Toward Deep Monocular View Generation and Omnidirectional Depth Estimation

Helmi Fraser

Submitted for the Degree of Doctor of Philosophy in Robotics and Autonomous Systems

THE UNIVERSITY *of* EDINBURGH

June 2023

Abstract

This thesis proposes new strategies for obtaining environmental depth representations from monocular perspective and omnidirectional vision. This research is inspired by the necessity for mobile autonomous systems to be able to sense their surroundings, which is frequently abundant in vital data necessary for planning, decision-making and action.

The methodologies presented here are primarily data-driven and based on machine learning, specifically deep learning.

Our first contribution is the generation of top-down, “bird’s eye view” representations of detected vehicles in a scene. This was achieved using only monocular, perspective view images. The novelty here was via an adversarial training scheme, which our experiments showed resulted in more robust models versus a strictly supervised baseline.

Our second contribution is a novel method for adapting view synthesis-based depth estimation models to omnidirectional imagery. Our proposal comprise three important facets. Firstly, a "virtual" spherical camera model is integrated into the training pipeline to facilitate model training. Secondly, we explicitly encode information of the spherical nature of the image format by adopting spherical convolutional layers to perform convolution operations, consequently compensating for the significant distortion. Thirdly, we propose an optical flow-based masking strategy to reduce the impact of undesired pixels during training, such as those originating from large, challenging visual areas of the image such as the sky. Our qualitative and quantitative findings indicate that these additions result in improved depth estimations versus earlier methods.

Our final contribution, broadly, is a method for incorporating LiDAR information into the training pipeline of an omnidirectional depth estimation model. We introduce a Bayesian optimisation-based extrinsic calibration method to match LiDAR returns with equirectangular images. Primarily, we weight the incorporation of this data via a frequency-based scheme dependent on the number of detected LiDAR projections. The results from this show that there is a tangible quantitative benefit in doing the aforementioned.

Acknowledgements

Foremost of my acknowledgments would be my supervisor, Dr Sen Wang, whose guidance and support over the years has been truly valuable. Without his help, I wouldn't be where I am today.

Inclusion of Published Works Form

Please note you are only required to complete this form if your thesis contains published works. If this is the case, please include this form within your thesis before submission.

Declaration

This thesis contains one or more multi-author published works. I hereby declare that the contributions of each author to these publications is as follows:

Citation details	*H. Fraser and S. Wang, "DeepBEV: A Conditional Adversarial Network for Bird's Eye View Generation," 2020 25th International Conference on Pattern Recognition (ICPR), 2021, pp. 5581-5586, doi: 10.1109/ICPR48806.2021.9412516.*
Author 1	Main author
Author 2	Review and guidance

Citation details	*H. Fraser and S. Wang, "Monocular Depth Estimation for Equirectangular Videos", 2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2022), 2022* [10.1109/IROS47612.2022.9982157](https://doi.org/10.1109/IROS47612.2022.9982157)
Author 1	Main author
Author 2	Review and guidance

Citation details	e. g. Author 1 and Author 2, Title of paper, Title of Journal, X, XX-XX (20XX)
Author 1	Contribution….
Author 2	Contribution….

Please included additional citations as required.

Contents

List of Figures

List of Tables

Part I

Introduction

1 Introduction

The ability to detect one's surroundings is an important part of how humans and animals traverse the physical world. As humans, we have many senses that span distinct modalities: sight, hearing and touch. Perhaps the most crucial is our sense of sight, which gives us the ability to traverse both familiar and unfamiliar environments. Our eyesight also gives us the ability to glean information from such settings, which may include not just environmental cues but also written words, pictures, and other forms of visual information.

The information we gather from our surroundings allows us to make judgments and take action. The same holds true for a wide range of non-biological systems such as robots. For a large number of robotic systems, having a way of sensing the environment is a fundamental requirement of safe operation. Mobile robotics and autonomous vehicles, in particular, simply cannot execute their intended functions without some sensing capability for mappping or navigation. Even many static systems, such as factory or assembly line robots often require sensory input in order to guide end effectors. Beyond this, augmented and virtual reality technology require sensory input to better place objects and such.

Taking the example of an autonomous vehicle, the detection of objects in 3D is of paramount importance. As autonomous vehicles need to operate in dynamic and fast-changing environments, this capability forms the crux of many safety-critical decision making processes.

Vision-based systems continue to be a popular choice for autonomous vehicle perception, whether it forms the entirety of the perception stack or plays only a limited role within it [3]. As the vast majority of current road infrastructure caters to humans, significant information is conveyed visually. Vision-based systems are uniquely able to leverage this pre-existing wealth of data and, in addition to this, can be relatively inexpensive to deploy owing to the ubiquity of cameras.

To date, typical methods of 3D perception for autonomous vehicles consist of utilising data from LiDAR, RADAR, a suite of cameras or a fusion of these. While effective, utilising data from these sensors can be costly, both in terms of the financial

cost associated with the sensor itself as well as the computational overhead required to process the data into a usable format.

Transforming and projecting 3D information to a top-down representation - a "bird's eye view" (BEV) - of *semantically significant* objects in the scene provides a number of benefits in this regard. For example, compared to a point cloud or an image, a BEV representation occupies less memory, as we are only interested in information at the object level as opposed to low-level information like pixels or LiDAR returns. In addition, it provides an easy-to-understand, interpretable representation of the scene at a given moment in time.

Following the theme of "doing more" with less, it is also possible to extract some estimate of depth directly from imagery. Humans are able to infer depth cues from our pair of eyes which form stereo vision, with our brains internally resolving disparity. Machine vision systems are similarly able to perform such feats, with plenty of examples of commercial stereo vision systems available.

Indeed, depth estimation from imagery has been the subject of a significant amount of research in the community. One area in particular which has seen a lot of work is monocular depth estimation, which as the name implies, aims to extract depth not from stereo pairs but instead from a single image. The benefits of being able to do this accurately are multiple. For one, a reduction in hardware requirements may translate to reduced operational costs. Secondly, reliable monocular depth estimation could act as a redundancy in the event of hardware failures, particularly in multi-camera or multi-sensor systems. However, it is an ill-posed problem and faces many challenges compared to typical stereo imagery.

There is however an implicit bias associated with much of the work so far, in that it typically targets cameras of a certain type: perspective pinhole. This usually arises from the model used and data employed.

1.1 Motivations

As previously stated, it is typically advantageous to be able to perceive the surroundings in some way in order to facilitate decision making. In many existing systems, achieving this can be done via leveraging a suite of expensive and physically cumbersome sensors, such as LiDAR or RADAR. By virtue of being costly, such sensing suites are a barrier to entry for many applications where either cost or size is of paramount importance.

If such sensing can be performed by much cheaper sensors through clever use of software, it could potentially address this issue, democratizing the technology and

encouraging further innovation.

There are a few current problems regarding environmental sensing for mobile robotics and autonomous vehicles. One is an issue of representation and the size associated with it. For example, the data obtained from a LiDAR sensor is often represented as a 3D point cloud which can quickly possess a large memory footprint, and is not a feasible option for large-scale environments, especially on resource constrained hardware.

Another issue is obtaining accurate depth information for the purposes of collision avoidance, mapping and so on. Many mobile robot platforms and autonomous vehicles use LiDAR for this purpose. Some of the advantages such sensors have are that they typically provide extremely accurate data and with a high (often horizontally omnidirectional) field-of-view (FOV). It is easy to imagine why such traits are desirable in the case of autonomous vehicles, which often operate in safety critical and dynamic environments. However, as mentioned they are usually expensive and physically bulky, and the data returned is sparse.

In recent years, cameras with wide FOV lenses are becoming increasingly commercially available. Often, these cameras have an all encompassing, "omnidirectional" FOV of 180° in the vertical and 360° in the horizontal. These cameras are often used for recreational purposes by members of the public.

In comparison to LiDAR, omnidirectional cameras give colour and texture information which LiDAR returns do not. If such cameras can accurately estimate depth, especially with only one camera, they might provide a low-cost sensing modality while maintaining rich information. Of course, many of the issues which plague depth estimation from monocular perspective pinhole cameras also effect omnidirectional cameras. Indeed, often these issues are exacerbated due to the high distortion and FOV. As the price of omnidirectional imaging technology continues to fall, in conjunction with the complete view of the scene that is supplied, this kind of imagery may hold promise for a wide range of robotics applications if it is utilised to its full potential.

1.1.1 Open Challenges

There are currently several open challenges when it comes to accurate sensing for mobile robotics applications, more than what this thesis attempts to address.

Some of these are:

1. Accurate and efficient environmental sensing on constrained hardware.
2. Leveraging the properties of omnidirectional cameras for sensing.
3. Depth estimation using omnidirectional cameras.
4. The comparative lack of high quality, publicly available omnidirectional image datasets.

For example, computing resources can be costly and power hungry, and many robotic platforms simply cannot host such resources onboard.

The current crop of omnidirectional cameras are becoming increasingly affordable and miniaturised, having generally shifted away from more traditional systems based on mirrors to multi-sensor, high field-of-view lens systems. While such cameras see a lot of recreational use, they seldom see much application in research and development. Especially in terms of the quantity of scientific literature regarding monocular depth estimation. The bulk of research so far on monocular depth estimation has been centered on perspective pinhole cameras, which creates an implicit bias toward that camera format.

Specifically because omnidirectional images are visually very different from perspective images, this implicit bias can cause issues. One issue is technical, as many depth estimation models were not developed with spherical geometries and severe distortion in mind. Another is an issue of data. Unfortunately, depth training datasets with accurate depth labels are uncommon for omnidirectional vision, with the majority being synthetic [4–6] and/or indoor scenes [7].

1.2 Contributions

Specifically, for Chapter 3, we draw inspiration from work in guided image generation, synthesis [8] and super-resolution. [9] Instead of solving the problem in a supervised manner by minimising the difference between the output and ground truth, we instead formulate the problem within the context of adversarial learning.

The main contribution of which is an adversarial approach to obtaining a BEV representation of detected vehicles from a monocular image. We seek to leverage the

generative capabilities of a Generative Adversarial Network (GAN) [10], specifically a Wasserstein GAN (WGAN) [11] with gradient penalty (WGAN-GP) [12]. Our model is composed of two sub-networks: a generator network and a critic network. The generator network is tasked with producing BEV representations from an image, while the critic network is designed to assign a "realness" score to this representation, distinguishing a generated BEV representation from its ground truth counterpart. Therefore, the BEV representation produced by the generator network is gradually trained to be similar to the ground truth. We show that a model trained in this way generalises better between datasets compared to baseline, strictly supervised models. In addition, our approach only requires data from a single monocular camera during inference, and does not require more complex sensors such as LiDAR or radar.

Likewise for Chapter 4, we present a monocular depth estimation technique that explicitly incorporates equirectangular geometry into a neural network training process. Firstly, we re-formulating the existing view synthesis based depth estimation models for equirectangular projection, injecting direct knowledge of camera geometry for accurate, dense depth estimation using omnidirectional vision. This is accomplished in part by including "spherical" convolutional layers into the depth estimation model, which takes into account the substantial visual distortion for the purposes of convolutional operations. Furthermore, we offer an optical flow-based efficient masking method to significantly reduce the noise added to the loss by "noisy" pixels produced by the full field of vision. Such as noisy pixels often result from wide, texture-less areas or moving objects. Compared with existing work in this area which mostly targets generated indoor scenes, to the best of our knowledge, this is the first work which demonstrates self-supervised learning-based, monocular omnidirectional vision for accurate depth estimation in real outdoor scenarios. This approach does not depend on ground truth depth data for training, which is rarely available for omnidirectional images. Experiments on two public datasets show state-of-the-art monocular depth estimation accuracy using omnidirectional videos.

Lastly, for Chapter 5, we propose an extrinsic calibration method for aligning a LiDAR sensor with an omnidirectional camera based on Bayesian optimisation. Additionally, we introduce a weighted loss function aimed at incorporating LiDAR returns as ground truth depth for omnidirectional depth training, which we call the Grid Loss. These contributions are used to extend the work shown in Chapter 4.

This thesis makes the following published contributions:

- The work undertaken in Chapter 3 in generating a BEV representation from a single image has been peer reviewed and accepted for publication at the *25th*

International Conference on Pattern Recognition (ICPR 2020).

- The work undertaken in Chapter 4 for a method of estimating depth from single omnidirectional image has been peer reviewed and accepted for publication at the *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2022).*

1.3 Outline

This thesis is structured as follows:

- Chapter 1: this chapter serves as an introduction and contextualises the work to follow, including an outline of motivations, open challenges and contributions.
- Chapter 2: this chapter details the necessary preliminary information which the thesis is built on as well as a review on relevant literature.
- Chapter 3: this chapter describes the work undertaken on generating a bird's eye view representation from a single image, including the methodology employed and experiments performed.
- Chapter 4: this chapter describes the work undertaken on omnidirectional monocular depth estimation, including details on methodology, data collection and experiments.
- Chapter 5: this chapter describes an extension to the methodology outlined in the previous chapter, incroporating LiDAR data to the training pipeline.
- Chapter 6: this chapter concludes the thesis, briefly summarising the important points made and results obtained.

2 | Background

2.1 Preliminary

This chapter will serve as a primer on the necessary background information and methods for the work contained within this thesis, as well as a review of the related state of the art literature.

2.1.1 Pinhole Camera Geometry

It is first important to discuss the fundamentals of camera geometry and image formation as a prerequisite to the work discussed in this thesis.

We assume that there is a coordinate system in the "world frame" in order to depict the environment surrounding the camera. We can express every single point as a three-dimensional point using this world frame coordinate (x, y, z).

Because an image is two-dimensional, it may be created by expressing the three-dimensional coordinates in the world frame in a coordinate frame centred on the picture – an image frame.

Image formation is based on capturing light rays that are reflected from an object in the world frame onto a medium in this image frame. One could simply place the medium in front of the object to "capture" it, but there would only be grey on the medium. This is due to the light rays from various points on the object overlapping on the medium, becoming incoherent.

To prevent this incoherence and achieve a one-to-one correspondence between the world points and the medium, a barrier with a small hole (a pinhole, or aperture) might be positioned between the object and the medium. This allows light rays to pass through the surface at one point. Due to the fact that objects in a scene reflect light rays in all directions, only one of these rays can travel through the aperture. In actuality, this is not the case and is more like a narrow beam of light is often formed by a collection of rays. Regardless, by doing this, it ensures that there is little overlap, though the image formed on the medium would be inverted. At a very

basic level, this is also how the human eye functions.

This type of projection involves converting three-dimensional points to two-dimensional points. In this scenario, we refer to it as forward projection.

Typically, this is accomplished through the use of a camera model. Described previously is the pinhole camera model at a basic level.

In order to mathematically model this, we must define a coordinate system. Allow the centre of projection, O, to be the origin of this coordinate system and is precisely at the centre of the pinhole. The Z axis is positive toward the perpendicular of the pinhole, "away" from the camera. The X and Y axes are along the plane of the pinhole surface.

We can then define an image plane (or projection plane), I, in this system as $Z = -f$, where f is the focal length and is the distance between the plane of the pinhole and the image plane.

Given some point $P = (X, Y, Z)$ in the world, we can derive the projected point, $P' = (x, y)$, on the image plane I as:

$$x = -f\frac{X}{Z}, y = -f\frac{Y}{Z}, \tag{2.1}$$

Here, we've defined the image plane to be behind the aperture, and thus form an inverted image. We can remove this inversion by using a "virtual" pinhole camera where the image plane is in front of the aperture, defined by $Z = f$. Now, the projection is defined as:

$$x = f\frac{X}{Z}, y = f\frac{Y}{Z}, \tag{2.2}$$

We can go a step further and use homogeneous coordinates to model an *ideal* pinhole camera by:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2.3}$$

Where coordinates X, Y, Z, x, y and the focal length, f, are defined in meters. The fourth column of the camera projection matrix represents the perspective division, which transforms a 3D point in the world coordinate system to a 2D point in the image coordinate system. This column is used to perform a division step that accounts for the fact that objects that are farther away from the camera appear smaller in the image. The division step scales the x and y coordinates proportion-

ally to the Z component, so that the resulting 2D coordinates accurately reflect the relative distances of the objects in the image. However, it is significantly more practical if coordinates are measured in pixels and pixel distances. To convert to pixel distances, we can introduce scale factors s_x and s_y:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} s_x f & 0 & 0 & 0 \\ 0 & s_y f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2.4}$$

Since x and y are now expressed in pixel coordinates, the focal lengths are often denoted as f_x and f_y respectively:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & 0 & 0 \\ 0 & f_y & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2.5}$$

Additionally, while in theory the origin of the coordinates are centered precisely at the aperture centre, in practice this is not often the case. Hence we can add translation component to the matrix, like so:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & u_0 & 0 \\ 0 & f_y & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2.6}$$

To increase generalizability, we can apply a skew factor, s, which compensates for any shear that may result from the optical axis not being precisely perpendicular to the aperture:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & s & u_0 & 0 \\ 0 & f_y & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2.7}$$

Thus, a pinhole camera can be geometrically modelled, where K is the intrinsic matrix:

$$K = \begin{bmatrix} f_x & s & u_0 & 0 \\ 0 & f_y & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{2.8}$$

However, the issue with this matrix is that it implies we can represent locations in three-dimensional space, in coordinates relative to the camera frame in which the Z axis is the optical axis. Unfortunately, it is difficult, if not impossible, to identify the camera's geometry in practice. Thus, we cannot determine what the coordinate axes are, short of opening up a camera to find out.

Let us assume we know the three-dimensional coordinates of a point – $P = (X, Y, Z, 1)^T$ – relative to an arbitrary frame, which we will refer to here as the world frame. The camera frame is simply a transformed version of this world frame, and a point in the world frame can be transformed into the camera frame by an extrinsic matrix:

$$P_{cam} = \begin{bmatrix} R & t \end{bmatrix} P \tag{2.9}$$

Where R is a $3x3$ rotation matrix, and t is a $3x1$ translation matrix. Hence the general mapping of a pinhole camera in the world frame is defined by:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = K \begin{bmatrix} R & t \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2.10}$$

The *intrinsic matrix*, K, defines the internal parameters of the camera and is fixed. The *extrinsic matrix*, $[R\,t]$, defines the external camera orientation and translation with respected to a world frame. Hence, a full geometrical model is obtained and a relationship between three-dimensional points in the world and an imaging medium can be described.

Fisheye calibration

It is also important to discuss fisheye calibration. Omnidirectional cameras are often composed of multiple sensors with their own lenses, in effect becoming a miniature multicamera system. These sensors are often coupled with wide-angle (or fisheye) lenses and the resulting images are stitched together.

Just as regular, perspective cameras can diverge from the ideal pinhole model, so too can cameras using fisheye lenses. In much the same way, these require undis-

tortion.

A point to note is that there is no *single* fisheye projection, but several. These are (confusingly) refered to as fisheye by various lens manufacturers.

The ideal fisheye projection is the *equidistant* projection. In this model, there is a direct relationship between the distance of pixels in an image and the light's angle of incidence:

$$R = f\theta \tag{2.11}$$

Where R is the radial distance of pixel from the principal point, f is the focal length and θ is the angle of incidence.

There are variious methods of incorporating distortion into the model above. A popular method is that incorporated by the OpenCV library, where θ in the projection equation is replaced by:

$$\theta_d = \theta + k_1\theta^3 + k_2\theta^5 + k_3\theta^7 + k_4\theta^9 \tag{2.12}$$

Where k_n denotes the radial distortion.

2.1.2 Equirectangular Camera Geometry

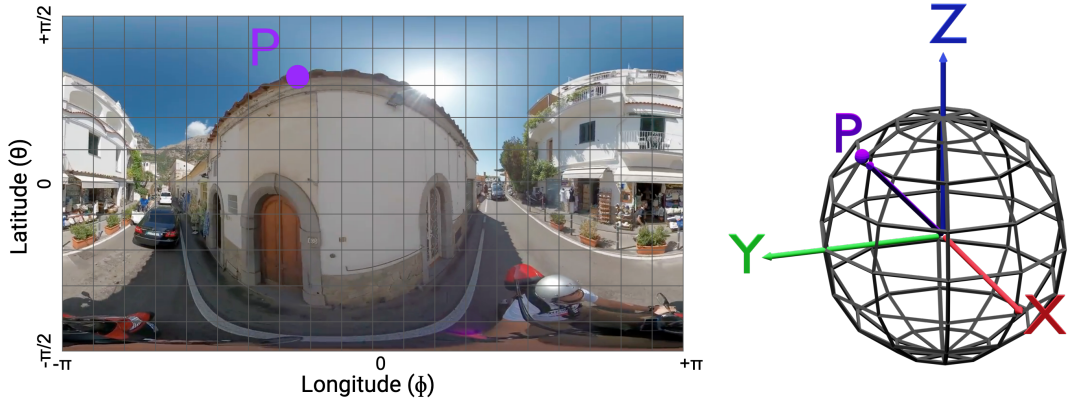


Figure 2.1: Equirectangular camera projection model

The equirectangular image format is one of several omnidirectional image formats and is quite popular, seeing usage within computer graphics and panoramic photgraphy. As such, many omnidirectional camera hardware produces images in such a format, so it is critical to understand and model.

This image format encapsulates a complete field-of-view, offering horizontal and vertical perspectives of 360° and 180°, respectively. Despite usually being captured through multiple sensors, the comprehensive scene coverage allows us to perceive it as if it were derived from a solitary "virtual" spherical sensor.

We can divide the process into two main steps: forward projection and backprojection

In our model, we propose that backprojection serves to transpose points from the sphere's surface to world coordinates, with forward projection being the inverse process.

Starting with the forward projection, we begin by drawing a ray from the world point to the center of the virtual spherical sensor

Thus, the projection of world to pixel coordinates, that is, forward projection, can be modeled as the intersection of a ray with the sphere's surface.

By introducing a unit sphere, the intersection point can be succinctly described via the ray's latitude (angle relative to the zenith/polar axis) and longitude (angle of rotation around the zenith). This model is illustrated in Figure 2.1, where a pixel, denoted by P, can be characterized in latitude-longitude coordinates and corresponded to a distinct point on a unit sphere.

Next, we find the intersection of this ray with the surface of our unit sphere.

Given the pixel dimensions of the equirectangular image, we can establish a

relationship between pixel coordinates and latitude-longitude coordinates. Once the latitude and longitude for each ray are known, we can identify the respective pixel location on the image. Specifically, a transformation from a Cartesian world point to a spherical latitude-longitude coordinate (considering a unit sphere) can be represented as follows:

$$\begin{bmatrix} \theta \\ \phi \end{bmatrix} = \begin{bmatrix} arctan\frac{\sqrt{x^2+y^2}}{z} \\ arctan\frac{y}{x} \end{bmatrix} \tag{2.13}$$

The above equation represents the transformation from Cartesian to spherical coordinates.

Here, θ symbolizes latitude, ϕ represents longitude, and x, y, z are the world points.

Once we have established a relationship between pixel coordinates and latitude-longitude coordinates, we can progress to the second step: backprojection.

For the inverse process, we handle each point, designated as $P = (x, y, z)$, for backprojection as if it resides on the surface of a sphere with radius r, at latitude θ and longitude ϕ. To streamline calculations, we adjust the central meridian and parallel by π and $\frac{\pi}{2}$ respectively, thus setting $\theta \in [0, \pi]$ and $\phi \in [0, 2\pi]$.

Here, we reverse our process by moving from spherical coordinates back to Cartesian. We calculate the position of a point on the sphere's surface, given its latitude and longitude.

For a pixel $p = (u, v)$ within an image of dimensions w, h, where $u \in [0, w]$, $v \in [0, h]$, P can be calculated as per the set of Equations 2.14, where r is the depth at p.

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} r\ cos(\frac{2\pi u}{w})\ sin(\frac{\pi v}{h}) \\ r\ sin(\frac{2\pi u}{w})\ sin(\frac{\pi v}{h}) \\ r\ sin(\frac{\pi v}{h}) \end{bmatrix} \tag{2.14}$$

Thus, we have achieved a transformation from pixel to world coordinates.

Consequently, employing Equations (2.13) and (2.14), we can construct a comprehensive model of a "virtual" equirectangular camera. This model serves as a robust framework, bridging the gap between world coordinates and pixel coordinates, hence facilitating an understanding of the image formation process in its entirety.

2.1.3 Camera Calibration

The science of camera calibration has been extensively studied [13] [14], with popular software packages available for use [15] [16].

In order to use the model described previously, it is essential that the parameters be determined. This can be done via calibration. In the context of imaging, *calibration* can mean different things: *colour* calibration, or *geometric* calibration. Since colour calibration is outside the scope of this thesis, we will focus only on *geometric* calibration.

Primarily, camera calibration involves identifying the internal camera parameters which influence the imaging process. There are several factors which will be taken into account and estimated: the principal point, the focal length, the scaling factors and the skew factor.

The principal point is the position of the centre of the image. Ideally, this will be directly in the centre of the frame. In practice, due to inevitable manufacturing inaccuracies, this is usually not the case. Figure 2.2 shows the effect of changing this value. Varying the principal point effects the intersection of the center line (green) with the focal plane (shown in yellow), where the top row shows a changing value in the x-axis and the bottom row shows a changing value in the y-axis.

The focal length is the distance between the optical centre of the aperture and the sensor, Figure 2.3 shows the effect of changing this value, where Changing the focal length changes the focal plane (shown in yellow) and effects the field-of-view of the camera. The top row depicts a low focal length value, while the bottom row shows a high focal length value.

The scaling factors are coefficients needed to convert from metric distances to pixel distances. Typically, this is bundled with the focal length, so that the focal length is expressed in pixel units.

The skew factor is the parameter which handles the amount of shear present. Figure 2.4 visualizes possible skew, which causes the focal plane (shown in yellow) to be non-rectangular.

The camera matrix described previously is only valid for an *ideal* pinhole camera. An ideal pinhole camera does not have a lens, and in practice the vast majority of cameras have one. Lenses inherently have distortion, and to accurately represent a real camera, the model must account for radial and tangential lens distortion.

Radial distortion results from light rays bending more towards a lens's borders than they do at its optical centre, with smaller lenses producing larger distortion. This is modelled by radial distortion coefficients. The distorted points are computed

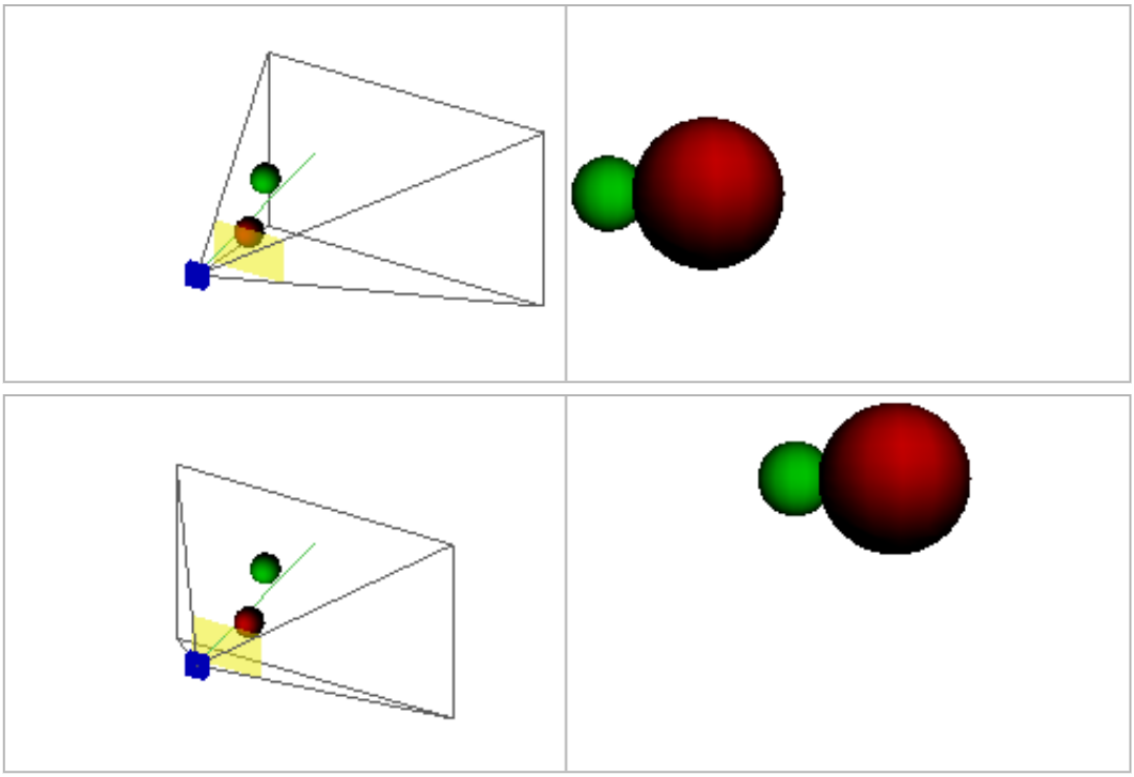


Figure 2.2: Effect of change in principal point.

as such:

$$x_{distorted} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{2.15}$$

$$y_{distorted} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{2.16}$$

Where:

- x and y are the undistorted pixel locations in normalized coordinates. These coordinates are determined from pixel coordinates by translating to the optical center and dividing by the focal length in pixels.
- k_1, k_2, k_3 are the radial distortion coefficients of the lens.
- $r^2 = x^2 + y^2$

In most circumstances, two distortion coefficients are adequate, but in cases of extreme distortion, three may be necessary (such as in wide-angle lenses).

Tangential distortion results from the lens and image plane not being parallel. Similarly, these are modelled by coefficients which describe the degree of distortion.

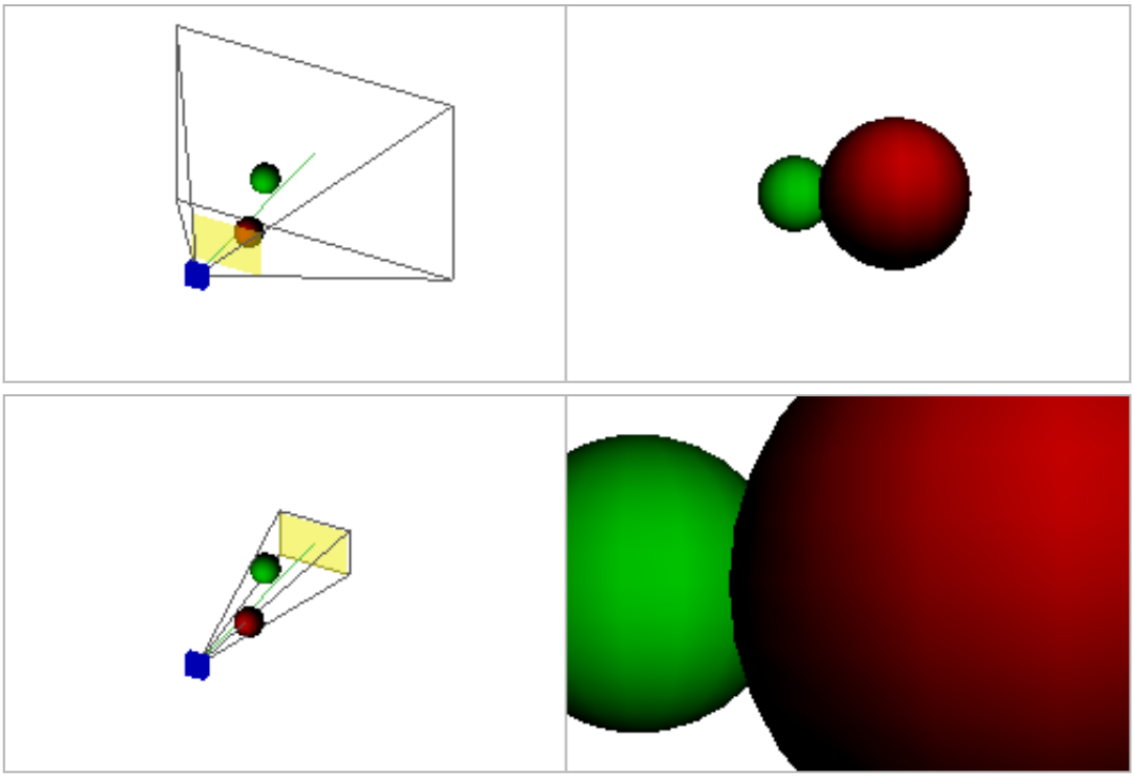


Figure 2.3: Effect of change in focal length.

The distorted points are computed as such:

$$x_{distorted} = x + (2 * p_1 * x * y + p_2 * (r^2 + 2 * x^2)) \tag{2.17}$$

$$y_{distorted} = y + (2 * p_2 * x * y * p_1 * (r^2 + 2 * y^2)) \tag{2.18}$$

Where:

- x and y are the undistorted pixel locations in normalized coordinates.
- p_1, p_2 are the tangential distortion coefficients of the lens.
- $r^2 = x^2 + y^2$

2.2 Neural Networks

As artificial neural networks (ANN) are one of the fundamental methods used in this thesis, it is necessary to understand how they operate.

These machine learning algorithms are loosely based on the behaviour of biological neural networks observed in humans and animals. In general, they try to

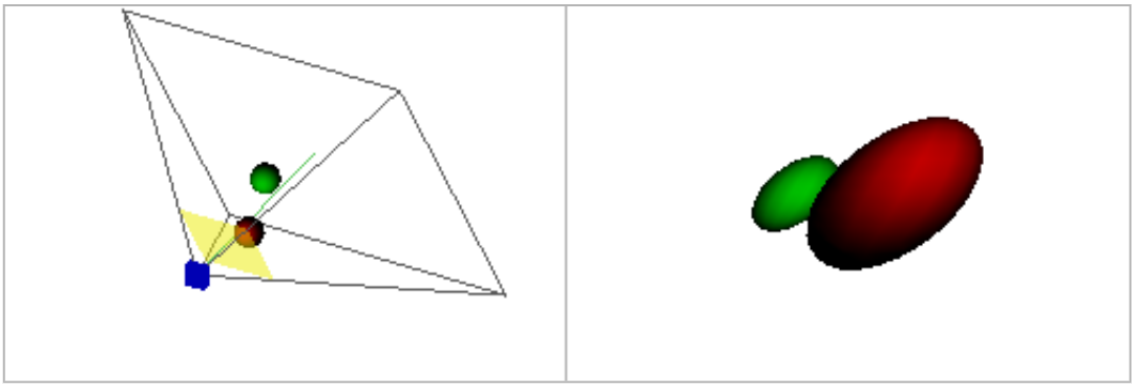


Figure 2.4: Effect of change in skew.

identify underlying correlations in a dataset by imitating how these biological networks function. Typically, such systems learn to execute tasks by analysing samples, without being explicitly designed with specific rules. For the remainder of this thesis, artificial neural networks will simply be referred to as neural networks.

Neural networks are comprised of many basic building blocks called *neurons*. Such neurons are simple but together are able to perform more complex operations.

2.2.1 Perceptron

Before diving deeper into more complex structures, let us look at a simple neural network model: perceptron.

The first and simplest neural network model, the perceptron, was devised as a mechanism for supervised learning. This network is considered elementary since it has just two layers: an input layer and an output layer. This structure has a single weight matrix, and all input layer units are connected to output layer units. The perceptron is a linear classifier for binary predictions, therefore it can classify or divide data into two categories.

The basic perceptron initially receives n input values $(x_1, x_2, ..., x_n)$ and is defined by $n + 1$ constants:

- n synaptic coefficients (weights: $w_1, w_2, ..., w_n$)
- Bias: a neuron with an activation function equal to one. As with other neurons, a bias interacts with the neurons of the layer before it via a weight.

Fundamentally, it operates on three basic steps, and an outline is shown in Figure 2.5. First, every input value is multiplied by their connection weight. Secondly, the result of all of these operations are summed to produce a *weighted sum*. Finally, this

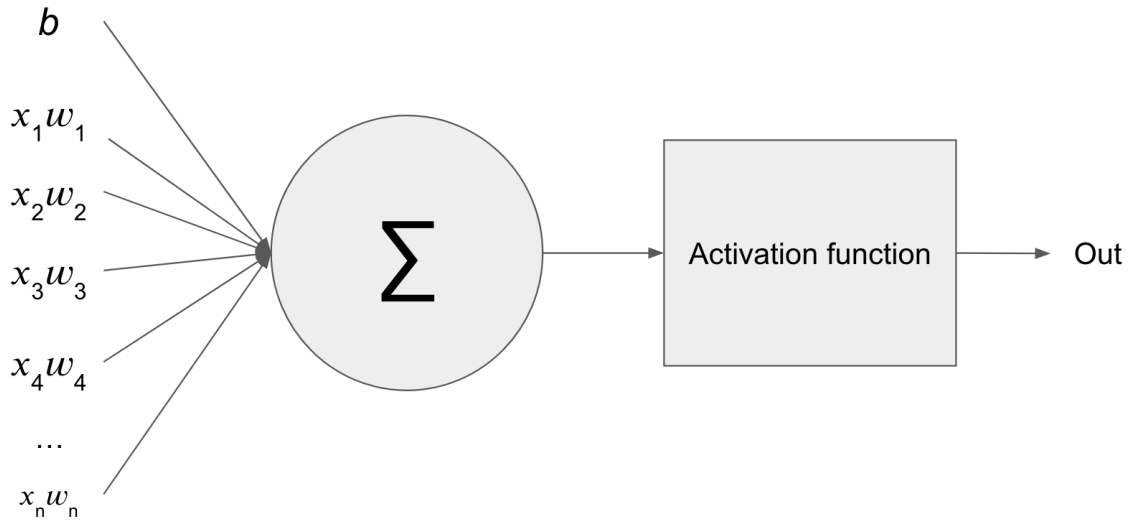


Figure 2.5: Perceptron model

sum is passed as input to an activation function, which tells us whether the neuron has "fired" or not. The bias term lets us alter the behaviour of this activation function, moving it up or down and acting as a threshold of sorts.

2.2.2 Multi-layer Perceptron

Consider, for a moment, a factory production line. On this production line, the first worker receives an item, modifies it, and then forwards it on to the next. Similarly, this next worker performs some action or modification, to then forward it on to the subsequent worker in the line. This procedure is continued until the final worker on the line completes the product, which is then exported out of the factory. In this model, the production line is composed of a sequence of steps, and items move between these steps as they are passed from worker to worker.

Following this analogy, a multi-layer perceptron is comparable to an assembly line with three steps: an input step, an intermediate (or hidden) step, and an output step. In the terminology, these steps are called 'layers'. Each layer has a predefined number of workers, or neurons.

Any input data is delivered to the MLP via the input layer, and a number of neurons handle the data in the hidden layer before transferring it to the output layer, which passes the final value(s) as output. These neurons modify input data via mathematical functions. Figure 2.6 depicts a simple MLP showing three inputs, a five node hidden layer and a single output.

There are weighted connections between network neurons (or nodes). These weights regulate the transmission of information from neuron to neuron. In other words, weights characterise the level of effect one neuron has on another.

When data leaves a node, it is conditioned by the connection weight and then modified by a "activation function", a mathematical function. Activation functions modify the input in a non-linear manner, enabling the network to learn complex data representations. The input values are multiplied by the weight values, followed by the addition of a bias term.

Multilayer perceptrons are commonly employed in supervised learning problems: they are trained on a set of input-output pairs and learn to reflect the correlations (or dependencies) between those inputs and outputs. Training involves adjusting the model's parameters, such as weights and biases, to reduce error. Backpropagation is used to adjust the weights and biases relative to the error, which may be evaluated using a number of techniques, including root-mean-squared error (RMSE).

MLPs, which are feedforward networks, are comparable to a game of tennis or ping pong. They operate largely in two phases, a continuous back and forth. This back-and-forth of guesses and replies may be viewed as a type of accelerated research, since each guess is a test of what we assume we know, and each response is feedback indicating how wrong we are.

In the forward pass, the signal flows from the input layer to the hidden layers to the output layer, and the judgement of the output layer is compared to the ground truth labels.

Backpropagation and the chain rule of calculus are employed in the backward pass to back-propagate partial derivatives of the error function with respect to the various weights and biases via the MLP. This procedure of differentiation provides a gradient, or error landscape, along which the parameters may be adjusted to get the MLP closer to the minimum error. This may be achieved via any gradient-based optimization approach, such as stochastic gradient descent. The network continues to engage in this tennis match until the error can no longer be decreased. This is called convergence.

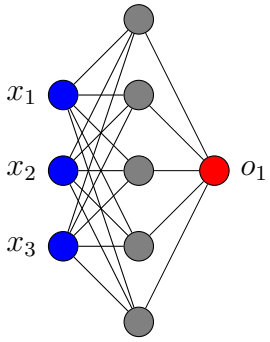


Figure 2.6: Example structure of a multi-layer perceptron.

Deep neural networks extend upon the MLP model by adding additional hidden layers to its core. Instead of an input layer, a hidden layer, and an output layer, there are several hidden layers in the middle, and the outputs of one hidden layer become the inputs for the next hidden layer until the data has been returned from the network's final output layer.

Several hidden layers of a deep neural network may grasp more complex patterns than a normal multilayer perceptron. Multiple layers of the deep neural network learn the patterns of diverse input segments. For instance, if the input data consists of images, the initial layer of the network may interpret the brightness or darkness of pixels, while successive layers may recognise shapes and edges that may be used to differentiate objects in the image.

However, MLPs do not scale well to larger images, requiring an exponentially larger number of connections as the image dimensions increase.

A much more fitting class of neural networks are *convolutional neural networks*.

2.2.3 Convolutional Neural Networks

Convolutional neural networks (CNNs) are a class of neural network design most frequently utilised in the development of learning-based computer vision systems. A CNN's structure allows them to process image information, transforming it into numbers which can be better processed by neural networks.

CNNs are built upon the convolutional layer. A convolution is a mathematical operation which merges sets of data. In this case, input data is *convolved* using a convolution filter to build a *feature map*.

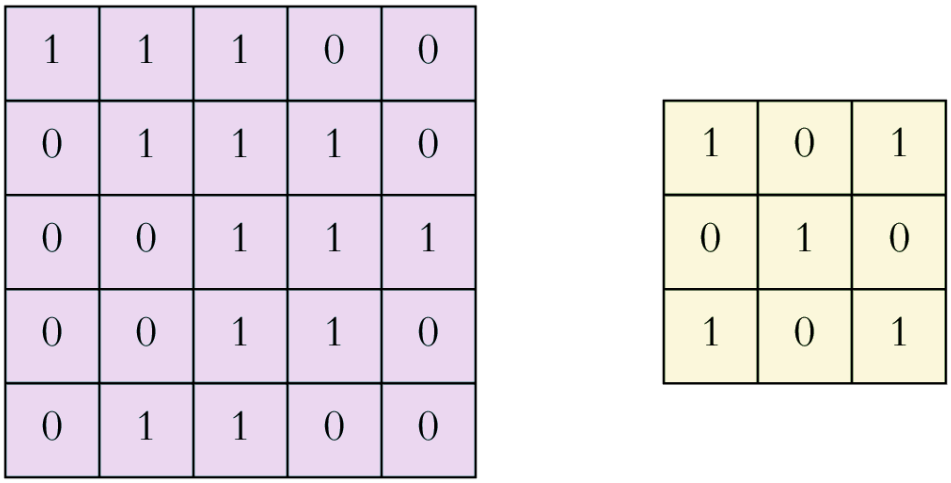


Figure 2.7: Example input (left) and example convolution filter (right)

Figure 2.7 depicts an example input and an example convolution filter. The input to the convolution layer, which could be input image for example, is located on the left. On the right is the convolution filter, which we shall refer to interchangeably as the kernel. Due to the filter's structure, this is known as a 3x3 convolution.

Convolution is achieved by applying this filter to the input. We multiply matrices element by element and total the results at each point. The receptive field of a neuron in a CNN is the region in the input feature map that has a direct influence on the output of that neuron. It is defined as the set of input pixels that contribute to the activation of that neuron. Given the size of the filter, the receptive field is also 3x3.

Shown in Figure 2.8 is the filter positioned in the top left corner of the input, and the result of the convolution operation (4) is displayed in the resultant feature map.

Then, we shift the filter to the right and repeat the operation, appending the result to the feature map, as shown in Figure 2.9.

This continues until all elements of the input have been covered and the feature map is complete, as shown in Figure 2.10. A 2D illustration of a convolution process

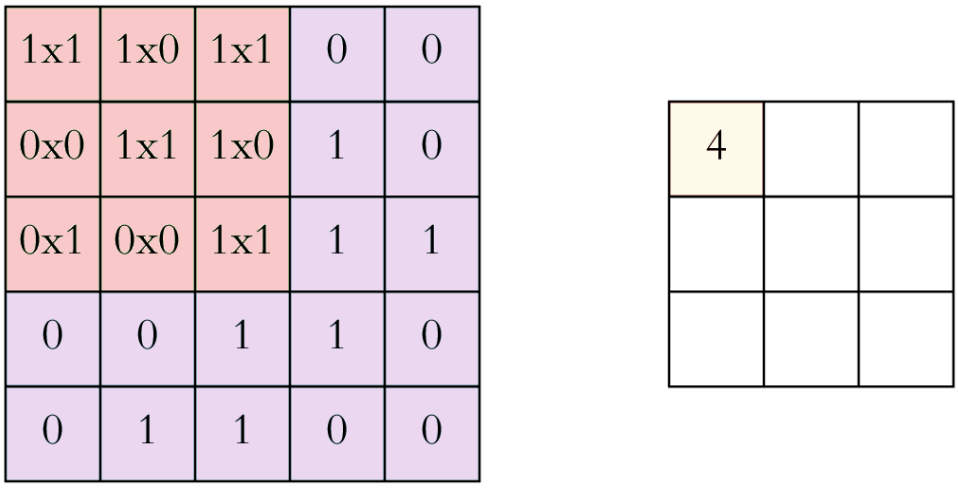


Figure 2.8: Convolution example: first step

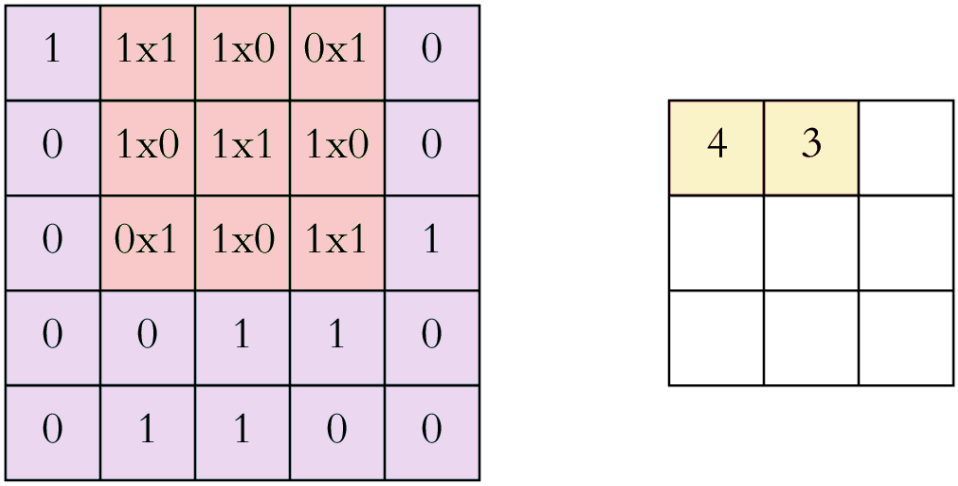


Figure 2.9: Convolution example: second step

with a 3x3 filter was shown. Yet, these convolutions are often implemented in 3D. An image is really represented as a three-dimensional matrix with height, width, and depth dimensions, where depth corresponds to colour channels (RGB). A convolution filter has a fixed height and width, such as 3x3 or 5x5, and must also be 3D because it encompasses the whole depth of its input.

Typically, many convolutions are executed on an input, with each convolution employing a separate filter and yielding a unique feature map. The result of stacking all of these feature maps is the final output of the convolution layer. First, let's depict a single-filter convolution.

Consider an input of 32x32x3 with a 5x5x3 filter, as shown in Figure 2.11. As can be seen, the depth of the image and convolution kernel are both 3. When the filter is in a specific position, it covers a restricted volume of the input, and the

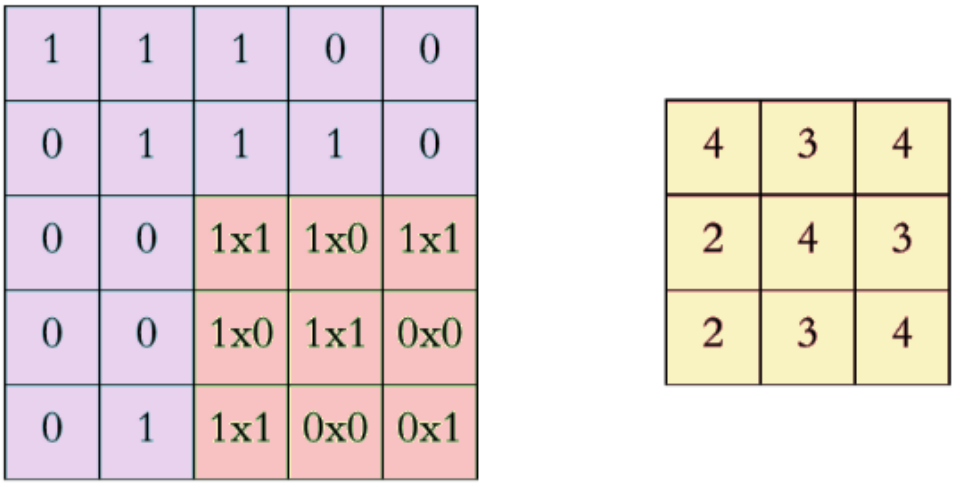


Figure 2.10: Convolution example: final

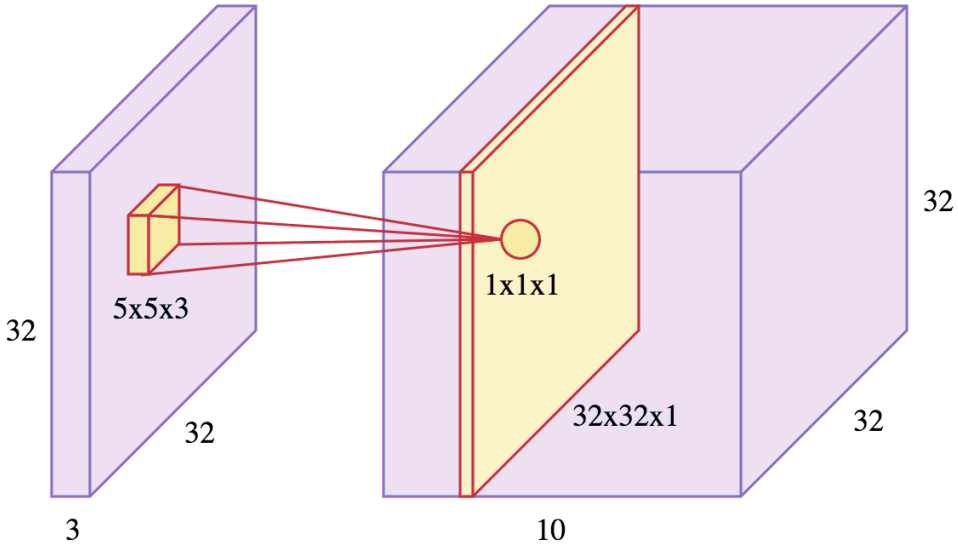


Figure 2.11: Convolution with additional dimensions

convolution procedure outlined previously is carried out. The only difference is that the summation of matrix multiplication is performed in 3D rather than 2D, but the output is still scalar. As explained previously, we slide the filter across the input and execute convolution at each location, accumulating the results in a feature map. As indicated by the red slice on the right, the dimensions of this feature map are 32×32x1.

Suppose ten distinct filters were utilised, resulting in ten 32x32x1 feature maps. By stacking them along the depth dimension, we would obtain the final result of the convolution layer: a 32x32x10 volume, as seen on the right. Due to padding, however, the height and width of the feature map remain identical at 32 pixels.

Padding will be explained at a later stage.

Similar to the 2D case, the kernel is shifted along the input and a scalar is acquired at each place; this scalar is then aggregated in the feature map.

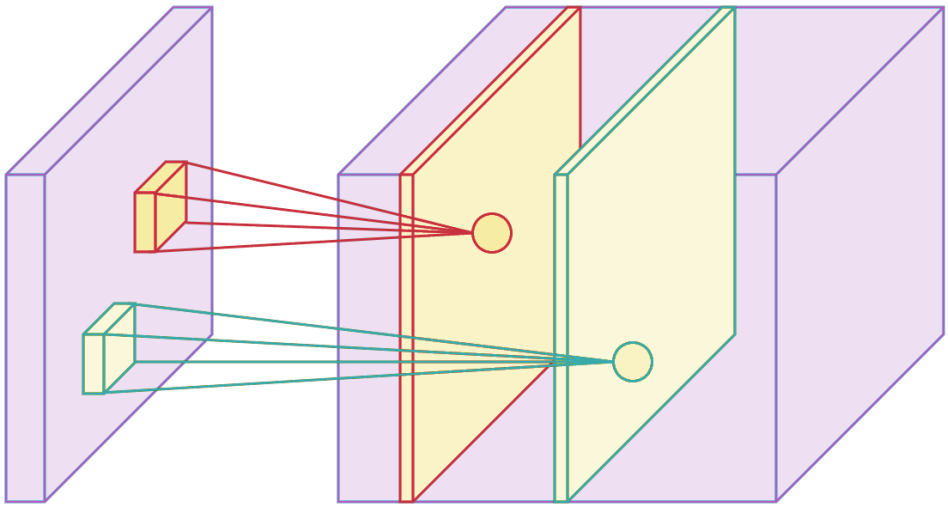


Figure 2.12: Visualisation of multiple feature maps

As illustrated in Figure 2.12, two feature maps are layered along the depth axis. Each kernel's convolution process is executed individually, resulting in separate feature maps.

For any neural network to be effective, it must incorporate nonlinearity. Similarly to an MLP, this is accomplished by applying an activation function to the weighted sum of its inputs. Therefore, the values in the final feature maps are not a true sum of the input data, but rather of the activation function that was applied to them. This was excluded from the preceding figures for simplicity.

There are some parameters which broadly control how convolution kernels behave. One such parameter is *stride*. Stride determines the amount by which the kernel is moved with each step. In the previous examples, a stride of one was used. In other words, the kernel moves one element at a time. If less overlap is sought between the receptive fields, it is possible to specify larger strides. This also reduces the size of the resultant feature map, since some locations are omitted.

If we wish to avoid a drop in the size of the feature map, we might surround the input with extra cells. This is known as padding.

Shown in Figure 2.13 is an example of how padding functions. The grey area surrounding the input represents the padding. Typically, padded values are either zero or the value of the adjacent cell. In this instance, the dimension of the feature map corresponds to the input. CNNs frequently utilise padding to maintain the

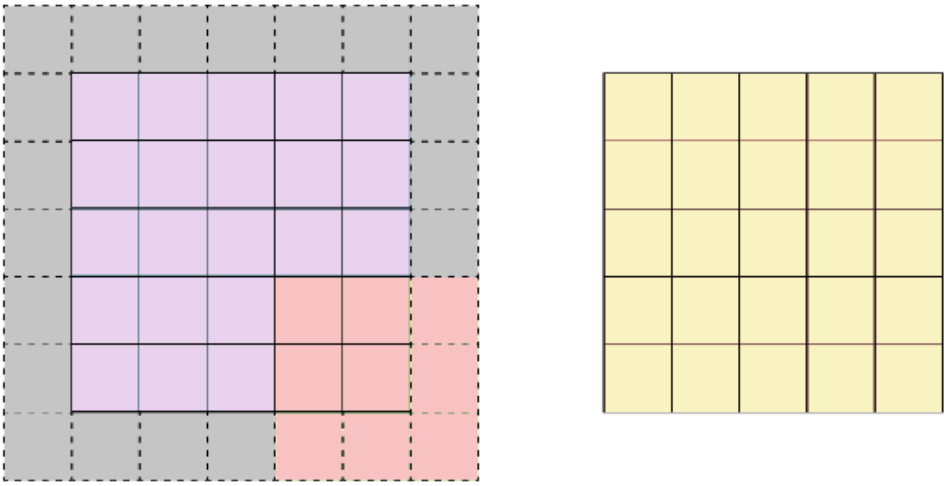


Figure 2.13: Visualisation of padding to obtain an equal dimension feature map

size of their feature maps. Otherwise, they would contract with each layer, which might be detrimental in some circumstances. The previously displayed convolution example figures utilised padding, resulting in the feature map having the same size as the input, with only the depth changing.

Another often employed operator is pooling. Typically, pooling layers are used following convolution to reduce dimensionality. This decreases the number of parameters, lowering training time and mitigating overfitting as a result. Each feature map is separately downsampled, resulting in height and breadth reductions while keeping depth.

A common technique of pooling is max pooling, which takes the maximum value in the pooling area. In contrast to convolution, pooling has no parameters. It just moves a window over its input and reads in the maximum value.

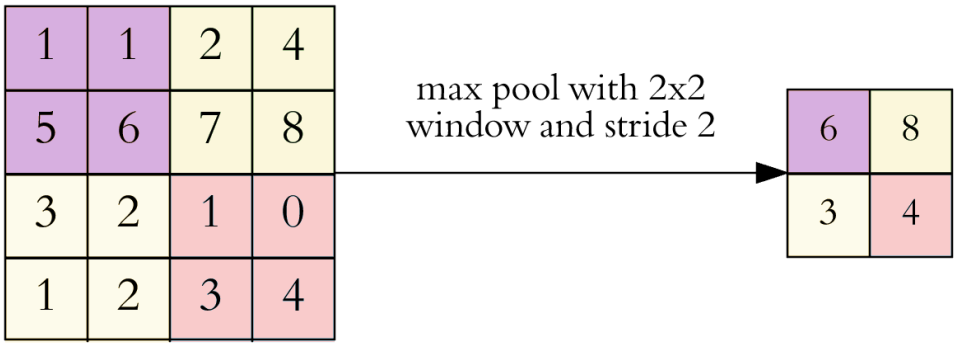


Figure 2.14: Visualisation of maximum pooling

Shown in Figure 2.14 is the result of max pooling using a 2x2 window with a stride of 2. Each colour represents a different window. Since the window dimensions

and stride are equal, the windows do not overlap.

This setup of window size and stride effectively halves the size of the feature map. The primary application of this operator is to downsample a feature map while retaining meaningful data. By cutting the height and width in half, we see a reduction in the number of weights to a quarter of the original input. Given that CNN systems typically include millions of weights, this reduction is considerable.

2.2.4 Common Network Architectures

Now that the basic "building blocks" have been described, our attention can now turn to describing some commonly used architectures.

Although the challenge of classifying objects is not new, researchers continue to work on it. LeNet-5 [17], proposed by LeCun et al. in 1998, was one of the first deep neural networks to address the optical character recognition problem. One of the primary reasons for this model's appeal was its design, which is clear and easily understandable.

AlexNet [18] was proposed to the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [19] by Krizhevsky et al. in 2012. AlexNet has a total of eight layers, including five convolution layers and three fully-connected layers. In contrast with LeNet, AlexNet employs extra kernels in each convolutional layer. The kernel sizes used by AlexNet were 11x11, 5x5, and 3x3. The network contains approximately 62 million parameters and was trained in parallel on the ImageNet dataset using two GPUs. That year, AlexNet won the ILSVRC competition with top-1 and top-5 accuracies of 57% and 80.3% respectively.

After AlexNet, VGG-16 [20] won the ImageNet (ILSVRC-2014) classification competition. There are several differences between AlexNet and VGG-16.

VGG-16 contains additional convolution layers, which showed a growing trend in which researchers were increasingly concentrating on increasing network depth. Additionally, only 3x3 kernels were used and as a consequence these smaller kernels are able to extract finer details from input data.

The architecture of the network consists of five blocks in total, and is shown in Figure 2.15. Each of the first two blocks of the network consists of two convolution layers and one max pooling layer. Each of the remaining three blocks consists of three convolution layers and one max pooling layer. Following block number 5, three fully connected layers are added to the network: the first two layers comprised of 4096 neurons, while the third layer comprised of 1000 neurons, conforming with the ImageNet class distribution. In addition, about 100 million of the network's 138

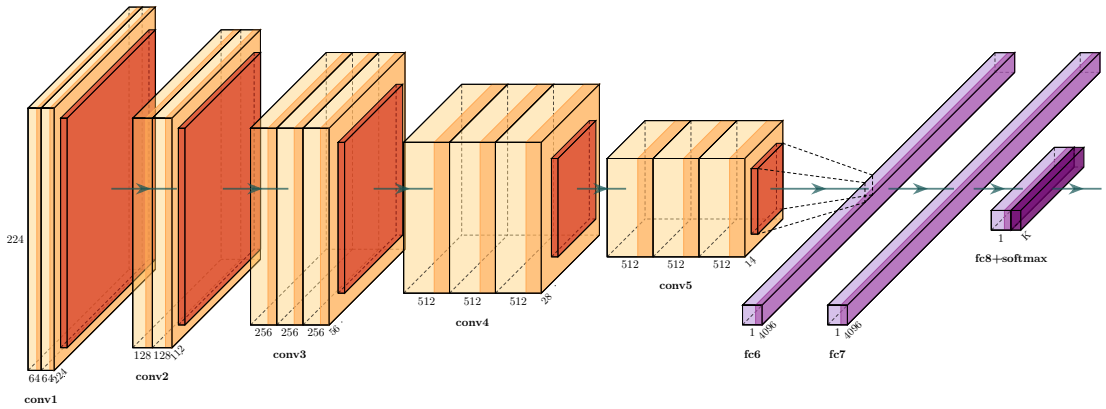


Figure 2.15: Visualisation of VGG-16 architecture. Convolution blocks are denoted by the labels 'conv1' to 'conv5'. Each block contains one or more convolution layers (denoted in yellow). Each of these are followed by a ReLU activation (denoted in orange). At the end of each block, the outputs are pooled (denoted by dark orange). Fully connected layers are denoted by the labels 'fc6' to 'fc8', in purple. Similarly, each of these are followed by a ReLU activation (denoted in dark purple).

million parameters are located in the first two fully-connected layers of VGG-16. The top-one and top-five accuracy of VGG-16 was 71.3% and 90.1%, respectively.

What these works highlight was a trend to make network architectures deeper in terms of layers in the pursuit of better performance. The reasoning was that deeper networks may capture more complex features, enhancing the robustness and effectiveness of models. Hence, explaining the breakthrough levels of accuracy seen in such competitions.

Researchers observed, however, that merely adding more layers did not work beyond a certain point. While training deeper networks, the issue of accuracy decline was observed. In other words, the addition of more network layers either saturated the network or drastically dropped the accuracy score. The accuracy loss was due to the vanishing gradient effect, which can only be observed in networks with greater depth.

The cause of this effect is that when additional layers with particular activation functions are added to neural networks, the gradients of the loss function approach zero, which increases training difficulty. For instance, the sigmoid function compresses an input into a range of 0 to 1. The effect of this is that a significant change in the input to the sigmoid function will result in a relatively minor change in the output. Consequently, the derivative trends towards increasingly smaller values.

Using activation functions in neural networks with only a few layers is typically

not a major concern. However, as the architecture becomes deeper, incorporating more layers, the gradient can become smaller and more difficult to manage.

While smaller networks with only a few layers and these activation functions may not exhibit significant problems, deeper networks are more likely to suffer from the vanishing gradient issue. This can make training the network more challenging and can lead to suboptimal solutions.

A method called backpropagation is utilised to compute neural network gradients. This method calculates the network's derivatives by traversing each layer in reverse order and using the chain rule.

In theory, this works fine. The problem arises when multiple small derivatives are multiplied together, which causes the gradient to exponentially decrease. This means the weights of the initial layers will not be efficiently updated with each training pass and leads to training difficulty and inaccuracies.

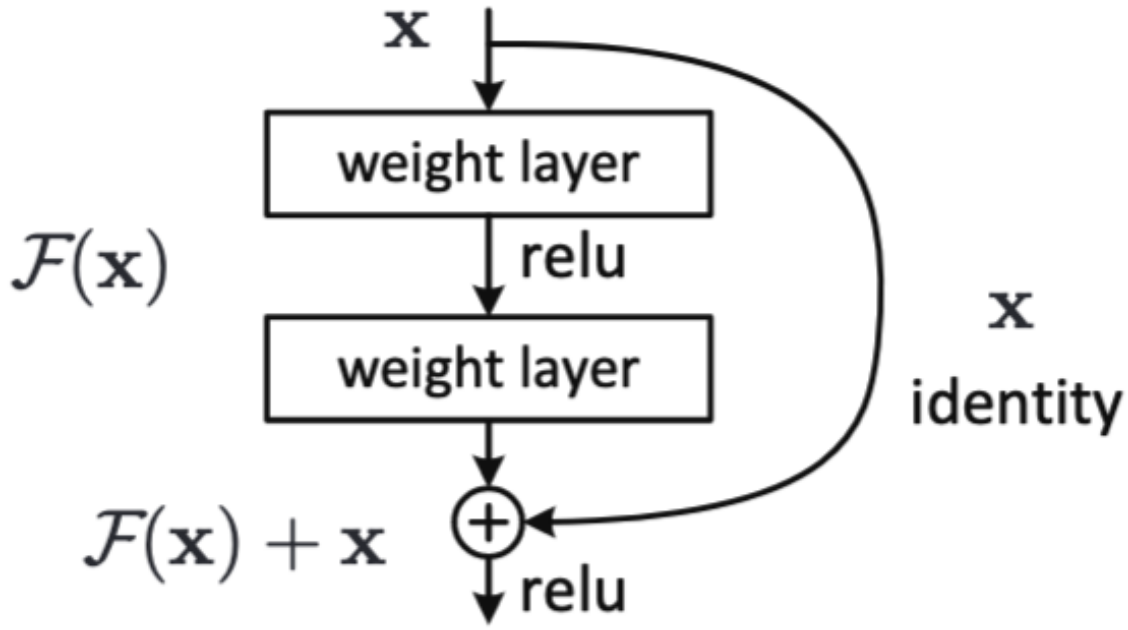


Figure 2.16: Residual block representation, sourced from [1]

Residual Networks, commonly known as ResNets, are a type of deep neural network architecture that were introduced in 2015 by He et al [1]. The key innovation of ResNets lies in their use of residual blocks, which allow for better flow of gradients during the training process. This helps to mitigate the vanishing gradient problem.

A residual block consists of several layers of neurons, with a "skip connection" added between the input and output. The skip connection is a direct pathway that allows the gradients to bypass the intermediate layers and flow directly from the input to the output. This helps to ensure that the gradients are not diminished

as they flow through the network, which can happen in traditional deep neural networks.

The output of a residual block is the sum of the input and the output of the layers. This is because the skip connection is added to the output of the layers, rather than being passed through an activation function. This allows the derivatives to remain unchanged, resulting in a net increase in gradient for the block.

The residual blocks are designed to learn the identity function, which helps to ensure that the later layers in the network perform at least as well as the earlier layers. This helps to prevent the network from getting stuck in suboptimal solutions during the training process.

These innovations make ResNets a popular choice for training deep neural networks for a variety of tasks, including image classification, object detection, and natural language processing.

2.2.5 Generative Adversarial Networks

Generative Adversarial Networks (GANs), first introduced by Goodfellow et al. in 2014 [10], are a class of deep neural networks that are used to estimate generative models. They operate based on a minimax game between two players: the generator ($G(z)$) and the discriminator ($D(x)$), both of which are typically neural networks.

The generator takes as input a vector of random values, z, and converts it into a data format, such as an image. The discriminator, on the other hand, accepts data as input, either real (x) or generated ($G(z)$), and outputs the probability, $P(x)$, of it being authentic. In the context of image generation, the goal of the generator is to produce an image that is indistinguishable from those in the ground truth training set, while the discriminator's task is to judge whether an image is real or generated.

However, standard GANs have a number of issues, such as difficulty in convergence, mode collapse, or the discriminator loss quickly converging to zero. The Wasserstein GAN (WGAN) was introduced as a solution to the difficulties faced by regular GANs, such as difficulty in convergence, mode collapse, or the discriminator loss quickly converging to zero. The WGAN uses an approximation of the Wasserstein distance as a loss function, which is more likely to provide useful gradients for generator updates compared to the original loss function used in GANs.

In a WGAN, the discriminator is referred to as a "critic", as its role is to evaluate the quality of the generator output, rather than simply providing a binary classification of real or fake. The critic is trained to estimate the Wasserstein distance between the real and generated distributions, which measures the amount of "mass"

that needs to be moved to transform one distribution into the other. The generator is trained to generate samples that minimize the Wasserstein distance, effectively making the generated samples more similar to the real data distribution.

An important advancement in the training of GANs was introduced by Gulrajani et al. in 2017 [12]. This method, called the Wasserstein GAN with Gradient Penalty (WGAN-GP), addresses a key limitation of regular GANs by providing a more stable and meaningful loss function. In particular, the WGAN-GP penalizes the norm of critic gradients with respect to the training samples, as opposed to the weight clipping approach used in regular WGANs.

The WGAN-GP improves the training process in several ways. First, it helps to stabilize the training process and prevent mode collapse, which is a common problem in GANs where the generator produces limited varieties of samples. Second, it promotes a more meaningful and stable optimization process for both the generator and the discriminator, resulting in better performance in generating high-quality samples. Finally, the gradient penalty further improves the stability of the training process by constraining the critic's gradients, which helps to prevent the generator from producing poor quality samples.

The WGAN-GP has been demonstrated to produce better results compared to regular GANs in various tasks, including image generation, text generation, and others. For example, it has been shown to produce more realistic and diverse images in tasks such as face generation, object generation, and style transfer. Moreover, it has been used successfully in text generation tasks such as dialogue generation and machine translation.

The WGAN-GP is a popular and effective approach for training generative models, thanks to its stable and meaningful loss function and improved training process. By penalizing the norm of critic gradients with respect to the training samples, it helps to prevent mode collapse and improve the quality and diversity of generated samples.

2.3 Literature Review

2.3.1 Perception and Vehicle Surround View

Much of the current work in parsing a vehicle's surroundings, specifically in the manner of surround view generation, are geometric and/or pure vision-based techniques.

An early approach to this was by stitching together the views from a suite of

fisheye cameras placed around the vehicle, thereby providing a single panoramic image of the immediate surroundings. [21] Similarly, Zhang *et al* propose utilising geometric and photometric alignment with composite view synthesis to generate a surround view image for embedded systems. [22] Iterations on this approach up the complexity significantly, as shown in [23], where an end-to-end system is proposed utilising a bank of eight cameras in conjunction with route planners.

Few learning-based approaches exist in the literature tackling this task. One such example is the approach proposed by Guindel *et al.* [24] The authors propose a computer vision based framework fusing the object detection and recognition capabilities of a deep network with a 3D reconstruction of the scene using a stereo camera. One disadvantage to this approach is that the accuracy of depth estimation decreases with distance. As noted from the results, acceptably low localisation errors are obtained in the 0-20m range but degrade further out.

Work specifically aimed at generating bird's eye views in an adversarial manner is sparse. One approach is BridgeGAN, [25] which generates bird's eye view images, not representations, through the use of an intermediary homography view. This approach uses three views - frontal, homography and bird's eye - and trains a GAN model for each, whose task is to learn a transformation from one view to the next. The homography view is obtained via homography estimation [26] from cropped frontal view images. Though this work applies adversarial training to the bird's eye view problem, there are many steps involved, such as obtaining an intermediary transformation and the need to train three separate GAN models.

On the other hand, MPV-Nets [27] does not have an adversarial component but instead obtains a bird's eye view representation through a series of convolutional neural networks. The authors show that it is possible to learn an action policy from this representation.

Another example is work conducted by Palazzi *et al.* [28] where the authors show a learning-based approach to generating a bird's eye occupancy map. This approach works by training a model to warp a detection from a dashboard camera to that of a top-down bird's eye view.

The work presented in this paper builds and extends on these examples in several significant ways. The generated bird's eye view detections from our approach are not axis aligned - as they are in [28] - and thus represent the environment in a finer level of detail. We predict the orientation of the detection.

In contrast to some approaches, [25] [27] [28] our approach is trained and tested on real data, as opposed to synthetic data. Generalisation and robustness of the proposed method are also validated by testing on an entirely different dataset, with

quantitative metrics. Our model is capable of predicting objects at distances greater than many of the examples discussed.

2.3.2 Monocular Depth Estimation

In recent years, depth estimation has proven to be a subject of great research interest, particularly in robotics and computer vision.

Monocular depth estimation is a challenging task as there are a lack of certain cues and hints, which other techniques can rely on. Much of the early body of work relied on exploiting geometric priors coupled with bespoke features.

As the problem is inherently ill posed, many recent works focuses on tackling it through learning methods. An early example is Saxena et al [29], who utilise a Markov Random Field trained via supervisory signals to model image depth cues and spatial relationships.

With the success of deep learning on image related tasks, many have switched from classical methods to neural network based methods. Eigen et al [30] were one of the first to use a deep neural network to estimate depth from a single image. In the same vein, Liu et al [31] propose a Conditional Random Field [32] based approach which does not rely on any geometric priors or similar.

Building on previous work, Eigen et al [33] show that it is possible to use a single architecture to perform depth estimation, surface normal estimation and instance labelling. Laina et al [34] propose a novel up-projection module to obtain finer estimation. Facil et al [35] propose incorporating camera parameters directly in the convolution operation to generalise depth estimation.

However, gathering suitable training data is challenging, especially fully annotated depth-RGB pairs. [36] mitigates this by showing that transfer learning is effective in this domain. Others have proposed utilising very sparse ground truth data obtained from depth sensors to guide training. [37] [38] [39]

Going further, researchers have pursued self-supervised and unsupervised approaches. Godard et al [40] exploit stereo geometry constraints to generate disparity by enforcing left-right consistency. The authors then improve on this by introducing a minimum reprojection loss to deal with occlusions, an auto-masking strategy to handle dynamic objects and multi-scale sampling to mitigate artifacting. [2] Guizilini et al [41] propose novel blocks for compressing/uncompressing representations using three dimensional convolutions. The authors use these blocks to produce high-quality depth maps from unlabelled monocular images, even outperforming some fully supervised approaches.

However, the vast majority of work published to date has been for perspective cameras. Given the abundance of perspective pinhole cameras - and that most datasets are captured using them - this is hardly surprising.

2.3.3 Omnidirectional Depth Estimation

In contrast with perspective pinhole cameras, there is a much smaller body of work for applying deep learning based methods to high field-of-view inputs.

The full surround vision would be useful in applications in robotics, autonomous vehicles and virtual reality. However, current methods find such panoramic inputs challenging, as well as the lack of available high quality datasets for this domain. Representing spherical co-ordinates on a rectangular grid results in significant distortions at the poles and violates some perspective pinhole assumptions.

Some sidestep this issue by using other projections. Cheng et al [42] utilise the cube map projection to map the equirectangular image to six perspective images (the faces of a cube). Similarly, Wang et al [5] use cubic projection with cube padding with a new indoor dataset.

Early attempts to solve the omnidirectional depth problem by Schönbein et al [43] [44] forgo correcting the distortion and constructing perspective views, and instead utilise a pair of calibrated catadioptric cameras with a plane-based model to perform three dimensional reconstruction. Huang et al [45] adapt standard structure-from-motion techniques for virtual reality related 360 images. Kumar et al [46] propose using sparse LiDAR supervision as a training signal for a regular CNN in the autonomous vehicles domain.

Recently, end-to-end trainable models have gained popularity in the literature, though there are few examples in the omnidirectional domain. Zioulis et al [4] propose one of the first works showing a CNN trained directly on equirectangular images to regress dense depth. The authors demonstrate how models trained on perspective pinhole images perform worse than those trained on equirectangular images directly. Zioulis et al [47] iterate on this work by utilising an improved synthetic dataset with multiple viewpoints to guide supervision through view synthesis. Wang et al [48] approach this problem via a stereo system, introduction of per pixel polar angle as input and a learned cost volume. BiFuse, proposed by Wang et al [49], seeks to regress for depth through a fusion of feature maps gained from equirectangular and cubic projections. The authors reason that the former provides a lot of spatial information owing to the wide field-of-view at the cost of high distortion, while the latter provides distortion free views at the cost of border discontinuities.

Efforts to handle the distortions caused by equirectangular projections in CNNs have been attempted. Su et al [50] propose to optimise a distortion aware network to learn to recreate regular filter outputs on equirectangular images. Coors et al [51] build on standard convolutional filters by incorporating the distortion directly at the filter level. This is accomplished by modifying the filter's sampling locations to conform to equirectangular geometry, thereby encoding invariance. Additionally, the filter preserves the "wrap-around" connective nature of these images, which is not present in perspective pinhole images.

Part II

Contributions

3 | Bird's Eye View Generation

3.1 Overview

The detection of objects in 3D is of paramount importance for an autonomous vehicle. As autonomous vehicles need to operate in dynamic and fast-changing environments, this capability forms the crux of many safety-critical decision making processes.

Vision-based systems continue to be a popular choice for autonomous vehicle perception, whether it forms the entirety of the perception stack or plays only a limited role within it [3]. As the vast majority of current road infrastructure caters to humans, significant information is conveyed visually. Vision-based systems are uniquely able to leverage this pre-existing wealth of data and, in addition to this, can be relatively inexpensive to deploy owing to the ubiquity of cameras.

To date, typical methods of 3D perception for autonomous vehicles consist of utilising data from LiDAR, RADAR, a suite of cameras or a fusion of these. While effective, utilising data from these sensors can be costly, both in terms of the financial cost associated with the sensor itself as well as the computational overhead required to process the data into a usable format.

While stereo depth systems have been explored in depth, it would still be prudent to explore monocular systems. Halving the number of potential sensors required would reduce costs. Additionally, in safety critical domains where human lives are at risk, monocular systems could act as a backup in the event of other system or hardware failures resulting from damage or faults.

Transforming and projecting 3D information to a top-down representation - a "bird's eye view" (BEV) - of *semantically significant* objects in the scene provides a number of benefits in this regard. For example, compared to a point cloud or an image, a BEV representation occupies less memory, as we are only interested in information at the object level as opposed to low-level information like pixels or LiDAR returns. In addition, it provides an easy-to-understand, interpretable

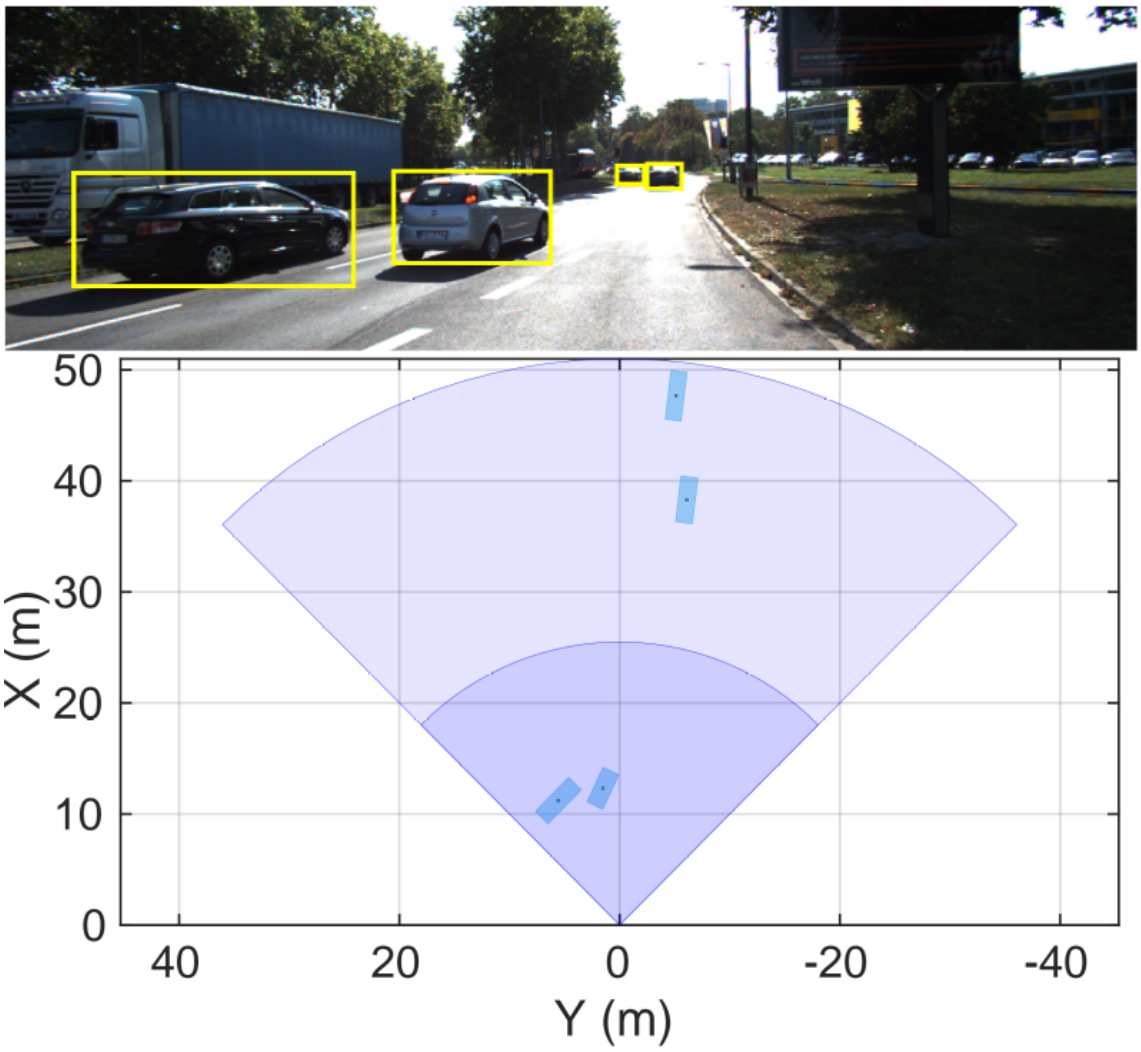


Figure 3.1: Outline of the bird's eye view problem from a single image.

representation of the scene at a given moment in time.

We draw inspiration from recent work in guided image generation, synthesis [8] and super-resolution. [9] Instead of solving the problem in a supervised manner by minimising the difference between the output and ground truth, we instead formulate the problem within the context of adversarial learning.

The main contribution of this chapter is an adversarial approach to obtaining a BEV representation of detected vehicles from a monocular image. We seek to leverage the generative capabilities of a Generative Adversarial Network (GAN) [10], specifically a Wasserstein GAN (WGAN) [11] with gradient penalty (WGAN-GP) [12]. Our model is composed of two sub-networks: a generator network and a critic network. The generator network is tasked with producing BEV representations from an image, while the critic network is designed to assign a "realness" score to

this representation, distinguishing a generated BEV representation from its ground truth counterpart. Therefore, the BEV representation produced by the generator network is gradually trained to be similar to the ground truth.

We show that a model trained in this way generalises better between datasets compared to baseline, strictly supervised models. In addition, our approach only requires data from a single monocular camera during inference, and does not require more complex sensors such as LiDAR or radar.

This chapter will provide an in depth examination of the methodology employed, experiments undertaken, as well as quantitative and qualitative evaluation. It will conclude with a discussion of key takeaways and future work.

3.2 Methodology

Here we introduce our method and outline the model, training data and training schema.

3.2.1 Model Architecture

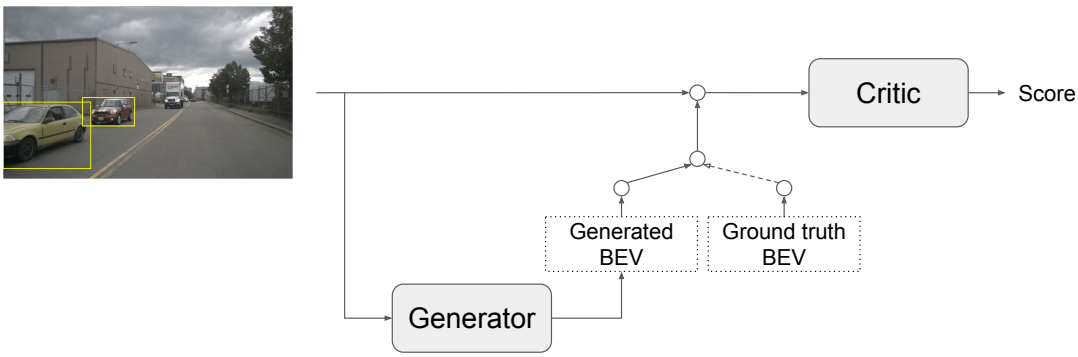
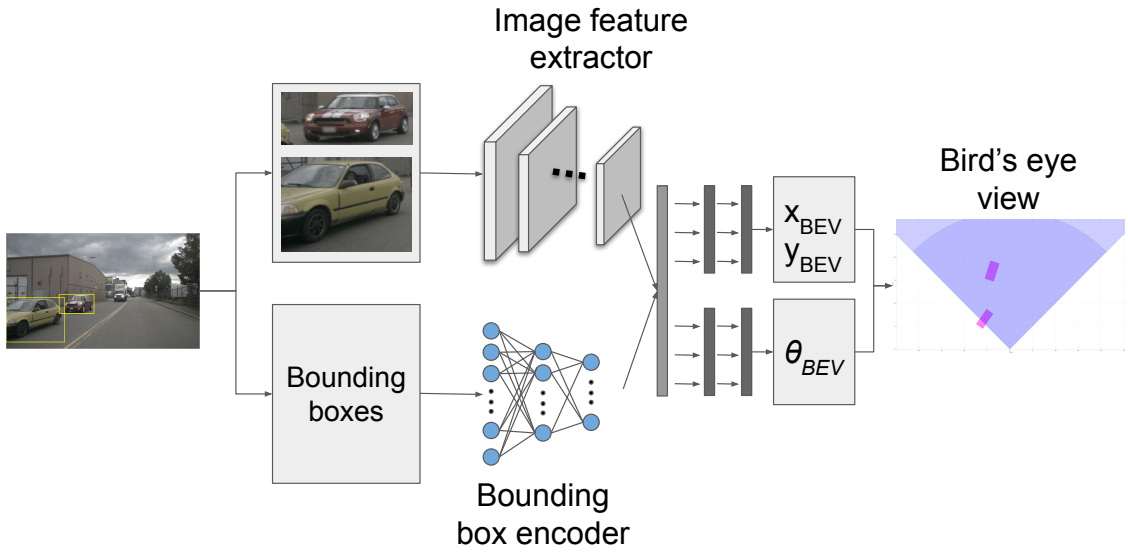
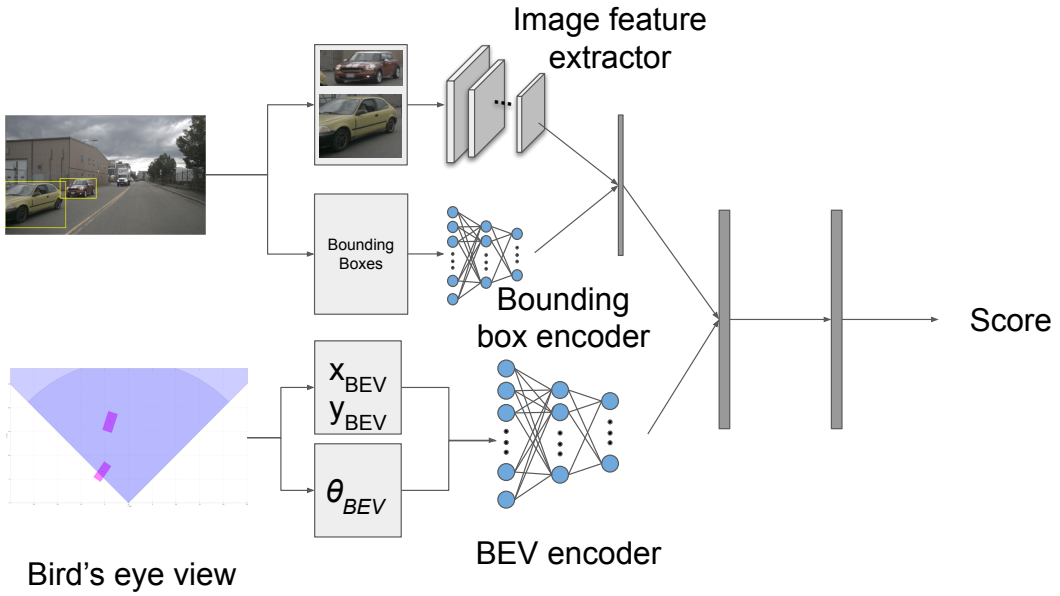


Figure 3.2: Overall system diagram. Generator and critic networks are shown in Figures 3.3a and 3.3b respectively.

The DeepBEV model is based on a WGAN-GP [12] framework, consisting of a generator network and a critic network, both based on ResNet-18 [1]. Specifically,



(a) Generator system diagram



(b) Critic system diagram

Figure 3.3: Generator and critic network system diagrams.

both networks employ a multi-input multi-output paradigm.

In general, the task of the generator network is to output the detected object's BEV translation and orientation, while the critic network compares this output with the image used to generate it and scores its quality - its "realness". Effectively, this critic learns a mapping between the scene and a plausible BEV representation. As seen in Figure 3.2, the output of the generator is an input to the critic.

For the generator, we employ parallel layers to process the image and the bounding box description, which is then concatenated into a feature vector. This feature vector is then used independently by two heads to produce a prediction for the object's translation and orientation. This is shown in Figure 3.3a.

For the critic, the same ResNet-18 based architecture is used as a feature extractor for the image. However, instead of inputting the detection's bounding box into a separate bounding box encoder, we encode the translation and orientation values via a dedicated encoder. The final output of the critic is a scalar score, evaluating the consistency of that image-value pair. Essentially, the critic network's aim is to discern if a BEV representation is plausible given an image and score it accordingly. This is shown in Figure 3.3b.

3.2.2 Loss Functions

Denote the loss function of the generator as:

$$L = \underset{\widetilde{x}\sim\mathbb{P}_g}{\mathbb{E}}[D(\widetilde{x})] \tag{3.1}$$

where $\mathbb{P}_g$ is the distribution of the generator model, $D()$ denotes the critic, and $\widetilde{x}$ are a set of generated samples (bird's eye view predictions). A modified form of the Wasserstein GAN objective function with a gradient penalty [12] is utilised as a loss function for the GAN training. Specifically, the loss function can be formulated as follows:

$$L = \underset{\widetilde{x}\sim\mathbb{P}_g}{\mathbb{E}}[D(\widetilde{x})] - \underset{x\sim\mathbb{P}_r}{\mathbb{E}}[D(x)] + \lambda \underset{\hat{x}\sim\mathbb{P}_{\hat{x}}}{\mathbb{E}}[(|| \bigtriangledown_{\hat{x}} D(\hat{x})||_2 - 1)^2] \tag{3.2}$$

The Wasserstein GAN with Gradient Penalty (WGAN-GP) is an improvement over the standard Wasserstein GAN (WGAN) that addresses some of the stability issues encountered during training. In WGAN-GP, a gradient penalty is used instead of weight clipping to enforce a Lipschitz constraint on the critic.

The gradient penalty is calculated as the norm of the gradient of the critic with respect to an interpolated distribution between the real and generated samples, $\mathbb{P}_{\hat{x}}$.

This interpolated distribution is created by randomly mixing real samples, x, with generated samples, $\widetilde{x}$, and the gradient penalty is calculated using the resulting set of interpolated samples, $\hat{x}$.

The goal of the gradient penalty is to enforce a Lipschitz constraint on the critic, such that the norm of the gradient should be 1 along the interpolated distribution, $\mathbb{P}_{\hat{x}}$. If the norm of the gradient diverges from a target norm of 1, the critic is penalized. This helps to stabilize the training process and prevent mode collapse, resulting in higher quality generated samples.

In the original WGAN-GP paper [12], a λ value of 10 was used as it was determined to work well across a range of architectures, datasets, and tasks. The λ value represents the trade-off between the Wasserstein distance and the gradient penalty, and can be adjusted to suit the specific needs of the task at hand.

In practice, this loss is implemented by computing the mean average critic score on ground truth and generated minibatch samples, while we uniformly sample and interpolate between these to calculate the gradient penalty. This is because, as defined by Equations 3.1 and 3.2, we seek to increase the difference between the values for real and generated samples.

3.2.3 Training Scheme

For the most part, we follow the training scheme outlined in [12]. However, there are two key distinctions: the use of pre-training and conditioning.

Pre-training

Table 3.1: Training hyperparameters

Parameter	Pretraining	Adversarial
Model	ResNet-18	-
Epochs	1000	1000
Batch size	32	32
Learning rate	0.0002	-
Generator learning rate	-	0.00001
Critic learning rate	-	0.0002
n_{critic}	-	1000
λ	-	10
Optimizer	Adam	Adam
β_1	0	0
β_2	0.999	0.999

The training scheme involves two passes on the training dataset. Firstly, the generator network is trained on the training dataset in a supervised manner. The network is given an image crop of an object along with its image bounding boxes and tasked with predicting that object's orientation and location. This serves to bring the model up to a level of proficiency on the BEV task. Table 3.1 shows the hyperparameters of the pre-training phase. Secondly, we perform adversarial training. The pre-trained weights are used for the generator, and the weights for the feature extractor are used in the critic. We justify this due to the fact that if we do not use pre-trained weights in the critic, there is an imbalance in performance between the two networks. The adversarial nature of the training scheme means that this raises the possibility of the generator network reaching a local minimum, as it relies on feedback from the critic.

We train the critic for n_{critic} more iterations than the generator, and this serves to bridge the performance gap between the pre-trained generator and semi-trained critic. In addition, this minimises the chance that the critic learns to significantly outperform the generator.

As this step is essentially fine-tuning, the learning rate for the generator is lowered so that performance is not negatively impacted. Similarly, the weights of the critic's feature extractor are frozen. The hyperparameters for adversarial training are outlined in Table 3.1.

Conditional GAN

In traditional GAN and WGAN implementations, a random latent vector is typically fed into the generator to generate fake samples. However, in the case of DeepBEV, the generator is conditioned solely on the input images and object bounding boxes, and does not require the use of a latent variable. This makes it easier to control the output of the generator based on specific input conditions, and may lead to better overall performance.

In Eq. 3.2, the goal is to maximize the difference between the critic scores of real and generated samples. This is typically achieved by training the critic to output high scores for real samples and low scores for fake samples. In order to label the samples for the critic during training, one can simply multiply the score of the critic by -1 if it is evaluating a ground truth sample. This effectively flips the sign of the score, making it negative, which is the desired labeling for real samples. This labeling can be done online during training, making it a simple and efficient method.

3.3 Evaluation

3.3.1 Dataset and Experimental Setup

We applied this approach to two datasets: KITTI [52] and nuScenes [53]. For all models presented, we trained on KITTI and evaluated on nuScenes. In this way, we ensure a testing environment that is wholly different when compared to the training environment. This ensures that dataset bias and overfitting is kept to a minimum, while also mimicking potential real-life deployment scenarios.

Specifically, the object detection benchmark of the KITTI dataset is used for training [52]. It is comprised of 7481 images taken in the city of Karlsruhe, Germany. For each image we:

1. extract detections of object class *Car*
2. filter for detections that are not occluded at all (KITTI occlusion level 0)
3. obtain image co-ordinates of detection bounding boxes and their corresponding 3D poses. Both of them are normalised to the range [0, 1].

The image plane object bounding boxes are represented as a length four vector describing its x and y centroid co-ordinate as well as bounding box width and height. Regarding 3D pose, the normalisation is between the maximum x and y object locations in metres, [0, 100] and [-50, 50] respectively. Similarly, the orientation was normalised between KITTI's range of $[-\pi, \pi]$. Since we are only interested in BEV related information, we disregard the object's height.

The evaluation dataset is comprised of 806 unique scenes (sequential recordings 20 seconds in length) from nuScenes, out of which are extracted entries at the object level. Each sample is one detected object in the image, with corresponding information such as image bounding box dimensions, translation, orientation and class. Out of the list of possible object classes, we only train and test on *car* classes. In our training dataset, KITTI, the *car* class is *"Car"* while in nuScenes it is *"vehicle.car"*. In total, the entire evaluation dataset comprises of over 60 thousand unique *"vehicle.car"* samples, split between two cities, at differing times of day and weather conditions. The data is formatted in the same manner as during training, with no augmentations applied.

DeepBEV is agnostic to the way objects are detected and bounding boxes obtained. For this reason, we use datasets which provide these annotations, but in principle any object detector can be used to obtain these. Our selection criteria for

bounding boxes are primarily occlusion based; we only utilise detections which have a very low level of occlusion.

In addition to nuScenes, we obtain some evaluation samples from Virtual KITTI 2 [54] - an adaptation of Virtual KITTI [55] - and the Surround Vehicle Awareness (SVA) [28] dataset. These are both synthetic datasets and will serve as an additional measure of robustness.

Table 3.2: Training dataset overview

Metric	Value
Images	7481
Occlusion level	0
Car detections	13456
Bounding box format	$[x_{centroid}\ y_{centroid}\ width_{box}\ height_{box}]$

3.3.2 Testing and Metrics

In terms of the testing scheme and metrics used for evaluation, we assess the performance on a per-object basis for the entire evaluation dataset. To compare the similarity between the ground truth and model predictions for each object, we compute the Euclidean distance between the two centroids and the difference in orientation. Using these metrics, we calculate the dataset-wide median and standard deviation (SD) values.

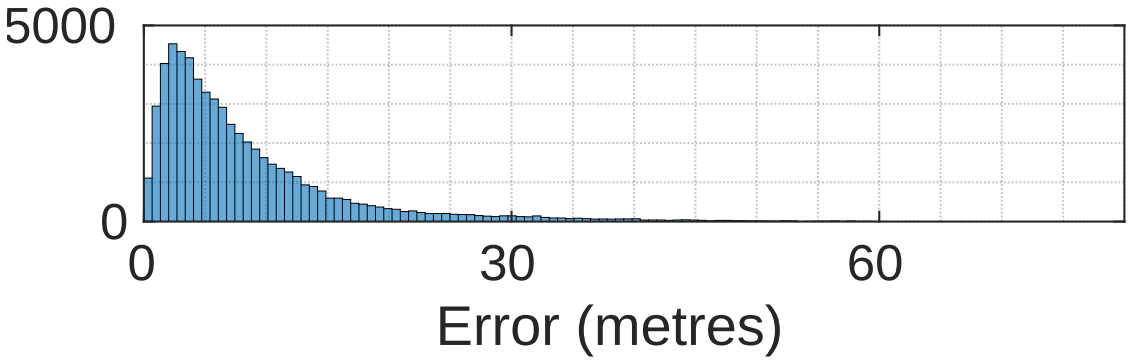
Since this is a specialized task, there are limited choices of pre-existing baseline models trained on real data available. Therefore, we evaluate DeepBEV against standard models trained in a supervised manner. To ensure a fair comparison, we train and test these models using the same data that was provided to DeepBEV. The benchmark models selected for comparison include various versions of standard ResNet [1], ResNeXt [56], and Wide ResNets [57], as shown in Table 3.3.

3.3.3 Quantitative Results

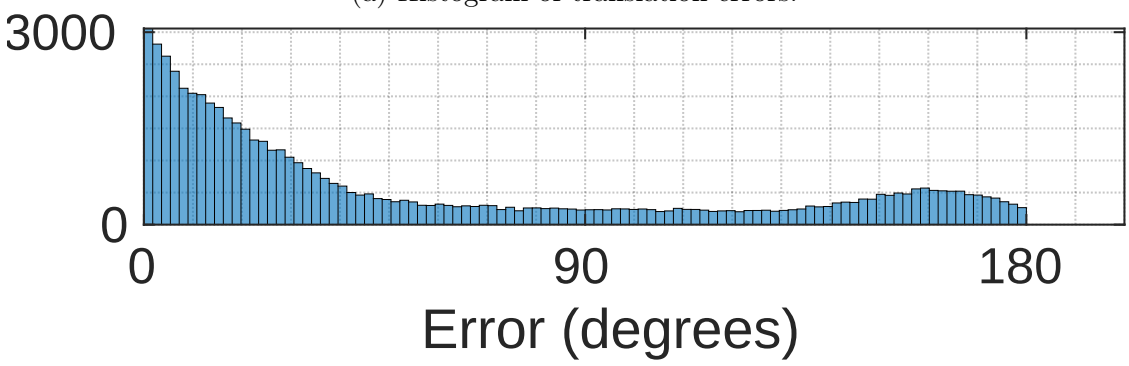
Table 3.3: Distance and orientation errors on nuScenes

Model	Distance Error (m)		Orientation Error (degrees)	
	Median	SD	Median	SD
DeepBEV	**5.91**	8.22	**28.67**	56.83
ResNet-18	8.62	7.11	30.70	57.40
ResNet-50	8.43	8.44	33.74	57.99
ResNet-101	7.58	9.24	**28.36**	59.29
ResNeXt-50	7.26	8.69	31.86	58.45
Wide ResNet-50	7.97	8.55	33.09	59.08

The distance and orientation test errors on the nuScenes are given in Table 3.3. It can be seen that DeepBEV has a median Euclidean distance error 22.8% lower than the next best baseline model, while being comparable in the orientation metric. Note that the generator portion of DeepBEV is a ResNet-18 model and its model size is only 20.6% of the ResNet-101, the only model which surpasses DeepBEV in orientation prediction by a median value of 0.318°.



(a) Histogram of translation errors.



(b) Histogram of rotation errors

Figure 3.4: Histogram of DeepBEV errors on nuScenes.

The histograms of the predicted distance and orientation errors are shown in Figures 3.4a and 3.4b. It is clear that most of the prediction errors are spread to the small error regions. While the results obtained show a somewhat high error value, the inaccuracies are present across all models tested. Curiously, the shifted error bars might indicate that there is systemic bias present – an average object distance in the dataset perhaps. This would require further work to investigate.

Our results show that solving the task in an adversarial manner as opposed to supervised improves model robustness and ability to generalise to novel data, both real and synthetic. Not only this, but we achieve superior performance with a greatly reduced model size compared to larger models like ResNet-101. The benefit of this is evident in the lower inference time of DeepBEV compared to the other models, as shown in Table 3.4.

Table 3.4: Per detection inference time in milliseconds, where a detection is a detected object in the image

Model	Median CPU (ms)	Median GPU (ms)	Total (ms)
DeepBEV	0.19	0.79	**0.98**
ResNet-18	0.19	0.80	0.99
ResNet-50	0.41	1.75	2.17
Wide ResNet-50	0.41	2.23	2.64
ResNeXt-50	0.75	2.86	3.61
ResNet-101	1.12	3.89	5.01

3.3.4 Qualitative Results

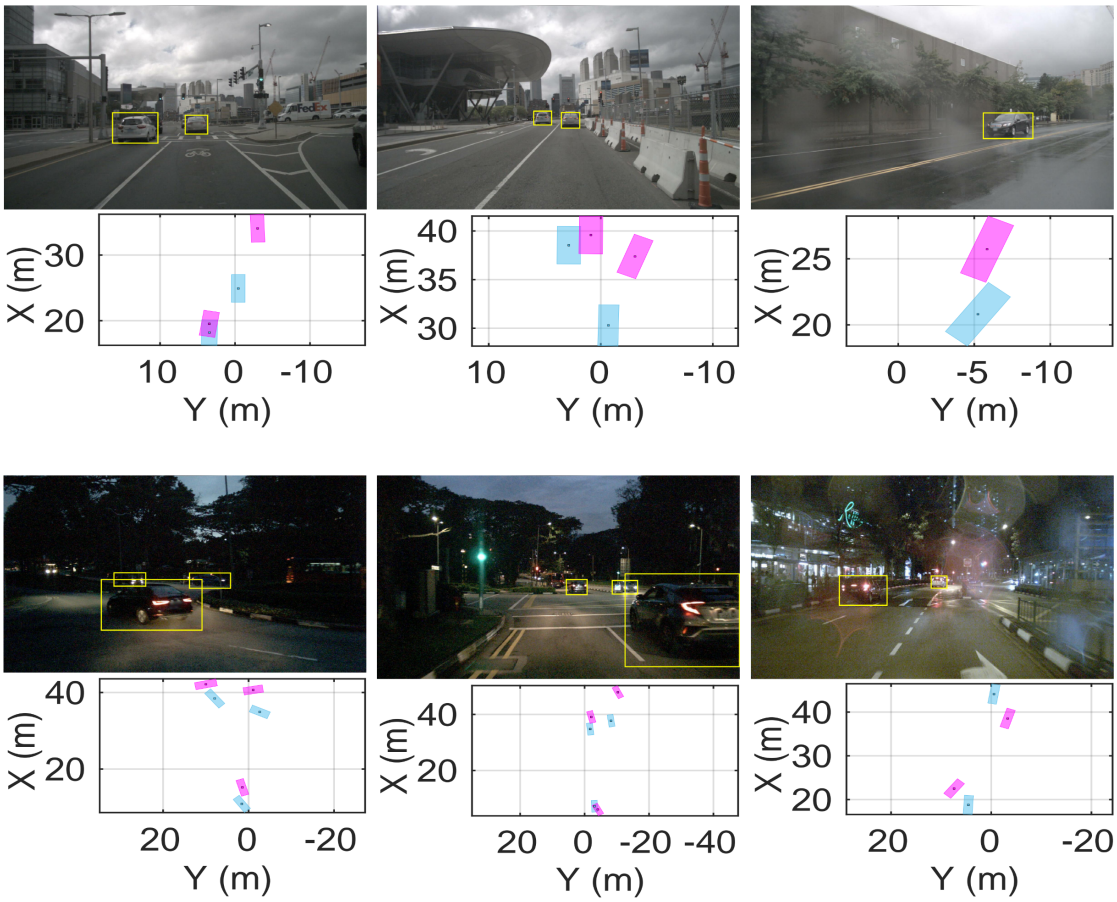


Figure 3.5: Samples from the evaluation set in daytime, rain and night-time conditions. Blue denotes ground truth pose, magenta denotes model prediction.

Shown in Figure 3.5 are example inferences from the nuScenes evaluation set in a variety of conditions. To reiterate, this was unseen during training time as the model was trained entirely on the KITTI dataset.

As can be seen, the results show a reasonable estimation of target object pose. Somewhat expected, the model produces better estimations for cars closer to the camera than further away. This can be seen in Figures 3.5 and 3.6, for example, where the detected object is <20m away. Additionally, some conditions were absent from the training dataset, such as rain on the lens or night time driving.

Adding to this robustness, is additional qualitative evaluation on two synthetic datasets: Virtual KITTI 2 and SVA. These are shown in Figures 3.6 and 3.7. Note that SVA does not contain object pose information in a similar manner to the other datasets. Despite both datasets being unseen at train time, the results look

plausible. Not only are the estimates reasonable, but it produces these estimates while bridging a real-to-simulation gap.

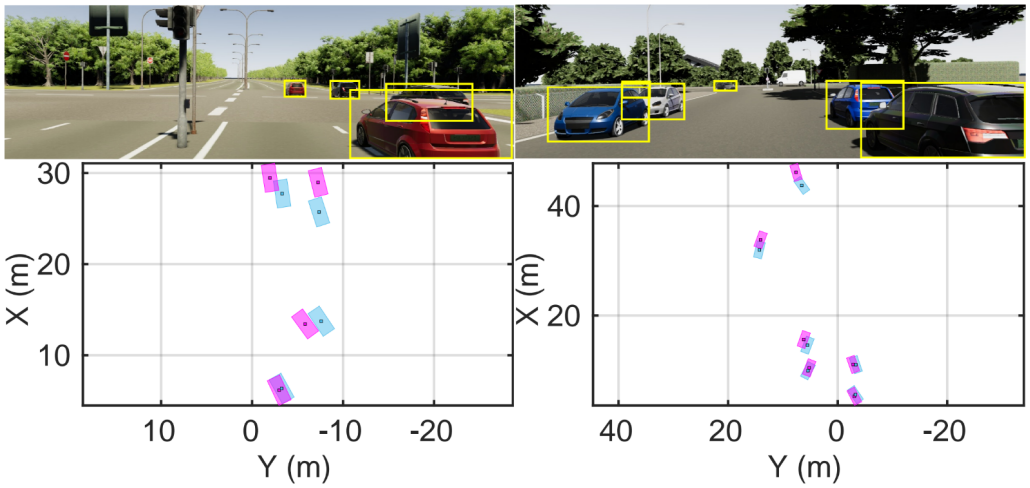


Figure 3.6: Samples from the Virtual KITTI 2 dataset.

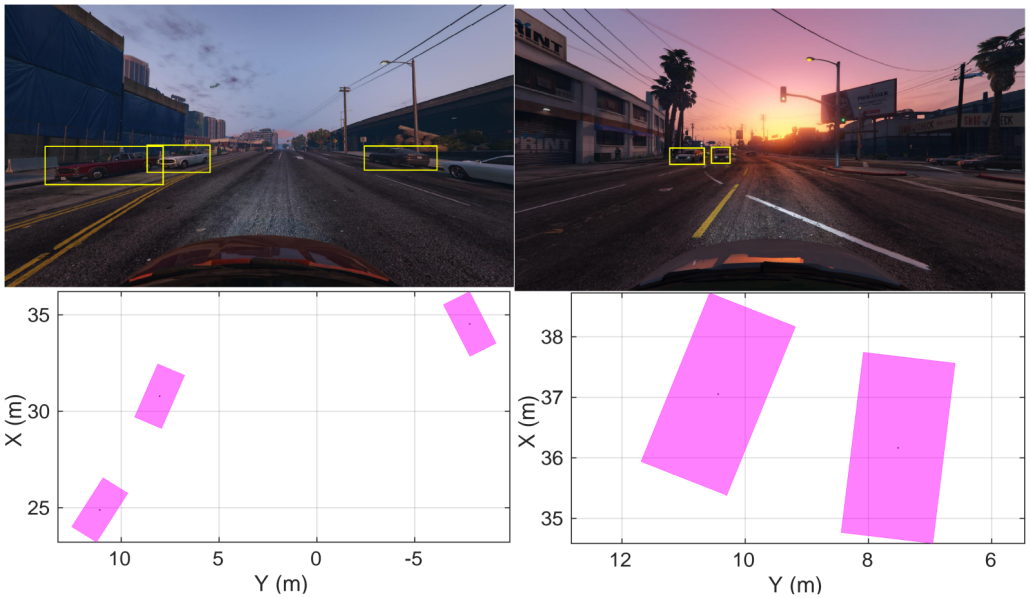


Figure 3.7: Samples from the SVA dataset.

3.4 Discussion

This chapter demonstrates an adversarial approach to generating a bird's eye view representation of a scene in the context of autonomous driving. Many solutions to this problem involve the use of multiple cameras or other range-finding sensors. Here, this is achieved using nothing more than a single image taken from a monocular camera and do not require more advance sensors like LiDAR or RADAR.

The novel aspect of this approach is that it formulates the bird's eye view problem as a conditional generative adversarial task and solves it with a WGAN-based network. This WGAN-based network comprised of a generator and a critic.

Rather than training a model in a supervised manner, such as using the difference between the model output and the ground truth, the generator is trained to produce plausible bird's eye view representations given an image and bounding box information. The error signal in this case is derived from a critic network, the task of which is to score a given bird's eye view representation in comparison to the inputs used to generate it.

The experiments on public datasets show that the proposed adversarial approach achieves better performance and robustness when tested on novel data than those trained in a typical supervised approach.

To clarify, this better performance was achieved on an entirely unseen dataset, captured using different camera hardware and containing unseen environmental conditions like rain and lens smearing. From a *robotics* perspective, robustness and run time speeds are highly important.

In this chapter, only unoccluded samples are used. Principally, there is nothing stopping the use of partially occluded samples. This could increase robustness further though it could be more difficult to train using highly occluded objects.

The results presented in this chapter imply that reframing the problem and solving it adversarially has tangible benefits with respect to model robustness, all else being equal. This robustness leads to DeepBEV outperforming much larger, "heavier" models on the same task, leading to a lower memory footprint and reduced inference time.

However, it is pertinent to also discuss the challenges and pitfalls present with this approach. Compared to direct supervision on ground truth data, adversarial training requires balancing the two competing "actors" on top of the usual considerations regarding dataset quality, model design and so on. If this is not taken into consideration, there is a risk of one aspect significantly outperforming the other and the overall system reaching a local minimum.

While the results in this study are promising, it is not explicitly clear whether using a different network for feature extraction, such as a larger ResNet variant, would lead to further improvements in performance. To answer this question, additional experiments would need to be conducted, comparing a variety of network sizes and the effect on performance. Additionally, the performance is sensitive to the quality of obtained bounding boxes. Bad bounding boxes can lead to bad estimations.

Expanding the model to other object classes will also require corresponding bounding boxes. There is nothing inherrent in the proposed solution which prevents it from being trained on an arbitrary number of bounding boxes types. The only limiting factor would be availability of data.

Though the approach presented here shows a quantifiable improvement versus direct supervision, it has only been tested in this specific task. It does not mean that the same approach will yield similar improvements in other domains.

Additionally, while the results show an improvement over direct supervision, in absolute terms it is still quite large. Given the monocular nature of the task, large metric errors can be expected, as inferring three dimensional depth from a two dimensional projection is fundamentally ill-posed. Such large errors are not suitable for safety critical systems and would require further work to improve accuracy and robustness.

Given the above, future work could involve a combination of the following:

- Improving the accuracy and robustness further to be suitable for deployment on an autonomous vehicle.
- Experiments to test whether adversarial training could provide benefits in other domains
- Extending the critic network as an error estimator for the model outputs

One fundamental limit to this approach is the field-of-view of the images used. In many autonomous vehicle or mobile robotics settings, it would be highly advantageous to have as wide a field-of-view as possible, as the system would be able to capture more information from the environment. However, such high field-of-view images are based on an entirely different projection model and come with their own set of problems. The next chapter explores how to tackle this problem from a depth estimation viewpoint.

4 | Omnidirectional Depth Estimation

4.1 Overview

Accurate depth information is critical for a wide variety of applications, ranging from robotic navigation to augmented and virtual reality. Therefore, depth estimation from images has been extensively investigated in the community. However, the vast majority of work on monocular depth estimation so far has been focused on perspective pinhole cameras, at least in terms of model design and data consumption. These were not designed with spherical geometries and severe distortion in mind, which means their performance may degrade significantly for omnidirectional imaging.

Earlier attempts at monocular depth estimation were not spherical and heavily depended on a supervised training signal delivered via labelled data, which was often captured using an accurate 360° LiDAR sensor. Unfortunately, such depth training datasets with accurate depth labels are uncommon for the omnidirectional vision, with the majority being synthetic [4–6] and/or indoor scenes [7]. As a result, this work leverages self-supervised and unsupervised approaches for monocular depth estimation using omnidirectional vision, benefiting from the considerably more available omnidirectional videos, e.g., on YouTube.

The majority of the publicly available omnidirectional videos, i.e., "in the wild" data, are in the equirectangular projection format. It is a planar representation of a spherical surface, resulting in severe distortion on the omnidirectional images. This means it violates some fundamental geometric assumptions held for perspective pinhole cameras. Since depth estimation is naturally a geometry problem, it is critical for the depth estimation systems to model this equirectangular representation and capable of tackling the severe distortion.

In this chapter, a monocular depth estimation technique is presented that explic-

Figure 4.1: This model results in good quality depth estimation on "in the wild" omnidirectional images.

itly incorporates equirectangular geometry into a neural network training process. This builds on themes outlined in the previous chapter and expands detected object pose estimation to per pixel depth estimation.

This is desirable for several reasons.

Firstly, that dense depth can be used as an enabler for other applications, such as in 3D reconstruction, mapping and localisation. Even consumer-grade, 1080p resolution 360° cameras have $1920 * 1080 = 2,073,600$ potential depth points, which is far denser than typical LiDAR systems. Harnessing this effectively and accurately would result in a significantly lower barrier to entry for many applications.

Secondly, the all encompassing field-of-view is a significant advantage – theoretically eliminating blind spots – compared to those provided by traditional perspective pinhole cameras.

Thirdly, there would be no need to perform an initial object detection phase, such as with the method proposed in the previous chapter. Such a method would be more difficult to scale to additional objects and would ultimately rely on the effectiveness of the object detector. For example, the method presented in the previous chapter was only tested on cars, and would need to be retrained to include other object classes.

The main contributions of this chapter are as follows:

- Re-formulating the existing view synthesis based depth estimation models for

equirectangular projection, injecting direct knowledge of camera geometry for accurate, dense depth estimation using omnidirectional vision.

- Incorporating "spherical" convolutional layers into the depth estimation model, considering the severe image distortion for convolutional operations.
- Proposing an optical flow based efficient masking strategy to dramatically mitigate the noise introduced to the loss from "noisy" pixels caused by the complete field of view, such as large, texture-less regions and dynamic objects.

In contrast with existing work in this area which mostly targets generated indoor scenes, to the best of the author's knowledge, this is the first work that demonstrates self-supervised learning-based, monocular omnidirectional vision for accurate depth estimation in real outdoor scenarios. In addition, this approach does not depend on ground truth depth data for training, which is rarely available for omnidirectional images. The experiments on two public datasets show state-of-the-art monocular depth estimation accuracy using omnidirectional videos.

4.2 Methodology

Here we introduce our method, depth estimation model, training schema and implementation.

4.2.1 Spherical Convolutions

The representation of a sphere on a two-dimensional plane through an equirectangular projection introduces substantial distortion, especially noticeable at the poles. The conventional convolutional filters and operations are primarily designed for perspective images and do not take into account such high levels of distortion. As a result, a typical convolutional filter, when applied to an equirectangular image, may fall short in fully capturing the image's geometric intricacies, a critical aspect for depth estimation.

Moreover, most existing depth estimation networks are conditioned on perspective images. This leads to inherent biases when such pre-trained standard convolutional kernel weights are employed on an equirectangular image.

Drawing inspiration from Spherenet [51], we extend the concept of spherical convolutions to depth estimation utilising equirectangular videos. A spherical convolution represents an alteration of the standard convolutional filter. Unlike the rigid

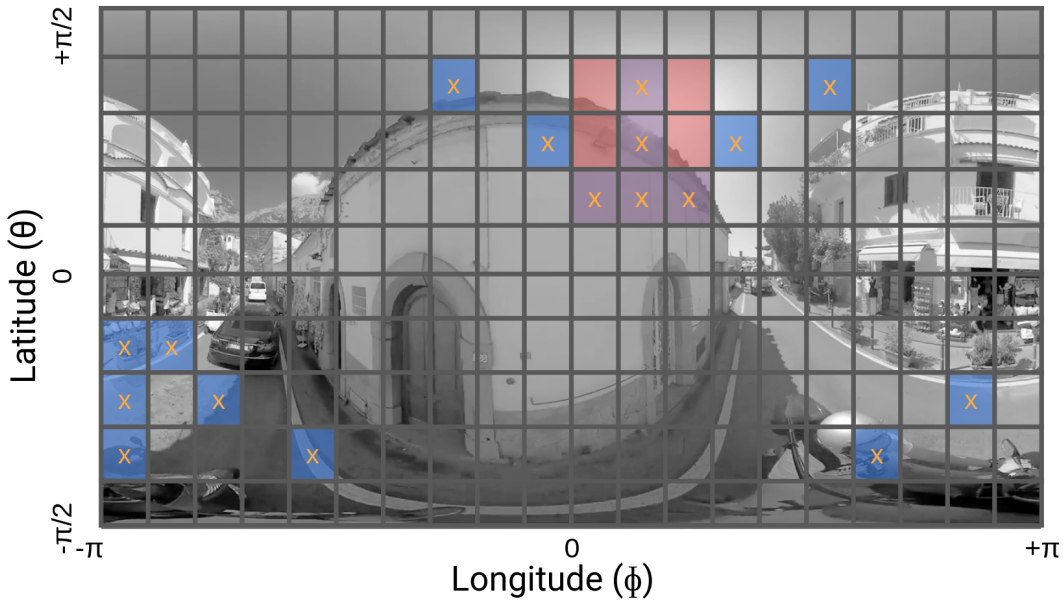


Figure 4.2: A comparison of sampling locations between the spherical convolution (blue) and a traditional kernel (red) using a 3x3 kernel as an example.

grid structure of the latter, spherical convolutions modify the kernel's sampling locations to align with the distortions caused by the equirectangular projection. Figure 4.2 visually demonstrates this transformation of sampling locations to cover varying portions of the image based on location. The red shaded cells denote the sampling positions of a traditional kernel, whereas the crossed, blue regions represent the adjusted sampling locations of a spherical kernel. Note that the kernel dimensions displayed are exaggerated for better visualization.

In practice, we can assume that the dimensions (resolution) of images typically do not change in a run, and we can therefore precompute sampling locations offline for all image locations. This is relatively expensive to compute, but since we would only need to do it once, utilisation at runtime should be significantly faster, enabling real-time applications.

This innovative approach of spherical convolution inherently incorporates invariance to distortion as the kernel navigates across the image. Simultaneously, it empowers the kernel to sample across the image boundary, leveraging the connective nature of 360° images, thereby mitigating discontinuities.

Here, our objective is to establish direct comparisons with conventional models. As a result, we strive to ensure equivalence in kernel dimensions, whenever feasible. Specifically, we adopt the filter dimensions of the Monodepth2 model.

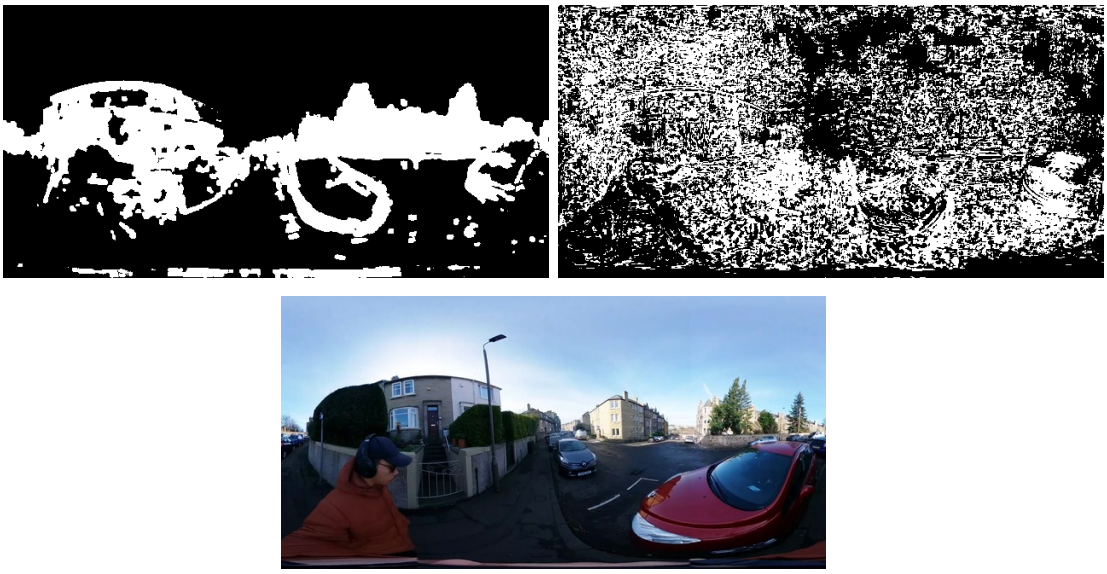


Figure 4.3: Our optical flow based binary masking (left) versus automasking (right) proposed in [2].

4.2.2 Optical Flow Masking

The complete surround view provided by 360° cameras offers many potential benefits, but with it some unique challenges not present with more traditional formats.

Specifically, for methods utilising appearance matching based losses such as this one, problems arising from non-Lambertian reflectances and inconsistent brightness are exacerbated. Such problematic pixels can occupy a larger proportion of equirectangular images compared to perspective images, introducing noise to the training signal if not dealt with. One particularly prominent example is the sky, which often makes up a significant portion of outdoor omnidirectional imagery.

[2] introduced a per-pixel automasking scheme to remove stationary pixels by only taking the loss of pixels where the warped photometric loss is less than that of the source, unwarped photometric loss. However, this can suffer greatly when applied to 360° images. As shown in Figure 4.3, many of the sky-region pixels are taken as valid and contribute to the loss calculation.

To overcome this, we turn to a masking strategy based on optical flow. The idea is to compute a binary, per pixel mask by taking only pixels above a certain optical flow magnitude threshold. We generate the mask, M_t, at time t by calculating the dense optical flow [58] for images I_{t-1} and I_t, normalizing the magnitudes, O_m, in the range $0 - 1$ and taking only pixels above a magnitude threshold, τ (empirically set to 0.1 in this case). 4.3 highlights the difference between our optical flow based binary masking strategy and the automasking proposed in [2]. White pixels are valid

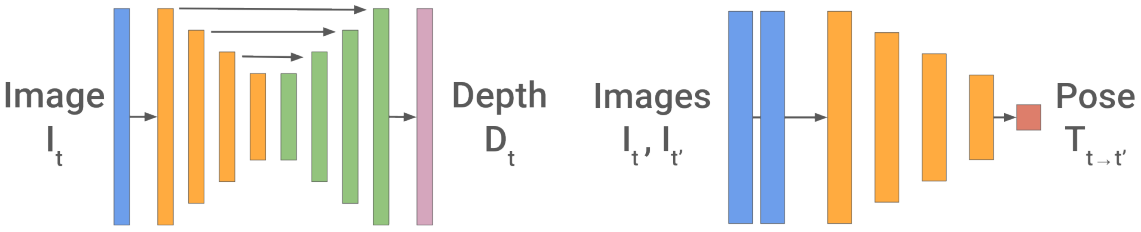
and will contribute to the loss, while black pixels will not. It can clearly be seen that this approach manages to largely filter for areas of importance while mitigating many challenging areas, such as the sky region.

$$M_t = O_m > \tau \tag{4.1}$$

In the interest of time, we compute this offline and apply it during training, though in principle it could be implemented online. Essentially, this is a data augmentation process and not real-time capable for mobile hardware. Though more computationally expensive (versus the aforementioned automasking), the result is a cleaner mask, devoid of most spurious sky pixels, non-Lambertian surfaces and similar velocity dynamic objects.

As only a binary mask is required of equal dimension to the image is required, implementation is agnostic to network architecture: we can replace the mask used by [2] with one generated via the method outlined above.

4.2.3 Model Architecture



(a) Depth (b) Pose

Figure 4.4: Encoder-decoder network for depth estimation, and a ResNet18 backbone model for pose estimation.

The Monodepth2 model is selected as the baseline model [2] for our depth estimation. The architecture consists of a depth estimation network and a separate pose estimation network, both taking RGB images as inputs. The former is based on the standard encoder-decoder U-Net with skip connections, accepting an equirectangular RGB image as an input. The output of the decoder is converted to depth by $D = 1/(a\sigma + b)$, choosing co-efficients a and b to restrict D to be within the range 0.1 and 100.

The pose estimation network is based on a ResNet18, outputting a 6 Degree-of-Freedom (DoF) relative pose from a pair of input images. This pose directly contributes to the photometric reconstruction error which guides the training process.

Pixels corresponding to a shared object in two distinct camera views should display similar visual characteristics. By establishing pixel correspondences between the views, we can ensure that their pixel intensities match. The goal is to minimize the photometric error by selecting values that result in uniform pixel intensities between the images. The pose is used here to perform warping between the two views, in this case consecutive image frames. The more correct this pose is, the lower the error.

When using spherical convolutional layers, we replace only the input layer of the network in question. More specifically, we performed a wholesale reimplementation of the network via the PyTorch library, which lends itself well to modular, ad-hoc replacements of various network details, such as custom layers. The replacement spherical convolutional layers can therefore be dropped in-place to the existing network and utilised with pre-trained weights.

4.2.4 Training and Implementation

We use weights from a pre-trained model for depth estimation for two reasons. Firstly, obtaining high-quality omnidirectional images with associated depth data is difficult. There are few publicly available datasets of this type in the literature, and those that do exist are mostly synthetic [4–6] and/or indoor scenes [7]. While these datasets are valuable, models trained on them face additional challenges in terms of bridging two domain gaps: the synthetic-to-real world gap and the indoor-to-outdoor gap. Secondly, we aim to demonstrate the applicability of our approach by adapting pre-existing depth estimation networks for the omnidirectional domain, providing greater flexibility in terms of selecting a depth estimation model.

We follow the self-supervised objective as outlined in [2, 40, 41, 59, 60], substituting the automasking procedure in [2] with our optical flow-based mask. In the self-supervised learning approach to depth estimation, the model is trained on a single image or video stream and learns to predict the corresponding depth map without using any external depth information. This is achieved by defining a suitable objective function that encourages the predicted depth map to be consistent with the input image or video stream.

This is using image reconstruction as a proxy for estimating depth. We can minimise the image reconstruction loss by projecting an image to nearby views using an estimated depth – this is the "view synthesis" aspect.

This approach has several advantages, including the ability to train models on large-scale datasets without the need for costly depth sensors and the flexibility to

apply the trained models to a wide range of tasks and environments.

The model is implemented using the PyTorch [61] framework, with the equirectangular projection model implemented as a PyTorch layer to enable drop-in replacement. We use Adam [62] with $\beta_1 = 0.9$ and $\beta_2 = 0.999$, a batch size of 8, and a learning rate of 10^{-8} for 10 epochs.

4.3 Evaluation

4.3.1 Dataset and Pre-Processing

As previously stated, there is a lack of publicly available omnidirectional depth datasets and benchmarking. Thus, we opted for a combined approach of gathering our own data and making use of publicly available internet videos.

The training data is entirely comprised of daytime footage collected by ourselves while walking around an urban area with a handheld Ricoh Theta Z1 camera. As such, there are numerous examples of both static and dynamic objects, such as buildings and motor vehicles and pedestrians. In terms of augmentation, we employ horizontal flips and jitter brightness, hue, saturation, and contrast to within 0.2 of their true values.

Additionally, we evaluate the use of the cube map, another omnidirectional projection. The environment is projected onto the faces of a cube in this scenario, and is generated online from the equirectangular image. This representation foregoes some of the equirectangular benefits in return for a distortion-free (but reduced field of view) image. This is solely for the pose network, which is fed the cube face in the direction of egomotion as an input.

We build additional validation sets using "in the wild" omnidirectional videos for qualitative evaluation. These were collected from publicly available YouTube videos, specifically two videos, each roughly two hours in length. One is comparable in that it is shot from the perspective of a pedestrian, while the other is captured from the perspective of a moped rider with a helmet mounted camera.

This is intended to assess the models' ability to adapt to a change in perspective, from pedestrian to vehicle. Additionally, because these images were captured with unfamiliar camera hardware, they will serve to verify the "virtual" camera model. These sets contain no frames that are used in the training pipeline. We perform no augmentation on these.

As the above have no ground truth depth, we further validate the designed model with the KITTI-360 [63] dataset. Though this dataset was not intended for omnidi-

rectional depth estimation, the data collection vehicle was equipped with a pair of side mounted fisheye cameras, which we form an equirectangular image from, and a Velodyne HDL-64E LiDAR sensor for sparse depth ground truth. The available LiDAR points are projected into the fisheye images, which form the ground truth information used for evaluation. While this provides a 360° field-of-view horizontally, the vertical is limited to the central 26.9°.

4.3.2 Competing Methods and Evaluation

To evaluate the effectiveness of our proposed approach, we compare it with several existing models in the literature that tackle the problem of omnidirectional depth estimation, as well as a comparable model for non-spherical imagery, namely BiFuse [49], Omnidepth [4], and Monodepth2 [2].

As described earlier, we stitch together a single equirectangular image from the pair of fisheye images available for KITTI-360 and perform depth estimation using our proposed approach. To ensure a fair comparison, we transform the predicted depth maps into dual fisheye format, mask them for existing ground truth LiDAR reprojections, and perform median scaling of the depth values (per image sample) following [60]. This scaling is done by multiplying the predicted depth maps by a scaling value, s, where $s = median(G_{depth})/median(P_{depth})$, with G_{depth} and P_{depth} representing the ground truth and predicted depth maps, respectively. This scaling ensures that the median depth value of the predicted depth maps matches that of the ground truth depth maps, allowing for a fair comparison across models.

By comparing the performance of our proposed approach with these existing models, we can demonstrate the superiority of our approach for omnidirectional depth estimation tasks.

4.3.3 Results

Quantitative Results

Table 4.1 shows quantitative results on common depth estimation metrics [30] of our models trained on self-collected data and tested directly on KITTI-360 without fine-tuning. These metrics quantify errors in terms of metre deviations from the ground truth as well as percentage accuracy within a certain threshold. To be clear, we evaluate only where ground truth LiDAR returns are available and therefore do not compare the whole scene evenly. For example, as there are no returns in the sky regions, these areas do not contribute to the quantitative results.

Model	Abs. Rel.	Sq. Rel.	RMSE	RMSE Log	$\delta < 1.25^1$	$\delta < 1.25^2$	$\delta < 1.25^3$
BiFuse	0.524	4.961	11.692	0.67	0.268	0.497	0.695
Omnidepth	0.750	10.327	12.885	0.815	0.232	0.437	0.606
Finetuned Monodepth2	0.392	4.00	9.57	0.522	0.383	0.661	0.820
Ours (CM, OM, SCp)	0.439	5.068	9.770	0.558	0.352	0.626	0.796
Ours (CM, OM, SCe)	0.440	5.234	9.749	0.556	0.363	0.630	0.796
Ours (CM, OM, SC)	0.423	4.770	9.531	0.542	0.372	0.643	0.807
Ours (CM, OM)	0.455	5.528	9.978	0.571	0.345	0.614	0.785
Ours (CM, OM, CP)	0.395	4.166	9.449	0.521	0.380	0.668	0.825
Ours (CM, OM, CP, SCe)	0.368	3.762	**9.184**	0.491	0.415	0.697	0.844
Ours (CM, OL, CP, SCe)	0.350	3.179	10.238	0.510	0.393	0.693	0.845
Ours (CM, OL, SC)	**0.339**	**3.066**	9.795	**0.485**	**0.421**	**0.717**	**0.859**

Table 4.1: Quantitative results on depth estimation metrics on KITTI-360 along with an ablation study of our model. Note the models are trained on self-collected dataset, without fine-tuning on KITTI-360. Where: **CM** signifies the use of the equirectangular camera model, **OM** the use of the optical flow mask *in conjunction* with established automasking, **OL** the use of *just* our optical flow mask, **SC** the use of spherical convolutional layers (where **e** and **p** signify their use in just the depth encoder or pose encoder respectively), **CP** the use of a rectified crop (cube patch) as input to the pose network.

As can be seen, our method and additions outperforms the others on nearly all metrics, with the overall best configuration a model using: 1) the proposed projection model, 2) our optical flow masking strategy in place of automasking and 3) spherical convolutional initial layers for *both* the depth encoder and pose encoder networks. This demonstrates the efficacy of the techniques proposed.

The histogram of the absolute errors of predicted depths, for those pixels where ground truth LiDAR reprojections are available, is given in Fig. 4.5. As can be seen, our approach compares favourably with other models, with the majority of predictions falling in the lower 3-metre error bins.

Qualitative Results

Figures 4.6, 4.7, and 4.8 show several qualitative depth prediction examples — from the validation set, the "in the wild" set and the KITTI-360 set respectively. For Figure 4.6, refer to Table 4.1, where **A** and **B** denote a configuration of [CM, OM, CP, SCe] and [CM, OM, CP] respectively. Lighter pixels indicate a closer depth value and vice versa. It is clear that the proposed model produces more accurate and meaningful depth maps. Particularly, the depths to some objects, like cars, are sharp and distinguishable from the background.

The models struggle with the sky and tunnel sections, resulting in erroneous estimations. The former may pose a challenge as it is a large region of mostly textureless pixels, which is exacerbated by the 360° nature of the images. The latter

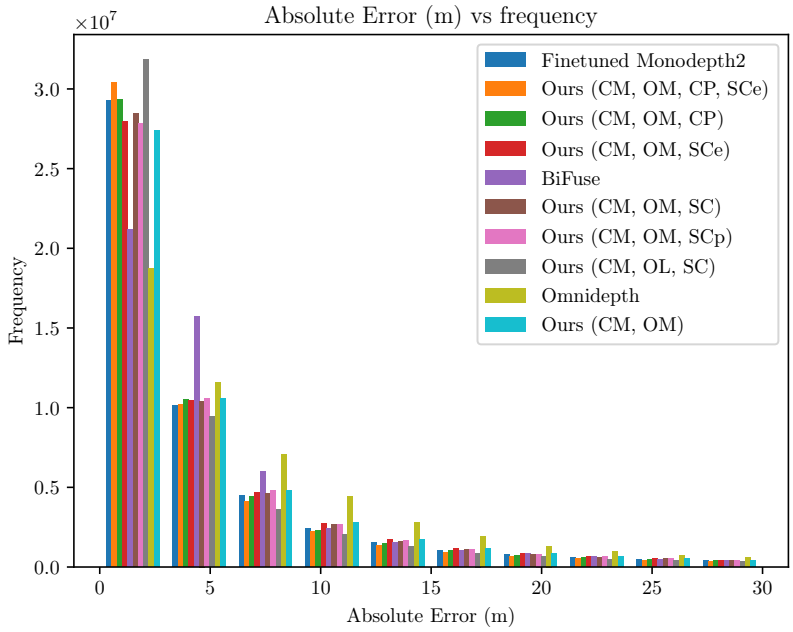


Figure 4.5: Histogram of errors on depth estimation.

Figure 4.6: Qualitative samples from the validation set for different configurations.

can be reasoned as the model struggles with the switch to an "indoor" scale, being predominantly trained on outdoor imagery. Likewise, [4] and [49] for the inverse reason. In future, we will investigate how to tackle these with more intelligent masking and outdoor-indoor scene cross training.

Despite this, the addition of our alternative masking strategy shows a qualitative improvement in depth consistency of the objects in the scene while preserving definition along the edges.

4.4 Discussion

In this chapter, we present a unique approach for extending view synthesis-based depth estimation models to 360° equirectangular projection images.

We achieve state-of-the-art results when compared to previous models by utilising

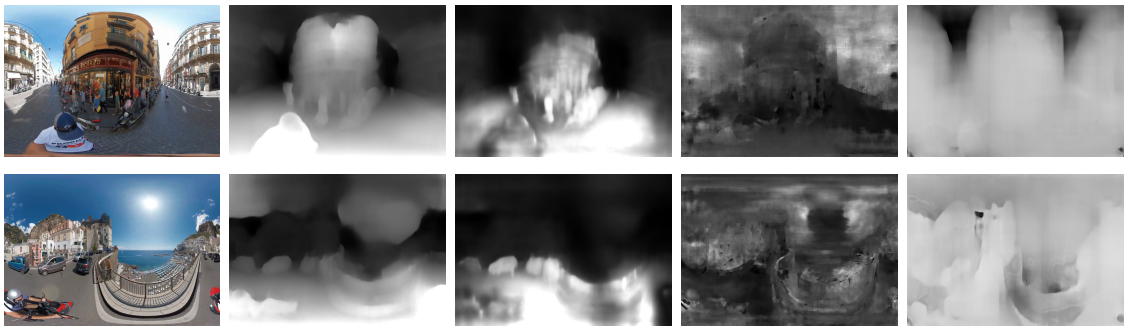


Figure 4.7: Qualitative samples from the additional "in the wild" validation set

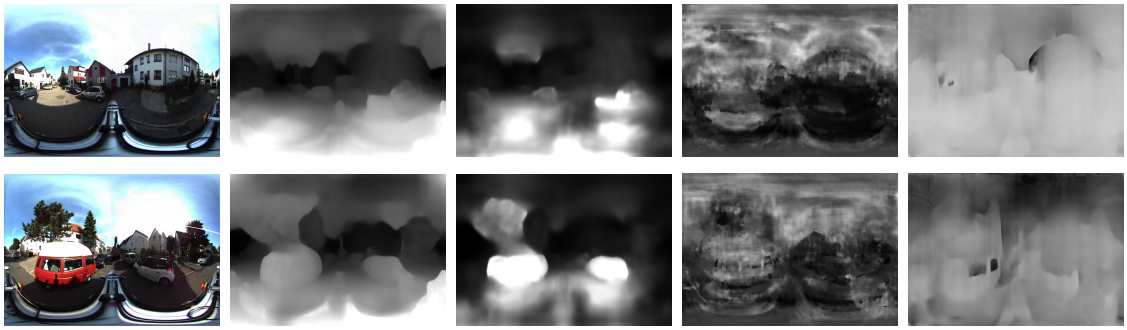


Figure 4.8: Qualitative samples from KITTI-360

a camera projection model that takes into consideration the spherical nature of the data, spherical convolutional input layers, and an optical flow-based binary masking approach.

The results presented in this chapter imply that existing networks are capable of adapting to an entirely different image format – even those containing severe distortions, as is the case with equirectangular images – if certain considerations are taken into account beforehand.

This is important in this case, as there exists a much broader body of literature on regular depth estimation and comparatively little for the omnidirectional equivalent. This leaves open the possibility of the field keeping pace with regular depth estimation research by adapting and extending the state-of-the-art methods and ideas.

Another key takeaway from the results presented here is that being selective with the data and training signal is important, moreso than with regular depth estimation. The all encompassing field-of-view is both a blessing and a curse, in the sense that there are no theoretical blind spots but a much higher "surface area" where noise can be injected into the training. Usually, this is handled via masking.

However, Figure 4.3 show how current automasking methods fail completely to filter for relevant pixels. By using an alternative masking scheme, we show state-of-the-art improvements over the previous (as detailed in Table 4.1). This underscores the need to take certain considerations into account while dealing with omnidirectional imagery.

It is also important to discuss the limitations with these results.

The most noticeable limitation are the depth estimates for the sky region. In many cases, these are obviously wrong, as can be seen for example in Figure 4.8, where these regions are estimated to be close. This issue is not unique to the omnidirectional domain and exists in regular depth estimation, but is exacerbated by the wide field-of-view, where the sky region occupies a far larger proportion of the image.

Another limitation on these results is whether this (or ideas along a similar vein) is applicable to other omnidirectional projection models. The results imply that similar adaptation using appropriate projection equations would yield similar results, but further experiments would be needed to validate this.

Additionally, this work highlights an urgent need in the field for high quality, publicly available datasets tailored for omnidirectional depth.

Future work could focus on refining erroneous outputs that appear in some portions of the image, such as the sky regions, and when travelling through difficult areas, such as tunnels.

Addressing these challenges may include using multi-modal data such as LiDAR returns for instance, or enhancing multi-task learning through the use of semantic segmentation.

In particular, the use of LiDAR data would be of interest. Many commercially available LiDAR sensors similarly have a 360°field-of-view, albeit usually just in the horizontal.

Therefore, LiDAR sensors and omnidirectional cameras are complementary, with the former providing a source of high quality depth estimation and the latter providing rich colour information.

Building on the work detailed here, the next chapter will explore the pairing of these two modalities for depth estimation. Crucially, the next chapter will explore how some of the limitations highlighted here can be overcome.

5 | Omnidirectional Depth Estimation With LIDAR

5.1 Overview

With the widespread commercial availability of omnidirectional cameras nowadays, they present a unique opportunity for mobile sensing applications, particularly for mobile robotics and autonomous vehicles. Such cameras provide an all encompassing field-of-view, theoretically leaving no blind spots in vision. This is of course a very attractive property for many systems to have.

However, this comes with the trade off of severe image distortions and very different properties compared to images obtained via regular perspective cameras. As such perspective cameras are the de facto standard, the vast majority of datasets (and consequently work built on them) have an implicit bias toward this format. These combined lead to much of the existing literature from various domains – such as depth estimation – being ineffecient or wholly incompatible with omnidirectional images. Consequently, the body of work based on omnidirectional images is much smaller in comparison. Translating techniques and insight gained from the perspective domain to the omnidirectional domain will help to bridge this gap. Further, verification of existing techniques but in the omnidirectional domain will help guide future work in this area.

One technique is the use of depth values from a LiDAR sensor as a ground truth source to guide the training of a depth estimation model. This can be beneficial as such sensors provide a source of high quality information and provide usable error signals. Pairing them with omnidirectional cameras seems like a natural fit, as many LiDAR sensors provide a 360°horizontal field-of-view. However, most are normally quite sparse and limited in their field-of-view. As they are often paired with perspective cameras, the field-of-view limitation is usually not too problematic.

However, this is exacerbated with the complete view afforded by omnidirectional

cameras with most LiDAR sensors only providing a relatively thin slice of depth for the entire image. Therefore, we hypothesize that utilising depth from LiDAR requires some thought, so as not to overweight this region during training.

The main contributions of this chapter are as follows:

- Introduce an extrinsic calibration method for aligning a LiDAR sensor with an omnidirectional camera based on Bayesian optimisation
- Propose a novel, weighted loss function aimed at incorporating LiDAR returns as ground truth depth for omnidirectional depth training, which we call the Grid Loss.

5.2 Methodology

Here we detail our method of fusing LiDAR and omnidirectional image data, outlining: extrinsic calibration, model architecture, loss functions and training scheme.

5.2.1 Extrinsic Calibration

While LiDAR sensors provide a high quality (albeit sparse) degree of depth information, utilising the data returned with image data alone will not yield sufficient results. This is due to the fact that the 3D points are not represented in the same coordinate frame, resulting in inaccurate projections.

As with any multi-sensor system, some level of calibration is necessary. As outlined in the previous chapter, a "spherical" camera model is used. What we would like is to derive a rigid transformation between points in the LiDAR frame to the camera frame, to then perform a projective transformation from the camera frame into pixel co-ordinates using this new camera model. This would allow the representation of high quality ground truth depth values in image space, establishing a direct link between pixel values and depth values. While it is true that the ratio of LiDAR points to image pixels will be sparse, it will at the very least be aligned.

Here, we take a modified form of the Perspective-n-Points (PnP) method. With a standard PnP setup, we would like to estimate the pose of a calibrated camera given a set of identified and observed three dimensional points and their two dimensional projections in the image. When it comes to 3D object tracking and camera localization and tracking, the PnP problem is relevant. It has been well studied and applied in the robotics and computer vision domains. It is frequently utilised

in applications such as Simultaneous Localization and Mapping (SLAM), Visual Odometry (VO), Structure-from-Motion (SfM) and image-based tracking pipelines.

However, it is less thoroughly investigated for omnidirectional imaging, as is the case with many well-known topics in this domain.

Starting with the data we have will help us outline the situation more clearly. Here, we have:

1. A camera projection model, which lets us project three dimensional points into pixel space.

2. Three-dimensional points collected using a LiDAR

Simply projecting the LiDAR points into pixel space is incorrect and will yield unsatisfactory results. The reason being is that the physical placement of the LiDAR sensor itself is away from the center of projection. Therefore, an initial transformation of the points is required prior to calibration, and this forms the reasoning behind extrinsic calibration. Estimating this transformation accurately is crucial as it will form the basis of the LiDAR ground truth data.

Intuitively, the approach described here can be thought of as "aligning" two sets of points with each other:

1. Select a set of pixels in the image.

2. Identify the corresponding three-dimensional LiDAR points.

3. Apply a transformation, T_i, to the points.

4. Project the transformed points back into the camera frame.

5. Determine alignment by using a distance metric (in this case, Euclidean).

6. If the points are in close proximity to each other, the process concludes, having found a suitable extrinsic calibration.

7. If not, the process repeats from step 3 with an adjusted transformation.

This bears some similarity to RANSAC but only in that it is an iterative process.

Regarding steps 1 and 2 in the above, the choice pixels and LiDAR correspondences are done manually, however these are fixed for the duration of alignment. These can in theory be arbitrary but a checkerboard was used as a guide.

The key to this process is the optimization of the transformation. Finding this transformation is not trivial, as it's a global optimization problem in a high-dimensional space without a clear gradient to guide the search. A random search

approach would not be effective due to the high-dimensionality, and we don't have the advantages that some optimization problems in machine learning have. For instance, methods like gradient descent leverage mathematical shortcuts and have access to derivatives of the function to expedite the evaluation. In our case, we need to find a more efficient way to guide the search for an optimal transformation.

To address this, we use Bayesian optimization, a powerful tool for optimizing expensive, complex functions. Bayesian optimization constructs a probability model for the objective function and then exploits this model to find the optimum by trading off exploration against exploitation. This will be explained in the next section.

Bayesian Optimisation Applied to Calibration

As outlined, optimising the transformation from LiDAR to pixel space is challenging. One attempt at solving this is Bayesian optimisation.

Bayesian optimization is a technique that utilizes Bayes' Theorem to guide the search for the minimum or maximum value of an objective function, denoted as f, within a bounded domain. This method is particularly well-suited for optimizing complex, computationally expensive, or noisy objective functions through a process called surrogate optimization.

In this context, the *objective function* is a measure of the alignment error between the projected LiDAR points and their corresponding pixels in the image. The goal is to minimize this alignment error by finding the optimal transformation, T_i, that aligns the 3D LiDAR points with the 2D image pixels most accurately.

The *surrogate function* is an approximation of the objective function, constructed based on the points sampled so far. The surrogate function is used in to model our belief about the objective function, and is where the Bayesian nature of this approach starts to show. The surrogate function serves as our posterior, which we use to guide future sampling. In other words, our belief about the current objective function influences the areas of the search space we focus on, as we aim to obtain better results.

In this case, a Gaussian Process is employed as the surrogate function. Gaussian Processes are flexible, non-parametric models that can capture complex, non-linear relationships and provide uncertainty estimates. This makes them suitable for modeling expensive or noisy objective functions, like the alignment error in this context.

The Bayesian optimization process iteratively refines the surrogate function by sampling new points, which are proposed by an *acquisition function*. The acquisition

function balances exploration (sampling points in areas with high uncertainty) and exploitation (sampling points where the surrogate function predicts low alignment errors). By efficiently updating the surrogate function based on the sampled points, Bayesian optimization narrows down the search for the optimal transformation T_i that minimizes the alignment error. Here, we use the Upper Confidence Bounds method for acquisition.

More formally, the objective function f will be sampled at:

$$x_t = \arg\max_x a(x|D_{1:t-1}) \tag{5.1}$$

where a is the acquisition function and $D_{1:t-1} = (x_1, y_1), ..., (x_{t-1}, y_{t-1})$ are the $t-1$ samples drawn from f thus far.

Our goal is to find a rigid transform from the LiDAR frame to the camera frame, with the search space consisting of possible (bounded) combinations of transformation parameters. In this context, the samples drawn represent sets of transformation parameters.

The optimization is performed in two stages: an *initial*, exploratory search followed by a secondary, *refining* (or fine-tuning) stage based on promising candidate solutions. Table 5.1 presents the optimization parameters and algorithmic bounds (constraints). In the first stage, the search space for optimization is restricted to lie within the specified ranges. The second stage refines the proposals from the first stage by further constraining the search to lie within a certain margin of the proposed values.

Figures 5.1 and 5.2 display examples of LiDAR-camera projections before and after calibration, respectively. Compare the view prior to calibration (top left) to that after extrinsic calibration (top right), as well as their corresponding colored LiDAR point clouds (bottom left and bottom right).

As shown, the uncalibrated projection, although close, is not accurate, which is confirmed by the colored point clouds. The misalignment would worsen if the LiDAR and camera were positioned farther apart or differently. However, the calibrated projection, although imperfect, is considerably more precise, as demonstrated by the alignment of various objects in the scene, such as the blue bin and traffic cones, and the alignement of the points on the checkerboard and chair shown in Figure 5.3.

Despite utilizing a top-of-the-line LiDAR sensor, the vertical field of view remains limited, covering only a relatively thin slice at the center.

Parameter	Value
Initial points	25
Number of initial iterations	15
Number of secondary iterations	10
Rotation (x)	[-5, 5]
Rotation (y)	[175, 185]
Rotation (z)	[175, 185]
Translation (x)	[-0.07, 0.07]
Translation (y)	[-0.07, 0.07]
Translation (z)	[-0.3, 0.3]
Rotation margin	0.05
Translation margin	0.005

Table 5.1: Hyperparameters and constraints for Bayesian Optimisation calibration, where values in *[...]* are allowable lower/upper bounds and *margin* is the allowable perturbation in the refining stage.

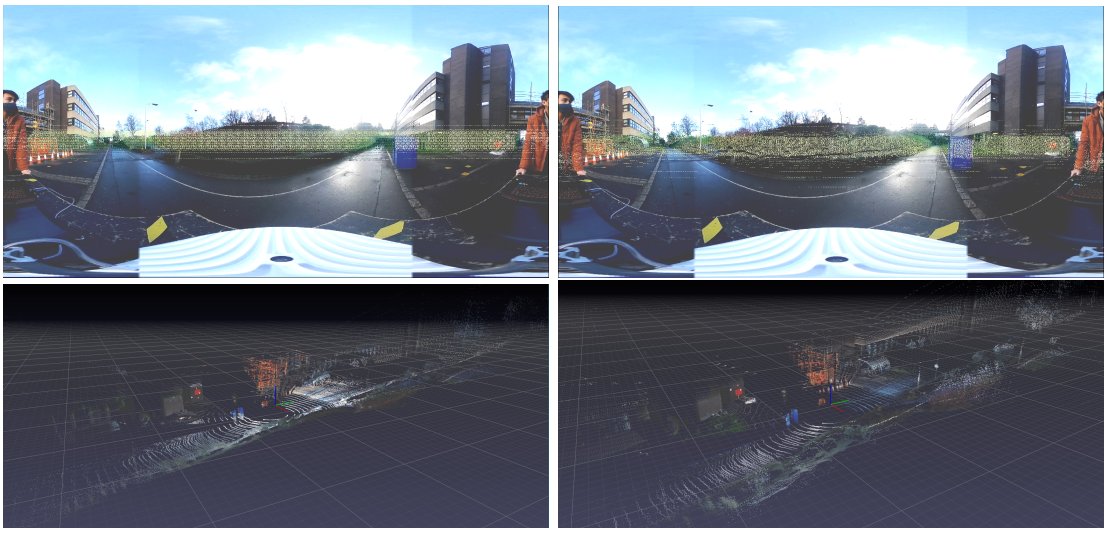


Figure 5.1: Outdoor LiDAR to camera projection.

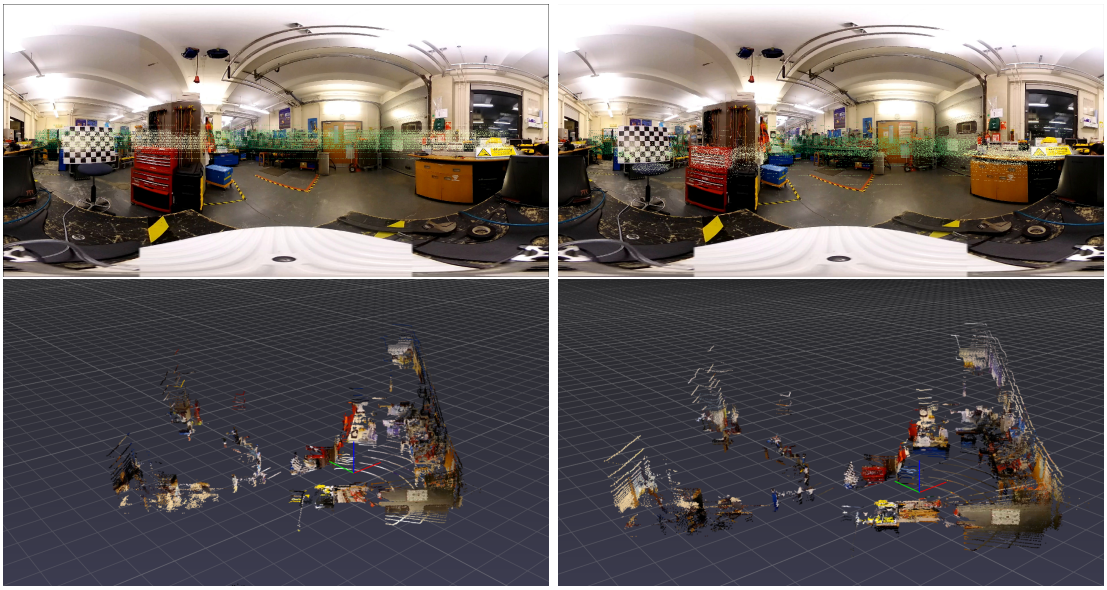
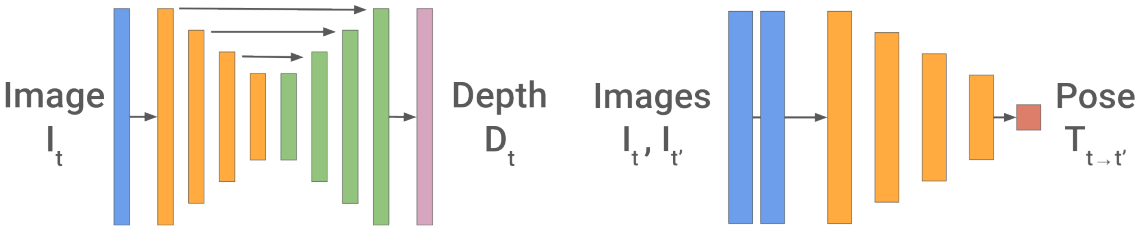


Figure 5.2: Indoor LiDAR to camera projection.

Figure 5.3: Emphasis on results of extrinsic calibration, using the checkerboard shown in Figure 5.2. Left image shows the uncalibrated projection, while the right is calibrated based on the calibration method outlined in this chapter.

5.2.2 Model Architecture



(a) Depth (b) Pose

Figure 5.4: Encoder-decoder network for depth estimation, and a ResNet18 backbone model for pose estimation.

As we would like to investigate the effect of incorporating ground truth depth information directly into an omnidirectional training pipeline, we choose a model architecture identical to the one described in the previous chapter. This enables fairer comparisons and conclusions to be drawn.

This architecture is made up of two networks: a depth estimation network and a separate pose estimation network, both of which are fed RGB images as inputs to the system and is shown in Figure 5.4. The former is based on an encoder-decoder U-Net with skip connections, and it accepts as an input an equirectangular RGB image. The output of the decoder is converted to depth using the equation $D = 1/(a\sigma + b)$, with the co-efficients a and b chosen to keep D within the range of 0.1 and 100, respectively. The pose estimation network is based on a ResNet18, which produces a relative 6 DOF pose from a pair of input photos.

An explanation of this architecture is provided in the previous chapter.

5.2.3 Grid Loss

As shown in Figures 5.1 and 5.2, pixels where ground truth depth data are available are relatively sparse. This is due to the small vertical field of view of the LiDAR sensor (40°) in comparison to the image as a whole. As such, integrating ground truth depth data into a depth estimation pipeline may require a different approach.

Again, this is an example where a limitation is exacerbated by the complete field of view of omnidirectional images. When using more traditional perspective images, this is usually much less of an issue as the camera's field of view is much smaller.

One potential solution to this problem is via additional hardware. The returns from multiple LiDAR sensors, if positioned such that each covers a portion of the image, can be fused together. Such an approach would provide complete coverage,

however it would be costly, as one would require at minimum five of such sensors to have at least 180°vertical coverage. As well as this, all additional sensors would need to undergo extrinsic calibration.

Here, we explore a mitigating measure for this scenario, which we call the *Grid Loss*.

Conceptually, it is quite simple and can be thought of as discretizing the image into a grid, and assigning greater importance to those grid cells with a higher density of ground truth points.

Specifically, we weight the per cell importance via the application of a unit softmax function, σ over all cells containing depth values, defined for $\sigma : \mathbb{R}^K \rightarrow (0,1)^K$ when K is greater than 1.

$$\sigma(c)_i = \frac{e^{c_i}}{\sum_{j=1}^{K} e^{c_j}} \text{ for } i = 1, ..., K \text{ and } c = (c_1, ..., c_k) \in \mathbb{R}^K \tag{5.2}$$

Where c is an input vector of length K, (non-empty grid cells) with each element comprising of the *frequency* of LiDAR points in that cell.

With each grid cell now having a corresponding weight, the final loss is the sum of *weighted* per-cell sum of absolute differences:

$$\text{Grid Loss} = \sum_{i=1}^{K} \sigma_i \sum_{j=1}^{n} |y_{in} - \hat{y_{in}}| \tag{5.3}$$

Where n is the number of LiDAR points in cell K, and y_{in} and $\hat{y_{in}}$ are the predicted and ground truth points respectively. When dealing with a batch, the loss is computed for each predicted-ground truth pair and the mean average of the batch is taken. To be clear, we are only considering cells which contain LiDAR points. We are effectively weighting and normalizing the cell importance via the application of a unit softmax function.

Shown in Figure 5.5 is a simplified visualisation of how the loss weighting works. For visualisation purposes, only three cells are shown. Each red dot is a ground truth depth value and the opacity of tint in the cell visualises the cell's contribution to the overall loss. As cell number 1 has the most points inside, it is weighted more, followed by cell number 2, then finally cell number 3. Hence, cell 1 contributes the most, then 2, then lastly 3.

As this is an example, only weighting based on three cells are shown. Of course, in actual usage all cells would be taken into account. To reiterate, these are not real LiDAR returns but merely an illustration.

This example highlights where this weighting may be useful, especially in an

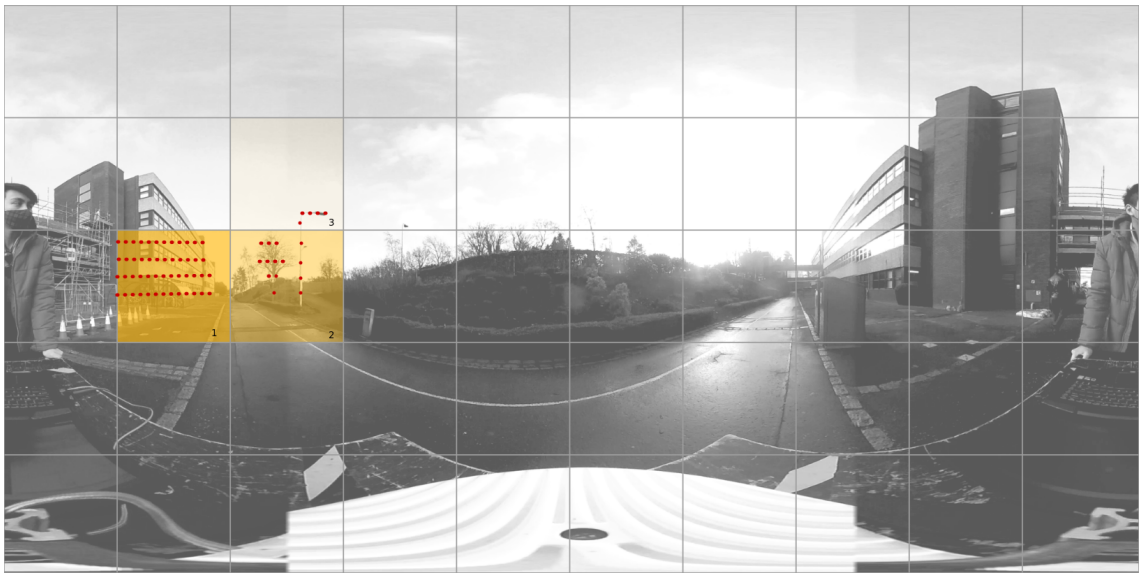


Figure 5.5: A simplified illustration of the loss weighting in action.

omnidirectional context. Note that there is no assumption on the *quality* of the points per se. Only that we assume that if more LiDAR points are available in an area, then we can be more confident in using that area to guide depth estimation.

5.2.4 Training and Implementation

We build on the work shown in the previous chapter by following a similar training and implementation procedure for many of the same reasons, as the challenges faced are also very similar.

As publicly available non-synthetic, outdoor, omnidirectional depth datasets are rare, we start with models pre-trained on the regular depth estimation task. We argue that this is beneficial as we minimise the domain gaps the model has to bridge versus using indoor, synthetic datasets.

Additionally, by choosing pre-existing models, we hope to demonstrate the ways our proposed method is applicable in the perspective-to-omnidirectional adaptation process.

Here, we follow the self-supervised objective as outlined in [59], [40], [60], [2], [41], substituting the automasking procedure in [2] with our optical flow based mask.

In addition to the above, we formulate our final loss by the addition of the Grid Loss. Crucially, we do not replace the self-supervised objective.

In practical terms, the training and validation process is implemented using the PyTorch [61] framework, building upon our previous work. We use Adam [62] with $\beta_1 = 0.9$ and $\beta_2 = 0.999$, batch size of 8, learning rate of 10^{-8}, for 10 epochs.

5.3 Evaluation

5.3.1 Dataset

To the best of our knowledge, a dataset containing high resolution equirectangular images with corresponding LiDAR returns or point clouds does not exist in the literature. Thus, we opted to collect our own data using a Velodyne Alpha Prime sensor and a Ricoh Theta Z1 camera mounted on a bespoke rig. At the time of writing, this LiDAR sensor is one of the highest quality sensors available, consisting of 128 channels.

The training data is entirely comprised of daytime footage collected by pushing the rig around an extra-urban, university campus setting. While collection was performed in daytime, the weather ranged from sunny to cloudy, providing a variety of lighting conditions.

Given the setting, there are numerous examples of both static and dynamic objects, such as buildings, pedestrians and vehicles. In terms of augmentation, we employ horizontal flips and jitter brightness, hue, saturation, and contrast to within 0.2 of their true values online at training time.

Similarly to the method discussed in the previous chapter, we incorporate the use of the cube map, another omnidirectional projection. This is solely for the pose network, which is fed the cube face in the direction of egomotion as an input.

In contrast with evaluation on the KITTI-360 dataset seen in the previous chapter, our collected dataset has more of a depth estimation focus in mind. Where we previously had to form an equirectangular image from two fisheye images as well as handling the corresponding LiDAR projections, here we have no need to. The RGB image captured by the camera is provided in an equirectangular format and LiDAR projections can be done directly to this. This avoids any potential artifacts or errors which could arise from warping between formats.

To be clear, all the data used for training and validation are self-collected using a bespoke rig, and is separate from the dataset collected in the previous chapter. While self-collected, the validation set is held out from training so as not to contaminate results.

5.3.2 Quantitative Results

Shown in Table 5.2 are quantitative results on common depth estimation metrics [30] of our models trained on the self-collected LiDAR dataset and tested on a held

Model	Abs. Rel.	Sq. Rel.	RMSE	RMSE Log	$\delta < 1.25^1$	$\delta < 1.25^2$	$\delta < 1.25^3$
Finetuned Monodepth2	1.185	5.821	4.077	0.865	0.284	0.498	0.652
Ours (CM, OL, CP, SC)	0.736	1.789	4.176	0.733	0.236	0.464	0.653
Ours (CM, OM)	0.734	3.680	4.674	0.744	0.258	0.482	0.650
Ours (CM, OL, SC)	0.722	1.761	3.864	0.690	0.248	0.496	0.699
Ours (GL, CM, OL, SC)	0.718	1.758	3.967	0.707	0.237	0.475	0.662
Ours (CM, OM, SC)	0.717	3.324	4.513	0.710	0.268	0.500	0.683
Ours (CM, OM, CP, SC)	0.681	2.836	4.342	0.666	0.247	0.529	0.731
Ours (CM, OM, CP)	0.659	2.758	4.277	0.677	0.265	0.524	0.721
Ours (GL, CM, OM, SC)	0.651	2.807	4.410	0.655	0.254	0.523	0.716

Table 5.2: Quantitative results on depth estimation metrics on the held out validation set along with an ablation study of our model. Where: **GL** signifies the use of the Grid Loss, **CM** the use of the equirectangular camera model, **OM** the use of the optical flow mask *in conjunction* with established automasking, **OL** the use of *just* our optical flow mask, **SC** the use of spherical convolutional layers **CP** the use of a rectified crop (cube patch) as input to the pose network.

out validation set. The metrics shown assess errors in metre values in addition to percentage accuracy within a certain threshold.

It can be seen that the addition of the proposed loss leads to improved performance in many metrics. The overall best configuration is one utilising: 1) the proposed Grid Loss, 2) the proposed camera model, 3) our optical flow masking strategy in place of automasking and 4) spherical convolutional initial layers for *both* the depth encoder and pose encoder networks.

However, overall conclusions are murky, as some configurations perform better than others in particular metrics. For example, the (GL, CM, OM, SC) configuration performs the best in Absolute Relative Error, while the (CM, OL, SC) configuration performs better in Squared Relative Error.

This could be due to a number of causes and leaves some questions unanswered. One straightforwad (if time consuming) question to answer is the effect of training data. As a rule of thumb, the more data that is available the better. In this case, perhaps the data available was simply not enough to train the network to fully leverage the new, more complex additions.

These findings suggest that the current approach would face challenges in real-world implementation, yet the modifications do lead to enhanced performance. For safety critical systems, these errors would be too high and further work is required to improve this.

The histogram of the absolute errors of predicted depths, for those pixels where ground truth LiDAR reprojections are available, is given in Fig. 5.6. These errors are drawn from the unseen validation set. As can be seen, our approach compares favourably with other models, with the majority of predictions falling in the lower

2.5-metre error bins.

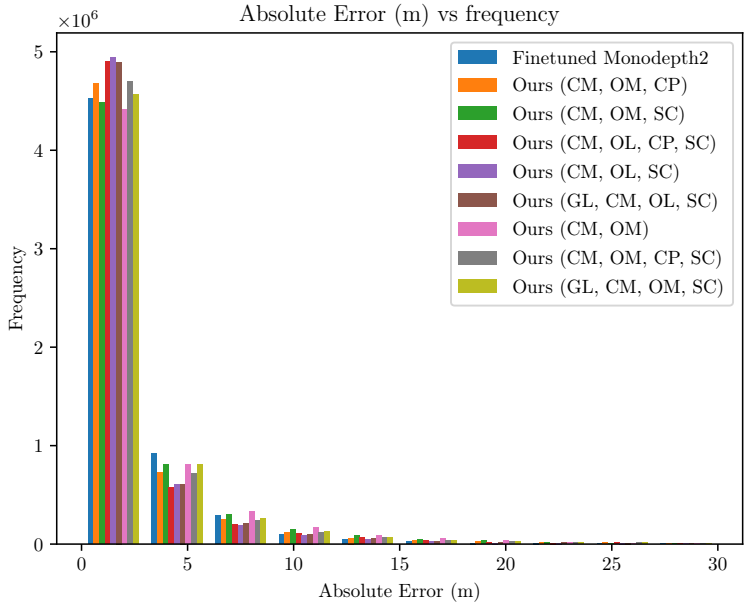


Figure 5.6: Histogram of errors on depth estimation.

5.3.3 Qualitative Results

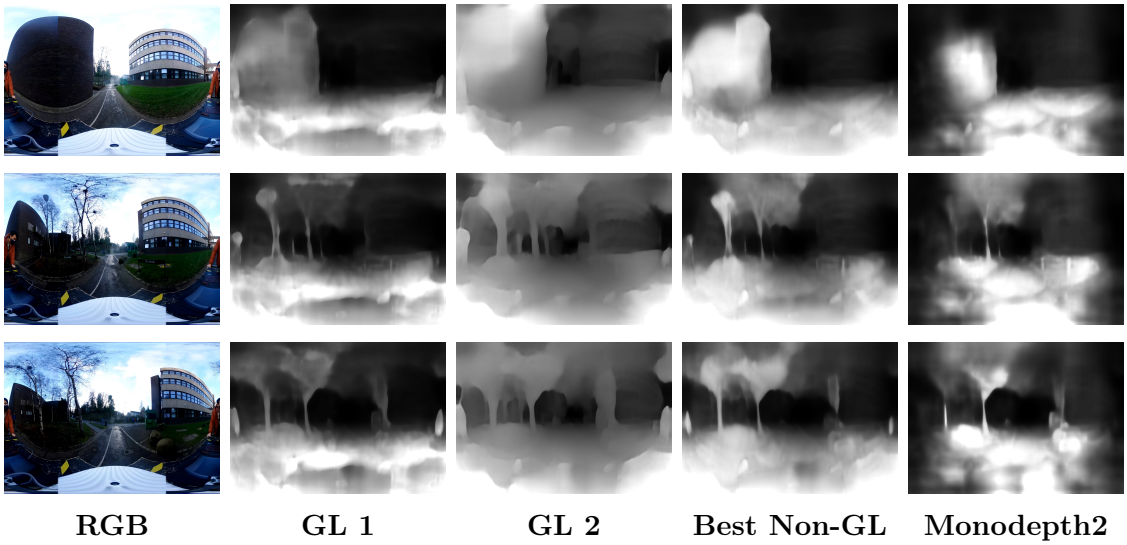


Figure 5.7: Qualitative samples from the held out validation set.

Figure 5.7 show several qualitative examples of depth estimation from the held out validation set, where Grid Loss 1 and 2 are the models trained with the new loss and utilising two permutations of the optical flow masking strategy, blended with

established automasking and without respectively. Here, **GL** signifies the use of the Grid Loss, with 1 being the model trained with optical masking *and* established automasking, and GL 2 being the model trained on *just* our optical flow mask. These are denoted by **OM** and **OL** respectively in Table 5.2. Grid Loss 1 is the best performing of the two in terms of raw metrics, as shown in Table 5.2.

As can be seen, qualitatively speaking, the results are sharper and contain finer details compared to a non adapted model (finetuned Monodepth2 in this case), making it easier to distinguish relatively thin objects such as trees.

In contrast, it also lacks the heavy vignetting effect present in the non adapted model.

Interestingly, the blended masking approach (Grid Loss 1) has far fewer spurious estimations in the sky region versus the solo approach, where it seems to struggle more.

Crucially, our alternative masking strategy coupled with the Grid Loss shows a qualitative improvement in quality, preserving more definition along the edges while showing smoother, more consistent depth regions.

5.4 Discussion

In this chapter, we present a unique approach for incorporating omnidirectional LiDAR data into the training pipeline of an omnidirectional depth estimation model.

Specifically, we introduce several items of interest.

First, we present a Bayesian optimisation based extrinsic calibration process which utilises the equirectangular camera model introduced in previous chapters. This process lets us pair LiDAR and omnidirectional images and obtain per-pixel ground truth depth information. Per-pixel depth data for omnidirectional imagery is extremely rare in the literature, to the best of our knowledge.

Obtaining data like this could be an enabler for future work and applications. Being theoretically applicable to any equirectangular image and LiDAR pair, this technique may help to bridge the data availability gap in the literature between perspective and omnidirectional images. Bridging this gap could spur further research and help bring parity to this domain.

Secondly, we show that the incorporation of this LiDAR information in the training pipeline of a depth estimation model is feasible and desirable. The primary technique to achieve this is by discretizing the omnidirectional image into a grid, and incorporating a weighting scheme to the LiDAR projections based on the frequency of depth points in each cell. We incorporate this information into the training by

introducing a supervised term which computes a weighted L1 loss – which we call the grid loss – to the self-supervised loss.

The results presented in this chapter imply that there are tangible benefits to incorporating LiDAR ground truth to omnidirectional depth estimation. Despite the sparsity of even top of the line LiDAR sensors in comparison to the image, even this is enough to improve performance.

If in future, hardware improves such that the LiDAR field-of-view increases significantly, further experiments can determine with greater certainty how much improvement the frequency-based weighting scheme provides. Alternatively, further work can be done to fuse and project multiple LiDAR sources to the same image which may be able to provide increased coverage.

In terms of limitations, the improvements gained versus strictly self-supervised training is not as big as was initially hypothesized. It is highly likely that this is an issue of data, and of not having enough training samples. The training set used in the previous example was a lot larger than the one presented in this chapter, and comparatively much easier to obtain. For such a data-driven method, having an adequate amount of data is important.

Future work could focus on repeating the experiments with more data, both in terms of raw samples and in density of LiDAR returns. This could then further examine grid loss efficacy and enable the community to explore other potential avenues of research, as such data is very rare.

Part III

Conclusions

6 | Conclusions

6.1 Conclusions

In conclusion, this thesis presents new strategies for extracting environmental depth representations from perspective and omnidirectional monocular vision. Such approaches are helpful in and of themselves, but they also demonstrate to the research community, particularly in the case of omnidirectional vision, that it is feasible to convert methodologies developed for one modality to another.

The work presented here is motivated by the fact that mobile autonomous systems must have the capacity to detect their surroundings. Frequently, the environment is rich with critical information required for decision making, planning, and action. Even more useful would be the ability to do so with modest hardware resources, as this would lower the barrier to entry for safe autonomous systems.

The ability to facilitate depth estimation on a variety of camera hardware is also a factor. As the price and availability of omnidirectional cameras grow more accessible for consumers, it would be especially advantageous for autonomous systems to utilise their capabilities. However, compared to traditional perspective cameras, these camera systems have received much less attention, particularly for depth estimation applications. Perspective systems are the de facto default system in the bulk of literature on computer vision. As such, many of the methodologies and datasets available have an implicit bias toward these systems.

Unfortunately, this makes it difficult to directly apply much of the existing work to omnidirectional cameras. Hopefully, the experiments shown herein highlight that it is indeed possible to leverage a lot of the existing techniques applicable to conventional perspective cameras for omnidirectional cameras.

Firstly, we present a method to generate object-scale top-down representations from a single perspective image. The key contribution is the generation of such a representation using an adversarial technique. As a consequence of the conducted experiments, a model trained in this manner generalises across datasets better than

baseline, strictly supervised models. We presented this approach at The 25th International Conference on Pattern Recognition (ICPR 2020).

After this, we shifted our focus to extending such capabilities to omnidirectional cameras. However, instead of a per-object depth schema, we instead opt for dense depth. Our reasoning is that the complete scene view afforded by such cameras contain a wealth of information. Omnidirectional depth could blend some of the best of properties of LiDAR systems (high FOV) and traditional camera systems (colour information and density of information). As such camera systems are becoming more affordable and lightweight, autonomous systems could benefit from high-quality, information rich yet low cost depth sensing. This would be an enabler for many other applications in the domain.

Lastly, we extend the above work to incorporate LiDAR training data in order to improve performance and accuracy. The difference here is that the LiDAR is complementary to the system and is not required during inference. It can be imagined that such a setup would be beneficial in the sense that only one LiDAR is required to improve the depth estimation models for many cameras and is itself not required to be deployed in the field. Crucially, we present an semi-auto calibration method for a LiDAR sensor and an omnidirectional camera. This lets us align the data between the two modalities correctly in order to extract correct depth values for training. Additionally, a weighted loss is proposed which takes into account the density of LiDAR points in a given region of the image in order to provide more local refinement.

6.2 Limitations and Future Work

There are limitations to the methodologies presented in this thesis.

With regards to bird's eye view generation, the proposed system requires additional training for extra object classes. Additionally, while there are quantifiable improvements versus baseline models, the absolute errors are still relatively large and therefore would require additional training and fine tuning for real world deployment. Another limitation is the object detection itself. While the system is agnostic to detection methodology, it is highly dependent on it and will vary with its accuracy. As with all camera based systems, it is susceptible to areas of low texture and extreme exposure (or lack thereof). Balancing the adversarial training can also be tricky, and an imbalance can result in getting stuck in a local minima.

Future work in this area could involve exploring how additional object classes are incorporated at training time in a scalable manner. For example, one could explore

if a strategy inspired by active learning [64] could be incorporated. In brief, this is a technique which enables a model to learn from a limited amount of labeled data by iteratively selecting the most informative samples for labeling. These samples are the ones which are expected to provide the most significant impact on the model's accuracy when they are labeled and added to the training set. In theory, this reduces (but does not eliminate) the overhead of manual labelling or data collection and lets the model ingest not only more data, but in a more tailored manner.

Additionally, it could also involve balancing the adversarial training better.

For the self-supervised omnidirectional depth work, some similar shortcomings are prevalent. It can be seen from the results that erroneous depth values are assigned to challenging regions, such as large sky regions and while traversing indoor/tunnel sections. Regarding the sky regions, it is a challenge for depth estimation using perspective pinhole, but is exacerbated in the omnidirectional case due to the much higher field-of-view capturing more of it in comparison. Additionally, this work has made it clear that the lack of high-quality public datasets tackling this problem is hindering future research.

Future work could involve tackling these two issues. A priority for future work should be a concentrated data collection effort. Futher research in this area can be enabled by gathering a diverse and expansive dataset that encompasses a broad range of environmental conditions. This data collection should focus on capturing images from the areas where the current depth estimation methods struggle the most - large sky regions and transitionary regions, such as outdoor-to-indoor or tunnel areas. By securing more data from these challenging regions, it could be that it can also address the issue of erroneous depth values.

Of course, the work incorporating LiDAR data into the training pipeline is also not without issues. Despite how sparse LiDAR returns can be, it is enough to provide a tangible benefit. However it was not as large as was intially hypothesized.

Future work could involve improving the density and coverage of LiDAR points, either through hardware improvements in the sensors themselves or via the fusion of multiple LiDAR sensors. This would also enable further experiments on the true efficacy of the proposed frequency-based weighting scheme. Additionally, this would provide more training data, which in turn should improve the performance.

Lastly, future work would involve investigating the *integration* of the methodologies outlined in this thesis into a cohesive system. For example, the omnidirectional depth methodologies could be integrated as a component within a self-driving or driver assistance system "stack". This would receive as input a video stream from a chassis-mounted camera and inform a collision avoidance systems of nearby haz-

ards. These hazards could be further corroborated by the object-level bird's eye view generation outlined in Chapter 3, which could act in tandem or as an independent component.

Bibliography

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *arXiv preprint arXiv:1512.03385*, 2015.

[2] Clément Godard, Oisin Mac Aodha, Michael Firman, and Gabriel J. Brostow. Digging into Self-Supervised Monocular Depth Prediction. *The IEEE International Conference on Computer Vision (ICCV)*, 2019.

[3] K. Bimbraw. Autonomous cars: Past, present and future a review of the developments in the last century, the present scenario and the expected future of autonomous vehicle technology. *2015 12th International Conference on Informatics in Control, Automation and Robotics (ICINCO)*, 01:191–198, July 2015.

[4] Nikolaos Zioulis, Antonis Karakottas, Dimitrios Zarpalas, and Petros Daras. OmniDepth: Dense Depth Estimation for Indoors Spherical Panoramas. *Computer Vision – ECCV 2018*.

[5] Fu-En Wang, Hou-Ning Hu, Hsien-Tzu Cheng, Juan-Ting Lin, Shang-Ta Yang, Meng-Li Shih, Hung-Kuo Chu, and Min Sun. Self-supervised Learning of Depth and Camera Motion from 360° Videos. *Computer Vision – ACCV 2018*.

[6] Lei Jin, Yanyu Xu, Jia Zheng, Junfei Zhang, Rui Tang, Shugong Xu, Jingyi Yu, and Shenghua Gao. Geometric Structure Based and Regularized Depth Estimation From 360° Indoor Imagery. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[7] Angel Chang, Angela Dai, Thomas Funkhouser, Maciej Halber, Matthias Niessner, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang. Matterport3D: Learning from RGB-D Data in Indoor Environments. *International Conference on 3D Vision (3DV)*, 2017.

[8] Ting-Chun Wang, Ming-Yu Liu, Jun-Yan Zhu, Andrew Tao, Jan Kautz, and Bryan Catanzaro. High-Resolution Image Synthesis and Semantic Manipulation with Conditional GANs. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018.

[9] Christian Ledig, Lucas Theis, Ferenc Huszar, Jose Caballero, Andrew P. Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, and Wenzhe Shi. Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network. *CoRR*, abs/1609.04802, 2016.

[10] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative Adversarial Nets. *Advances in Neural Information Processing Systems 27*, pages 2672–2680, 2014.

[11] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein Generative Adversarial Networks. *Proceedings of the 34th International Conference on Machine Learning*, 70:214–223, 06–11 Aug 2017.

[12] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved Training of Wasserstein GANs. *Advances in Neural Information Processing Systems 30*, pages 5767–5777, 2017.

[13] Z. Zhang. A flexible new technique for camera calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(11):1330–1334, 2000.

[14] Janne Heikkilä and Olli Silvén. A Four Step Camera Calibration Procedure with Implicit Image Correction. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 22:1106–, 01 1997.

[15] J.-Y. Bouguet. Camera Calibration Toolbox For Matlab. 2001.

[16] Gary Bradski and Adrian Kaehler. *Learning OpenCV - computer vision with the OpenCV library: software that sees.* 01 2008.

[17] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[18] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, may 2017.

[19] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[20] K Simonyan and A Zisserman. Very deep convolutional networks for large-scale image recognition. pages 1–14. Computational and Biological Learning Society, 2015.

[21] Chien-chuan Lin and Ming-shi Wang. A vision based top-view transformation model for a vehicle parking assistant. *Sensors 2012*, pages 4431–4446.

[22] B. Zhang, V. Appia, I. Pekkucuksen, Y. Liu, A. U. Batur, P. Shastry, S. Liu, S. Sivasankaran, and K. Chitnis. A Surround View Camera Solution for Embedded Systems. *2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 676–681, June 2014.

[23] Simon Hecker, Dengxin Dai, and Luc Van Gool. End-to-End Learning of Driving Models with Surround-View Cameras and Route Planners. *Computer Vision – ECCV 2018*, pages 449–468, 2018.

[24] Carlos Guindel, David Martín, and José María Armingol. Traffic scene awareness for intelligent vehicles using ConvNets and stereo vision. *Robotics and Autonomous Systems*, 112:109–122, 2019.

[25] Xinge Zhu, Zhichao Yin, Jianping Shi, Hongsheng Li, and Dahua Lin. Generative Adversarial Frontal View to Bird View Synthesis. *CoRR*, abs/1808.00327, 2018.

[26] Simon Baker, Ankur Datta, and Takeo Kanade. Parameterizing homographies, 2006.

[27] Dequan Wang, Coline Devin, Qi-Zhi Cai, Philipp Krähenbühl, and Trevor Darrell. Monocular Plan View Networks for Autonomous Driving. *CoRR*, abs/1905.06937, 2019.

[28] A. Palazzi, G. Borghi, D. Abati, S. Calderara, and R. Cucchiara. Learning to map vehicles into bird's eye view. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 10484:233–243, 2017.

[29] Ashutosh Saxena, Min Sun, and Andrew Y. Ng. Make3D: Learning 3D Scene Structure from a Single Still Image. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31, 2009.

[30] David Eigen, Christian Puhrsch, and Rob Fergus. Depth Map Prediction from a Single Image Using a Multi-Scale Deep Network. *Advances in Neural Information Processing Systems*, 2014.

[31] Fayao Liu, Chunhua Shen, and Guosheng Lin. Deep Convolutional Neural Fields for Depth Estimation From a Single Image. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[32] John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. *Proceedings of the Eighteenth International Conference on Machine Learning*, 2001.

[33] David Eigen and Rob Fergus. Predicting Depth, Surface Normals and Semantic Labels with a Common Multi-scale Convolutional Architecture. *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015.

[34] Iro Laina, Christian Rupprecht, Vasileios Belagiannis, Federico Tombari, and Nassir Navab. Deeper Depth Prediction with Fully Convolutional Residual Networks. *International Conference on 3D Vision (3DV)*, 2016.

[35] Jose M. Facil, Benjamin Ummenhofer, Huizhong Zhou, Luis Montesano, Thomas Brox, and Javier Civera. CAM-Convs: Camera-Aware Multi-Scale Convolutions for Single-View Depth. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[36] Ibraheem Alhashim and Peter Wonka. High Quality Monocular Depth Estimation via Transfer Learning. *arXiv e-prints*, abs/1812.11941, 2018.

[37] Fangchang Ma and Sertac Karaman. Sparse-to-Dense: Depth Prediction from Sparse Depth Samples and a Single Image. *The IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

[38] Zhao Chen, Vijay Badrinarayanan, Gilad Drozdov, and Andrew Rabinovich. Estimating Depth from RGB and Sparse Sensing. *Computer Vision – ECCV 2018*, 2018.

[39] Shreyas S. Shivakumar, Ty Nguyen, Ian D. Miller, Steven W. Chen, Vijay Kumar, and Camillo J. Taylor. DFuseNet: Deep Fusion of RGB and Sparse Depth Information for Image Guided Dense Depth Completion. *The IEEE International Conference on Intelligent Transportation Systems (ITSC)*, 2019.

[40] Clément Godard, Oisin Mac Aodha, and Gabriel J. Brostow. Unsupervised Monocular Depth Estimation with Left-Right Consistency. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[41] Vitor Guizilini, Rareș Ambruș, Sudeep Pillai, Allan Raventos, and Adrien Gaidon. 3D Packing for Self-Supervised Monocular Depth Estimation. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[42] Hsien-Tzu Cheng, Chun-Hung Chao, Jin-Dong Dong, Hao-Kai Wen, Tyng-Luh Liu, and Min Sun. Cube Padding for Weakly-Supervised Saliency Prediction in 360° Videos. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[43] Miriam Schönbein, Tobias Strauss, and Andreas Geiger. Calibrating and Centering Quasi-Central Catadioptric Cameras. *The IEEE International Conference on Robotics and Automation (ICRA)*, 2014.

[44] Miriam Schönbein and Andreas Geiger. Omnidirectional 3D Reconstruction in Augmented Manhattan Worlds. *The IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2014.

[45] Jingwei Huang, Zhili Chen, Duygu Ceylan, and Hailin Jin. 6-DOF VR videos with a single 360-camera. *The IEEE International Conference on Virtual Reality and 3D Interfaces (VR)*, 2017.

[46] Varun Ravi Kumar, Stefan Milz, Christian Witt, Martin Simon, Karl Amende, Johannes Petzold, Senthil Yogamani, and Timo Pech. Monocular Fisheye Camera Depth Estimation Using Sparse LiDAR Supervision. *The IEEE International Conference on Intelligent Transportation Systems (ITSC)*, 2018.

[47] Nikolaos Zioulis, Antonis Karakottas, Dimitris Zarpalas, Federic Alvarez, and Petros Daras. Spherical View Synthesis for Self-Supervised 360° Depth Estimation. *International Conference on 3D Vision (3DV)*, September 2019.

[48] Ning-Hsu Wang, Bolivar Solarte, Yi-Hsuan Tsai, Wei-Chen Chiu, and Min Sun. 360SD-Net: 360° Stereo Depth Estimation with Learnable Cost Volume . *The IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

[49] Fu-En Wang, Yu-Hsuan Yeh, Min Sun, Wei-Chen Chiu, and Yi-Hsuan Tsai. BiFuse: Monocular 360 Depth Estimation via Bi-Projection Fusion. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[50] Yu-Chuan Su and Kristen Grauman. Learning Spherical Convolution for Fast Features from 360° Imagery. *Advances in Neural Information Processing Systems*, 2017.

[51] Benjamin Coors, Alexandru Paul Condurache, and Andreas Geiger. SphereNet: Learning Spherical Representations for Detection and Classification in Omnidirectional Images. *Computer Vision – ECCV 2018*, 2018.

[52] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite. *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[53] Holger Caesar, Varun Bankiti, Alex H. Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuScenes: A multimodal dataset for autonomous driving. *arXiv preprint arXiv:1903.11027*, 2019.

[54] Yohann Cabon, Naila Murray, and Martin Humenberger. Virtual KITTI 2, 2020.

[55] Adrien Gaidon, Qiao Wang, Yohann Cabon, and Eleonora Vig. Virtual worlds as proxy for multi-object tracking analysis. *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 4340–4349, 2016.

[56] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated Residual Transformations for Deep Neural Networks. *arXiv preprint arXiv:1611.05431*, 2016.

[57] Sergey Zagoruyko and Nikos Komodakis. Wide Residual Networks. *Proceedings of the British Machine Vision Conference (BMVC)*, pages 87.1–87.12, September 2016.

[58] Gunnar Farnebäck. Two-frame motion estimation based on polynomial expansion. *Image Analysis*, 2003.

[59] Ravi Garg, BG Vijay Kumar, Gustavo Carneiro, and Ian Reid. Unsupervised CNN for Single View Depth Estimation: Geometry to the Rescue. *Computer Vision – ECCV 2016*, 2016.

[60] Tinghui Zhou, Matthew Brown, Noah Snavely, and David G. Lowe. Unsupervised Learning of Depth and Ego-Motion from Video. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

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

[62] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *CoRR*, abs/1412.6980, 2015.

[63] Jun Xie, Martin Kiefel, Ming-Ting Sun, and Andreas Geiger. Semantic Instance Annotation of Street Scenes by 3D to 2D Label Transfer. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[64] Burr Settles. Active learning literature survey. 2009.