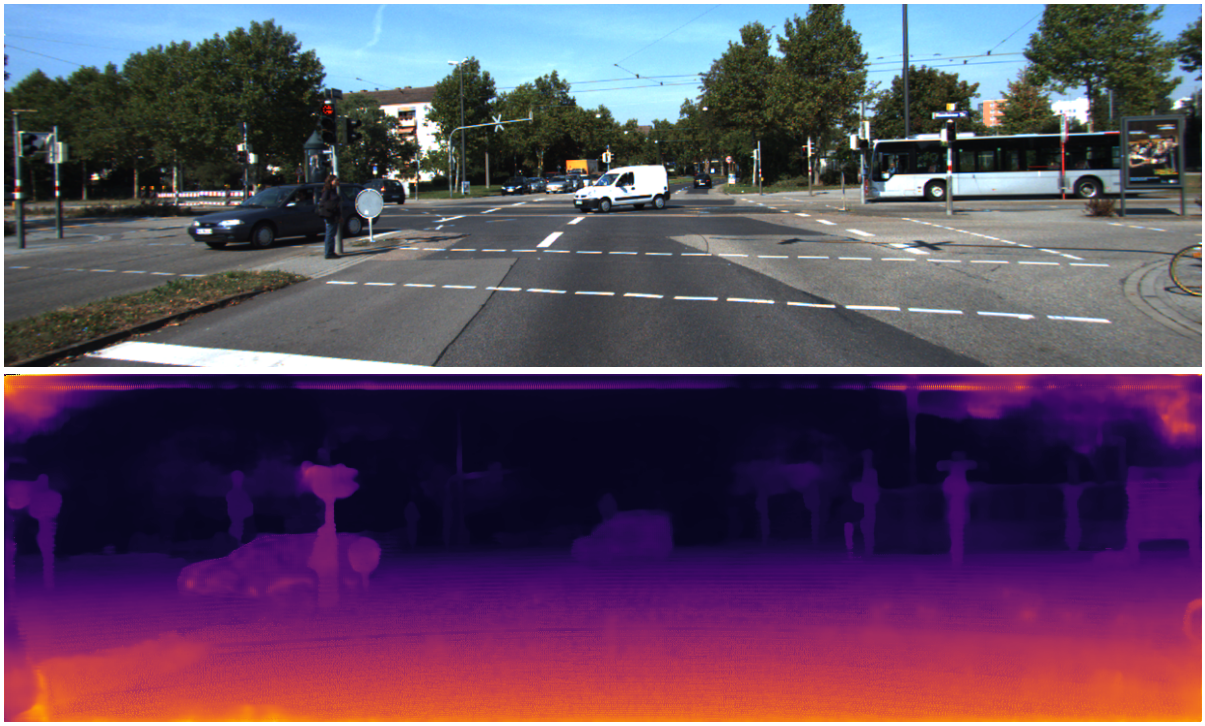

Figure 4.17: The qualitative results from SemAttNet [161] show the top image as the left RGB image from the first example scene from the KITTI-2015 stereo dataset. The corresponding disparity map estimation is shown in the bottom image. The raw dataset was used for the LiDAR data mapped onto the scenes in the stereo dataset, and then the results were converted from depth values to disparity using the calibration parameters.

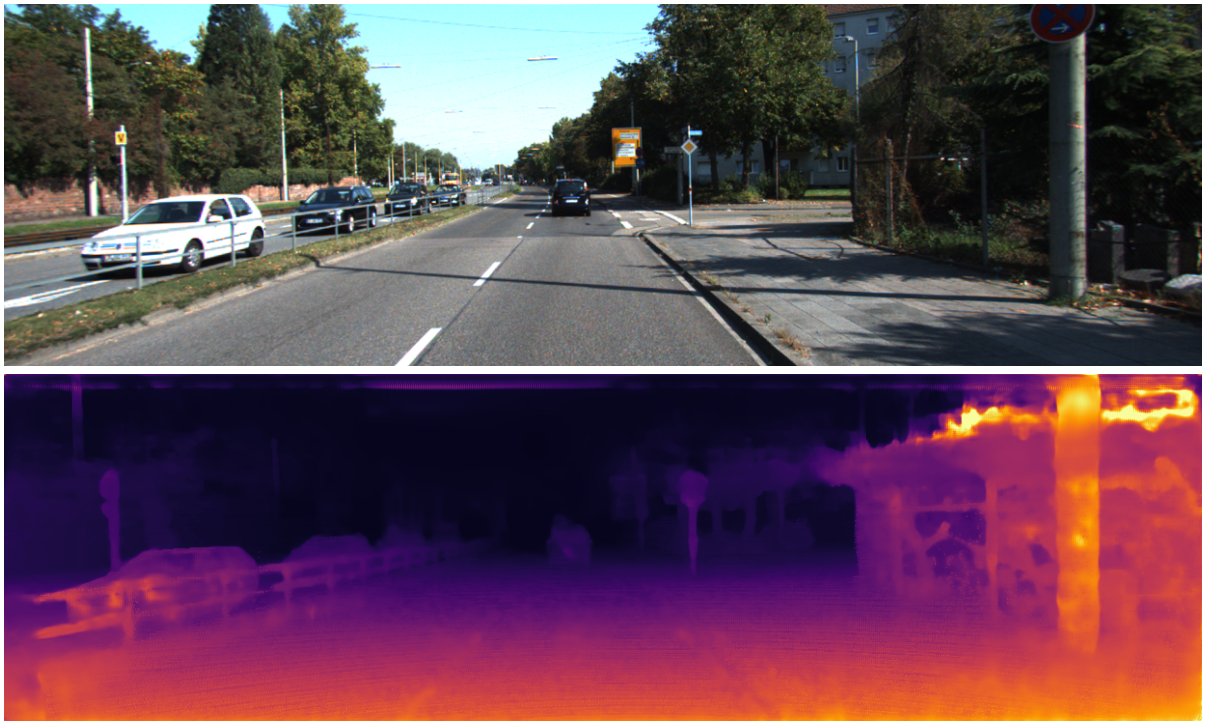

Figure 4.18: The qualitative results of SemAttNet on the second example scene.

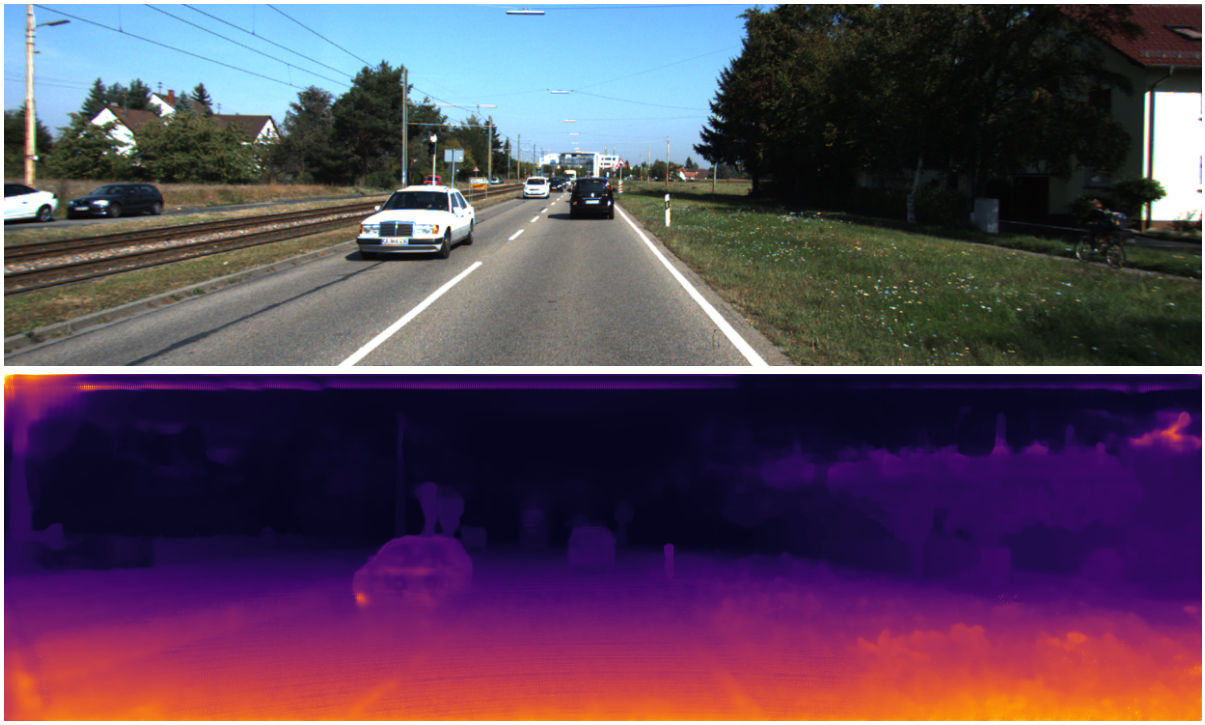

Figure 4.19: The qualitative results of SemAttNet on the third example scene.

The qualitative results in Figures 4.17, 4.18, 4.19 show similarities to PENet, where the disparity map has noise patterns in the colour disparity scale, which is visible in the road.

4.3.3 Dynamic Spatial Propagation Network for Depth Completion

Dynamic Spatial Propagation Network (DySPN) [165] computes a depth map from a sparse LiDAR depth map with a single RGB image as well for guidance. A coarse, dense depth map is initially computed with a convolutional neural network, and then the coarse depth map is refined using a dynamic spatial propagation network.

The initial coarse depth map is computed from the sparse LiDAR depth map using a UNet-like [79] encoder-decoder network with ResNet-34 [157] as the backbone. This network will output attention maps, the initial depth map, an affinity matrix and a confidence estimation for the input depth [56, 166, 158].

The depth map is updated with adaptive affinity weights to refine the depth map over N steps. This can be defined as $V^{t+1} = G^t V^t$ for $t \in \{0, 1, 2, \ldots, N\}$ with the depth map reshaped to one-dimensional vector $V^t \in \mathbb{R}^{mn}$ the global adaptive affinity matrix $G^t \in \mathbb{R}^{mn \times mn}$ which contains all adaptive affinity weights defined as follows:

$$G^t = \begin{bmatrix} 1 - \tilde{\lambda}^t_{0,0} & \tilde{w}^t_{0,0}(1,0) & \ldots & \tilde{w}^t_{0,0}(m,n) \\ \tilde{w}^t_{1,0}(1,0) & 1 - \tilde{\lambda}^t_{1,0} & \ldots & \tilde{w}^t_{1,0}(m,n) \\ \vdots & \vdots & \ddots & \vdots \\ \tilde{w}^t_{m,n}(1,0) & \tilde{w}^t_{m,n}(1,0) & \ldots & 1 - \tilde{\lambda}^t_{m,n} \end{bmatrix} \tag{4.10}$$

where $\tilde{w}^t_{i,j}(a,b) = \pi^t_{i,j}(a,b) w_{i,j}(a,b)$ is the adaptive affinity weight which is computed using the fixed affinity weights $w_{i,j}(a,b)$ and the global attention $\pi^t_{i,j}(a,b)$. $\tilde{\lambda}^t_{0,0} = \sum_{a \neq i, b \neq j}$ so that each row will sum to 1.

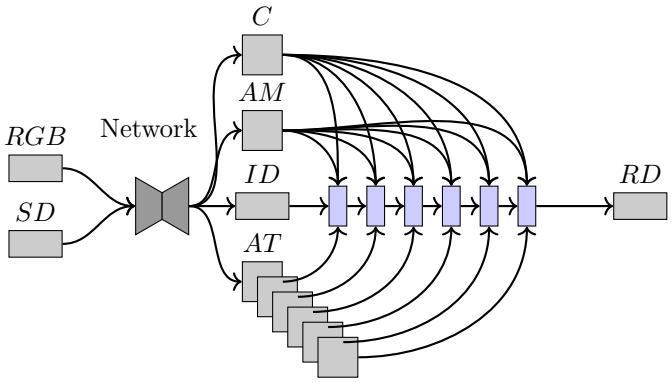

Figure 4.20: The architecture for DySPN. The inputs shown on the left are the RGB image and the sparse depth map SD captured by the LiDAR sensor. The encoder-decoder network outputs the confidence shown as C, the affinity matrix as AM, the initial depth estimation as ID and the attention maps as AT. The initial depth is refined over six steps of the DySPN shown in blue to get the final refined depth map shown as RD.

The DySPN reduces the computational complexity of the global adaptive affinity update by only computing for a neighbourhood. The affinity weights for neighbours of the same distance are set to be equal, and the neighbourhood is also deformable [167].

The L_1 and L_2 loss are both used to compute the training loss, which is defined as $Loss(h^{gt}, h^N) = L_1(h^{gt}, h^N) + L_2(h^{gt}, h^N)$ where h^N is the depth map after N steps and h^{gt} is the ground truth depth map.

Performance Evaluation

This method was tested on the KITTI depth completion validation dataset described in Section 4.1.2. The method resulted in 192.71 for the MAE, 709.12 for the RMSE, 0.82 for the iMAE and 1.88 for the iRMSE.

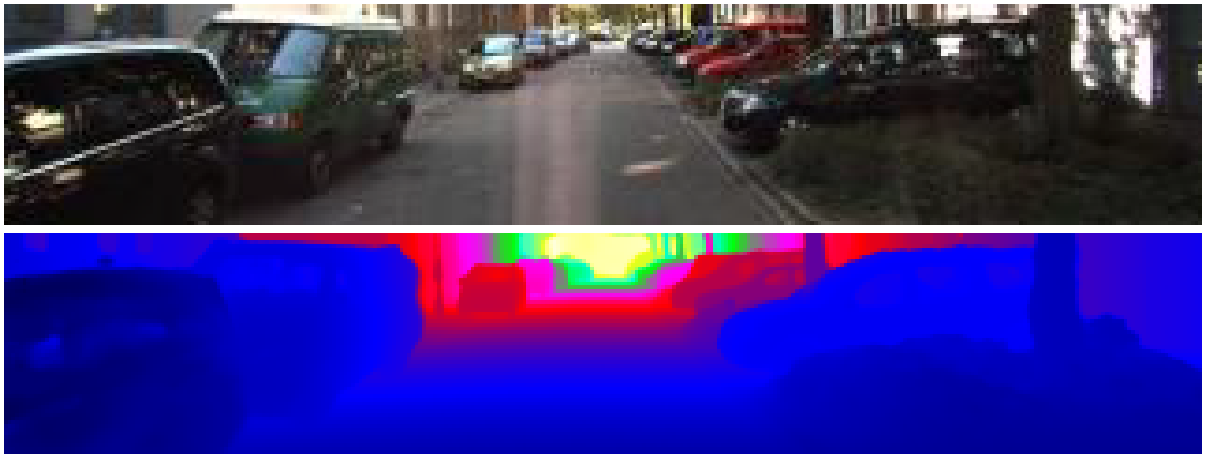

Figure 4.21: The qualitative results from DySPN [165] show the top image as the left RGB image from the first example scene from the KITTI depth completion dataset. The corresponding depth map estimation is shown in the bottom image. The colour scale is shown differently because it represents the depth estimations instead of the disparity estimations.

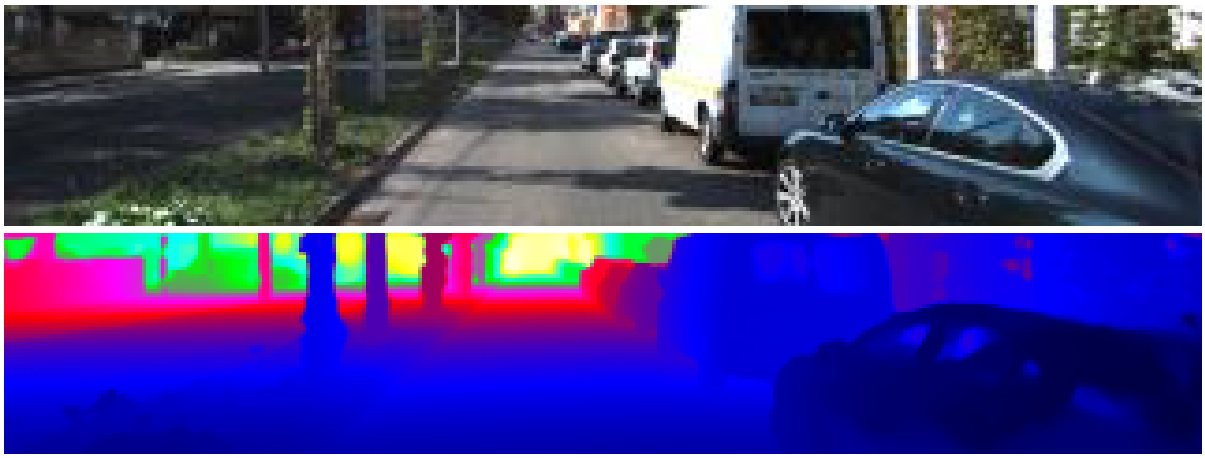

Figure 4.22: The qualitative results of DySPN on the second example scene.

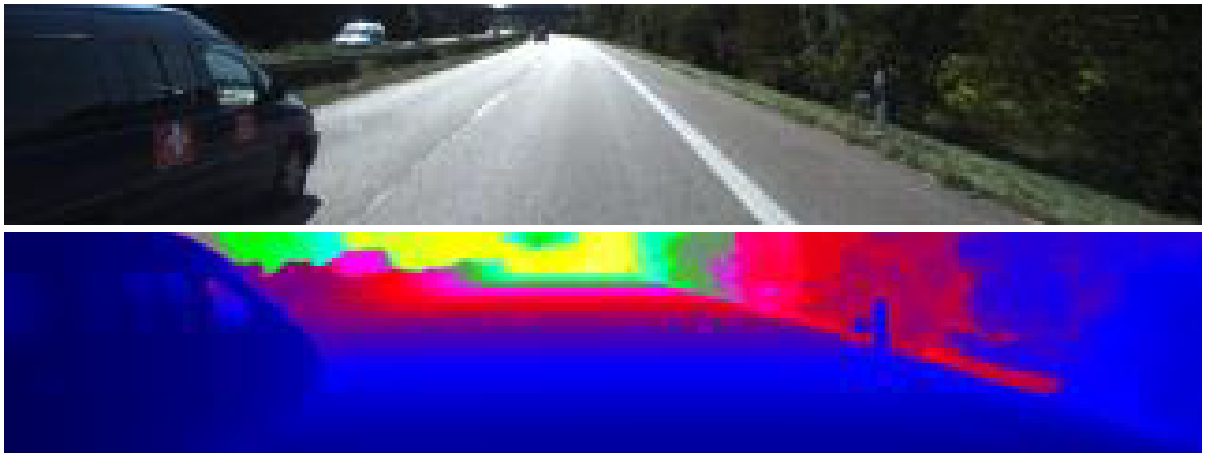

Figure 4.23: The qualitative results of DySPN on the third example scene.

The qualitative results in Figures 4.21, 4.22, 4.23 also show less noise in the depth map compared to the other depth completion methods, with smoother depth colour scale gradients on a flat plane surface, such as the road.

4.4 Methods for data fusion with RGB stereo and LiDAR

The stereo and LiDAR methods use both stereo RGB images captured from a 2 RGB camera setup to capture images taken from views next to each other, as well as the depth map from a LiDAR scanner projected onto one of the RGB camera views. These methods will use the 2 RGB stereo images and the sparse LiDAR depth map to compute a more accurate depth map estimation by using sensor fusion. In Chapter 2 Section 2.2, techniques which focused on RGB cameras and LiDAR sensors were explored, which informed that recent methods are primarily deep learning-based and end-to-end trained. This section evaluates the state-of-the-art deep learning-based stereo and LiDAR methods and investigates the network architecture and performance on the KITTI dataset.

4.4.1 Volumetric Propagation Network: Stereo-LiDAR Fusion for Long-Range Depth Estimation

Volumetric Propagation Network (VPN) [168] computes a depth map using the RGB stereo images and a LiDAR point cloud of the scene. First, a fusion volume is constructed to represent the scene as a 3-dimensional voxel grid evenly distributed along the depth range. The fusion volume is filled with information from the stereo images first by using a feature extraction network, then converting the voxel coordinates onto the stereo images using the

calibration parameters. Next, the point clouds are also converted into the fusion volume and stored as a binary occupancy representation.

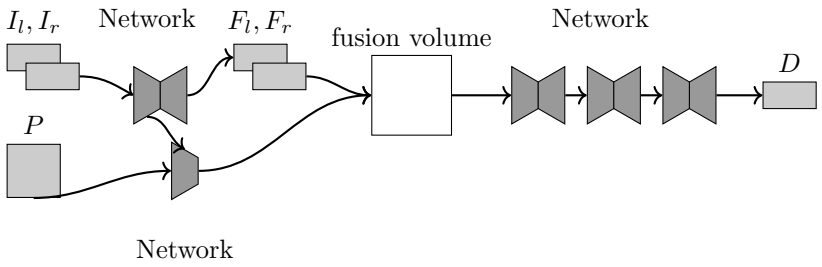

Figure 4.24: The model architecture for VPN. The inputs shown are the point cloud P and the left-right RGB image pair I_l, I_r used to compute feature maps F_l, F_r and the final depth map D is computed from the fusion volume.

The final information embedded into the fusion volume is point cloud features, which are extracted using FusionConv layers that cluster neighbouring points from the point cloud that lie within a voxel window. These points are fused with the corresponding features in the left RGB image feature map, and then the weighted geometric distance is computed to get the weights for the convolution operation on the neighbouring point features.

The final fusion volume is fed through a stacked hourglass network [27, 169] that consists of multiple encoder and decoder networks, which compute multiple cost volume estimates. The depth is computed using the soft-min function. The training total loss is computed with a weighted sum of the loss for each depth map estimate in the hourglass network.

Performance Evaluation

This method was tested on the KITTI depth completion validation dataset described in Section 4.1.2. The method resulted in 205.1 for the MAE, 636.2 for the RMSE, 0.9870 for the iMAE and 1.8721 for the iRMSE.

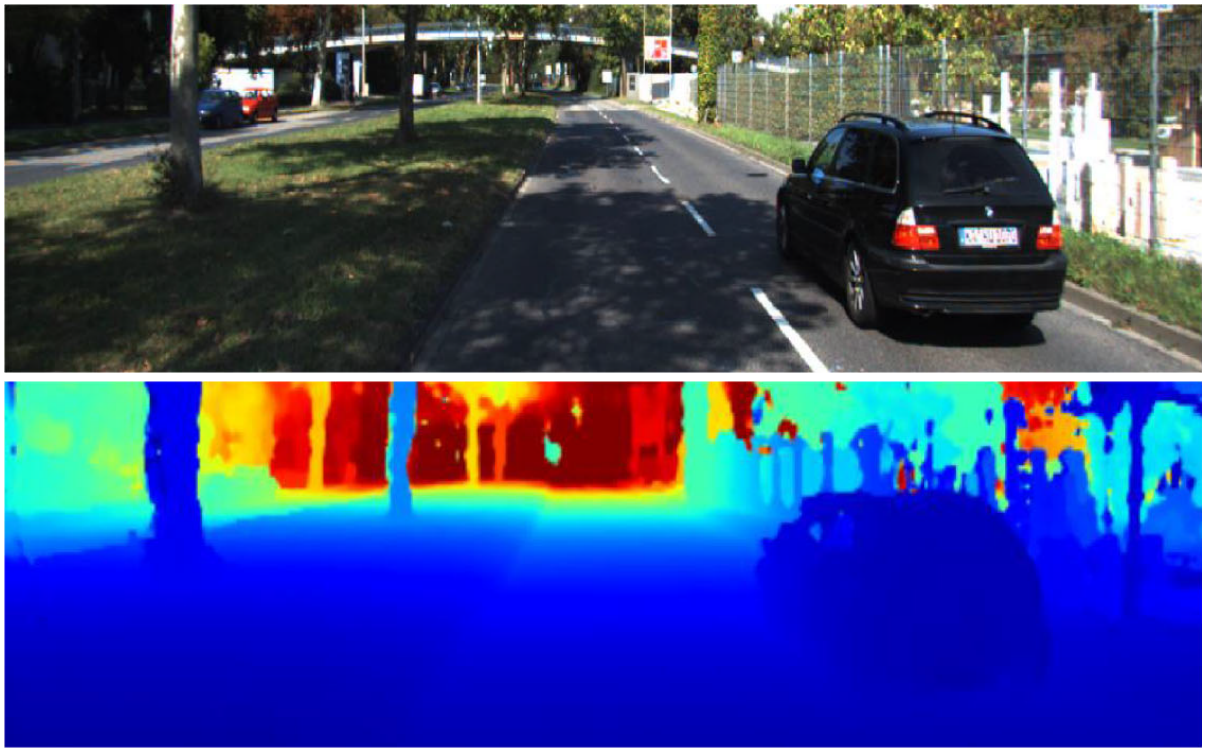

Figure 4.25: The qualitative results from VPN [168] show the top image as the left RGB image from the first example scene from the KITTI raw dataset. The corresponding depth map estimation is shown in the bottom image.

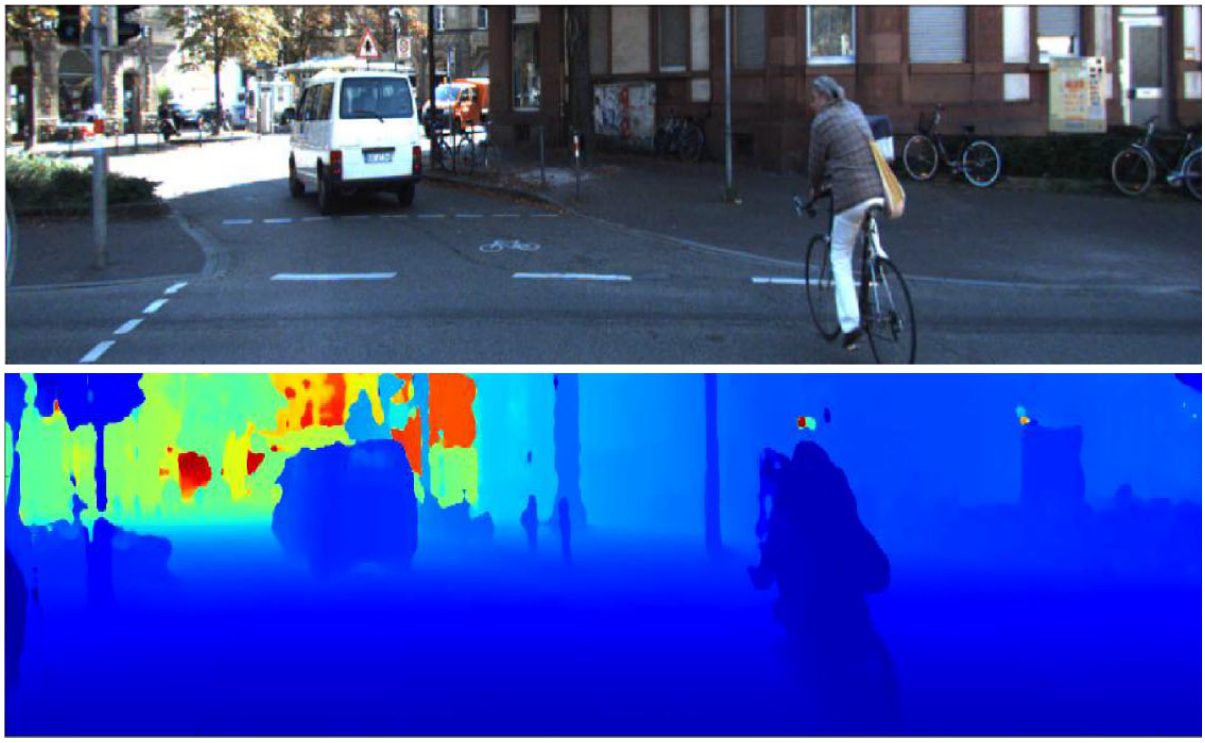

Figure 4.26: The qualitative results of VPN on the second example scene.

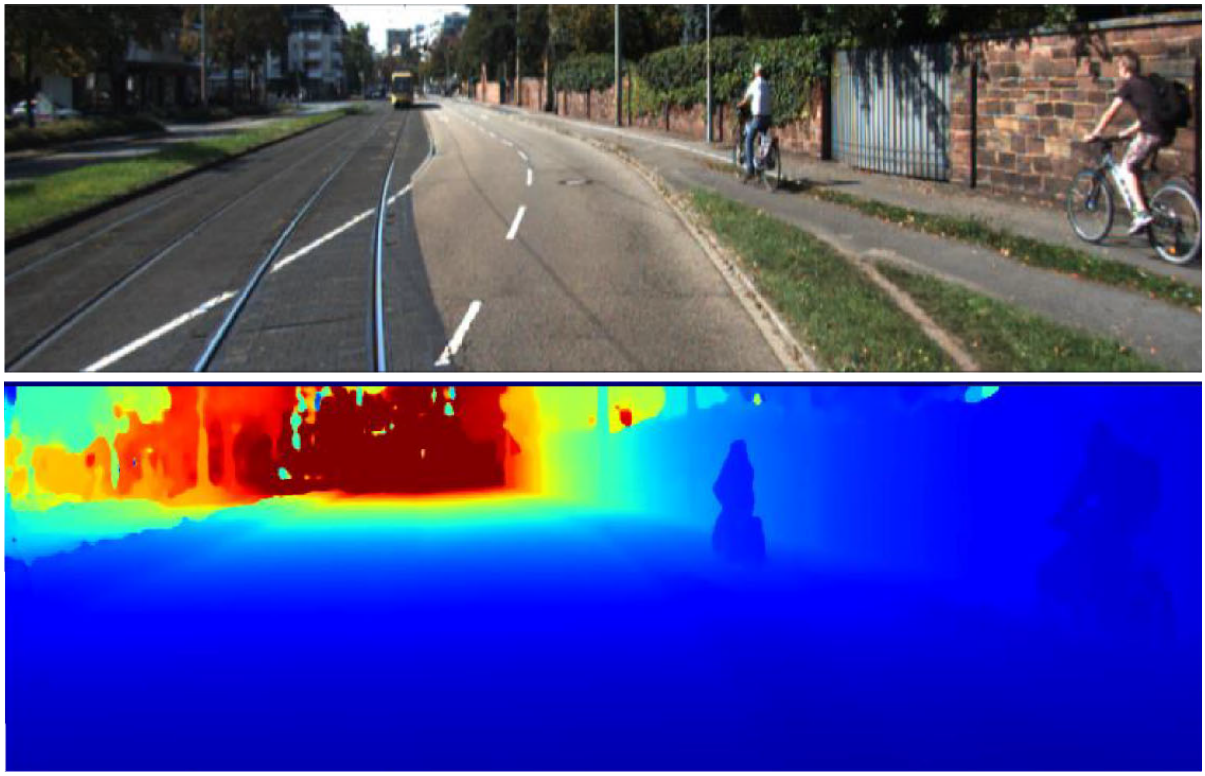

Figure 4.27: The qualitative results of VPN on the third example scene.

The qualitative results in Figures 4.25, 4.26, 4.27 show that the depth map has less noise than the depth completion methods with smooth depth colour scale gradients on the roads since it also uses stereo matching.

4.4.2 3D LiDAR and Stereo Fusion using Stereo Matching Network with Conditional Cost Volume Normalisation

The method "3D LiDAR and Stereo Fusion using Stereo Matching Network with Conditional Cost Volume Normalisation" (CCVN) [170] computes a disparity map by using the RGB images and sparse disparity maps of a LiDAR point cloud projected onto the RGB image views. The stereo and LiDAR data are fused by concatenating the RGB image with the LiDAR disparity map to create a 4-channel image. Then, this is input into a single feature extraction network using 2D convolutional layers to get feature maps for both left and right images. The features are used to construct a 4D feature volume which contains all potential matches. Next, a matching network is used to apply regularisation by using 3D convolutional layers to get the final cost volume. Finally, the soft-min function can be used to get the final disparity map.

The cost volume is normalised by using conditional cost volume normalisation, which is inspired by conditional batch normalisation [171, 172]. In conditional batch normalisation, the learnable parameters are defined as functions of conditional information and for conditional cost volume normalisation, the information used is the pixels in the LiDAR data. For the pixels with no values, a learnable constant value is used instead. The conditional cost volume method is approximated with a hierarchical extension to reduce the number of parameters by computing an intermediate vector conditioned on each LiDAR pixel and then modulating this with learnable parameters for each depth in the volume. This conditional cost volume regularisation is used in the matching network to compute the final cost volume from the feature volume.

Performance Evaluation

This method was tested on the KITTI depth completion validation dataset described in Section 4.1.2. The method resulted in 252.5 for the MAE, 749.3 for the RMSE, 0.8069 for the iMAE and 1.3968 for the iRMSE.

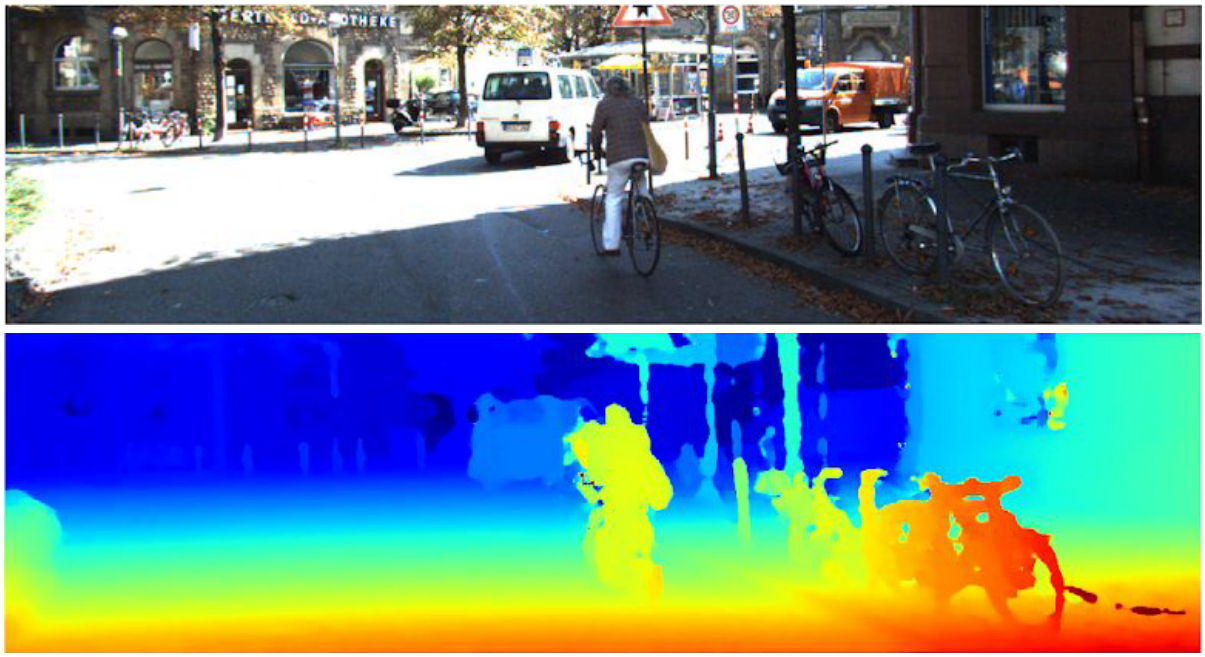

Figure 4.28: The qualitative results from CCVN [170] show the top image as the left RGB image from an example scene from the KITTI depth completion dataset. The corresponding depth map estimation is shown in the bottom image.

The qualitative results in Figure 4.28 show less noise than depth completion methods. However, the visual quality is lower, as seen in the signposts, compared to the methods which only use stereo data.

4.4.3 Noise-Aware Unsupervised Deep LiDAR-Stereo Fusion

The method "Noise-Aware Unsupervised Deep LiDAR-Stereo Fusion" [25] fuses a pair of stereo RGB images with the LiDAR projected disparity maps using an unsupervised training method without the need for training data with a ground truth disparity map.

The method uses a deep convolutional neural network that computes the final disparity maps using multiple stages similar to [26]. The first stage extracts features from the stereo images and features from LiDAR disparity maps. The network uses sparsity invariant convolutional layers [173] for the sparse LiDAR disparity maps. The stereo and LiDAR feature maps are concatenated into a single feature map for both left and right images. The next stage constructs a 4D feature volume using both feature maps, and then feature matching is computed using 3D convolutions to compute the cost volume. Finally, the disparity map is computed using a soft-argmin on the cost volume.

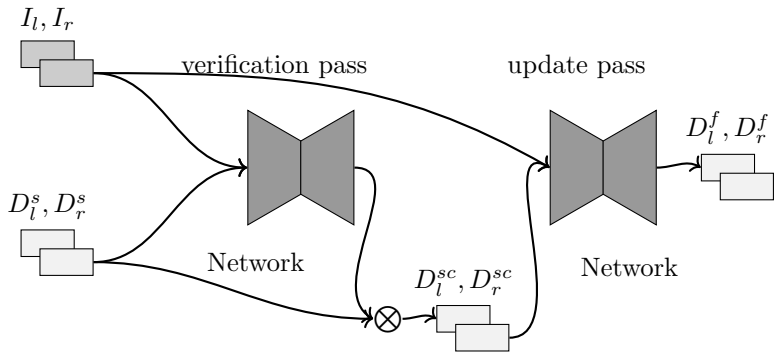

Figure 4.29: The inputs shown are the left-right RGB images I_l, I_r and the left-right sparse disparity maps D_l^s, D_r^s after the verification pass the filtered disparity maps are computed D_l^{sc}, D_r^{sc} this removes disparities that are not consistent. After the update pass, the fused disparity maps are computed D_l^f, D_r^f.

The data is passed through this network twice. The first produces initial disparity maps that are used to remove noisy LiDAR disparities, so the filtered LiDAR disparity maps are used in the second pass to compute the final disparity map. The network is trained using an unsupervised loss function that is the sum of an image warping loss, a LiDAR loss, a smoothness loss and a plane fitting loss. The image warping loss computes photometric consistency between the stereo images using the final disparity map. The LiDAR loss computes the distance between the filtered LiDAR disparity map and the final disparity map. The smoothness loss computes the smoothness of the final disparity map. The plane fitting loss computes the distance between the final disparity map and a disparity

map created by projecting the disparity values onto a local plane. The network is only updated after the LiDAR disparities have been filtered. Backpropagation will only go through the final network pass after filtering.

Performance Evaluation

This method was tested on the KITTI-141 subset described in Section 4.1.3. The method resulted in 0.0350 for the Abs rel, 0.0287 for $> 2px$, 0.0198 for $> 3px$, 0.0126 for $> 5px$ and 0.9872 for $\delta < 1.25$.

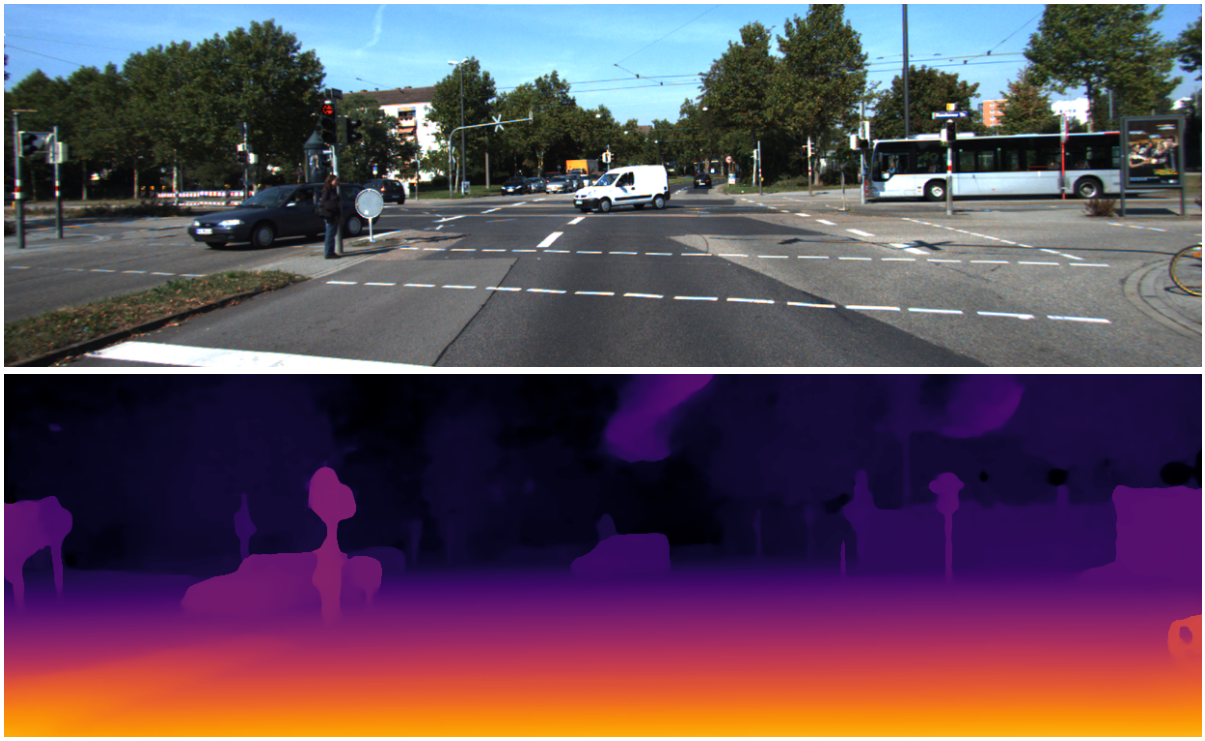

Figure 4.30: The qualitative results from LiDARStereoNet [25] show the top image as the left RGB image from the first example scene from the KITTI-2015 stereo dataset. The corresponding disparity map estimation is shown in the bottom image. The raw dataset was used for the LiDAR data mapped onto the scenes in the stereo dataset, and then the results were converted from depth values to disparity using the calibration parameters.

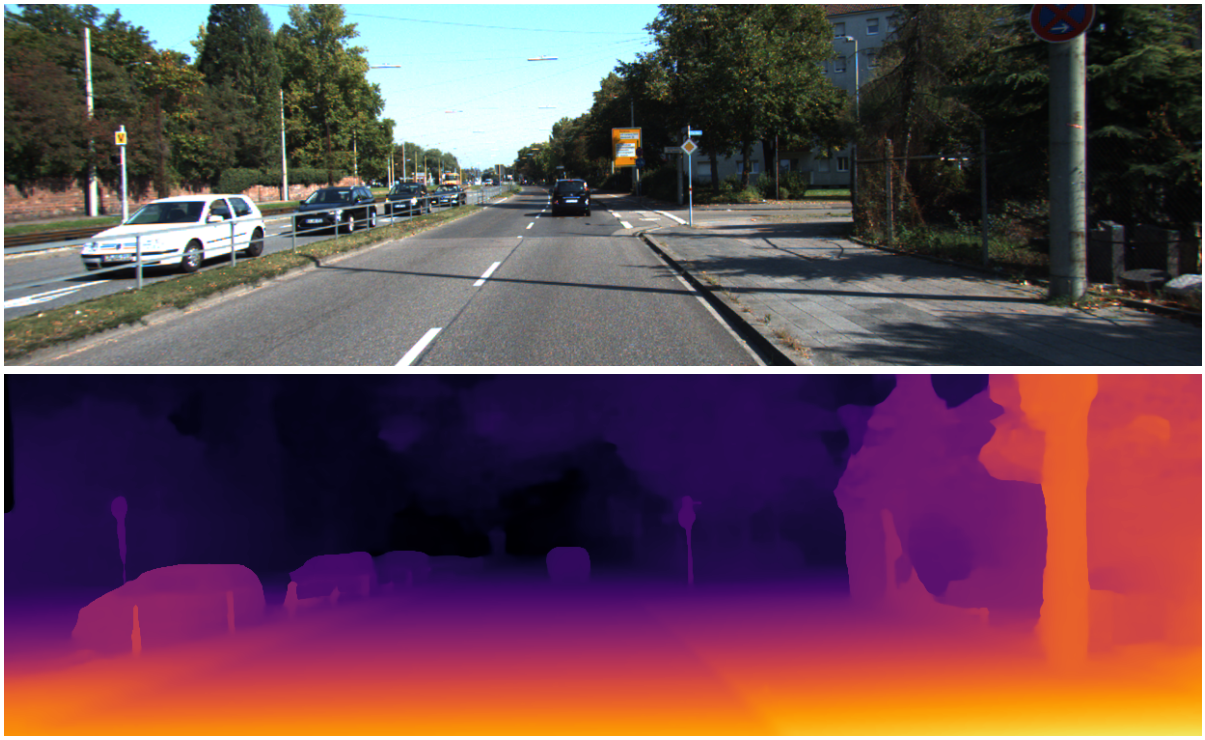

Figure 4.31: The qualitative results of LiDARStereoNet on the second example scene.

Figure 4.32: The qualitative results of LiDARStereoNet on the third example scene.

The qualitative results in Figures 4.30, 4.31, 4.32 show less fine detail in the disparity map than [150, 152], visible in

the left of Figure 4.31 in the fence and sign posts. However, it shows less noise than the depth completion methods with smoother colour disparity scale gradients.

4.5 Results, discussions and conclusions

In this section, the performance of the multi-modal fusion methods is compared, all evaluated using the KITTI dataset [146], which is developed for the application of autonomous driving. The dataset was created using sensors mounted to a vehicle and contains diverse scenes of urban, rural and highways. The KITTI depth completion dataset and the KITTI stereo dataset were used, which have been adapted from the raw data to evaluate the tasks of stereo and depth completion, where stereo predicts the disparity of each pixel between the stereo images and predicts the depth distance for each pixel.

Depth completion dataset

The following methods are compared: the Volumetric Propagation Network (VPN) method, the "3D LiDAR and Stereo Fusion using Stereo" Matching Network with Conditional Cost Volume Normalisation" (CCVN) method, the "PENet: Towards Precise and Efficient Image Guided Depth Completion" (PENet) method, the "SemAttNet: Towards Attention-based Semantic Aware Guided Depth Completion" (SemAttNet) method and the "Dynamic Spatial Propagation Network for Depth Completion" (DySPN) on the KITTI depth completion dataset.

Method name	MAE (mm)	RMSE (mm)	iMAE (1/km)	iRMSE (1/km)
VPN [168]	205.1	**636.2**	0.9870	1.8721
CCVN [170]	252.5	749.3	**0.8069**	**1.3968**
PENet [158]	209.0	757.2	0.9	2.22
SemAttNet [161]	205.49	709.41	0.90	2.03
DySPN [165]	**192.71**.0	709.12	0.82	1.88

Table 4.1: Comparison of the methods that have been tested on the KITTI depth completion dataset showing the mean absolute error (MAE) and root mean squared error (RMSE), both in millimetres, also the inverse mean absolute error (iMAE) and inverse root mean squared error (iRMSE), both in 1 / kilometres. The lowest error in each column is shown in bold.

Stereo dataset

The following methods are compared: the "RAFT-Stereo: Multilevel recurrent field transforms for stereo matching" (RAFT-Stereo) method, the "Noise-Aware Unsupervised Deep LiDAR-Stereo Fusion" (LiDARStereoNet) method and the "Hierarchical Neural Architecture Search for Deep Stereo Matching" (LEAStereo) method on the KITTI

stereo dataset.

Method name	3.0 px
RAFT-Stereo [150]	1.96
LiDARStereoNet* [25]	1.98
LEAStereo [158]	**1.65**
CREStereo [152]	1.69

Table 4.2: Comparison of the methods that have been tested on the KITTI stereo dataset showing the percentage of pixels with an error greater than 3 pixels. The lowest error in each column is shown in bold. *LiDARStereoNet was evaluated on a subset of the stereo dataset containing 141 scenes out of 200.

Comparison of accuracy

The compared results in Table 4.1 show the methods tested on the KITTI depth completion dataset, which has LiDAR and stereo data. The other results in Table 4.2 show the methods tested on the KITTI stereo dataset and evaluated using a pixel threshold error, which is the percentage of values with a greater error than 3 pixels.

The LEAStereo method shown in Table 4.2 has the highest performance on the stereo dataset using only a pair of RGB stereo images. This shows the success of the RGB stereo-only approach LEAStereo used compared to the LiDAR and RGB stereo approach that LiDARStereoNet used. This shows that Stereo-only methods can outperform LiDAR stereo methods, though using better network architectures and training approaches despite having less data.

In Table 4.1, the VPN method has the highest performance with the RMSE metric, and the CCVN method with the iMAE and iRMSE metrics. These are evaluated on the depth completion dataset, which has LiDAR and RGB, so the accuracy will be higher than RGB only. However, the DySPN method has the highest performance with the MAE metric, which uses LiDAR and only 1 RGB image. This shows that current state-of-the-art LiDAR stereo fusion methods are able to use the additional stereo data that the LiDAR-only methods do not have access to and provide lower error with most of the error metrics. However, it also shows that they could show lower performance in the MAE metric due to the LiDAR-only methods using improved network architectures and training approaches.

Comparison of strengths and limitations

Methods	Strengths	Limitations
RAFT-Stereo [150]	• lower memory usage from indexing features • model is available to download	• only RGB sensors used • lower accuracy
CREStereo [152]	• lower memory usage from indexing features • fast inference time of 0.41 seconds on a NVIDIA RTX 2080Ti GPU • higher accuracy • model is available to download	• only RGB sensors used
LEAStereo [154]	• fast inference time of 0.30 seconds on a NVIDIA V100 GPU • higher accuracy • model is available to download	• higher complexity from using neural architecture search • higher memory usage due to using 3D convolutions • only RGB sensors used
PENet [158]	• multiple sensor modalities used • lower memory usage due to only using 2D convolutions • very fast inference time 0.032 seconds on a NVIDIA RTX 2080Ti GPU • model is available to download	• lower accuracy
SemAttNet [161]	• lower memory usage due to only using 2D convolutions • multiple sensor modalities used • very fast inference time of 0.2 seconds on a NVIDIA A100 GPU • model is available to download	• higher complexity from using attention based modules • lower accuracy
DySPN [165]	• multiple sensor modalities used • lower memory usage due to only using 2D convolutions • very fast inference time of 0.16 seconds on a NVIDIA RTX 2080Ti GPU • higher accuracy	• model is not available to download
VPN [168]	• multiple sensor modalities used • higher accuracy	• higher memory usage due to using 3D convolutions • slow inference time of 1.4 seconds on a NVIDIA 1080Ti GPU • model is not available to download
CCVN [170]	• multiple sensor modalities used • higher accuracy	• higher memory usage due to using 3D convolutions • slow inference time of 1.011 seconds on a NVIDIA 1080Ti GPU • model is not available to download
LiDARStereoNet [25]	• multiple sensor modalities used • model is available to download	• higher complexity from using an unsupervised training loss • higher memory usage due to using 3D convolutions • lower accuracy

Table 4.3: The strengths and limitations of each method.

The compared methods (Table 4.3) show the strengths and limitations of each method. The methods with slower inference time take over 1 second to compute the depth map using a NVIDIA 1080Ti GPU due to using stereo

and LiDAR data as input. These are computed on an image size of (1242×375); however, if these models are trained on larger image sizes, then inference time would scale significantly with an $O(n^2)$ complexity scaling for the 2D feature extraction stage in the architectures and a $O(n^3)$ complexity scaling for the 3D Volume stages in the architecture. However, downsampling to the fixed size (1242×375) would prevent this and allow use of the model without retraining for a different input image size.

The methods with very fast inference time take less than 0.25 seconds to compute the depth map due to using LiDAR and RGB without stereo data. The methods that have high memory usage will be limited by the memory available when computing the depth maps, so downsampling is used to reduce the memory usage, which can reduce accuracy. Memory usage and inference time are important for these tasks, as having lower memory usage and inference times opens up the model to being used for real-time applications and reduces hardware requirements, allowing integration into real-time detection pipelines. This informs which approaches can be used for real-time use without needing to wait until offline computation is done.

Shown in Table 4.3, the methods VPN, CCVN, LiDARStereoNet and LEAStereo are limited by the high memory usage since they all use a feature volume to compute the final depth map, which has high memory cost. This can limit the methods to outputting a depth map at a lower resolution and then upsampling it, which will result in losing fine 3D detail. This loss in detail will result in small structures not being represented in the output depth map, so important information could be ignored.

These methods perform well on the KITTI dataset, with VPN having the highest accuracy, MAE and CCVN having the highest accuracy for the iMAE and IRMSE. LEAStereo also has the highest accuracy on the KITTI stereo dataset. However, these methods would not be able to efficiently compute depth maps for datasets with larger resolution data.

The methods RAFT-Stereo and CREStereo use a recurrent architecture that indexes features to improve computational and memory efficiency, so increasing the resolution of the RGB input images would not have as high a memory cost as the feature volume methods.

The methods PENet, SemAttNet and DySPN do not use stereo RGB images as inputs. Instead, they only use a single RGB image, which reduces the memory cost of the computation and would be more efficient at computing

depth maps for higher-resolution inputs.

The specific area these methods can be applied to is computing a 3D model of a road environment to be used for autonomous driving. The methods RAFT-Stereo, CREStereo and LEAStereo all rely on feature matching from stereo images, so they would only be suitable for autonomous driving in environments with texture, where features can be easily matched. These methods would perform worse for autonomous driving in environments with texture-less areas, such as building interiors with walls without any patterns. The methods using the LiDAR depth map as an input, such as PENet, SemAttNet, DySPN, VPN, CCVN, and LiDARStereoNet, would be more suitable for these indoor environments.

4.6 Confidence estimation for LiDAR and stereo data fusion

A novel LiDAR stereo data fusion pipeline was developed for computing depth maps from RGB stereo and LiDAR data that builds on the existing state-of-the-art methods. This follows from the work done in Chapter 3 with the method for stereo and ToF data fusion.

From the work done reviewing the state-of-the-art methods, it was shown that the state-of-the-art data-fusion methods could be outperformed by the stereo or LiDAR-only methods due to using a better network architecture or training approach. To address this issue, a new method was investigated that would aim to overcome this limitation by incorporating these better state-of-the-art methods into a data fusion pipeline that could then use the outputs of these methods to produce a final output that is even lower in error. This approach would also allow it to be more general, as it could be used with any state-of-the-art method, including ones later developed that have higher performance than existing ones today.

First LEAStereo method is used as with stereo data, it has shown to have higher performance over the LiDARStereoNet method, which used both LiDAR and Stereo input data. The LEAStereo method is used to compute a disparity map from the 2 RGB images to produce a stereo estimated disparity map. The LiDAR data was used to produce a separate disparity map by projecting it onto the stereo image views and calculating the disparity for each pixel.

To fuse these disparity maps, the approach used in Chapter 3. In that chapter, it was used for fusing stereo and ToF data, so it is modified to be used for LiDAR and stereo data. The fusion method uses the same convolutional

neural network architecture with the inputs modified. This method will output confidence estimations for each disparity value, which can be used to decide which disparity value to use from the stereo or LiDAR disparity map at each pixel location.

4.6.1 Convolutional Neural Network

The LiDAR data is projected to a sparse disparity map, with most pixels having no disparity value. To get a dense disparity map from the LiDAR data, interpolation is used to fill in the missing pixels using the neighbouring disparity values. The inputs used for the network are the disparity map that has been computed by the LEAStereo method from the stereo image pair, the projected disparity map from the LiDAR data and a warped difference image. This warped difference is computed by converting the RGB images to grayscale, then warping the right grayscale image using the LiDAR disparity map, and then taking the difference between the left grayscale image and the right warped grayscale image.

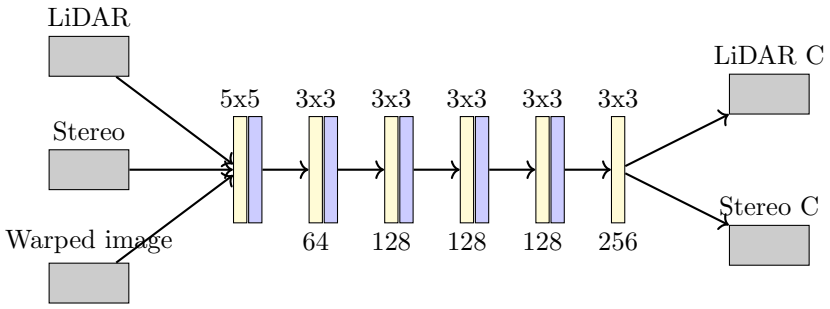

Figure 4.33: The network architecture of the convolutional neural network used for confidence estimation. The LiDAR disparity map, the stereo disparity map and the warped image are used as inputs. The yellow blocks represent a convolutional layer with the kernel size shown above and the input channel size shown beneath. The blue blocks represent the PReLU activation functions. The outputs are the LiDAR and stereo confidence maps.

These inputs are fed into the network as a single 3-channel image to output the two confidence maps. The network uses the same architecture used in Chapter 3 Section 3.3 with the input being modified to use a 3-channel input shown in Figure 4.33. This network will output with two channels for the two confidence maps for the stereo and LiDAR disparity maps.

4.6.2 Training the network

The training dataset is taken from the KITTI Vision Benchmark Suite described in Section 4.1. This contains 32918 training scenes where each has an RGB stereo image pair, a LiDAR disparity map and a ground truth disparity

map. The data are augmented by extracting random square patches with a width of 128 pixels and doing random vertical and horizontal flips and random rotations of $\pm 5°$, doing identical augmentations on the RGB and LiDAR disparity maps. The ground truth disparity maps are obtained using the provided ground from the KITTI dataset, converted to disparity map format. The loss function calculation and the training parameters are followed from what was used in Chapter 3 for the ToF and stereo fusion method.

4.6.3 Fusion using confidence

To fuse the LiDAR and stereo disparity maps, two fusion strategies are considered. The first strategy is to use the maximum confidence values only. The second strategy is to use a weighted sum using the confidence values.

For the maximum fusion strategy, the confidence maps are used to find the best disparity values for each pixel by taking the maximum as follows:

$$D_F(i,j) = \begin{cases} D_L(i,j), & \text{if } C_L(i,j) \geq C_S(i,j) \\ D_S(i,j), & \text{otherwise} \end{cases} \tag{4.11}$$

where $D_L(i,j)$, $D_S(i,j)$ are the disparity values at pixel i,j from the stereo and LiDAR disparity maps. $C_L(i,j), C_S(i,j)$ are the confidence values at pixel i,j for the stereo and LiDAR confidence maps outputted by the network.

For the weighted fusion strategy, the confidence maps are normalised and used to weight the disparity values as follows:

$$D_F(i,j) = D_L(i,j) \times \frac{e^{(C_L(i,j))}}{e^{(C_L(i,j))} + e^{(C_S(i,j))}} + D_S(i,j) \times \frac{e^{(C_S(i,j))}}{e^{(C_L(i,j))} + e^{(C_S(i,j))}} \tag{4.12}$$

where $e^{(\cdot)}$ is the natural exponential function.

4.6.4 Performance evaluation

The accuracy of the implemented method is evaluated by using quantitative results, and then the qualitative results are evaluated, which are 3D projections of the depth maps.

Quantitative results

The implemented method and LEAStereo are tested to provide a baseline, as the method uses LEAStereo to perform data fusion with the LiDAR data. Having the comparison of the method with LEAStereo and the LiDAR will show if the data fusion can provide a reduction in error compared to the input data. The other stereo methods are not considered here, as the LEAStereo method was shown to have the highest performance.

Testing is done on the KITTI depth completion dataset with the stereo dataset training scenes removed to get 1528 scenes. Each scene in the dataset has an RGB stereo image pair, a LiDAR scan projected onto a depth map and a ground truth depth map, which is obtained by combining multiple frames of LiDAR scans of the scene into a single depth map. The methods tested compute a disparity map for each scene that estimates the true disparity at each pixel.

To objectively evaluate the accuracy of the estimated disparity maps, the following errors were used: the mean absolute error (MAE), the root mean squared error (RMSE) and a pixel threshold error for {1px, 2px, 3px, 5px} that computes the percentage of disparity estimates that are within the threshold. These metrics are computed for each scene with the ground truth and then averaged. The LiDAR data is also used to compare with the methods by interpolating the missing values in the LiDAR disparity map to get disparity values at each pixel.

Method name	MAE	RMSE	1px error	2px error	3px error	5px error
LEAStereo	0.71857	**1.36208**	3.72739%	**1.39395**%	**0.75764**%	**0.41516**%
LiDAR	0.62999	2.30649	3.94646%	2.74662%	2.03827%	1.44598%
Presented Method Maximum	**0.47225**	1.38002	**2.86182**%	1.52125%	0.84817%	0.48718%
Presented Method Weighted	0.59479	1.42881	3.49172%	2.29013%	1.67999%	1.07144%

Table 4.4: The results of the methods tested on the KITTI dataset with the lowest errors are shown in bold. "Presented Method Maximum" is the method that uses only the values with maximum confidence, and "Presented Method Weighted" is the method that weights the values with the confidence values and sums them together. The lowest error from each column is shown in bold.

The results from Table 4.4 show that both LiDAR and Stereo Fusion methods reduced the MAE compared to the disparity map computed by LEAStereo and the LiDAR disparity map. The "Presented Method Maximum" method, which uses the maximum confidence disparity values, had the lowest error when compared to the "Presented Method Weighted" method, which weights the disparity values. The RMSE and the pixel threshold errors all show that LEAStereo has the lowest error, apart from the 1-pixel threshold error.

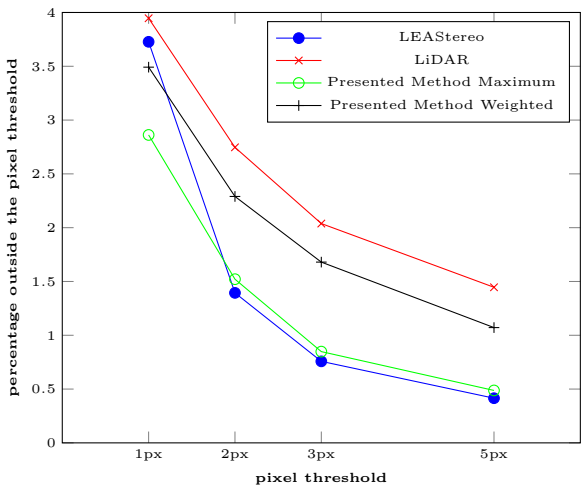

Figure 4.34: The threshold error plotted. "Presented Method Maximum" is the method that uses only the values with maximum confidence, and "Presented Method Weighted" is the method that weights the values with the confidence values and sums them together. The X-axis shows the pixel threshold used to compute the error, and the Y-axis shows the percentage of the pixels with an error that is outside of the threshold.

The graph in Figure 4.34 shows that both the Presented Methods produce the lowest error for the 1-pixel threshold error. This means that for fine detailed disparity estimation at the single pixel level, the presented methods perform best. However the graph also shows the LEAStereo method has a slightly better performance for compared to the "Presented Method Maximum" for disparities that have an error of 2 pixels or greater compared to the "Presented Method Maximum" method. This means LEAStereo has a slight advantage in reducing the larger outlier errors that are at the multi-pixel level.

Qualitative results

In this section, the results of the presented method are visually displayed by converting the disparity map estimated by the method, converting it to depth values, projecting it into a 3D point cloud, and using the corresponding RGB values in the RGB image to determine the colour of that point.

Figure 4.35: Left RGB images for 3 scenes from the KITTI dataset.

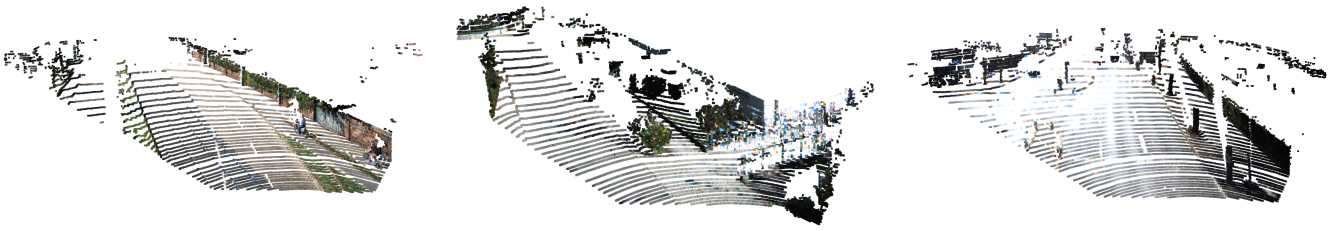

Figure 4.36: Sparse LiDAR disparity maps for the 3 scenes from the KITTI dataset.

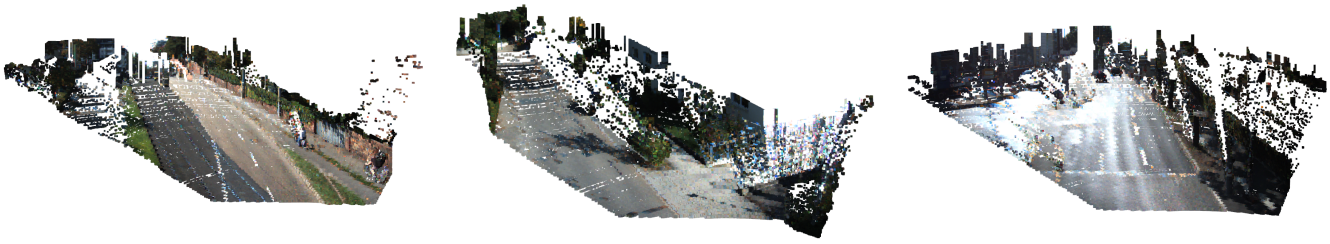

Figure 4.37: LiDAR disparity maps for the 3 scenes from the KITTI dataset interpolated.

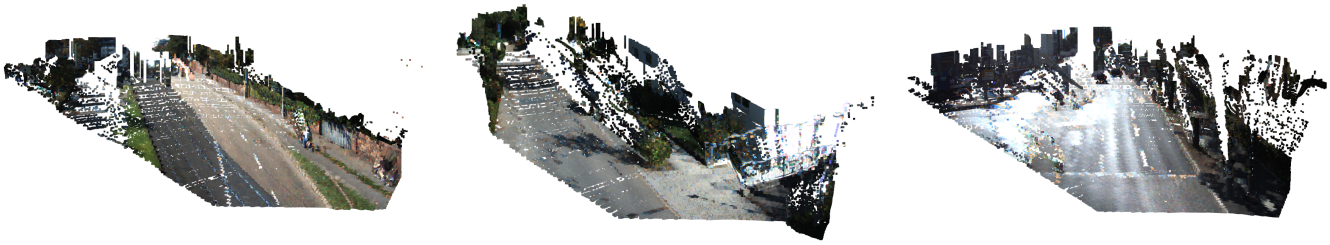

Figure 4.38: "Presented Method Maximum" results for the 3 scenes from the KITTI dataset.

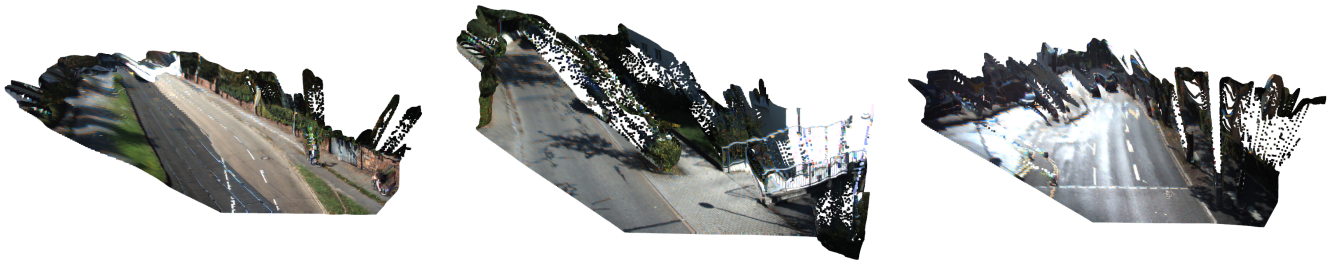

Figure 4.39: LEAStereo results for the 3 scenes from the KITTI dataset.

The three images in Figure 4.35 show the left RGB stereo images from 3 example scenes. The left image is from the first scene in the dataset. The middle image is from the scene with the lowest MAE for the "Presented Method Maximum" method. The right image is from the scene with the highest MAE for the "Presented Method Maximum" method.

The images in Figure 4.36 show the sparse LiDAR data for each of the three scenes. The LiDAR disparity values

have been projected into a 3D point cloud with RGB colours from the RGB images.

The images in Figure 4.37 show the LiDAR data after the missing pixels have been interpolated and then projected into a 3D point cloud. The images in Figure 4.38 show the 3D point cloud projection from the disparity map from the presented method that fused the LiDAR and Stereo disparity maps using the maximum confidence.

The images in Figure 4.39 show the 3D point cloud projection from the disparity map computed by LEAStereo.

To further validate the perceptual quality of the result, pairwise comparisons were obtained from random unique evaluators to provide a more rigorous quantitative evaluation. Amazon Mechanical Turk was used to conduct the comparison, which allows access to random, unique evaluators to complete the evaluation task defined. The evaluation was set up to display the results of the visual point cloud projection for both LEAStereo and the "Presented Method Maximum". These comparisons were displayed to the evaluators side-by-side in a randomised order with a reference as the RGB image of the scene provided. Each evaluator was given 3 scenes to evaluate to determine which point cloud projection best matched the reference image for each scene.

Method name	Scene 1	Scene 2	Scene 3	Total	Mean opinion score
LEAStereo	**18**	**16**	**16**	**50**	**0.833**
Presented Method Maximum	2	4	4	10	0.167

Table 4.5: The results of the visual evaluation by 20 random unique evaluators from the Amazon Mechanical Turk platform for three scenes. The better evaluation for each column is shown in bold.

The results here (Table 4.5) show that LEAStereo has significantly better visual quality than the "Presented Method Maximum" method, which uses the maximum fusion strategy. The weighted fusion strategy was not tested here because it shows significantly lower performance on the error metrics, so only the best fusion strategy was compared.

4.6.5 Fusion using depth completed LiDAR data

To further build on this LiDAR stereo data fusion pipeline, the LiDAR data is improved by using a LiDAR depth completion method. This method is used to reduce the error in the input disparity map provided from the LiDAR data. This will modify the presented data fusion pipeline so that it will use the stereo disparity map provided from LEAStereo, as well as the LiDAR disparity map that has been improved with a depth completion method, and perform data fusion using the presented confidence fusion model.

This approach is tested using the PENet method [158] as the method chosen to reduce the error in the LiDAR

disparity maps before fusion with the stereo disparity maps computed by LEAStereo [154]. The PENet method is chosen since, at the time of this research, it is the most accurate method that has the model publicly available.

The confidence estimation network is also retrained, which is used for fusion on the disparity maps produced by the PENet method, and compared in the table below.

Method name	MAE	RMSE	1px error	2px error	3px error	5px error
LEAStereo	0.71857	1.36208	3.72739%	1.39395%	0.75764%	0.41516%
LiDAR	0.62999	2.30649	3.94646%	2.74662%	2.03827%	1.44598%
PENet	0.36403	**0.91875**	1.62626%	0.92506%	0.61626%	0.34327%
Presented Method	0.37705	1.06877	1.77653%	1.01569%	0.65431%	0.37870%
Presented Method Retrained	**0.35721**	0.92201	**1.52902**%	**0.82854**%	**0.52235**%	**0.27263**%

Table 4.6: The confidence estimation fusion results. Here, the "Presented Method" uses the confidence estimation on the PENet and LEAStereo disparity maps. The "Presented Method Retrained" retrains the confidence estimation on the PENet disparity maps. The lowest error in each column is shown in bold.

The results in Table 4.6 show that the "Presented Method Retrained", which is trained on the PENet data, has the lowest error for the MAE and the pixel threshold errors.

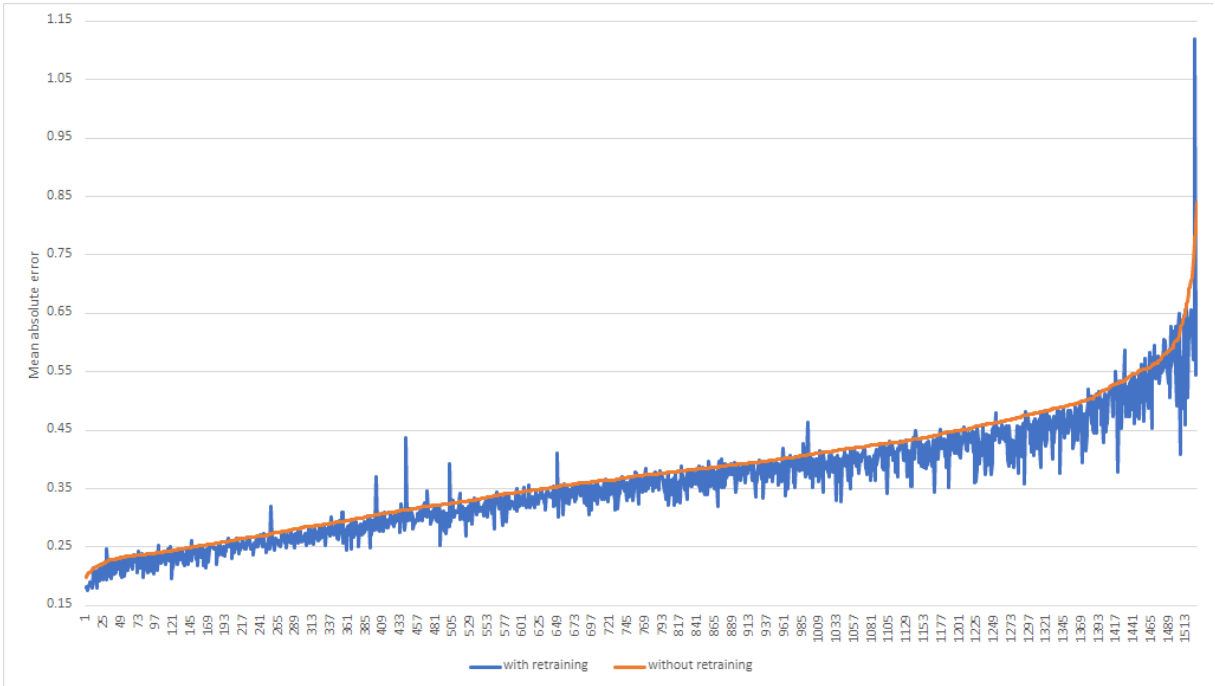

Figure 4.40: This graph shows the MAE of the fused method trained on the PENet data in blue compared with the fused method trained on the LiDAR data without depth completion in orange. The graph plots all the scenes sorted by the method without training on PENet data.

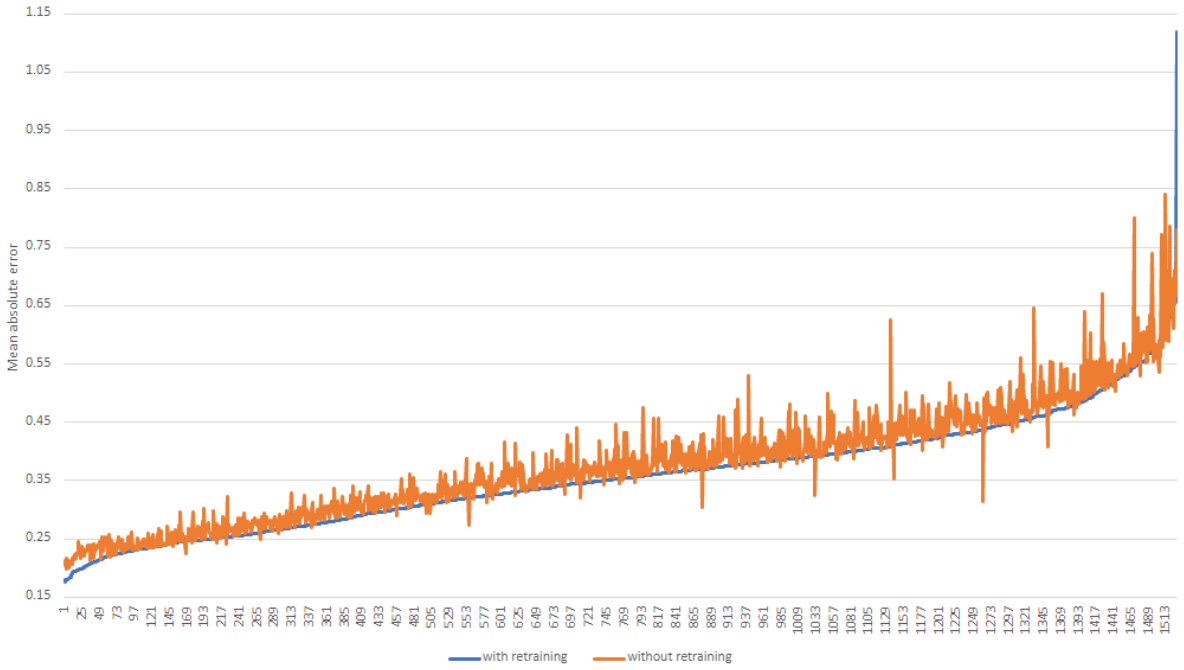

Figure 4.41: This graph shows the same data as Figure 4.40, but it has been sorted by the method that has been trained using the PENet data

To further visualise the results, the retraining graph plots for the error in each scene are displayed. To generate this graph, the MAE was recorded for each scene, computed for both the presented method without retraining and the presented method with retraining, as follows:

$$S = \{(r_n, p_n) \mid n \in \{1, 2, \ldots, N\}\} \tag{4.13}$$

where r_n is the MAE of the method with retraining at scene n and p_n is the MAE of the method without retraining at scene n.

The MAE has variation between the scenes, so to display this data in a graph, it needs to be sorted by error. For the best comparison, this is done for both the presented method without retraining and the presented method with retraining, so the same error data is shown in two separate graphs.

To do this set S is then sorted by both r_n and p_n to obtain:

$$L_r = \text{sort}(\{r_n \mid (r_n, p_n) \in S\}, \text{ascending}) \tag{4.14}$$

$$L_p = \text{sort}(\{p_n \mid (r_n, p_n) \in S\}, \text{ascending}) \tag{4.15}$$

The lists are plotted as two graphs, where Figure 4.40 shows the list L_p plotted and Figure 4.41 shows the list L_r plotted. These graphs both show the same data, with the difference being which method's error they were sorted by. From these graphs, it can be seen that the presented method with retraining has consistently lower error than the presented method without retraining.

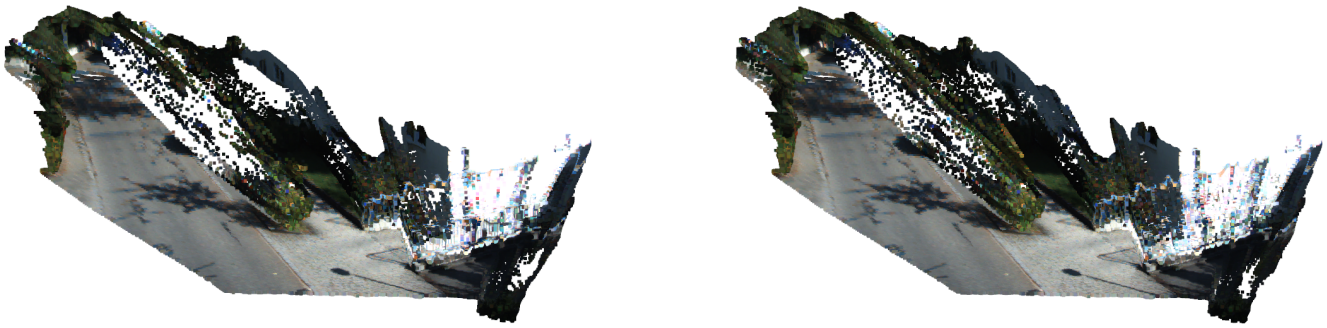

Figure 4.42: A 3D view of the scene with the best improvement in performance of the MAE.

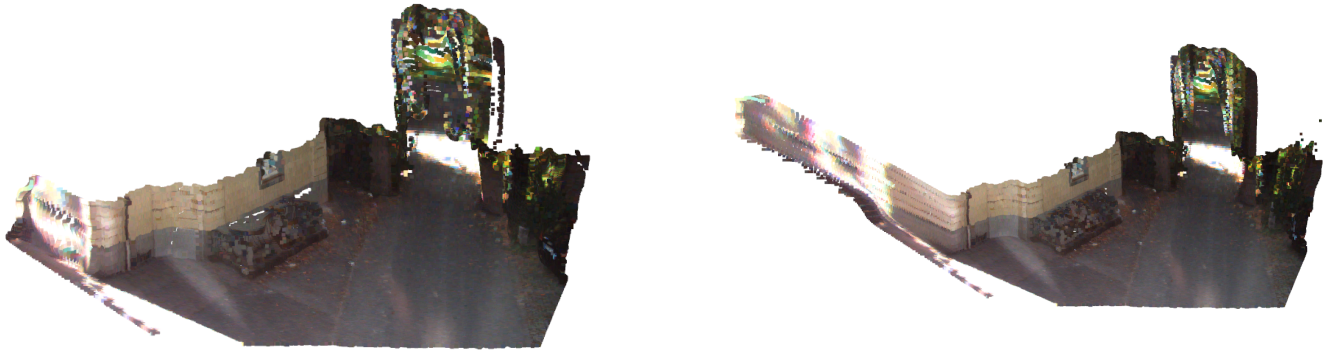

Figure 4.43: A 3D view of the scene with the least improvement in performance of the MAE.

These images (Figure 4.42, 4.43) show the method without training using the PENet data on the left and the method trained using the PENet data on the right. Figure 4.42 shows the scene where the left result had the greatest improvement in performance with the MAE. Figure 4.43 shows the scene with the lowest improvement in performance.

4.7 Conclusions

In this chapter, state-of-the-art computing disparity or depth maps using RGB and LiDAR data were reviewed. It was found that the methods that used an RGB stereo and LiDAR data fusion approach were lagging in performance behind the latest RGB stereo-only or LiDAR-only methods. This highlighted a gap that could be addressed in developing a novel LiDAR stereo data fusion pipeline, which could leverage the strengths of the latest RGB stereo only or LiDAR only methods by using the work done in Chapter 3 and applying it to this area.

The work done developing the data fusion pipeline covered first incorporating the LEAStereo method to get low error disparity map estimates from the stereo data. Then the PENet method was applied to get low error disparity map estimates from the LiDAR data. The pipeline was able to use the confidence fusion method from Chapter 3 to perform data fusion and produce lower error metric disparity map estimates than either of the inputs.

The results from the evaluation showed that the presented confidence fusion method, with just the LEAStereo method and simple interpolation for the LiDAR data, was able to produce lower error in multiple error metrics, such as the MAE and pixel threshold error, when the maximum fusion strategy was used. The error was further reduced by improving the LiDAR data, where the PENet method was used instead of simple interpolation. The data fusion pipeline presented, which used both the LEAStereo and PENet methods, showed a reduction in the error metrics when compared with both the LEAStereo and PENet individually, which demonstrated the success of the data fusion approach for this depth information. This pipeline can be used for providing lower error depth map data to a full crack detection pipeline, which can use this data to make estimates about the presence of cracks with lower error.

Chapter 5

Crack Detection

The research aim is to develop methods to detect and analyse flaws such as cracks by using a multi-sensor fusion approach. The previous chapters covered improving the quality of RGB image data and reducing errors in depth data through data fusion methods. This chapter addresses the detection of flaws from RGB image data and depth data. This will enable a crack detection pipeline that can use the data fusion methods from the previous chapters to improve the input data used in the detection stage of the pipeline.

One common type of flaw is the presence of cracks on surfaces, which can significantly compromise the structural integrity of the materials or components. This Chapter focuses on the development of a crack segmentation approach that aims to identify and assess the significance of cracks based on their depth within the surface. The work done in this Chapter resulted in a publication [174]. The method of segmentation is chosen for analysing cracks over the other methods, such as detection or bounding box estimation, because it performs pixel-level estimation to give more information about the shape of the crack than the other methods.

The approach chosen to determine whether a crack is significant is to analyse its depth. Deeper cracks are generally more critical and require immediate attention to prevent potential failures. To estimate the depth of a crack solely from observations of the surface, the relationship between the crack's length and depth is utilised.

5.1 Segmentation methods

Deep learning segmentation methods are used to extract the shape of cracks from images [89, 80, 88] by training deep learning architectures such as convolutional neural networks (CNNs) or Vision transformers (ViTs). These can be trained on labelled datasets containing images of cracked surfaces with manually annotated segmentation maps that give a ground truth for training. These models can learn to distinguish crack patterns from the surrounding surface by learning from the data. The resulting segmentation map provides a pixel-wise classification, indicating the regions corresponding to cracks up to pixel-level accuracy.

Once the segmentation masks are obtained, further analysis can then be done to extract meaningful information about the cracks. By drawing a path between the tips of the crack using the information from the segmentation map, the length of the crack can be calculated to give an estimate of the depth of the crack.

To validate the effectiveness of the crack segmentation approach, both real-world datasets of concrete cracks and synthetic segmentation maps are used. The use of real-world datasets ensures that the approach will work for real-world scenarios. Synthetic data is also used to test for specific crack shapes to verify that the approach will work for different shapes that are not included in the real-world data and can be used to inform the future direction of the research.

Recent methods for crack segmentation include CNN architecture approaches such as an encoder-decoder [102, 103, 104] network architecture, as well as using the UNet architecture [105, 106, 107], which is a specific encoder-decoder architecture that includes skip connections going directly from the earlier encoder layers to the later decoder at the same resolutions. ViTs have also been used for crack segmentation [120, 121], showing state-of-the-art performance; however, they require significant data and compute to train, as they lack the inductive bias present in the CNN architecture.

5.1.1 Datasets

The crackseg9k dataset [175, 176] is used for training and evaluation of the crack segmentation methods in this research. This dataset is selected as it is the largest, most diverse and consistent crack segmentation dataset constructed so far, which provides a valuable resource for training and evaluating crack detection models. It

consists of 9,159 available RGB crack images sourced from multiple open-source datasets, which encompass different environments with different material surfaces and different lighting conditions, as well as covering various crack types. The dataset is collected from the following 10 sub-datasets:

- Masonry: Cracks in masonry walls.
- CFD: Road surface cracks.
- CrackTree200: Pavement cracks.
- Volker: Cracks in walls and roads.
- DeepCrack: Cracks in roads and buildings.
- Ceramic: Cracks in ceramic building facades.
- SDNET2018: Non-crack images only.
- Rissbilder: Cracks in buildings.
- Crack500: Road cracks.
- GAPS384: Asphalt road cracks.

The images have all been resized to 400×400 pixels, and the ground truth images have been preprocessed. The preprocessing is done as a data refinement process to reduce distortions and discontinuities in the ground truth data, where morphological operations such as Gaussian blurring, erosion and dilation were used to improve the ground truth data. The dataset has been split into 7,332 images for training and 1,827 images for testing, which is a standard 80% train 20% test split.

5.1.2 Training a segmentation model

To compute segmentation masks for cracks in surfaces, various approaches were explored, which started by implementing a deep learning model. The initial attempt was to train a model that could produce depth maps for crack segmentation.

To guide the implementation, the method outlined in the paper titled "Downstream Semantic Segmentation Model for Low-Level Surface Crack Detection" [101] was used for reference. This method was chosen due to its approach

using an off-the-shelf neural network architecture, which has better availability and can speed up the development, allowing the method to determine the crack significance by assessing the crack length using the segmentation results. The model described in the paper used the DeepLabV3 [177], which is a state-of-the-art segmentation model using ResNet-101 [157] as the backbone.

To train the model, the parameters started from a random initialisation and the crackseg9k dataset described in Section 5.1.1 was used as the training data. During the training process, the methodology of the original paper was followed. This involved using the binary focal loss and using data augmentation to further increase the dataset size. This is done by using transformations such as random cropping, flipping and blurring, which will add additional diversity to the training data.

After training the model, its performance was evaluated on the separate testing dataset split described in Section 5.1.1. The trained model produced the results shown in Figure 5.1.

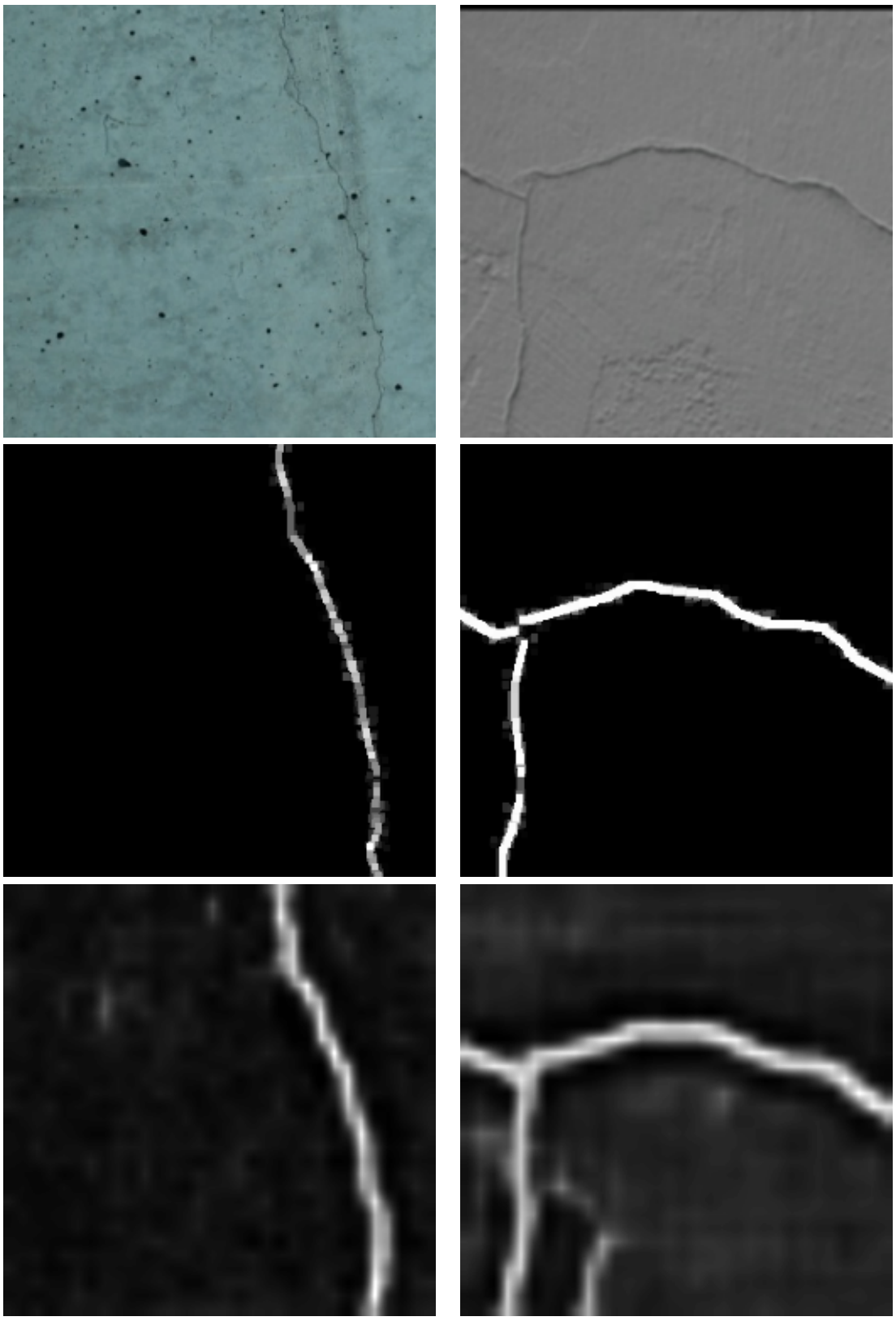

Figure 5.1: The top row shows two examples of the input RGB crack images, the middle row shows the corresponding ground truth segmentation maps, and the bottom row shows the estimated semantic maps by the network

The trained segmentation model results shown in Figure 5.1 show limited performance. This is due to the complexity of the cracks shown in the dataset, as well as the size and diversity. To overcome this, a larger model architecture and more computing power would be needed, so the use of pre-trained crack segmentation models was explored to improve the results in the subsequent stages of the research.

5.1.3 Using a pre-trained segmentation model

To overcome the limitations of training the presented crack segmentation model, which was constrained by computational resources, the utilisation of a pre-trained model that had been trained with greater computational resources was explored. This approach allowed the leveraging of the increased training time used for the pre-trained model to improve the accuracy of the crack segmentation task. For the research, a pre-trained model from the paper titled "CrackFormer: Transformer Network for Fine-Grained Crack Detection" [121] was selected. This particular model differs from traditional CNN models as it utilises a ViT architecture, which has demonstrated remarkable performance in various computer vision tasks.

The pre-trained model, which was used, had been trained on three distinct datasets specifically focused on concrete cracks. These were the CrackTree260 [178] dataset, the CrackLS315 [179] dataset and the Stone331 [104] dataset, which consist of 260, 315 and 331 labelled images of cracks in concrete surfaces such as roads or walls, respectively. The datasets were used to train a separate model for each dataset.

The pre-trained model was used to estimate segmentation maps from the input images, which were more accurate since they had been trained with significantly greater computational resources.

5.1.4 Using segmentation maps to estimate crack depth

Having obtained segmentation maps from the pre-trained model, these maps were used to estimate the depth of a crack. The relationship between the crack length and crack depth serves as a key factor in this estimation process. To estimate the crack depth, a path between the crack tips is first plotted within the segmentation mask generated by the pre-trained model. The path is plotted by collecting all points on the crack border defined by the crack mask. For each of these points, the shortest path is found. Then the crack length is found as the longest of those possible shortest paths. This works as a simple way to get an estimate of the length of the crack from the segmentation mask, where the path represents the approximate trajectory of the crack within the surface.

For a more detailed explanation of how it is computed, given B as the set of pixel coordinates on the boundary between the crack and non-crack pixels, the length estimate is computed as follows:

$$P = \{\{p_i, p_j\} \subseteq B \mid p_i \neq p_j\} \tag{5.1}$$

$$p^* = \arg\max_{p_{i,j} \in P} d_M^*(p_{i,j}) \tag{5.2}$$

Where the shortest path finding function $d^*(p)$ is computed on a pixel coordinate pair $p = \{p_i, p_j\}$ to find the shortest path between the points constrained by the crack mask M for the function $d_{shortest,M}(p_{i,j})$. This is computed for all pixel coordinate pairs from P, which finds the shortest path p^* between the most distant points p_i, p_j corresponding to the crack tips.

By calculating the length of this path, an estimate of the crack length proportional to the scale of the image is obtained. However, to convert this length into real-world measurements, the image scale needs to be calibrated based on the known distance at which the image was taken. To determine the true length using these parameters, 3D projection is used, which was covered in previous sections. This would use the pose and camera parameters to project the 2D path of the crack on the image into the real-world 3D coordinate space, and then the true length is computed from the 3D path.

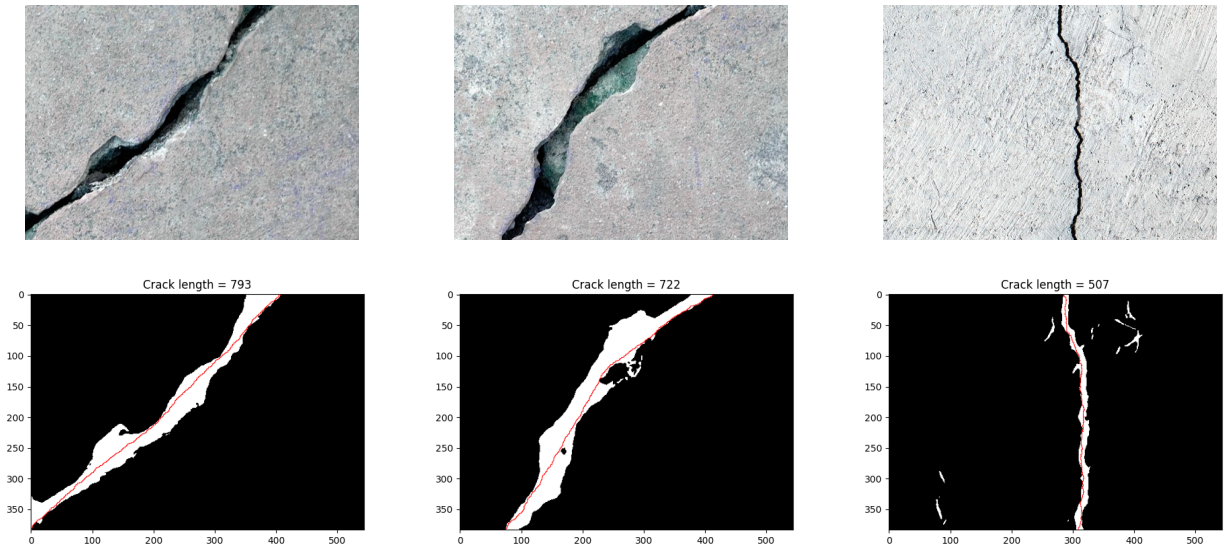

Figure 5.2: The top row shows three examples of RGB input crack images, and the bottom row shows the corresponding estimated semantic maps from the pre-trained network with the crack path in red and the length calculated relative to the pixels. The vertical and horizontal axes are labelled with the pixel scale.

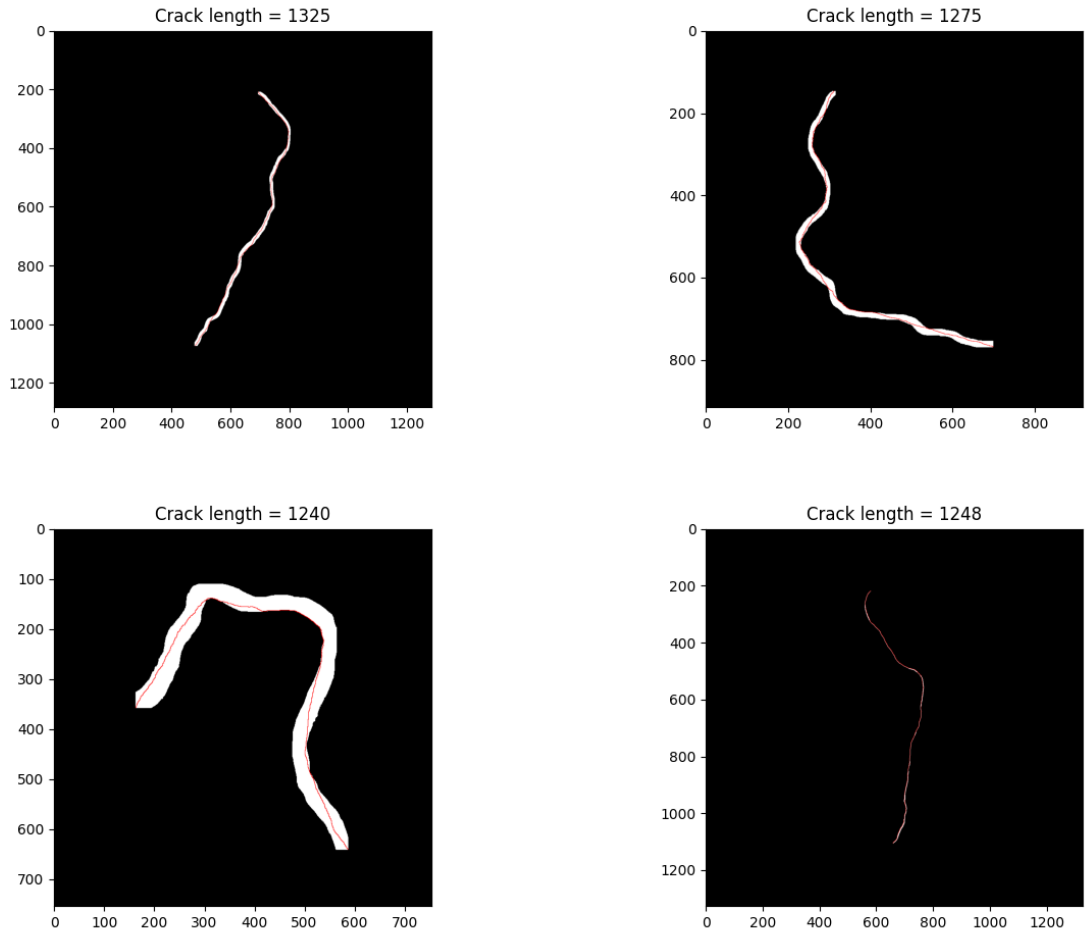

Figure 5.3: Four examples of the synthetically generated segmentation maps for a crack with the path shown as a red line with the length calculated relative to the pixels. The vertical and horizontal axes are labelled with the pixel scale.

To further explore the accuracy of the crack length measurement technique, a method for generating synthetic cracks with random shapes was used, shown in Figure 5.3. This allowed the creation of segmentation maps specifically for testing the measurement of crack length.

5.2 Synthetic data generation experiments

For the generation of the synthetic data, Blender [127] is used to produce realistic RGB image renders of 3D environments that will be close to real-world data. The Blender Sensor Simulation Toolkit (BlenSor) [180] is used to simulate the depth sensors by scanning the constructed 3D scene to produce realistic depth sensor data.

5.2.1 Overview of process

For the process of generating the synthetic data, a similar method to the one used in [126] is followed to construct the crack scene in Blender and then render the RGB images and the ground truth segmentation map. The same scene is then created in BlenSor, where the depth data is simulated and then projected onto the same view as the RGB image.

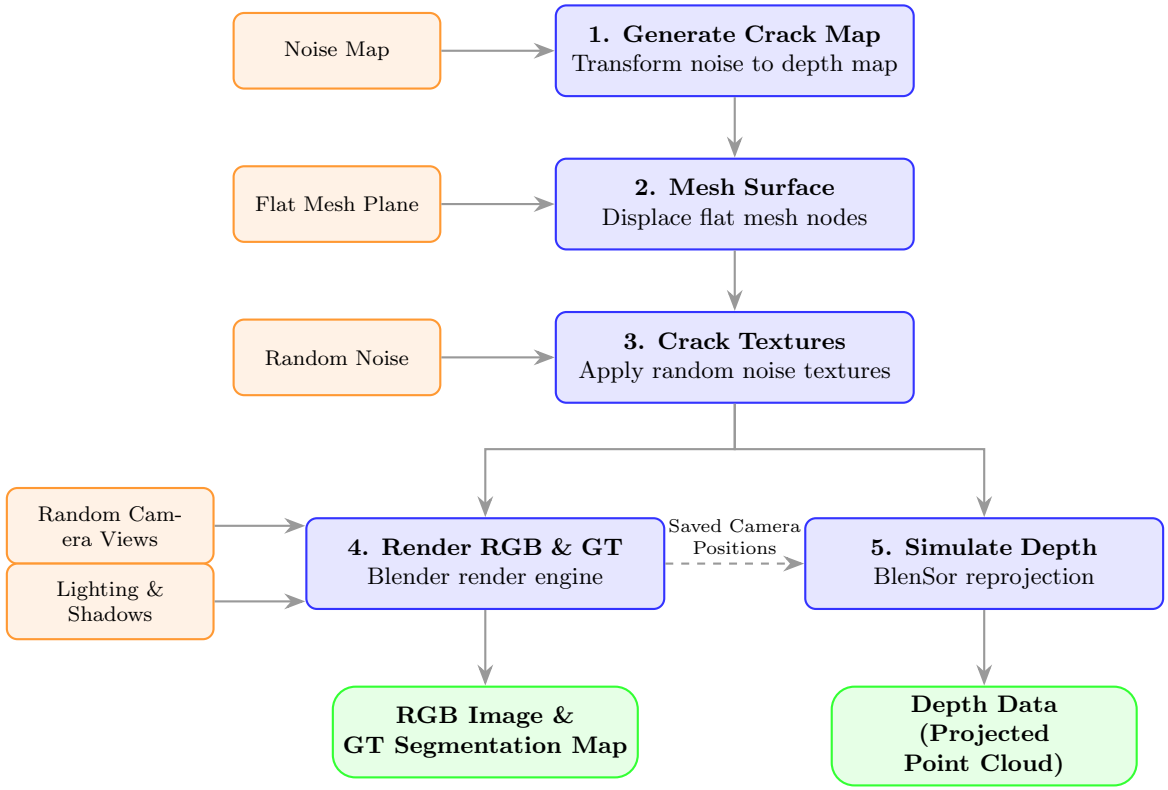

Figure 5.4: Overview of the synthetic data generation pipeline: constructing the 3D scene, applying textures, and branching to generate paired RGB/GT (Blender) and depth data (BlenSor)

As shown in Figure 5.4, to generate the crack scene, the following process is used:

1. Generate Crack map: First, a crack map is generated, which represents the depth of the crack from the surface. This is done by transforming a noise map to get a crack pattern.

2. Mesh surface: The 3D geometry is generated using a mesh surface, which is later used for rendering the scene. A flat mesh plane is initialised, and then the crack map is used to displace the mesh nodes to get the crack geometry.

3. Crack textures: Textures are then generated using random noise and applied to the mesh to make the

segmentation task more challenging.

4. Render RGB images and ground truth: Data diversity is further increased by adding randomised lighting and shadows to mimic real-world conditions, and then rendered from random camera views to get the final RGB image and ground truth segmentation map.

5. Simulate depth data: The same process is then used to generate the same scene in BlenSor using the saved camera positions to determine the view to use when simulating the depth sensor to capture the point cloud and reproject it onto the view of the RGB image.

5.2.2 Crack map generation

The crack map is generated using the method from [126], which results in a 2D image of pixel values that represent the depth of the crack from the surface plane, with a value of 0 for where there is no crack.

The first stage is to produce 2D noise maps that are realistic and natural-looking. To do this, OpenSimplex noise [181] is used to create natural appearing noise textures. OpenSimplex noise was developed as an open algorithm that is an improvement on Perlin noise [182], which suffered from directional artefacts and higher computational cost. This improvement enables OpenSimplex noise to match real-world cracks more closely without the presence of artefacts. The reduced computational cost will allow a larger synthetic dataset to be generated with a fixed amount of compute.

The noise maps produced with OpenSimplex noise are then used to produce a more realistic noise map by implementing Fractal Brownian Motion (fBm) using the noise, which adds the multi-scale irregularity found in natural cracks. This is done by accumulating multiple layers of these OpenSimplex noise maps, referred to as octaves. For each subsequent octave, the intensity is decreased by a multiplier referred to as persistence, and the frequency is increased by a multiplier referred to as lacunarity.

The process is done by first scaling the pixel coordinates for the intended image so they are normalised to be independent of that image resolution, which is done as follows:

$$u_i = \frac{i}{N_x} \cdot f_x, \quad v_j = \frac{j}{N_y} \cdot f_y, \quad \text{for } i = 0, 1, \ldots, N_x - 1 \text{ and } j = 0, 1, \ldots, N_y - 1 \tag{5.3}$$

where f_x, f_y are the frequencies for x and y, N_x, N_y are the dimensions of the noise map.

Next, the noise map is generated using multiple octaves for all points i, j and takes the absolute values as follows:

$$Z_{i,j} = |\sum_{k=0}^{O-1} \left(\mathcal{N}\left(u_i \cdot L^k, v_j \cdot L^k\right) \cdot P^k\right)| \tag{5.4}$$

where $\mathcal{N}(x, y)$ is the simplex noise function, L is the lacunarity, P is the persistence and O is the number of octaves.

Next, a binary mask of the crack is obtained using a threshold on the absolute noise map as follows:

$$M_{i,j} = \begin{cases} 1, & \text{if } Z_{i,j} < \text{threshold} \\ 0, & \text{otherwise} \end{cases} \tag{5.5}$$

Finally, the crack displacement is obtained using the mask and the absolute noise map as follows:

$$D_{i,j} = M_{i,j} \cdot |Z_{i,j} - \text{threshold}| \tag{5.6}$$

The noise map is transformed into the crack map by first taking absolute values, which will result in a valley-like pattern. The values then need to be offset by the threshold and then flipped by taking the absolute values again, to now get a mountain range-like pattern with the peaks being equal to the threshold. Then, anything less than 0 will be outside the crack and is set to 0 using a binary mask. The generated crack displacement map represents the depth of the crack at each point, which is used for the geometry of the crack, and the binary crack mask is used for the ground truth segmentation map. An example of this code can be seen in the linked Codebase [130].

5.2.3 Mesh generation

The 3D structure of the crack scene is represented using a mesh, specifically a quad mesh, which is a collection of vertices, each with a 3D coordinate position. Each vertex has 4 edges connected to its 4 neighbouring vertices to form a grid-like pattern. This mesh is initially a flat 2D plane; then, using the crack map, the mesh can be displaced using the corresponding pixel values to determine how much to displace the vertices in the plane's normal direction.

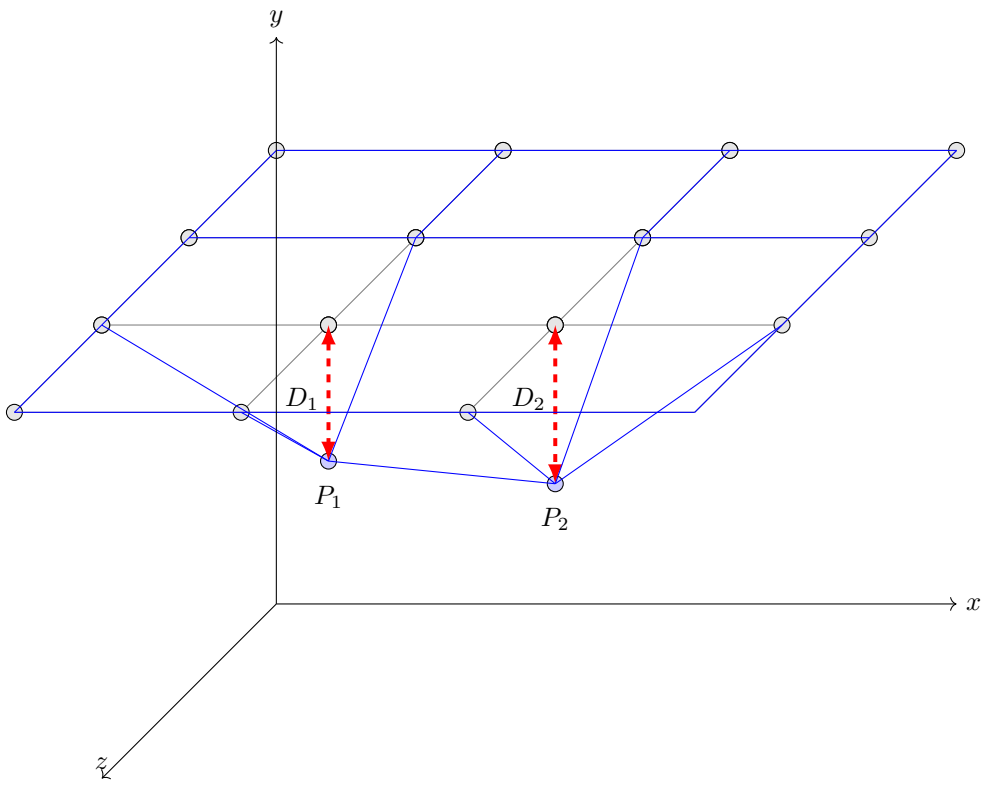

Figure 5.5: This shows how vertex points P_1, P_2 would be displaced by displacement values D_1, D_2 in the y-axis to get the displaced mesh that represents the crack geometry

5.2.4 Texture generation

For rendering the mesh, a material is needed to inform how the light will interact with the surface when the light paths are simulated. To create this material, different textures are created, which result in different rendering interactions with the surface. A colour map texture is used to control the amount and wavelength of the light reflected, resulting in a specific colour being rendered. A roughness map is used to control the simulated effect of small surface bumps on the light reflection, which adds further detail to the synthetic data. A normal map is used to control what the surface-normal direction should be when simulating the light reflection without needing to adjust the mesh, allowing for even finer detail to be rendered.

These texture maps are procedurally generated to ensure that the model cannot overfit to a specific texture, making it unable to generalise for unseen textures. The textures are generated with a combination of noise maps to add complexity to the detection task, which will improve the ability of models trained on it to generalise to real-world scenes.

To generate the ground truth segmentation map, a separate material has to be generated that will inform the render simulation to render white in locations inside the crack and black where it is outside the crack. This can be done without simulating the light paths and instead using simple rasterised rendering, which improves the ground truth accuracy as well as being faster to render. The crack segmentation texture just uses the crack mask that was computed in the process of creating the crack map. The generation of an accurate segmentation map ground truth is essential for the training of models on this dataset.

5.2.5 Scene lighting and rendering

To complete the render process, a light source is needed to generate the light rays that will be simulated and interact with the material mesh and will reach the camera view to be rendered as an RGB image. The light sources are randomised with their position and intensity to provide further diversity to the dataset. Additionally, the light source is randomly blocked with a randomly generated texture mask that will obscure some of the light paths, which allows models trained on this data to perform better in real-world scenarios where shadows are present. Finally, the camera positions are randomised to view the crack from different angles.

The depth data is simulated using BlenSor to simulate a Time-of-flight (ToF) sensor. The mesh is constructed in the BlenSor software using the same process. Then, the camera positions are used to determine the position for the simulated ToF sensor, which is positioned slightly offset from the RGB camera view to simulate real-world setups where the RGB and ToF sensors are side-by-side. The BlenSor software will generate a 3D point cloud, which is a collection of 3D coordinates where the sensor has scanned the surface. These points are then projected onto the RGB image using the projection matrices, using the camera intrinsics and camera extrinsics to determine where the points will land on the image pixel space.

This is done as follows:

$$P_i^W = [X_i^W, Y_i^W, Z_i^W]^T, \text{ for all } P_i^W \text{ in } \{P_1^W, P_2^W, ..., P_n^W\} \tag{5.7}$$

where P_i^W is a point in the point cloud with n total points, having world coordinates X_i^W, Y_i^W, Z_i^W in the world space W.

Next, the depth is found by transforming to the camera coordinates P_i^C in the camera space C as follows:

$$P_i^C = \begin{bmatrix} X_i^C \\ Y_i^C \\ Z_i^C \end{bmatrix} = [R|t] \begin{bmatrix} X_i^W \\ Y_i^W \\ Z_i^W \\ 1 \end{bmatrix} \tag{5.8a}$$

$$\text{depth}_i = Z_i^C \tag{5.8b}$$

where $[R|t]$ is the 3×4 extrinsic matrix containing the 3×3 rotation matrix R and the 3×1 translation matrix t. This is used to multiply the 4×1 homogeneous world coordinates $[X_i^W, Y_i^W, Z_i^W, 1]^T$. The depth is then found as just the Z^C coordinate in the camera space.

The pixel coordinates are found as follows:

$$K = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{5.9a}$$

$$\begin{bmatrix} u_i' \\ v_i' \\ w_i \end{bmatrix} = K \begin{bmatrix} X_i^C \\ Y_i^C \\ Z_i^C \end{bmatrix} \tag{5.9b}$$

$$u_i' = \frac{u_i}{w_i}, \quad v_i' = \frac{v_i}{w_i} \tag{5.9c}$$

where K is the intrinsic matrix consisting of f_x, f_y (the focal lengths in the x and y directions), s (the skew coefficient), and c_x, c_y (the principal points for the x and y coordinates).

This gives the pixel coordinates on the depth map u_i', v_i' (rounded to the nearest integer) and the depth_i. This results in a sparse depth map, where only a percentage of the pixels in the depth map have depth values and the rest have no values, which can be represented as -1. If multiple depth values exist for the same pixel, the smallest is chosen so that the depth map represents the distance to the closest surface.

This generates a full data sample with the RGB and sparse depth map inputs as well as the ground truth segmen-

tation map. This can be repeated to generate completely new, randomly generated data samples to construct a large dataset.

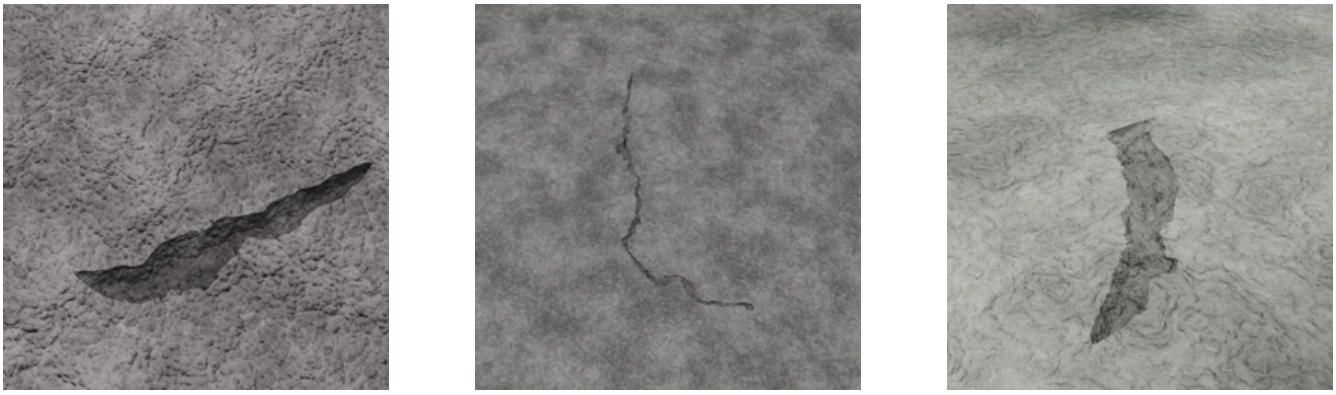

Figure 5.6: The rendered RGB synthetic data images.

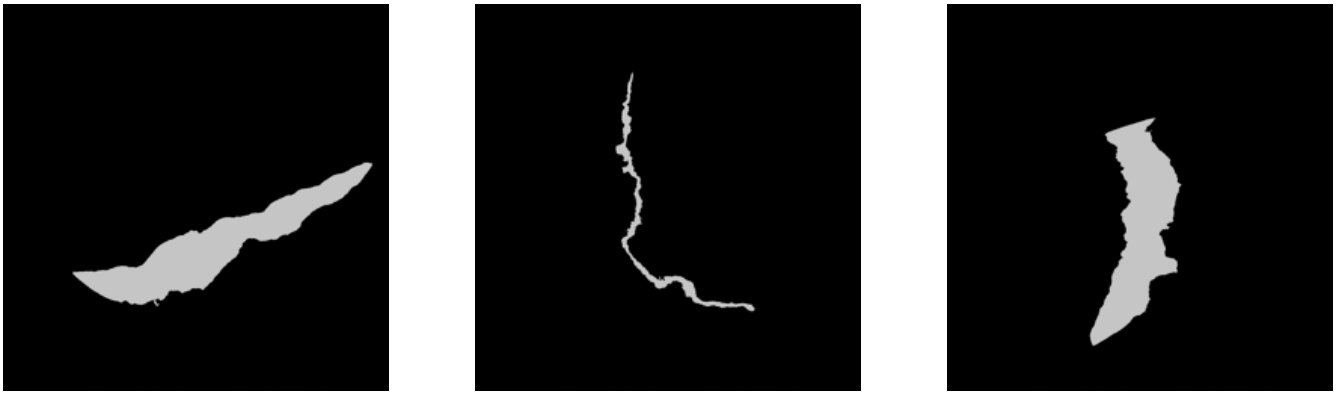

Figure 5.7: The ground truth synthetic data segmentation maps.

5.3 Segmentation methods with data fusion using a deep neural network

The synthetic data was then used to train a state-of-the-art neural network architecture to predict the segmentation map. The DeeplabV3 architecture [177] was chosen since it is easily available in the deep learning framework PyTorch, as well as having state-of-the-art performance in segmentation tasks. The model was then trained on the synthetic data as well as some real-world data, and then evaluated on some real-world data it did not see in training.

The ResNet-50 backbone is chosen as it is shown to have similar performance to the larger ResNet-101 backbone

while using significantly fewer computational resources. In Section 5.1.2, ResNet-101 backbone was used instead due to the higher complexity and diversity of the crackseg9k dataset described in Section 5.1.1. However, this synthetic dataset is simpler as it does not cover as many types of cracks and materials, so the smaller ResNet-50 backbone is used to reduce the amount of computational resources needed to train the model. The pretrained DeeplabV3 model is not used to avoid performance instability due to the high-level natural images it is trained to segment, compared to the low-level characteristics of the crack images.

5.3.1 DeeplabV3

The DeeplabV3 model is a deep convolutional neural network that improves on the DeepLab architecture, using Atrous Convolutions to capture information from features at multiple scales, which allows it to segment effectively at low-level image scale as well as high-level image scale. The network architecture is an encoder-decoder based structure, with a backbone encoder to extract relevant features using a state-of-the-art image recognition model such as ResNet [157] and a DeepLab head decoder that will take the features at multiple scales to output the segmentation map.

ResNet-50

ResNet-50 is a deep convolutional neural network developed for image classification tasks, which uses residual connections to allow the inputs to skip layers, allowing for the training of deeper architectures.

The architecture of ResNet-50 includes the following components:

- Initial Convolutional Layer: The network begins with a convolutional layer that will increase the input dimensionality from the 3-channel input to output 64 channels using 64 convolutional filters. This first layer also uses batch normalisation, a ReLu activation and a max pooling operation.
- Residual Blocks: The next 48 layers are from 16 residual blocks called bottleneck blocks, where each block is composed of three convolutional layers with a skip connection to allow the next layer to get information from the previous layer.

  The structure of the 3 layers within each block is as follows:

  - The first layer uses 1×1 filters to reduce the dimensionality.

- The second layer uses 3×3 filters to process the features at a lower dimensionality.
- The third layer uses 1×1 filters to restore the dimensionality.

This bottleneck design allows the network to reduce the number of channels for the 3×3 convolutional layer, reducing the number of filters, which will reduce the network size with a smaller number of parameters. This will decrease the computational cost of the forward pass. The convolutional layers are all followed by a batch normalisation layer, and the ReLu activation function is used at the end of the bottleneck block. As the network deepens, the spatial dimensions of the feature maps are reduced by using a stride of 2 in some of the second convolutional layers of the bottleneck blocks to downsample the features by a factor of 2, as well as increasing the number of channels by a factor of 2.

- Final Layer: The final 50th layer of ResNet-50 is a fully connected layer. However, for DeepLabV3, this layer is ignored as it just takes the features with spatial information from the final convolution layer.

The ResNet-50 model has also been modified to limit the downsampling to only a reduction in the spatial dimensions by a factor of 8, as well as removing the final fully connected layer. This is done to limit the loss of low-level detail and allow for more accurate low-level segmentation.

DeepLabv3 Head

Next, DeepLabV3 takes the features output at the final convolution layer from the backbone and uses the next model, which is the DeepLabV3 classifier head, to predict the segmentation map at the same resolution as the input.

The forward process is as follows:

- Atrous Spatial Pyramid Pooling (ASPP): The first layer is split into 5 branches that utilise convolutional layers with 4 different dilation rates to process information ranging from fine low-level details to larger high-level details. The 5th branch uses adaptive average pooling to process global information from the feature maps.
- Further Convolutional layers: The outputs from the ASPP branches are concatenated into a single tensor, and then a 1×1 convolution layer is applied to process the features from different scales. Then, two more

convolutional layers are used to generate the final output.

- Upsampling: The outputted segmentation map is still at the scaled-down resolution of the backbone feature maps, so to get a segmentation map with the same spatial dimensions as the input image, bilinear upsampling is used.

5.3.2 Training

To train the DeeplabV3 using RGBD data, the first layer is simply modified to have 4 channels, and the RGB and depth images are concatenated to get a 4-channel image. The diversity of training data is further increased by using data augmentation, colour jitter, flipping, rotation, resizing and cropping. Colour jitter is applied only to the RGB image, and a brightness and contrast transform is applied to the depth image.

Loss function

The model is trained using the binary cross-entropy loss function, which is defined as follows:

$$BCE(x, y) = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(x_i) + (1 - y_i)\log(1 - x_i)] \tag{5.10}$$

where N is the batch size, x is the predicted output, and y is the ground truth labels, which are either 0 or 1. This loss function requires the estimations to be between 0 and 1, since when x = 0 or x = 1, it would be undefined, so the Sigmoid function was used to ensure estimations between 0 and 1. The Sigmoid function is defined as follows:

$$\text{Sigmoid}(x) = \frac{1}{1 + e^{-x}} \tag{5.11}$$

A batch size of 16 was used to train the model for 30 epochs using the Adam optimiser with a learning rate of 1×10^{-4}.

Training data

The synthetic data generation process was used to generate 10,000 total data samples, which were then split into 8,000 samples for training and 2,000 for validation. The model was trained for 50 epochs on an NVIDIA RTX 2070 GPU.

A real-world dataset [183] was also used, which consisted of road scenes featuring both cracks and potholes. The synthetic generation method presented in this research is limited to only generating cracks due to the added complexity of potholes, so only the data samples containing cracks are used for evaluation.

These scenes in the dataset were captured as videos using the Luxonis OAK-D camera, which records RGB videos at a resolution of 1920×1080, and the depth information is captured as a disparity map at a resolution of 640×400. Both are recorded at a frame rate of 15 frames per second. The disparity map is obtained by the camera using stereo matching, and then some simple filtering is used to obtain disparity maps with some invalid pixels. The real-world data contains ground truth masks, which were obtained using a semi-automatic labelling process. This process was assisted by disparity maps and traditional computer vision algorithms. The dataset is split into a training and test dataset to get 1,527 samples extracted from 66 videos for training and 721 samples extracted from 27 videos for testing.

However, with the provided test-train split, there was some data contamination due to overlapping scenes where the same crack scene had been captured from a different position. To ensure there was no data contamination, SIFT keypoints [184] were computed to compare all the scenes and then group the videos so that there was no overlap between groups. Then, a combination of groups was found to get a data split that resulted in 1,124 training samples and 1,124 test samples.

To ensure consistency with real-world data and synthetic data, the disparity maps are converted into depth maps and normalised to be between 0 and 1 for both the synthetic and real-world data.

To convert disparity values to depth, the provided camera parameters are used for the focal length f and the horizontal distance between the stereo cameras, referred to as the baseline b. These are used to compute the depth values Z as follows:

$$Z = \frac{f \cdot B}{d} \tag{5.12}$$

The real-world data has different resolutions for the disparity and RGB images, so the disparity map is resized and padded to make sure each pixel is aligned. The disparity map contains some invalid pixels from the filtering set to -1, so nearest neighbour interpolation is used to avoid affecting neighbouring pixels. For training, the synthetic and real-world training datasets are combined. The smaller real-world dataset is repeated N times, such that the

real-world dataset is the largest size that is still smaller than the synthetic dataset size.

5.4 Results, discussions, conclusions

5.4.1 Results

To evaluate the segmentation map predicted by the model, the metrics Precision, Recall, Accuracy, F1-score, and Intersection over Union (IoU) were used. The model produces a probability map where each pixel has a value between 0 and 1, estimating the probability that the pixel is a crack. To evaluate the segmentation maps, a binary segmentation map is needed, which is obtained by using some threshold between 0 and 1 as follows:

$$S(x,y)\begin{cases}1, & \text{if } P(x,y) \geq Threshold \\ 0, & \text{otherwise}\end{cases} \tag{5.13}$$

where $S(x,y)$ is the binary segmentation map at pixel (x,y), $P(x,y)$ is the probability map outputted by the model. Depending on the threshold chosen, this can affect some of the metrics. A higher threshold results in fewer positive estimations, leading to higher precision because it only predicts "true" for the pixels where it has the highest confidence. However, this trades off recall, as it predicts "false" for more pixels and misses more pixels that are "true" according to the ground truth. Conversely, a lower threshold would result in higher recall and lower precision. The F1-score balances both of these measures, making it useful for determining the best threshold to use.

To compute the metrics, first the confusion matrix is calculated to get the 4 following four values computed over the entire test dataset:

- **True Positives (TP)**: The number of pixels correctly classified as positive, so the model predicted true and the ground truth was true.
- **False Positives (FP)**: The number of pixels incorrectly classified as positive, so the model predicted true, and the ground truth was false.
- **True Negatives (TN)**: The number of pixels correctly classified as negative, so the model predicted false

and the ground truth was false.

- **False Negatives (FN)**: The number of pixels incorrectly classified as negative, so the model predicted false and the ground truth was true.

Using these values, the metrics can be computed as follows:

- Accuracy: This measures the overall percentage of correct estimations, taking into account both positive and negative classifications. The formula for accuracy is:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{TN} + \text{FN}} \tag{5.14}$$

- Precision: This is the accuracy of just the positive estimations made by the model. It is calculated using the formula:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{5.15}$$

- Recall(also known as sensitivity): This is the accuracy of estimations on just the pixels that were true according to the ground truth. It is calculated as:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{5.16}$$

- F1-score: This is the harmonic mean of precision and recall, providing a single measure that represents both measures. It is calculated as:

$$\text{F1-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{5.17}$$

- Intersection over Union (IoU): This metric evaluates the overlap between the predicted and ground truth masks. It is calculated as:

$$\text{IoU} = \frac{\text{TP}}{\text{TP} + \text{FP} + \text{FN}} \tag{5.18}$$

The accuracy measurement is affected by the imbalance between the number of crack and non-crack pixels. It would show a higher-than-average accuracy if the model predicted all pixels as non-crack, given the larger proportion of non-crack pixels compared to crack pixels. In contrast, the IoU metric is less affected by this imbalance because it specifically considers the overlap between predicted crack pixels and ground truth crack pixels. Therefore, the better metrics to evaluate the model are the F1-score and the IoU.

The presented model is then compared with other models from the SHREC contest [185], which were developed by two teams as part of a contest on pothole and crack detection in roads and pavements. The competition involved experimenting with different architectures and approaches to achieve the best performance on the provided dataset. Each team was given a training set composed of both semantic segmentation image/mask pairs and an additional RGB-D dataset.

Models submitted by Team 1: Team 1 use a loss function based on active contour theory to train the deep-learning architectures MANet [186], UNet [79], and UNet++ [187]. These three architectures were trained with the same approach to produce the three separate models.

Models submitted by Team 2: Team 2 use a variety of approaches to train the following models:

- **SegFormer**: A pretrained SegFormer [188] model was fine tuned on the data.
- **DeepLabV3+**: This was trained using the DeepLabV3+ [189] architecture and used pretrained EfficientNet [190] as the backbone.
- **MaskedSoftCPSDLUnet-Unet++**: This model was trained using the UNet++[187] architecture and uses an approach that improves on Cross Pseudo Supervision (CPS) [191], and it can use both the annotated and non-annotated data for training.
- **MaskedSoftCPSDLUnet-Deeplabv3+**: This model was trained simultaneously with the previous model as a part of the training approach and uses DeepLabV3+ [189] as the model architecture.

The models MaskedSoftCPSDLUnet-Unet++ and MaskedSoftCPSDLUnet-Deeplabv3+ were developed to be used as an ensemble. However, implementing the full method was challenging, so the models were evaluated separately.

Baseline Model

- **Baseline-DeepLabV3+**: This model, developed by the contest organisers, utilises the DeepLabV3+[189] architecture with a ResNet-101 encoder [157].

Figure 5.8: The RGB images in the real dataset.

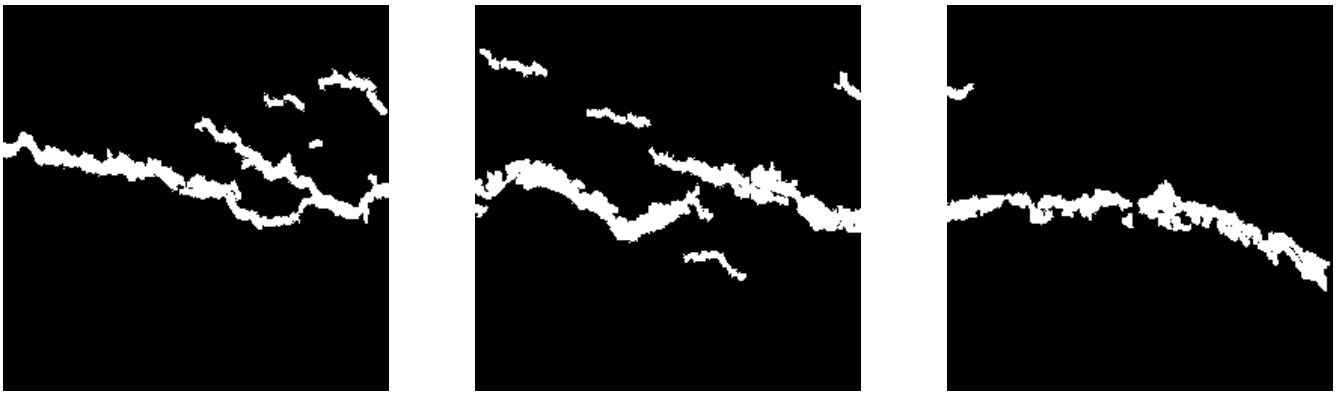

Figure 5.9: The ground truth segmentation maps.

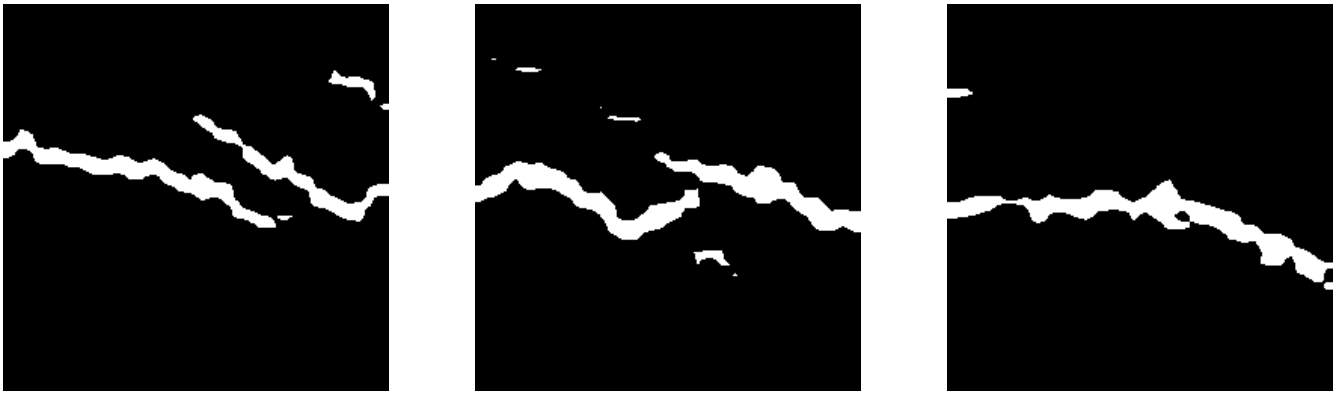

Figure 5.10: The predicted segmentation maps by the presented model.

Model name	Accuracy	IoU	Precision	Recall	F1-score
MAnet	0.915	0.362	0.484	0.59	0.532
UNet	0.937	0.475	0.598	0.697	0.644
UNet++	0.907	0.319	0.441	0.536	0.484
SegFormer	0.936	0.407	0.618	0.544	0.579
DeepLabV3+	0.933	0.469	0.569	0.728	0.638
MaskedSoftCPSDLUnet-Unet++	0.929	0.401	0.561	0.584	0.573
MaskedSoftCPSDLUnet-deeplabV3+	0.924	0.423	0.527	0.682	0.594
Baseline-DeepLabV3+	0.905	0.351	0.442	0.63	0.519
Presented Model RGB-D	**0.96**	**0.605**	**0.758**	**0.75**	**0.754**
Presented Model RGB	0.951	0.534	0.695	0.697	0.696

Table 5.1: The models compared by accuracy, IoU, precision, recall and F1-score. The best values for each column are shown in bold.

Table 5.1 shows the models with the best result for each metric in bold. For each of the metrics, the "Presented Model RGB-D" has the highest performance, followed by the "Presented Model RGB" as the second highest performance. Compared to the existing models used for comparison, the "Presented Model RGB-D" had an improvement of 2.45% for accuracy, 27.37% for IoU, 22.65% for precision, 7.6% for recall and 17.08% for F1-Score, compared to the next best model, which was UNet for accuracy, UNet for IoU, SegFormer for precision, UNet for recall and UNet for F1-Score.

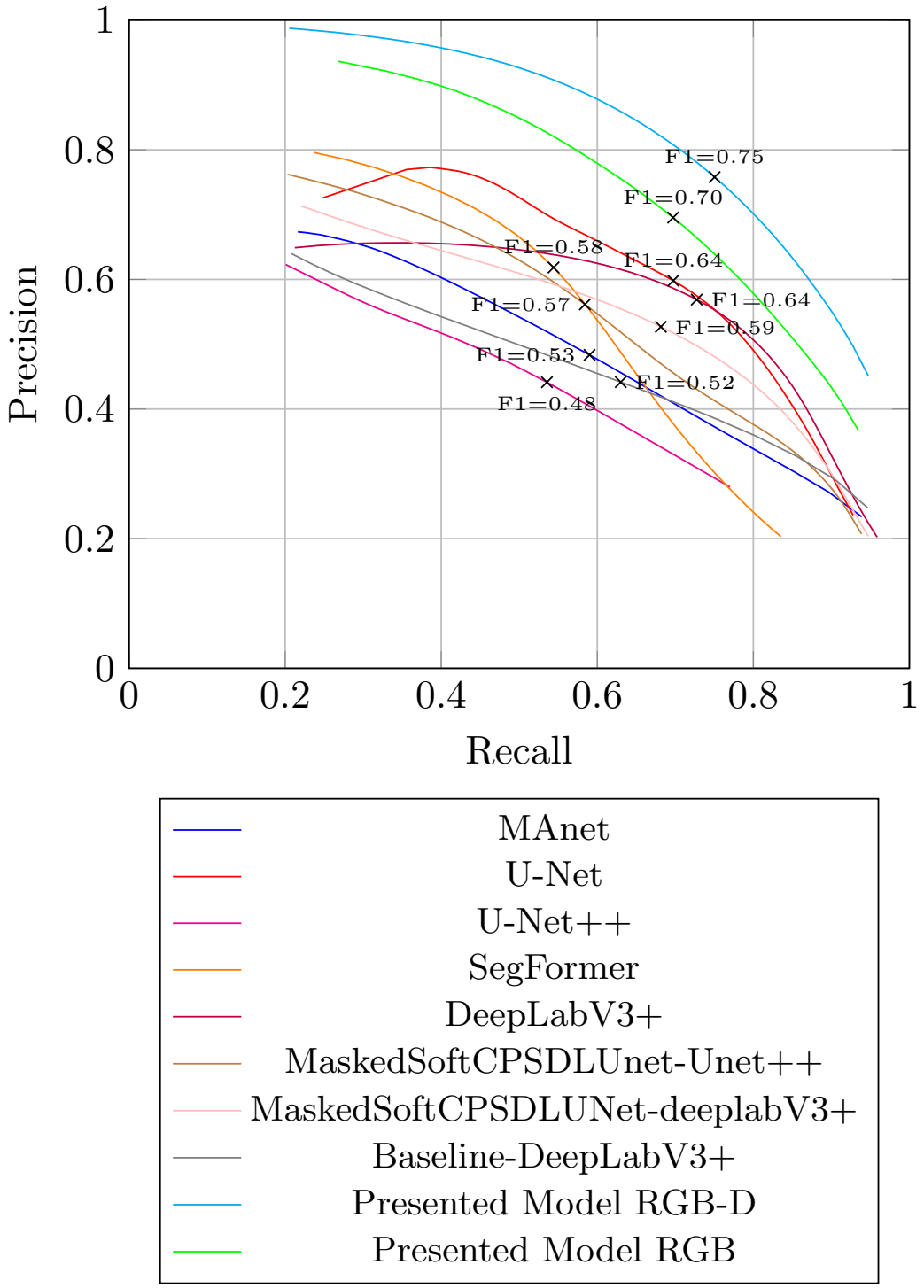

Figure 5.11: The precision-recall curves for all the models on the test dataset with the F1-scores marked on the graph. The F1-scores are also displayed in Table 5.1

Figure 5.11 shows the precision-recall curves generated by applying different thresholds to the probability map output by the model, with thresholds ranging from 0.0 to 1.0 in increments of 0.01. These thresholds were used to create binary segmentation maps, and the best threshold was selected as the one with the highest F1 score.

The graph shows the presented model trained with just the synthetic data and tested on RGB only as "Presented Model RGB" and tested with a combined dataset of the synthetic and real data using the full RGB-D data as "Presented Model RGB-D". The "Presented Model RGB-D", trained with the combined dataset shown in light blue, outperforms all the other methods, achieving the highest F1-score of 0.754, which is a 17.08% improvement over the UNet model, which had an F1-score of 0.644, which was the highest out of the models used for comparison. The "Presented Model RGB" trained with only the synthetic RGB data shown in light green also outperforms all the other methods and achieves an F1-score of 0.696, which is an 8.07% improvement over the UNet model.

The models here relate to the previous chapters by using both RGB images and depth images as the inputs. This connects to the work done in Chapter 3, where methods that improve RGB image quality with data fusion were covered, and then Chapter 4 covered methods to fuse depth maps to reduce error. Together, this builds a full data fusion pipeline for crack detection where the models covered in this chapter can take the outputs of the models covered in the previous chapter to compute the detection of the crack shape.

5.4.2 Discussions

The method implemented in the research for crack segmentation using RGB-D data and synthetic data generation significantly outperformed the other state-of-the-art methods when evaluated by IoU and F1 scores. The use of depth maps, combined with RGB data, allowed for more accurate pixel-level segmentation by providing additional information. The use of synthetic data was also demonstrated to be effective at accurately modelling real-world data, as was shown in the results from the "Presented Model RGB", which only used the synthetic data for training and outperformed the other models. The improved performance is due to the larger dataset size for training using synthetic data compared to the other methods, which were limited by smaller datasets of only real-world cracks. The depth information combined with the RGB image gives more information for the model to use, which helps it in image regions where it is hard to determine if a pixel is a crack from the RGB image alone.

The models used for comparison with the presented models in this research were provided with the trained parameters, which were trained on the real-world dataset, and none of them used the depth map data. This gives us a direct comparison as to the improvement that both the synthetic and depth data contributed to when achieving higher performance on the segmentation metrics. However, the research did not cover retraining the compared models on the synthetic or depth data to determine how it would affect the different architectures and training

methods to get a like-for-like comparison on those models, as they each used complex training methods and network architectures, which require significant work to implement.

The results from this method covered the analysis of cracks by estimating a segmentation mask, which gives information about the shape of a crack. This can then be analysed by estimating the length of the crack described in Section 5.1.4. This length can then be used to determine which cracks are significant, as longer cracks are often deeper and more severe.

The application areas for this are in infrastructure maintenance, where efficient and accurate detection of road and pavement cracks or other structures, such as bridges, is essential for timely repairs and cost savings. The segmentation maps produced can also provide further analysis of the cracks to determine the severity of a crack and prioritise where action should be taken.

Future work could explore optimising data fusion techniques within the neural network to better use depth information with improved network architectures that have been designed for RGB-D segmentation. Additionally, the synthetic data generation process could be improved by increasing the data diversity with the modelling of more complex surface geometry and more accurate materials, also using different modalities for synthetic depth maps in addition to the ToF, such as LiDAR or Stereo depth maps. This would make the model trained from this data more generalisable to variations in depth sensors with real-world RGB-D data.

5.4.3 Conclusion

A novel approach to crack segmentation is used, which has depth maps and RGB images with synthetic data used to train a state-of-the-art deep learning architecture, DeeplabV3, which is modified to take the depth maps as inputs. The implemented method is compared with state-of-the-art crack segmentation methods' results on a real-world dataset of road cracks and outperformed the other methods, achieving a 17.08% increase in F1-Score compared to the next best model.

Chapter 6

Conclusions and Future Work

The aim of this research was to develop a method for detecting flaws, which were chosen to be cracks in road surfaces. Methods for multimodal imaging were assessed, and image fusion methods were worked on, where improvements were made on existing methods. These methods would just improve the quality of RGB images but not create any new analysis, so they would be used for capturing information representing the object that could contain a flaw, and then analysis methods could be applied. Then this research was followed by looking at other modalities than mono RGB with stereo RGB and LiDAR depth maps, and developed a method for combining state-of-the-art methods for stereo and LiDAR to output a more accurate depth map than either of the input methods. Then the research looked at the final stage to detect the flaw, focusing on cracks, and developed a novel method that could segment cracks in road surfaces from the RGB image and depth maps. A synthetic dataset was generated to increase training data due to limited real-world data availability. The generated data was required to be diverse and represent the real-world data accurately, so various techniques were utilised, such as random variations in lighting and shadows. Then the research demonstrated how these segmentation maps would be used to determine the severity of cracks by measuring the crack length.

6.1 Image fusion

RGB sensors with image fusion methods were investigated that would fuse multiple 2D images captured of the same scene into a single image of higher visual quality. Existing image fusion methods were implemented to later look at

modifying the methods to find improvements.

One of the implemented methods was "Image Fusion with Guided Filtering" [11], which used guided filtering [129] to identify high-quality information in the images for fusion. Another approach, "Infrared and Visible Image Fusion using a Deep Learning Framework" [12], used a pre-trained deep learning model (VGG-19) [142] to generate feature maps. These feature maps were then used to compute weight maps, assigning more weight to higher-quality information within the images. These methods were then modified to make novel contributions.

To evaluate the effectiveness of the implementations of these methods, they were tested on datasets containing multiple images of the same scene captured under different variations, such as exposure time [138], focal length [131], and wavelength [139]. Using the implementations of the methods, the existing bi-level fusion approach was modified by introducing a three-level fusion technique, which involved splitting each image into one base layer and two detail layers.

To provide an objective evaluation, visual quality metrics were implemented, which utilised three input images: the first two were the source images used by the fusion method, and the third was the resulting fused image. These metrics included the Q_Y metric [132], which used the structural similarity index measure (SSIM) [133] to compute a score, the Q_C metric [134], which utilised the universal image quality index (UIQI) [135], the Q_G metric [136], which measured edge information preservation, and the Q_{MI} metric, which utilised mutual information [137]. By comparing the scores obtained from the original method using bi-level fusion and the proposed novel modified method using three-level fusion, it was consistently observed that the proposed method achieved higher quality scores across all metrics. The improvement shown in all the metrics implemented on all of the datasets tested clearly demonstrates the superiority of the modified method compared to the original.

The research done on methods for 3D reconstruction helped inform this research on the relevant 3D methods for fusion, since having an accurate 3D representation of objects that could contain a flaw will be necessary for accurate detection. Techniques for computing 3D representations from multiple data sources were explored. These sources included RGB cameras for stereo depth map computation [192, 22, 193, 194], as well as RGB and ToF (time of flight) [18] sensors capable of capturing videos [31, 30, 34, 36, 33, 195, 196, 40, 41] that could be used to reconstruct a 3D mesh of an object. Various 3D representations, such as voxels, surfels, and signed distance fields [13], were also investigated as they could be converted into meshes [9].

The research specifically focused on fusing depth maps as the 3D representation with ToF and stereo RGB data obtained from a setup composed of one ToF sensor and two RGB cameras. This approach connects to the methods implemented for RGB image fusion since it uses the same image array data format for its inputs. A dataset was essential to be able to research this area, so a synthetic dataset was used [128], consisting of two depth maps: one computed from simulated stereo images and the other from a simulated ToF sensor, both projected onto the same view as the stereo depth map. The synthetic dataset was generated by rendering 3D scenes in Blender, ensuring accurate ground truth information derived from the 3D environment.

Initially, the image fusion methods that were previously used on ToF and RGB data were evaluated. Both the guided filtering method and the deep learning fusion method were tested, with the evaluation done using the mean absolute error (MAE) between the depth maps and the ground truth. However, the results showed that neither of these methods demonstrated an increase in accuracy when applied to 3D data fusion. So a suitable method to build from with "Stereo and ToF Data Fusion by Learning from Synthetic Data" [128] was implemented. This method was used since it demonstrated a successful approach using 2 image modalities and fusing them to get a more accurate image output, which was the task this research was focused on. This method used a convolutional neural network (CNN) trained directly to generate fusion weight maps using synthetic data paired with ground truth information. The results showed improved accuracy after fusion compared to the MAE obtained from either input depth map.

Then, the implemented CNN method was modified by using filtering methods, with "Block Match 3D (BM3D)" [144] and "Contrast Limited Adaptive Histogram Equalisation (CLAHE)" [143]. The evaluation of these modifications showed that the BM3D method had the greatest improvement in accuracy, evaluated by the MAE. These methods were evaluated by testing both the input depth maps and the output fused depth map to determine the best way to apply the filtering methods, finding that applying them to the fused depth map after fusion achieved the best results. The results for the novel modified method showed a significant improvement, evidenced by the reduction in error. This was evaluated as the original method implemented compared with the presented novel modified method, which incorporated filtering. The reduction in error was from using BM3D filtering on the depth map after fusion. This resulted in improvements in both the mean squared error (MSE) and mean absolute error (MAE). This improvement is due to the filtering methods reducing the noise in the measured values.

6.2 RGB stereo and LiDAR

The work on RGB stereo and LiDAR gave a comparison of the state-of-the-art methods using the KITTI Depth Completion dataset. This dataset is intended for the task of depth completion; however, it can also be used for fusion methods by also using the 2 RGB stereo images as inputs. This dataset is used to see the accuracy and performance of different fusion methods compared to methods specialised for one modality. A method was also developed that used these state-of-the-art methods for one modality and combined the outputs to get a more accurate resulting depth map.

The comparison of the state-of-the-art methods for RGB stereo and LiDAR data showed that the methods that just focus on one modality, with either RGB stereo methods or LiDAR depth completion methods, have higher accuracy compared to the RGB stereo and LiDAR fusion methods due to the single modality methods being more widely researched. The presented model in this research addressed this by allowing existing state-of-the-art methods to be fused by combining the output estimations.

While methods like VPN, CCVN, LiDARStereoNet, and LEAStereo performed well on the KITTI dataset, they were limited by high memory usage. These methods, which rely on feature volumes, are more suitable for datasets with lower resolutions. On the other hand, methods such as RAFT-Stereo and CREStereo exhibited computational and memory efficiency due to their recurrent architectures. This makes them more viable for datasets with higher resolution inputs, as they can handle the increased memory cost more efficiently.

Methods like PENet, SemAttNet, and DySPN, which rely solely on single RGB images, offer an advantage in terms of memory cost. These methods can efficiently compute depth maps for higher-resolution inputs without compromising on accuracy. This makes them particularly suitable for environments lacking texture, such as indoor settings. However, due to the lack of stereo RGB images, it misses out on being able to estimate denser depth values for highly textured areas since it relies on the sparse LiDAR depths to interpolate the missing values.

Feature matching-based approaches, such as LEAStereo, perform well in textured environments, making them suitable for autonomous driving scenarios with diverse visual features. Whereas LiDAR-based methods, such as PENet, SemAttNet, DySPN, VPN, CCVN, and LiDARStereoNet, performed better in texture-less areas where visual information alone may be insufficient.

For the presented method, the objective evaluation was done using mean absolute error (MAE), root mean squared error (RMSE), and pixel threshold error metrics. These metrics give a comparison of the performance of estimated disparity maps across different scenes. The results consistently showed that the resulting fused output of the data-fusion method, which was the output disparity map, outperformed the inputs to the data-fusion method, which were the input disparity maps. This improvement is evidenced by the reduction in the MAE metric, where the resulting output of the method had a reduced error with the ground truth compared to the input. This demonstrated that the fusion methods improved the performance of the depth estimation task by reducing errors.

The fusion method using max confidence values showed lower errors compared to the method that weighted the disparity values. This shows that using the most confident values during fusion can lead to improved accuracy over always using a weighted combination of both. However, LEAStereo generally achieved the lowest errors, except for the 1-pixel threshold error. This showed that LEAStereo performs better than the fusion methods at reducing the number of outliers that have a significant error, but was not as good with the larger number of inaccurate estimations that are closer to the true value.

A visual evaluation was performed through pairwise comparisons, which showed that LEAStereo was better in terms of visual quality. So, to improve the fused depth map accuracy, the input LiDAR depth map accuracy needed to be improved. To enhance the accuracy of LiDAR depth maps, the PENet method was used before fusion with the LEAStereo disparity maps. This approach resulted in improved accuracy, with a lower MAE in the fused depth maps. This shows that using depth completion methods like PENet contributes to a more accurate fused depth map due to having 2 input sources that were closer in accuracy.

6.3 Crack detection

To complete the research, it needed to look at flaw detection and the applications, so the research was decided to focus on cracks specifically as the flaws to detect and analyse. This would do a simple analysis of the cracks by using the surface crack shape to determine the significance of a crack, as the surface shape has been shown to indicate the depth of the crack and predict future growth. The properties of the crack shape, such as the width or length [197], have been shown to be correlated with the depth of the crack [198] or the growth of the crack, with the length being modelled as directly infusing the stress intensity factor [199], which is used to predict future growth.

To be able to get the shape of the crack, a segmentation map would need to be estimated first, which gives full information about the surface shape. Then, using this crack segmentation map, specific shape properties can be detected, such as the crack length, by having a line drawn from the crack points and measuring that line to get a value for the crack length. This could be used with the calibrated camera parameters and distance to the surface to give the exact length of the crack.

Initial experiments to demonstrate the method stages were done. First, a segmentation model was trained using the method outlined in "Downstream Semantic Segmentation Model for Low-Level Surface Crack Detection." [101]. The DeepLabV3 [177] architecture was trained from randomly initialised weights on the crackseg9k dataset [175], which contains ground truth segmentation maps for training and evaluation. Then, a method was implemented for testing that used a pre-trained model where it had been trained with more computational resources than was accessable for this research with "CrackFormer: Transformer Network for Fine-Grained Crack Detection" [121], which used a vision transformer (ViT) architecture which differs from the traditional convolutional neural network (CNN) architecture that was trained, and this showed superior accuracy for segmentation.

The earlier research enabled this stage of research since crack segmentation methods could be looked at using RGB-D data, where the depth map could be obtained from a fusion method, covered in the earlier stage of research combining multiple sensor modalities, and then there would be further data fusion for the method to use both the RGB image and depth map to generate the segmentation map. However, there was a lack of data availability compared to the RGB-only crack data, with only small datasets publicly available for use [183], so to overcome this, methods for synthetic data generation were worked on.

For synthetic data, a novel method was implemented, which used both Blender [127] and BlenSor [180] to generate this large dataset of 10,000 data samples and further increase diversity through data augmentation. It was verified that the synthetic data accurately represented the real-world cracks it was modelling by comparing it to other state-of-the-art methods on the smaller real-world dataset. The results showed that the presented method, which used the synthetic data for training, outperformed all the other methods when evaluated with both the IoU and F1-score. This shows that the use of both the extra synthetic training data as well as the extra modality with RGB-D data resulted in improved segmentation accuracy by using data fusion.

6.4 Limitations and future directions

The presented work demonstrated the different stages of a method for analysing cracks in surfaces, where a crack would be captured with RGB cameras and depth sensors. Next, the presented method from this research could be used for combining state-of-the-art methods for RGB stereo and depth completion to get a highly accurate depth map as the 3D representation of the scene. Then, using the RGB image with the modified image fusion method, an RGB-D image of the scene was obtained, which is high-quality and low error. Then, the model trained with synthetic data would use the RGB-D image to produce a segmentation map of the crack, which would then be used to estimate the crack length and would be used to determine the severity of the crack.

Due to the limitations with the availability of data, the method in full could not be tested, so each stage had to be tested separately. Future work would be to create a larger real-world dataset for the specific task of crack segmentation with multiple sensor modalities used to capture the cracks and produce depth maps, which would allow the full method to be verified and provide a direction for future research to improve upon each stage.

The method uses deep learning, which limits its application for other flaws, such as cracks in steel, since it would need to be trained using relevant data on these flaws, as the cracks would have different characteristics compared to the cracks in roads, which the presented method from this research used for the training data. Computational resources are also a limitation for deep learning models, since although using the synthetic data generation method could potentially generate unlimited data, computational resources were still needed to generate the data and train the model, and it also limits the complexity of the model architecture, since models with more parameters would need more computational resources to train.

Future research should aim to find improvements in the fusion method and the crack segmentation by producing results with reduced error and increasing computational efficiency. Further improvements can be found with robustness to challenging scenarios such as dynamic environments, diverse materials or difficult weather conditions, which are important for creating a system that can be deployed in real-world conditions. Improvements can be found by using newer, more advanced sensors that allow higher-quality input data to be used. Additionally, improvements with computational efficiency, by reducing memory usage and speeding up the computation time, enable the real-time deployment of the process in embedded systems.

The research showed that data is crucial to finding these improvements. The success of the larger synthetic dataset demonstrates that data quantity is essential. The common theme was that end-to-end deep learning models consistently outperformed methods that rely on specific preprocessing or post-processing steps, as well as using larger models and model architectures that scale better with more training data.

Therefore, this informs a future path for the research that involves creating a larger and higher-quality dataset that will be the foundation for research in this task. The progress in sensor technology should be leveraged to do this using newer sensor models and exploring data collection with integrated sensor units that simultaneously capture with multiple modalities to simplify the calibration, improving the alignment of this data. Following this data, improvements can be made to synthetic data generation using the real-world data for reference and validation. More advanced synthetic data generation methods should be explored, such as using neural-generated data using GANs [200] or Diffusion models [201] instead of just using software-based simulation. The data can also enable the development of techniques that can better leverage the increased training data, such as end-to-end deep learning methods that would directly estimate the depth values instead of using a weighted average or max confidence approach. The data would also allow deep learning architectures that scale better with more data, such as the transformer architecture [202].

References

[1] David L Hall and James Llinas. An introduction to multisensor data fusion. *Proceedings of the IEEE*, 85(1):6–23, 1997.

[2] Suleyman Kivanc Ekici: *NRM Mühendislik*. Manual vs. automated inspection: Striking the right balance for quality and throughput. `https://www.linkedin.com/pulse/manual-vs-automated-inspection-striking-right-balance-ekici-assef`, 2025.

[3] Averroes: *Averroes.ai*. Understanding defect detection in manufacturing in 2025. `https://averroes.ai/blog/defect-detection-in-manufacturing`, 2025.

[4] Sanjay Pichaiah: *Akridata.ai*. Manual vs automated inspection quality control: Why automated inspection is the future of manufacturing. `https://akridata.ai/blog/automated-vs-manual-inspection-quality-control/`, 2025.

[5] Yashika: *Robro Systems*. The evolution of defect detection: From traditional methods to machine vision and ai. `https://www.robrosystems.com/blogs/post/the-evolution-of-defect-detection-from-traditional-methods-to-machine-vision-and-ai`, 2024.

[6] Yao Yao, Shue-Ting Ellen Tung, and Branko Glisic. Crack detection and characterization techniques—an overview. *Structural Control and Health Monitoring*, 21(12):1387–1413, 2014.

[7] Jinglong Wei. Based on an overview of crack detection. *Scientific Journal of Technology*, 6:133–137, 03 2024.

[8] Dihao Ai, Guiyuan Jiang, Siew-Kei Lam, Peilan He, and Chengwu Li. Computer vision framework for crack detection of civil infrastructure—a review. *Engineering Applications of Artificial Intelligence*, 117:105478, 2023.

[9] William E Lorensen and Harvey E Cline. Marching cubes: A high resolution 3d surface construction algorithm. *ACM siggraph computer graphics*, 21(4):163–169, 1987.

[10] Michael Kazhdan and Hugues Hoppe. Screened poisson surface reconstruction. *ACM Transactions on Graphics (ToG)*, 32(3):1–13, 2013.

[11] Shutao Li, Xudong Kang, and Jianwen Hu. Image fusion with guided filtering. *IEEE Transactions on Image processing*, 22(7):2864–2875, 2013.

[12] Hui Li, Xiao-Jun Wu, and Josef Kittler. Infrared and visible image fusion using a deep learning framework. In *2018 24th international conference on pattern recognition (ICPR)*, pages 2705–2710. IEEE, 2018.

[13] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 303–312, 1996.

[14] Carlo Dal Mutto, Pietro Zanuttigh, Stefano Mattoccia, and Guido Cortelazzo. Locally consistent tof and stereo data fusion. In *European Conference on Computer Vision*, pages 598–607. Springer, 2012.

[15] Carlo Dal Mutto, Pietro Zanuttigh, and Guido Maria Cortelazzo. Probabilistic tof and stereo data fusion based on mixed pixels measurement models. *IEEE transactions on pattern analysis and machine intelligence*, 37(11):2260–2272, 2015.

[16] Giulio Marin, Pietro Zanuttigh, and Stefano Mattoccia. Reliable fusion of tof and stereo depth driven by confidence measures. In *European Conference on Computer Vision*, pages 386–401. Springer, 2016.

[17] Matteo Poggi, Gianluca Agresti, Fabio Tosi, Pietro Zanuttigh, and Stefano Mattoccia. Confidence estimation for tof and stereo sensors and its application to depth data fusion. *IEEE Sensors Journal*, 20(3):1411–1421, 2019.

[18] Young Min Kim, Christian Theobalt, James Diebel, Jana Kosecka, Branislav Miscusik, and Sebastian Thrun. Multi-view image and tof sensor fusion for dense 3d reconstruction. In *2009 IEEE 12th international conference on computer vision workshops, ICCV workshops*, pages 1542–1549. IEEE, 2009.

[19] Yasutaka Furukawa and Jean Ponce. Accurate, dense, and robust multiview stereopsis. *IEEE transactions on pattern analysis and machine intelligence*, 32(8):1362–1376, 2009.

[20] Jorge Nocedal. Updating quasi-newton matrices with limited storage. *Mathematics of computation*, 35(151):773–782, 1980.

[21] Li Ding and Gaurav Sharma. Fusing structure from motion and lidar for dense accurate depth map estimation. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1283–1287. IEEE, 2017.

[22] Johannes L Schonberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4104–4113, 2016.

[23] Andriy Myronenko and Xubo Song. Point set registration: Coherent point drift. *IEEE transactions on pattern analysis and machine intelligence*, 32(12):2262–2275, 2010.

[24] Gehua Yang, Charles V Stewart, Michal Sofka, and Chia-Ling Tsai. Registration of challenging image pairs: Initialization, estimation, and decision. *IEEE transactions on pattern analysis and machine intelligence*, 29(11):1973–1989, 2007.

[25] Xuelian Cheng, Yiran Zhong, Yuchao Dai, Pan Ji, and Hongdong Li. Noise-aware unsupervised deep lidar-stereo fusion. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6339–6348, 2019.

[26] Yiran Zhong, Yuchao Dai, and Hongdong Li. Self-supervised learning for stereo matching with self-improving ability. *arXiv preprint arXiv:1709.00930*, 2017.

[27] Jia-Ren Chang and Yong-Sheng Chen. Pyramid stereo matching network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5410–5418, 2018.

[28] Alex Kendall, Hayk Martirosyan, Saumitro Dasgupta, Peter Henry, Ryan Kennedy, Abraham Bachrach, and Adam Bry. End-to-end learning of geometry and context for deep stereo regression. In *Proceedings of the IEEE international conference on computer vision*, pages 66–75, 2017.

[29] Radhakrishna Achanta, Appu Shaji, Kevin Smith, Aurelien Lucchi, Pascal Fua, and Sabine Süsstrunk. Slic superpixels compared to state-of-the-art superpixel methods. *IEEE transactions on pattern analysis and machine intelligence*, 34(11):2274–2282, 2012.

[30] Sungjoon Choi, Qian-Yi Zhou, and Vladlen Koltun. Robust reconstruction of indoor scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5556–5565, 2015.

[31] Christian Kerl, Jürgen Sturm, and Daniel Cremers. Robust odometry estimation for rgb-d cameras. In *2013 IEEE international conference on robotics and automation*, pages 3748–3754. IEEE, 2013.

[32] Radu Bogdan Rusu, Nico Blodow, and Michael Beetz. Fast point feature histograms (fpfh) for 3d registration. In *2009 IEEE international conference on robotics and automation*, pages 3212–3217. IEEE, 2009.

[33] Robert Maier, Kihwan Kim, Daniel Cremers, Jan Kautz, and Matthias Nießner. Intrinsic3d: High-quality 3d reconstruction by joint appearance and geometry optimization with spatially-varying lighting. In *Proceedings of the IEEE international conference on computer vision*, pages 3114–3122, 2017.

[34] Matthias Nießner, Michael Zollhöfer, Shahram Izadi, and Marc Stamminger. Real-time 3d reconstruction at scale using voxel hashing. *ACM Transactions on Graphics (ToG)*, 32(6):1–11, 2013.

[35] Angela Dai, Matthias Nießner, Michael Zollhöfer, Shahram Izadi, and Christian Theobalt. Bundlefusion: Real-time globally consistent 3d reconstruction using on-the-fly surface reintegration. *ACM Transactions on Graphics (ToG)*, 36(4):1, 2017.

[36] Michael Zollhöfer, Angela Dai, Matthias Innmann, Chenglei Wu, Marc Stamminger, Christian Theobalt, and Matthias Nießner. Shading-based refinement on volumetric signed distance functions. *ACM Transactions on Graphics (TOG)*, 34(4):1–14, 2015.

[37] Georg Klein and David Murray. Parallel tracking and mapping for small ar workspaces. In *2007 6th IEEE and ACM international symposium on mixed and augmented reality*, pages 225–234. IEEE, 2007.

[38] Bill Triggs, Philip F McLauchlan, Richard I Hartley, and Andrew W Fitzgibbon. Bundle adjustment—a modern synthesis. In *International workshop on vision algorithms*, pages 298–372. Springer, 1999.

[39] Jorge Nocedal and Stephen J Wright. *Numerical optimization*. Springer, 2006.

[40] Zachary Teed and Jia Deng. Droid-slam: Deep visual slam for monocular, stereo, and rgb-d cameras. *Advances in Neural Information Processing Systems*, 34, 2021.

[41] Zachary Teed and Jia Deng. Raft: Recurrent all-pairs field transforms for optical flow. In *European conference on computer vision*, pages 402–419. Springer, 2020.

[42] Wenshan Wang, Delong Zhu, Xiangwei Wang, Yaoyu Hu, Yuheng Qiu, Chen Wang, Yafei Hu, Ashish Kapoor, and Sebastian Scherer. Tartanair: A dataset to push the limits of visual slam. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4909–4916. IEEE, 2020.

[43] Brian Karis and Epic Games. Real shading in unreal engine 4. *Proc. Physically Based Shading Theory Practice*, 4(3):1, 2013.

[44] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and service robotics: Results of the 11th international conference*, pages 621–635. Springer, 2017.

[45] Xiaoyang Guo, Kai Yang, Wukui Yang, Xiaogang Wang, and Hongsheng Li. Group-wise correlation stereo network. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3273–3282, 2019.

[46] Yao Yao, Zixin Luo, Shiwei Li, Tian Fang, and Long Quan. Mvsnet: Depth inference for unstructured multi-view stereo. In *Proceedings of the European conference on computer vision (ECCV)*, pages 767–783, 2018.

[47] Feihu Zhang, Victor Prisacariu, Ruigang Yang, and Philip HS Torr. Ga-net: Guided aggregation net for end-to-end stereo matching. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 185–194, 2019.

[48] Feihu Zhang, Xiaojuan Qi, Ruigang Yang, Victor Prisacariu, Benjamin Wah, and Philip Torr. Domain-invariant stereo matching networks. In *European Conference on Computer Vision*, pages 420–439. Springer, 2020.

[49] Vladimir Tankovich, Christian Hane, Yinda Zhang, Adarsh Kowdle, Sean Fanello, and Sofien Bouaziz. Hitnet: Hierarchical iterative tile refinement network for real-time stereo matching. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14362–14372, 2021.

[50] Zhaoshuo Li, Xingtong Liu, Nathan Drenkow, Andy Ding, Francis X Creighton, Russell H Taylor, and Mathias Unberath. Revisiting stereo depth estimation from a sequence-to-sequence perspective with transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6197–6206, 2021.

[51] Sameh Khamis, Sean Fanello, Christoph Rhemann, Adarsh Kowdle, Julien Valentin, and Shahram Izadi. Stereonet: Guided hierarchical refinement for real-time edge-aware depth prediction. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 573–590, 2018.

[52] Gengshan Yang, Joshua Manela, Michael Happold, and Deva Ramanan. Hierarchical deep stereo matching on high-resolution images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5515–5524, 2019.

[53] Jure Zbontar, Yann LeCun, et al. Stereo matching by training a convolutional neural network to compare image patches. *J. Mach. Learn. Res.*, 17(1):2287–2318, 2016.

[54] Nikolaus Mayer, Eddy Ilg, Philip Hausser, Philipp Fischer, Daniel Cremers, Alexey Dosovitskiy, and Thomas Brox. A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4040–4048, 2016.

[55] Xinjing Cheng, Peng Wang, and Ruigang Yang. Depth estimation via affinity learned with convolutional spatial propagation network. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 103–119, 2018.

[56] Xinjing Cheng, Peng Wang, Chenye Guan, and Ruigang Yang. Cspn++: Learning context and resource aware convolutional spatial propagation networks for depth completion. *arXiv preprint arXiv:1911.05377*, 2019.

[57] Lina Liu, Xibin Song, Xiaoyang Lyu, Junwei Diao, Mengmeng Wang, Yong Liu, and Liangjun Zhang. Fcfr-net: Feature fusion based coarse-to-fine residual learning for depth completion. *arXiv preprint arXiv:2012.08270*, 2020.

[58] Zhiqiang Yan, Kun Wang, Xiang Li, Zhenyu Zhang, Baobei Xu, Jun Li, and Jian Yang. Rignet: Repetitive image guided network for depth completion. *arXiv preprint arXiv:2107.13802*, 2021.

[59] Jiaxiong Qiu, Zhaopeng Cui, Yinda Zhang, Xingdi Zhang, Shuaicheng Liu, Bing Zeng, and Marc Pollefeys. Deeplidar: Deep surface normal guided depth prediction for outdoor scene from sparse lidar data and single color image. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3313–3322, 2019.

[60] Yun Chen, Bin Yang, Ming Liang, and Raquel Urtasun. Learning joint 2d-3d representations for depth completion. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10023–10032, 2019.

[61] Shanshan Zhao, Mingming Gong, Huan Fu, and Dacheng Tao. Adaptive context-aware multi-modal network for depth completion. *IEEE Transactions on Image Processing*, 30:5264–5276, 2021.

[62] Byeong-Uk Lee, Kyunghyun Lee, and In So Kweon. Depth completion using plane-residual representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13916–13925, 2021.

[63] Chongzhen Zhang, Yang Tang, Chaoqiang Zhao, Qiyu Sun, Zhencheng Ye, and Jürgen Kurths. Multitask gans for semantic segmentation and depth completion with cycle consistency. *IEEE Transactions on Neural Networks and Learning Systems*, 32(12):5404–5415, 2021.

[64] Kihong Park, Seungryong Kim, and Kwanghoon Sohn. High-precision depth estimation with the 3d lidar and stereo fusion. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2156–2163. IEEE, 2018.

[65] Junming Zhang, Manikandasriram Srinivasan Ramanagopal, Ram Vasudevan, and Matthew Johnson-Roberson. Listereo: Generate dense depth maps from lidar and stereo imagery. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7829–7836. IEEE, 2020.

[66] Will Maddern and Paul Newman. Real-time probabilistic fusion of sparse 3d lidar and dense stereo. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2181–2188. IEEE, 2016.

[67] Jan Petrich, Zack Snow, David Corbin, and Edward W Reutzel. Multi-modal sensor fusion with machine learning for data-driven process monitoring for additive manufacturing. *Additive Manufacturing*, 48:102364, 2021.

[68] Omer Kullu and Eyup Cinar. A deep-learning-based multi-modal sensor fusion approach for detection of equipment faults. *Machines*, 10(11):1105, 2022.

[69] Kushal Virupakshappa, Michael Marino, and Erdal Oruklu. A multi-resolution convolutional neural network architecture for ultrasonic flaw detection. In *2018 IEEE International Ultrasonics Symposium (IUS)*, pages 1–4. IEEE, 2018.

[70] Oskar Siljama, Tuomas Koskinen, Oskari Jessen-Juhler, and Iikka Virkkunen. Automated flaw detection in multi-channel phased array ultrasonic data using machine learning. *Journal of Nondestructive Evaluation*, 40(3):67, 2021.

[71] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[72] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[73] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*, pages 21–37. Springer, 2016.

[74] Luka Posilović, Duje Medak, Marko Subašić, Tomislav Petković, Marko Budimir, and Sven Lončarić. Flaw detection from ultrasonic images using yolo and ssd. In *2019 11th International Symposium on Image and Signal Processing and Analysis (ISPA)*, pages 163–168. IEEE, 2019.

[75] Yongmin Guo, Zhitao Xiao, Lei Geng, Jun Wu, Fang Zhang, Yanbei Liu, and Wen Wang. Fully convolutional neural network with gru for 3d braided composite material flaw detection. *IEEE Access*, 7:151180–151188, 2019.

[76] Guoyong Zhang, Zhaohui Tang, Jin Zhang, and Weihua Gui. Convolutional autoencoder-based flaw detection for steel wire ropes. *Sensors*, 20(22):6612, 2020.

[77] Afshin Shoeibi, Marjane Khodatars, Mahboobeh Jafari, Navid Ghassemi, Parisa Moridian, Roohallah Alizadehsani, Sai Ho Ling, Abbas Khosravi, Hamid Alinejad-Rokny, Hak-Keung Lam, et al. Diagnosis of brain diseases in fusion of neuroimaging modalities using deep learning: A review. *Information Fusion*, 93:85–117, 2023.

[78] Velmurugan Subbiah Parvathy, Sivakumar Pothiraj, and Jenyfal Sampson. Optimal deep neural network model based multimodality fused medical image classification. *Physical Communication*, 41:101119, 2020.

[79] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*, pages 234–241. Springer, 2015.

[80] Younes Hamishebahar, Hong Guan, Stephen So, and Jun Jo. A comprehensive review of deep learning-based crack detection approaches. *Applied Sciences*, 12(3):1374, 2022.

[81] Highways England and Atkins. Economic growth and the strategic road network. Evidence review, Highways England, 2016.

[82] T. Buckland, C. Parkman, C. Booth, and R. Abell. Valuing the benefits of road maintenance. Client Project Report CPR2137, Transport Research Laboratory (TRL), Crowthorne, UK, 2015. Prepared for the Department for Transport.

[83] RoadVision. The hidden costs and inefficiencies of manual road inspections. `https://www.roadvision.ai/blog/the-hidden-costs-and-inefficiencies-of-manual-road-inspections`, 2026. Accessed: 2026-05-06.

[84] Ikhlas Abdel-Qader, Osama Abudayyeh, and Michael E Kelly. Analysis of edge-detection techniques for crack identification in bridges. *Journal of computing in civil engineering*, 17(4):255–263, 2003.

[85] Henrique Oliveira and Paulo Lobato Correia. Automatic road crack segmentation using entropy and image dynamic thresholding. In *2009 17th European Signal Processing Conference*, pages 622–626. IEEE, 2009.

[86] Prateek Prasanna, Kristin J Dana, Nenad Gucunski, Basily B Basily, Hung M La, Ronny Salim Lim, and Hooman Parvardeh. Automated crack detection on concrete bridges. *IEEE Transactions on automation science and engineering*, 13(2):591–599, 2014.

[87] Yong Shi, Limeng Cui, Zhiquan Qi, Fan Meng, and Zhensong Chen. Automatic road crack detection using random structured forests. *IEEE Transactions on Intelligent Transportation Systems*, 17(12):3434–3445, 2016.

[88] Raza Ali, Joon Huang Chuah, Mohamad Sofian Abu Talip, Norrima Mokhtar, and Muhammad Ali Shoaib. Structural crack detection using deep convolutional neural networks. *Automation in Construction*, 133:103989, 2022.

[89] Hongxia Li, Weixing Wang, Mengfei Wang, Limin Li, and Vivian Vimlund. A review of deep learning methods for pixel-level crack detection. *Journal of Traffic and Transportation Engineering (English Edition)*, 9(6):945–968, 2022.

[90] Lei Zhang, Fan Yang, Yimin Daniel Zhang, and Ying Julie Zhu. Road crack detection using deep convolutional neural network. In *2016 IEEE international conference on image processing (ICIP)*, pages 3708–3712. IEEE, 2016.

[91] Markus Eisenbach, Ronny Stricker, Daniel Seichter, Karl Amende, Klaus Debes, Maximilian Sesselmann, Dirk Ebersbach, Ulrike Stoeckert, and Horst-Michael Gross. How to get pavement distress detection ready for deep learning? a systematic approach. In *2017 international joint conference on neural networks (IJCNN)*, pages 2039–2047. IEEE, 2017.

[92] Zhun Fan, Yuming Wu, Jiewei Lu, and Wenji Li. Automatic pavement crack detection based on structured prediction with the convolutional neural network. *arXiv preprint arXiv:1802.02208*, 2018.

[93] Suguru Yokoyama and Takashi Matsumoto. Development of an automatic detector of cracks in concrete using machine learning. *Procedia engineering*, 171:1250–1255, 2017.

[94] Young-Jin Cha, Wooram Choi, Gahyun Suh, Sadegh Mahmoudkhani, and Oral Büyüköztürk. Autonomous structural visual inspection using region-based deep learning for detecting multiple damage types. *Computer-Aided Civil and Infrastructure Engineering*, 33(9):731–747, 2018.

[95] Mark David Jenkins, Thomas Arthur Carr, Maria Insa Iglesias, Tom Buggy, and Gordon Morison. A deep convolutional neural network for semantic pixel-wise segmentation of road and pavement surface cracks. In *2018 26th European signal processing conference (EUSIPCO)*, pages 2120–2124. IEEE, 2018.

[96] Fahimeh Fooladgar and Shohreh Kasaei. A survey on indoor rgb-d semantic segmentation: from hand-crafted features to deep convolutional neural networks. *Multimedia Tools and Applications*, 79(7):4499–4524, 2020.

[97] Bolei Zhou, Agata Lapedriza, Jianxiong Xiao, Antonio Torralba, and Aude Oliva. Learning deep features for scene recognition using places database. *Advances in neural information processing systems*, 27, 2014.

[98] Bowen Yin, Xuying Zhang, Zhongyu Li, Li Liu, Ming-Ming Cheng, and Qibin Hou. Dformer: Rethinking rgbd representation learning for semantic segmentation. *arXiv preprint arXiv:2309.09668*, 2023.

[99] Liang Yang, Bing Li, Wei Li, Howard Brand, Biao Jiang, and Jizhong Xiao. Concrete defects inspection and 3d mapping using cityflyer quadrotor robot. *IEEE/CAA Journal of Automatica Sinica*, 7(4), 2020.

[100] Eric Bianchi and Matthew Hebdon. Visual structural inspection datasets. *Automation in construction*, 139:104299, 2022.

[101] Thitirat Siriborvornratanakul. Downstream semantic segmentation model for low-level surface crack detection. *Advances in Multimedia*, 2022:1–12, 2022.

[102] Wenting Qiao, Qiangwei Liu, Xiaoguang Wu, Biao Ma, and Gang Li. Automatic pixel-level pavement crack recognition using a deep feature aggregation segmentation network with a scse attention mechanism module. *Sensors*, 21(9):2902, 2021.

[103] Jacob König, Mark David Jenkins, Mike Mannion, Peter Barrie, and Gordon Morison. Weakly-supervised surface crack segmentation by generating pseudo-labels using localization with a classifier and thresholding. *IEEE Transactions on Intelligent Transportation Systems*, 23(12):24083–24094, 2022.

[104] Jacob König, Mark David Jenkins, Mike Mannion, Peter Barrie, and Gordon Morison. Optimized deep encoder-decoder methods for crack segmentation. *Digital Signal Processing*, 108:102907, 2021.

[105] Vladimir Polovnikov, Dmitriy Alekseev, Ivan Vinogradov, and George V Lashkia. Daunet: Deep augmented neural network for pavement crack segmentation. *IEEE Access*, 9:125714–125723, 2021.

[106] Gui Yu, Juming Dong, Yihang Wang, and Xinglin Zhou. Ruc-net: A residual-unet-based convolutional neural network for pixel-level pavement crack segmentation. *Sensors*, 23(1):53, 2022.

[107] Jacob König, Mark David Jenkins, Peter Barrie, Mike Mannion, and Gordon Morison. Segmentation of surface cracks based on a fully convolutional neural network and gated scale pooling. In *2019 27th European signal processing conference (EUSIPCO)*, pages 1–5. IEEE, 2019.

[108] Firdes Çelik and Markus König. A sigmoid-optimized encoder–decoder network for crack segmentation with copy-edit-paste transfer learning. *Computer-Aided Civil and Infrastructure Engineering*, 37(14):1875–1890, 2022.

[109] Qipei Mei and Mustafa Gül. A cost effective solution for road crack inspection using cameras and deep neural networks. *arXiv preprint arXiv:1907.06014*, 2019.

[110] Guijie Zhu, Zhun Fan, Jiacheng Liu, Duan Yuan, Peili Ma, Meihua Wang, Weihua Sheng, and Kelvin CP Wang. Rha-net: An encoder-decoder network with residual blocks and hybrid attention mechanisms for pavement crack segmentation. *arXiv preprint arXiv:2207.14166*, 2022.

[111] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C Berg, Wan-Yen Lo, et al. Segment anything. *arXiv preprint arXiv:2304.02643*, 2023.

[112] Wenhai Wang, Jifeng Dai, Zhe Chen, Zhenhang Huang, Zhiqi Li, Xizhou Zhu, Xiaowei Hu, Tong Lu, Lewei Lu, Hongsheng Li, et al. Internimage: Exploring large-scale vision foundation models with deformable convolutions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14408–14419, 2023.

[113] Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.

[114] Wenhui Wang, Hangbo Bao, Li Dong, Johan Bjorck, Zhiliang Peng, Qiang Liu, Kriti Aggarwal, Owais Khan Mohammed, Saksham Singhal, Subhojit Som, et al. Image as a foreign language: Beit pretraining for all vision and vision-language tasks. *arXiv preprint arXiv:2208.10442*, 2022.

[115] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[116] Yuanduo Hong, Huihui Pan, Weichao Sun, Xinghu Yu, and Huijun Gao. Representation separation for semantic segmentation with vision transformers. *arXiv preprint arXiv:2212.13764*, 2022.

[117] Zhe Chen, Yuchen Duan, Wenhai Wang, Junjun He, Tong Lu, Jifeng Dai, and Yu Qiao. Vision transformer adapter for dense predictions. *arXiv preprint arXiv:2205.08534*, 2022.

[118] Feng Li, Hao Zhang, Huaizhe Xu, Shilong Liu, Lei Zhang, Lionel M Ni, and Heung-Yeung Shum. Mask dino: Towards a unified transformer-based framework for object detection and segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3041–3050, 2023.

[119] Yixuan Wei, Han Hu, Zhenda Xie, Zheng Zhang, Yue Cao, Jianmin Bao, Dong Chen, and Baining Guo. Contrastive learning rivals masked image modeling in fine-tuning via feature distillation. *arXiv preprint arXiv:2205.14141*, 2022.

[120] Xiaochen Ju, Xinxin Zhao, and Shengsheng Qian. Transmf: Transformer-based multi-scale fusion model for crack detection. *Mathematics*, 10(13):2354, 2022.

[121] Huajun Liu, Xiangyu Miao, Christoph Mertz, Chengzhong Xu, and Hui Kong. Crackformer: Transformer network for fine-grained crack detection. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3783–3792, 2021.

[122] Zhiying Lu, Hongtao Xie, Chuanbin Liu, and Yongdong Zhang. Bridging the gap between vision transformers and convolutional neural networks on small datasets. *Advances in Neural Information Processing Systems*, 35:14663–14677, 2022.

[123] IA Kanaeva and Ju A Ivanova. Road pavement crack detection using deep learning with synthetic data. In *IOP Conference Series: Materials Science and Engineering*, volume 1019, page 012036. IOP Publishing, 2021.

[124] Rodrigo Rill-García, Eva Dokladalova, and Petr Dokládal. Syncrack: Improving pavement and concrete crack detection through synthetic data generation. In *VISIGRAPP (4: VISAPP)*, pages 147–158, 2022.

[125] Aleksei Boikov, Vladimir Payor, Roman Savelev, and Alexandr Kolesnikov. Synthetic data generation for steel defect detection and classification using deep learning. *Symmetry*, 13(7):1176, 2021.

[126] Guanghao Zhai, Yasutaka Narazaki, Shuo Wang, Shaik Althaf V Shajihan, and Billie F Spencer Jr. Synthetic data augmentation for pixel-wise steel fatigue crack identification using fully convolutional networks. *Smart Struct Syst*, 29(1):237–250, 2022.

[127] Blender Foundation. blender.org - Home of the Blender project - Free and Open 3D Creation Software — blender.org. https://www.blender.org/.

[128] Gianluca Agresti, Ludovico Minto, Giulio Marin, and Pietro Zanuttigh. Stereo and tof data fusion by learning from synthetic data. *Information Fusion*, 49:161–173, 2019.

[129] Kaiming He, Jian Sun, and Xiaoou Tang. Guided image filtering. *IEEE transactions on pattern analysis and machine intelligence*, 35(6):1397–1409, 2012.

[130] Benedict Marsh. Codebase and Data. https://github.com/benedictmarsh2/AI-enabled-flaw-detection-using-multi-sensory-data-fusion, 2026.

[131] Antoine Mousnier, Elif Vural, and Christine Guillemot. Partial light field tomographic reconstruction from a fixed-camera focal stack. *arXiv preprint arXiv:1503.01903*, 2015.

[132] Cui Yang, Jian-Qi Zhang, Xiao-Rui Wang, and Xin Liu. A novel similarity based quality metric for image fusion. *Information Fusion*, 9(2):156–160, 2008.

[133] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

[134] Nedeljko Cvejic, Artur Loza, David Bull, and Nishan Canagarajah. A similarity metric for assessment of image fusion algorithms. *International journal of signal processing*, 2(3):178–182, 2005.

[135] Zhou Wang and Alan C Bovik. A universal image quality index. *IEEE signal processing letters*, 9(3):81–84, 2002.

[136] CS Xydeas, , and Vladimir Petrovic. Objective image fusion performance measure. *Electronics letters*, 36(4):308–309, 2000.

[137] M Hossny, S Nahavandi, and D Creighton. Comments on'information measure for performance of image fusion'. *Electronics letters*, 44(18):1066–1067, 2008.

[138] Kanita Karaduzovic-Hadziabdic, J Hasic Telalovic, and Rafal Konrad Mantiuk. Multi-exposure image stacks for testing hdr deghosting methods. https://www.repository.cam.ac.uk/handle/1810/261766, 2017. Apollo - University of Cambridge Repository.

[139] Fumihito Yasuma, Tomoo Mitsunaga, Daisuke Iso, and Shree K Nayar. Generalized assorted pixel camera: postcapture control of resolution, dynamic range, and spectrum. *IEEE transactions on image processing*, 19(9):2241–2253, 2010.

[140] Gianluca Agresti, Ludovico Minto, Giulio Marin, and Pietro Zanuttigh. Dataset - stereo and tof data fusion by learning from synthetic data. `https://lttm.dei.unipd.it/paper_data/realfusion/#dat`, 2019.

[141] Heiko Hirschmuller. Stereo processing by semiglobal matching and mutual information. *IEEE Transactions on pattern analysis and machine intelligence*, 30(2):328–341, 2007.

[142] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[143] Stephen M Pizer, E Philip Amburn, John D Austin, Robert Cromartie, Ari Geselowitz, Trey Greer, Bart ter Haar Romeny, John B Zimmerman, and Karel Zuiderveld. Adaptive histogram equalization and its variations. *Computer vision, graphics, and image processing*, 39(3):355–368, 1987.

[144] Kostadin Dabov, Alessandro Foi, Vladimir Katkovnik, and Karen Egiazarian. Image denoising by sparse 3-d transform-domain collaborative filtering. *IEEE Transactions on image processing*, 16(8):2080–2095, 2007.

[145] Benedict Marsh, Abdul Hamid Sadka, and Hamid Bahai. A critical review of deep learning-based multi-sensor fusion techniques. *Sensors*, 22(23):9364, 2022.

[146] Moritz Menze and Andreas Geiger. Object scene flow for autonomous vehicles. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3061–3070, 2015.

[147] Moritz Menze, Christian Heipke, and Andreas Geiger. Object scene flow. *ISPRS Journal of Photogrammetry and Remote Sensing (JPRS)*, 2018.

[148] Moritz Menze, Christian Heipke, and Andreas Geiger. Joint 3d estimation of vehicles and scene flow. In *ISPRS Workshop on Image Sequence Analysis (ISA)*, 2015.

[149] Jonas Uhrig, Nick Schneider, Lukas Schneider, Uwe Franke, Thomas Brox, and Andreas Geiger. Sparsity invariant cnns. In *International Conference on 3D Vision (3DV)*, 2017.

[150] Lahav Lipson, Zachary Teed, and Jia Deng. Raft-stereo: Multilevel recurrent field transforms for stereo matching. In *2021 International Conference on 3D Vision (3DV)*, pages 218–227. IEEE, 2021.

[151] Edward H Adelson, Charles H Anderson, James R Bergen, Peter J Burt, and Joan M Ogden. Pyramid methods in image processing. *RCA engineer*, 29(6):33–41, 1984.

[152] Jiankun Li, Peisen Wang, Pengfei Xiong, Tao Cai, Ziwei Yan, Lei Yang, Jiangyu Liu, Haoqiang Fan, and Shuaicheng Liu. Practical stereo matching via cascaded recurrent network with adaptive correlation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16263–16272, 2022.

[153] Xizhou Zhu, Han Hu, Stephen Lin, and Jifeng Dai. Deformable convnets v2: More deformable, better results. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 9308–9316, 2019.

[154] Xuelian Cheng, Yiran Zhong, Mehrtash Harandi, Yuchao Dai, Xiaojun Chang, Hongdong Li, Tom Drummond, and Zongyuan Ge. Hierarchical neural architecture search for deep stereo matching. *Advances in Neural Information Processing Systems*, 33:22158–22169, 2020.

[155] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018.

[156] Chenxi Liu, Liang-Chieh Chen, Florian Schroff, Hartwig Adam, Wei Hua, Alan L Yuille, and Li Fei-Fei. Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 82–92, 2019.

[157] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[158] Mu Hu, Shuling Wang, Bin Li, Shiyu Ning, Li Fan, and Xiaojin Gong. Penet: Towards precise and efficient image guided depth completion. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 13656–13662. IEEE, 2021.

[159] Jie Tang, Fei-Peng Tian, Wei Feng, Jian Li, and Ping Tan. Learning guided convolutional network for depth completion. *IEEE Transactions on Image Processing*, 30:1116–1129, 2020.

[160] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015.

[161] Danish Nazir, Marcus Liwicki, Didier Stricker, and Muhammad Zeshan Afzal. Semattnet: Towards attention-based semantic aware guided depth completion. *arXiv preprint arXiv:2204.13635*, 2022.

[162] Fahimeh Fooladgar and Shohreh Kasaei. Multi-modal attention-based fusion model for semantic segmentation of rgb-depth images. *arXiv preprint arXiv:1912.11691*, 2019.

[163] Zifeng Wu, Chunhua Shen, and Anton Van Den Hengel. Wider or deeper: Revisiting the resnet model for visual recognition. *Pattern Recognition*, 90:119–133, 2019.

[164] Wouter Van Gansbeke, Davy Neven, Bert De Brabandere, and Luc Van Gool. Sparse and noisy lidar completion with rgb guidance and uncertainty. In *2019 16th international conference on machine vision applications (MVA)*, pages 1–6. IEEE, 2019.

[165] Yuankai Lin, Tao Cheng, Qi Zhong, Wending Zhou, and Hua Yang. Dynamic spatial propagation network for depth completion. *arXiv preprint arXiv:2202.09769*, 2022.

[166] Xinjing Cheng, Peng Wang, Yanqi Zhou, Chenye Guan, and Ruigang Yang. Omnidirectional depth extension networks. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 589–595. IEEE, 2020.

[167] Zheyuan Xu, Hongche Yin, and Jian Yao. Deformable spatial propagation networks for depth completion. In *2020 IEEE International Conference on Image Processing (ICIP)*, pages 913–917. IEEE, 2020.

[168] Jaesung Choe, Kyungdon Joo, Tooba Imtiaz, and In So Kweon. Volumetric propagation network: Stereo-lidar fusion for long-range depth estimation. *IEEE Robotics and Automation Letters*, 6(3):4672–4679, 2021.

[169] Alejandro Newell, Kaiyu Yang, and Jia Deng. Stacked hourglass networks for human pose estimation. In *European conference on computer vision*, pages 483–499. Springer, 2016.

[170] Tsun-Hsuan Wang, Hou-Ning Hu, Chieh Hubert Lin, Yi-Hsuan Tsai, Wei-Chen Chiu, and Min Sun. 3d lidar and stereo fusion using stereo matching network with conditional cost volume normalization. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5895–5902. IEEE, 2019.

[171] Ethan Perez, Harm De Vries, Florian Strub, Vincent Dumoulin, and Aaron Courville. Learning visual reasoning without strong priors. *arXiv preprint arXiv:1707.03017*, 2017.

[172] Harm De Vries, Florian Strub, Jérémie Mary, Hugo Larochelle, Olivier Pietquin, and Aaron C Courville. Modulating early visual processing by language. *Advances in Neural Information Processing Systems*, 30, 2017.

[173] Jonas Uhrig, Nick Schneider, Lukas Schneider, Uwe Franke, Thomas Brox, and Andreas Geiger. Sparsity invariant cnns. In *2017 international conference on 3D Vision (3DV)*, pages 11–20. IEEE, 2017.

[174] Benedict Marsh and Ruiheng Wu. Crack segmentation in roads using synthetic data and rgb-d data fusion. *Computer Vision and Image Understanding*, page 104452, 2025.

[175] Siddharth Sharma, Dhananjay Balakrishnan, Shreyas Kulkarni, Shreyas Singh, Saipraneeth Devunuri, and Sai Chowdeswara Rao Korlapati. Crackseg9k: A Collection of Crack Segmentation Datasets, 2022.

[176] Shreyas Kulkarni, Shreyas Singh, Dhananjay Balakrishnan, Siddharth Sharma, Saipraneeth Devunuri, and Sai Chowdeswara Rao Korlapati. Crackseg9k: A collection and benchmark for crack segmentation datasets and frameworks. In *European conference on computer vision*, pages 179–195. Springer, 2022.

[177] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587*, 2017.

[178] Qin Zou, Yu Cao, Qingquan Li, Qingzhou Mao, and Song Wang. Cracktree: Automatic crack detection from pavement images. *Pattern Recognition Letters*, 33(3):227–238, 2012.

[179] Qin Zou, Zheng Zhang, Qingquan Li, Xianbiao Qi, Qian Wang, and Song Wang. Deepcrack: Learning hierarchical convolutional features for crack detection. *IEEE Transactions on Image Processing*, 28(3):1498–1512, 2018.

[180] Michael Gschwandtner, Roland Kwitt, Andreas Uhl, and Wolfgang Pree. Blensor: Blender sensor simulation toolbox. In *Advances in Visual Computing: 7th International Symposium, ISVC 2011, Las Vegas, NV, USA, September 26-28, 2011. Proceedings, Part II 7*, pages 199–208. Springer, 2011.

[181] Kurt Spencer. Like simplex noise but don't like the patent? introducing opensimplex noise! `https://jvm-gaming.org/t/like-simplex-noise-but-dont-like-the-patent-introducing-opensimplex-noise/51080`, 2014.

[182] Ken Perlin. An image synthesizer. *ACM Siggraph Computer Graphics*, 19(3):287–296, 1985.

[183] Andrea Ranieri, Elia Moscoso Thompson, and Silvia Biasotti. Pothole mix, 2022.

[184] David G Lowe. Object recognition from local scale-invariant features. In *Proceedings of the seventh IEEE international conference on computer vision*, volume 2, pages 1150–1157. Ieee, 1999.

[185] Elia Moscoso Thompson, Andrea Ranieri, Silvia Biasotti, Miguel Chicchon, Ivan Sipiran, Minh-Khoi Pham, Thang-Long Nguyen-Ho, Hai-Dang Nguyen, and Minh-Triet Tran. Shrec 2022: Pothole and crack detection in the road pavement using images and rgb-d data. *Computers & Graphics*, 107:161–171, 2022.

[186] Tongle Fan, Guanglei Wang, Yan Li, and Hongrui Wang. Ma-net: A multi-scale attention network for liver and tumor segmentation. *IEEE Access*, 8:179656–179665, 2020.

[187] Zongwei Zhou, Md Mahfuzur Rahman Siddiquee, Nima Tajbakhsh, and Jianming Liang. Unet++: A nested u-net architecture for medical image segmentation. In *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support: 4th International Workshop, DLMIA 2018, and 8th International Workshop, ML-CDS 2018, Held in Conjunction with MICCAI 2018, Granada, Spain, September 20, 2018, Proceedings 4*, pages 3–11. Springer, 2018.

[188] Enze Xie, Wenhai Wang, Zhiding Yu, Anima Anandkumar, Jose M Alvarez, and Ping Luo. Segformer: Simple and efficient design for semantic segmentation with transformers. *Advances in neural information processing systems*, 34:12077–12090, 2021.

[189] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *Proceedings of the European conference on computer vision (ECCV)*, pages 801–818, 2018.

[190] Mingxing Tan. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*, 2019.

[191] Xiaokang Chen, Yuhui Yuan, Gang Zeng, and Jingdong Wang. Semi-supervised semantic segmentation with cross pseudo supervision. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 2613–2622, 2021.

[192] Qingshan Xu and Wenbing Tao. Multi-scale geometric consistency guided multi-view stereo. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5483–5492, 2019.

[193] Andreas Kuhn, Christian Sormann, Mattia Rossi, Oliver Erdler, and Friedrich Fraundorfer. Deepc-mvs: Deep confidence prediction for multi-view stereo reconstruction. In *2020 International Conference on 3D Vision (3DV)*, pages 404–413. IEEE, 2020.

[194] Johannes L Schönberger, Enliang Zheng, Jan-Michael Frahm, and Marc Pollefeys. Pixelwise view selection for unstructured multi-view stereo. In *European Conference on Computer Vision*, pages 501–518. Springer, 2016.

[195] Shuran Song, Samuel P Lichtenberg, and Jianxiong Xiao. Sun rgb-d: A rgb-d scene understanding benchmark suite. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 567–576, 2015.

[196] Thomas Schops, Torsten Sattler, and Marc Pollefeys. Bad slam: Bundle adjusted direct rgb-d slam. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 134–144, 2019.

[197] Xiangqian Fan and Shaowei Hu. Influence of crack initiation length on fracture behaviors of reinforced concrete. *Applied clay science*, 79:25–29, 2013.

[198] Yue Li, Juhui Zhang, Zhongguo Guan, and Youliang Chen. Experimental study on the correlation between crack width and crack depth of rc beams. *Materials*, 14(20):5950, 2021.

[199] Paul Paris and Fazil Erdogan. A critical analysis of crack propagation laws. *Journal of basic engineering*, 85(4):528–533, 1963.

[200] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks. *Communications of the ACM*, 63(11):139–144, 2020.

[201] Prafulla Dhariwal and Alexander Nichol. Diffusion models beat gans on image synthesis. *Advances in neural information processing systems*, 34:8780–8794, 2021.

[202] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.