technische universität dortmund

Fraunhofer IML

Master Thesis

A Monocular Vision Architecture for Free Volume Estimation in Shipping Containers

Ashwin Nedungadi

Matrikelnummer: 230397

Major: M.Sc. Automation & Robotics

Specialization: Cognitive Systems

Issued On:

01.07.2024

Submitted On:

11.02.2025

Supervisors:

Prof. Dr.-Ing. Alice Kirchheim

Christopher Rest, M.Sc.

Technische Universität Dortmund

Fakultät Maschinenbau

Lehrstuhl für Förder- und Lagerwesen

http://flw.mb.tu-dortmund.de/

“Dedicatum amori fabricandi machinas intelligentes.”

Abstract

Accurately estimating point clouds from a single image using monocular depth estimation remains a critical challenge in computer vision, with downstream applications in 3D reconstruction and areas such as robotics, autonomous navigation, and augmented reality. This thesis introduces an architecture to estimate the free volume within shipping containers using monocular images captured by mobile devices. The aim is to enhance cargo optimization and warehouse management in logistics through a cost-effective and scalable solution for 3D reconstruction and volume estimation. Building on recent advancements in deep learning, particularly monocular depth estimation, we evaluate monocular depth estimation models on domain-specific data to address key logistical challenges, such as inefficiencies in manual measurements and the high costs of specialized instruments. Traditional methods for optimizing cargo loading rely on manual measurements or specialized instruments, which are often inefficient or costly. In contrast, the proposed approach leverages computer vision techniques to overcome these challenges by reconstructing watertight meshes of container interiors using a single image, which enables the precise measurement of free volume for optimized loading configurations.

The architecture addresses two key sub-problems: converting images into point clouds and transforming point clouds into watertight meshes. The thesis is divided into three stages: data collection and benchmarking of monocular depth estimation models, 3D reconstruction and volume estimation, and fine-tuning with evaluation of the results. We demonstrate the feasibility of applying monocular depth estimation to real-world logistics tasks while providing a framework and highlighting its advantages over traditional methods. By improving monocular depth estimation in our domain and assessing its performance in practical applications in logistics, this research contributes to both computer vision and supply chain efficiency. Ultimately, it aims to revolutionize operational workflows by enabling precise volume estimations for sustainable and economically efficient cargo loading.

Acknowledgements

I would like to express my sincere gratitude to my advisor, Christopher Rest, for his invaluable guidance and constructive feedback throughout the process of writing this thesis. His support has been instrumental in shaping this work's final outcome.

I am also sincerely thankful to my colleagues at Fraunhofer IML for fostering engaging discussions on robotics, which greatly enriched my understanding of the subject. I am particularly grateful for the freedom I was given during my time there, which allowed me to explore, experiment, and innovate over the course of the year. I am deeply thankful to my friends in Dortmund for their unwavering support, encouragement, and kindness, which have been a source of strength and comfort throughout this journey.

I want to thank Laura for her constant support and encouragement throughout this journey. Her belief in me has motivated me during challenging times. Her creative insights have broadened my understanding of art and inspired me to see robotics as a fascinating intersection of engineering, design, and artistic expression. Finally, I want to express my deepest gratitude to my parents: to my dad, whose curiosity I inherited, and to my mom, whose unwavering work ethic continues to inspire me. Their influence has significantly shaped who I am today.

Contents

1 Introduction

1.1 Overview

Humans have, for so long, tried to replicate and represent what we see in nature. From early primal cave paintings to the masterful depth showcased in Caravaggio's Renaissance masterpieces, we have always sought to capture the essence of our three-dimensional world in our two-dimensional creations[1]. The past two decades have witnessed a remarkable shift in this pursuit, with the rapid advancement of computing technology enabling us to not only capture the world in 2D images but also directly reconstruct 3D information using active sensors like LiDAR and structured light cameras or through passive methods like stereopsis, structure from motion (SfM), and shape from shading (SfS) [1–3]. This ability to reconstruct and represent 3D objects and environments from 2D data has become a transformative technology with far-reaching applications in diverse fields, including computer vision [4], robotics [5–7], augmented and mixed reality [8,9], medicine [10,11], and archaeology [12,13]. By creating accurate digital representations of the natural world, 3D reconstruction empowers us to spatially augment and utilize information in novel ways, unlocking possibilities previously inconceivable and enabling advancements across fields.

This work delves into the challenge of two inherently ill-posed problems: metric 3D reconstruction using monocular depth estimation and surface reconstruction to obtain a watertight mesh whose volume can be computed to estimate a container's space utilization by calculating the free volume. Monocular depth estimation has only recently matured due to incorporating more evolved multi-scale deep learning models in computer vision tasks. It aims to accurately recover metric depth information from a single image using deep learning models trained with extensive data. In the context of our specific use case of estimating the free space of loaded containers at logistic warehouses, we fine-tune existing foundation models on RGB and Depth data captured by a high-precision NavVis 3D scanner as well as data collected using a LiDAR sensor from an iOS device. The data is processed for consistency and used to fine-tune selected monocular depth estimation models following recent works [14, 15]. However, reconstructing 3D information solely from a single image presents significant challenges compared to stereo vision or LiDAR, where depth information can be obtained directly or indirectly. The inherent ambiguity of estimating depth from a single viewpoint makes monocular depth estimation an ill-posed problem, as a 2D image may have multiple valid interpretations when projected into 3D space. Similarly, estimating a continuous surface from a set of unordered points introduces challenges in defining surface manifolds, as multiple interpolations can exist. Additionally, these models can suffer from accuracy issues when presented with distorted, blurry, occluded, or skewed images,further complicating their real-world deployment. Despite these challenges,

[1] See appendix A

monocular depth estimation offers significant advantages due to its cost-effectiveness, scalability, and efficiency, particularly when using readily available RGB cameras on edge devices.

This research presents a novel perception-based method to estimate the unutilized space in loaded cargo containers through monocular depth estimation and 3D reconstruction. The goal is to estimate the free volume in shipping containers by utilizing cutting-edge deep learning architectures and integrating domain-specific knowledge. In the specific use case of estimating the free space of loaded containers in logistics warehouses, we present data generation and processing pipelines for fine-tuning foundational monocular depth estimation models. The training data is processed to ensure quality and consists of RGB and depth information captured by a high-precision NavVis 3D scanner, a LiDAR-equipped iOS device, and a handheld scanner developed at Fraunhofer IML. We evaluate various monocular depth estimation models, comparing their accuracy and feasibility for our use case before selecting and fine-tuning the most suitable model with our dataset. Through this process, we develop and present a monocular vision architecture for 3D reconstruction, tailored for space utilization in shipping containers but adaptable for other applications such as robot navigation. Our work demonstrates the potential of monocular depth estimation for smart optimization within logistics operations, showcasing its real-world applicability.

1.2 Motivation

The efficient loading of cargo trucks and shipping containers is a fundamental aspect of logistics with far-reaching consequences for the entire supply chain [16,17]. It directly affects operational efficiency, cost reduction, and resource utilization, making it a key area of research [18–20]. Depth estimation from a single image offers a fast, cost-effective, and scalable approach to 3D reconstruction and volume estimation, essential for optimizing warehouse cargo loading and cross-docking operations[2]. By developing new system architectures with these novel techniques and re-framing the problem as a computer vision-based perception challenge for robotics, we aim to leverage RGB images for 3D reconstruction using edge devices, such as mobile or mixed-reality wearables. These techniques can generate accurate 3D cargo models within containers. This enables precise volume measurements, which can be integrated into cargo loading algorithms to optimize space utilization and improve logistics efficiency [16]. Real-time volume estimation eliminates the need for manual measurements, allows dynamic adjustments based on cargo size, enables cargo sharing, and enhances supply chain automation. Moreover, the 3D data obtained from reconstruction can support digital twin development for warehouses and enhance robotic perception in downstream tasks, such as navigation, object detection, and automated warehouse management. These innovations contribute to intelligent logistics solutions within Industry 4.0, enabling smarter data-driven decision-making in supply chain operations.

[2]Cross-docking is the process of unloading goods from incoming vehicles at a logistics facility and promptly transferring them to trucks for outgoing shipments, requiring minimal or no storage time in between.

1.3 Related Work

The task of 3D reconstruction is of paramount importance in robotics for a wide range of tasks such as localization [21], navigation [6, 22], mapping [23, 24], as well as in autonomous interaction and manipulation of environments [25, 26, 5]. Historically, robotic systems have relied on active sensing modalities such as LiDAR [27], structured light [28], Time-of-Flight (ToF) sensors, and stereo vision to infer 3D spatial geometry [29, 30]. While these methods provide high precision, they come with substantial hardware costs and exhibit robustness limitations under challenging conditions such as reflective surfaces, dynamic lighting, and fog [31, 32]. Recent advancements in computing hardware and deep learning have spurred the development of vision-based approaches to 3D perception that augment or replace traditional sensors, positioning monocular and multi-view perception as key frontiers in autonomous system design [7].

3D reconstruction techniques in computer vision can generally be categorized into two main paradigms: single-view [33] and multiview [34] approaches. Multi-view methods, in particular, have been extensively studied and encompass traditional techniques, such as structure-from-motion (SfM) [2], Multi-View Stereo (MVS) [35], and shape-from-shading (SfS) [3]. These methods rely on the geometric correspondences between overlapping images to infer 3D structures while mitigating scale ambiguity. Although advancements in deep learning have led to notable improvements in multi-view reconstruction [36–38] many of the challenges inherent to traditional approaches persist.

While multi-view methods have demonstrated significant potential in producing detailed 3D reconstructions by leveraging geometric correspondences between overlapping images, they face notable challenges. High computational demands, sensitivity to camera calibration errors, and dependency on imaging from multiple viewpoints limit their applicability in real-time, resource-constrained, or dynamic environments such as on edge devices or robotics applications [39–41]. In contrast, single-view 3D reconstruction bypasses multi-view constraints by exploiting learned priors from large-scale data to infer depth and 3D geometry directly from monocular inputs [42, 43]. This makes monocular depth estimation particularly advantageous in scenarios where multiple views or specialized sensors are impractical or unavailable [23, 44]. However, while eliminating multi-view dependencies, single-view approaches inherently struggle with scale ambiguity, occlusion handling, and domain shift sensitivities, which are exacerbated by dataset-specific biases in zero-shot scenarios [45, 46].

Early single-view depth estimation methods relied on geometric cues such as vanishing points, perspective views, shading, and focused on handcrafted features [1, 47–49] which failed to adapt and scale to be used in complex, dynamic environments. The advent of deep learning has revolutionized the field, significantly narrowing the performance gap between single-view

and multi-view approaches. Transformer architectures with attention [50, 51] were introduced to capture long-range dependencies and enhance global feature aggregation. Implicit neural representations, including Neural Radiance Fields [52, 53] and Signed Distance Functions (SDFs) [54], improve continuous scene representations [55]. Despite these advancements, challenges remain in terms of real-time inference, fine detail recovery, and cross-domain adaptability. Hybrid methodologies fusing data-driven priors with geometric constraints have also been actively explored to mitigate these limitations [56, 57].

An important distinction in single-view depth estimation is between absolute depth estimation and relative depth estimation. Absolute depth-estimation approaches aim to recover metric distances between the camera and pixels in the scene, typically framed as a regression [42] or classification [58] problem. In regression-based approaches, convolutional neural networks or vision transformers directly predict depth values at the pixel level, often supervised by ground truth LiDAR or RGB-D data [42]. Classification-based methods discretize the depth range into predefined 'bins', treating depth prediction as a multi-class problem to mitigate the sensitivity of regression-based approaches to unbounded output scales.

Relative depth estimation, in contrast, focuses on predicting depth relationships between pixels by determining whether one pixel is closer or farther than its neighboring pixels without recovering exact distances to the camera [59]. Early approaches leveraged fully convolutional networks [42, 60, 61], which used relative depth regression on large-scale, in-the-wild datasets to improve generalization. More recent models [62–64] integrate vision transformers (ViTs) [50] to improve depth ordering accuracy. The advent of generative diffusion models, such as [59], has advanced relative depth estimation by leveraging iterative refinement strategies on large-scale synthetic and internet data. Hybrid approaches [65], including scale-aware architectures [66] and self-supervised frameworks, aim to bridge the gap between relative and absolute depth estimation by incorporating geometric consistency and domain adaptation. This emerging paradigm, exemplified by models like ZoeDepth [14] and Depth Anything [43, 15], enhances cross-domain adaptability by pre-training on large-scale synthetic relative depth data and fine-tuning for metric depth prediction.

Architectural advancements have played a crucial role in improving single-view depth estimation, whereas generalization to unseen domains (zero-shot metric depth estimation) has emerged as a key challenge. Early methods such as CNN (Convolutional Neural Network) based models [42] and probabilistic approaches using MRFs/CRFs (Markov Random Fields/-Conditional Random Fields) [48, 67] evolved into encoder-decoder frameworks [58, 14] that fuse multi-scale features for a balance between local and global depth cues. While hierarchical designs [68, 69] further improved robustness via multi-scale feature fusion [70], the introduction of ViT based depth decoders [71, 64] revolutionized the field by capturing long-range depth

dependencies through self-attention. Simultaneously, generalization strategies have become increasingly central to depth estimation, with models such as [61, 43] pioneering large-scale relative depth training and incorporating scale-invariant objectives to improve the zero-shot performance [72].
More recent approaches leverage self-supervision [45] and unlabeled data at scale [43], whereas [73, 74] normalize inputs to a canonical space using focal length to enhance metric consistency. Contemporary models [75, 15] are combined with metric fine-tuning or specialized camera-aware modules for scale recovery. Additionally, recent work has addressed the challenge of unknown camera intrinsics with solutions such as dedicated intrinsic-aware networks [56] and spherical embeddings [76], highlighting the critical role of intrinsic reasoning in achieving accurate depth estimation in unconstrained environments. Collectively, these advancements underscore the interplay between architectural innovations and scalable training, demonstrating that explicit geometric reasoning about camera properties is pivotal for achieving accurate, generalizable, and metric-aware depth prediction in next-generation models.

While multi-view and single-view depth estimation methods provide a good understanding of 3D geometry from 2D images, translating these depth inferences into coherent, high-fidelity 3D surfaces requires complementary techniques in surface reconstruction. Surface reconstruction aims to interpolate surfaces between point cloud outputs, converting sparse or noisy point data into smooth, detailed meshes. Traditional methods in computer graphics, such as Poisson surface reconstruction [77], utilize mathematical techniques to interpolate implicit surfaces from oriented points, offering both efficiency and reliability. In contrast, modern research has unveiled learning-based methods for surface reconstruction. One notable method is Point2Mesh [78], which employs self-priors to deform an initial mesh, effectively shrink-wrapping it around a single input point cloud through iterative, self-supervised optimization. Neural Kernel Surface Reconstruction (NKSR) [79] also leverages learned neural kernels to adaptively interpolate surfaces from unstructured point clouds. Unlike traditional methods that use fixed kernels, NKSR optimizes kernel weights via gradient descent to fit the input data, allowing it to infer spatially varying priors that better capture local geometric patterns.
However, while these learning-based methods automate robust feature extraction and reduce manual parameter tuning, they often require significant computational resources during inference and struggle to generalize to out-of-domain data. In addition, the lack of large-scale 3D datasets has further limited the use of such models for surface reconstruction. This dichotomy underscores the enduring relevance of traditional approaches like Poisson reconstruction, which offers a more robust surface reconstruction solution that outperforms learning-based approaches in practical applications requiring broad generalization [80].

1.4 Objectives

The primary objective of this study is to evaluate the feasibility of monocular depth estimation for container space utilization using 3D reconstruction. The secondary objective is to collect, filter, and process a multimodal RGB-D dataset to fine-tune a suitable monocular depth estimation model on domain-specific data. Finally, we aim to integrate the fine-tuned model into a pipeline for volume estimation and assess its performance and limitations. To achieve these objectives, we structure this work into the following key components:

- Data Collection and Processing: RGB-D data were gathered at DB Schenker using a high-precision NavVis 3D scanner, an iOS device with LiDAR, and a handheld scanner, followed by filtering and pre-processing of the data for fine-tuning.
- Benchmarking monocular depth estimation models: Evaluate various monocular depth estimation models to determine the most suitable candidate for fine-tuning in our specific logistics use case.
- Surface Reconstruction & Volume Estimation Pipeline: Prototype suitable methods to reconstruct a watertight 3D mesh of a shipping container's interior and estimate its free volume using monocular depth estimation.
- Evaluation & Performance Analysis: Assess the fine-tuned model's accuracy and limitations through quantitative benchmarking, including depth prediction metrics, point-cloud metrics, and final volume estimation accuracy.

1.5 Thesis Structure

The remainder of this thesis is organized as follows:

1. **Chapter 2** Provides the theoretical background, covering the various camera models used in this work and the transformations between them. It also introduces the underlying architecture for monocular depth estimation and explains its working principles. In addition, this chapter presents an overview of the surface reconstruction techniques and methods for processing sensor depth data.

2. **Chapter 3** Details the data collection process, including the hardware used for data acquisition and their calibration procedures followed by raw data extraction. It then discusses the data-processing pipelines used to refine and structure the collected data into a usable format. The resulting dataset, along with the publicly available datasets used for training, is described in detail. The chapter concludes with a discussion of the data augmentation strategies employed to enhance the robustness of the fine-tuning process.

3. **Chapter 4** Contains the main contribution of the thesis, and begins with zero-shot benchmarking of various monocular depth-estimation models on a domain-specific dataset. It then provides an in-depth analysis of the selected models, followed by a discussion on the fine-tuning strategy used to adapt them to the logistics use case. The latter sections of this chapter focus on the remaining components of the 3D reconstruction pipeline, including surface reconstruction and volume estimation.

4. **Chapter 5** Presents the evaluation of each component in the pipeline using appropriate metrics, and compares the performance before and after fine-tuning with domain-specific data. This chapter also discusses the results and their interpretation.

5. **Chapter 6** Concludes the thesis by summarizing key findings, discussing the strengths and limitations of the proposed pipeline, and outlining potential directions for future research.

2 Theoretical Background

This chapter establishes the foundational concepts and principles underlying the methodologies employed in this thesis. It begins with exploring camera models, including the Pinhole, Fisheye, and Omnidirectional camera models, due to their relevance to the devices utilized in data collection. We discuss the role of focal length in scale-aware 3D reconstruction alongside methods for estimating depth from single-view images. The chapter also provides an overview of the model architectures used, emphasizing the challenges and requirements for training and fine-tuning models in this domain. Finally, the chapter addresses the surface reconstruction algorithms used in this study and metrics for comparing predicted depth maps and point clouds to their ground truths.

2.1 Camera Parameters and Calibration

All cameras can be thought of as a mapping between the 3D world and the 2D image plane [81]. Camera calibration plays an important role in computer vision by estimating the camera's intrinsic parameters such as focal length, camera center, and distortions, and the camera's extrinsic parameters, which describe the camera's position and orientation relative to the 3D world coordinate system. Understanding the underlying camera models and camera parameters that enable the transformation between raw image data and 3D world points is crucial for addressing single-image 3D reconstruction (Figure 2.1).

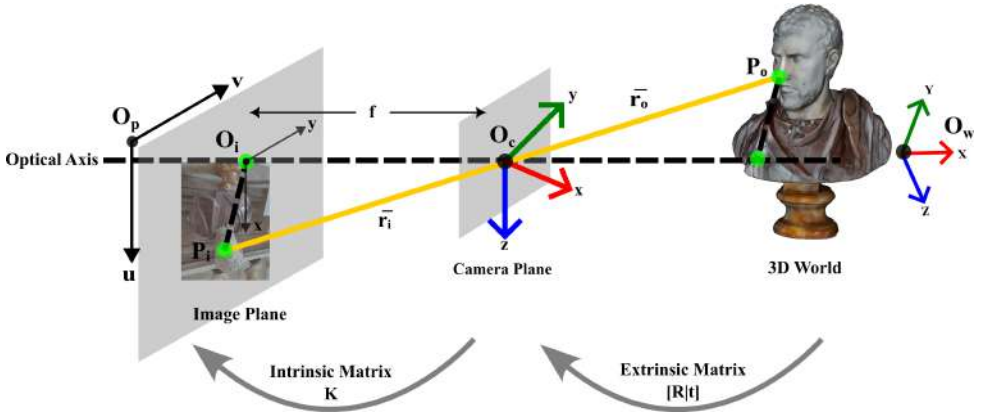


Figure 2.1: An illustration of the pinhole camera model and its four coordinate systems: world, camera, image, and pixel. The intrinsic and extrinsic parameters enable transformations between these coordinate frames, facilitating 3D-to-2D projection.

2.1.1 Camera Projection Matrix

The camera projection matrix is a fundamental representation in computer vision that describes the mapping between the 3D world coordinates and the 2D image coordinates. It encapsulates both the intrinsic and the extrinsic properties of the camera, forming a linear transformation that

projects a point in 3D space onto the image plane [1]. The projection matrix P is defined as:

$$P = K \cdot [R|t], \tag{2.1}$$

where

- K is the intrinsic matrix containing the internal parameters of the camera (i.e. how the camera views the world), such as focal length and principal point coordinates.
- $[R|t]$ is the extrinsic matrix, representing the camera's rotation R and translation vector t relative to the world coordinate system.

Camera Intrinsic Parameters

A camera's intrinsic parameters determine its internal properties in comparison to a pinhole camera model and are essential for rectifying distortions. The intrinsic matrix allows the transformation of 3D world points onto 2D coordinates on an image plane using the pinhole camera model. The intrinsic matrix K can be expressed as:

$$K = \begin{bmatrix} F_x & 0 & C_x \\ 0 & F_y & C_y \\ 0 & 0 & 1 \end{bmatrix}, \tag{2.2}$$

where

- F_x and F_y are the focal lengths in pixels along the x-and y-axes, respectively.
- C_x and C_y are the coordinates of the principal point, typically near the center of the image sensor.

Camera Focal Length

The focal length of a camera measures the optical distance (usually measured in mm) from the point where the light meets inside the lens to the camera's sensor. Changing a camera's focal length changes the way objects appear in the image. A short focal length, such as 24 mm (Figure 2.2b), provides a wide angle of view, allowing more of the scene to enter the frame (Figure 2.2), and affecting depth perception by exaggerating distances between objects. A long focal length, like 200 mm (Figure 2.2b), in contrast, compresses the depth and results in a narrower field of view, focusing on more distant objects and effectively zooming in on them [1]. This physical property of camera systems makes 3D reconstruction without known camera intrinsics ill-posed.

Camera Extrinsic Parameters

The camera extrinsics describe the camera's position and orientation relative to the world coordinate system. It tells you where in the world the camera is located in 3D space and describes

(a) Focal lengths and their effect on field of view

(b) Object size with respect to different focal lengths

Figure 2.2: The effect of focal length in distance perception [82].

how a point in 3D space is transformed onto the camera coordinates (Figure 2.1). The extrinsic matrix combines the rotation matrix R, which is a 3x3 matrix that defines the camera's orientation, and the translation vector T, which is a 3x1 vector representing the position of the world coordinate system's origin in camera coordinates [1]. The extrinsic matrix is usually represented together as a 3x4 matrix as:

$$E = [RT]. \tag{2.3}$$

The extrinsic matrix can also be derived from the camera pose by inverting the pose matrix (Equation 2.3) where the camera position C is the location of the camera center in world coordinates and R_c is the rotation matrix describing the camera's orientation relative to the world.

$$E = [R_c|C]^{-1}. \tag{2.4}$$

2.1.2 Camera Calibration

Camera calibration is a fundamental process in computer vision to estimate the intrinsic and extrinsic parameters of a camera, enabling the accurate mapping of 3D world points to 2D image coordinates. Although most commercially available cameras adhere to the pinhole model, real-world lenses introduce distortions and optical imperfections caused by various factors, such as imperfections in lens manufacturing or physical limitations of the lens design [83]. These distortions (Figure 2.3) are modeled by the Brown-Conrady [84] distortion model and can be categorized into radial and tangential distortions.

1. Radial distortion causes straight lines to appear curved, particularly near the edges of the image. The distortion increases with distance from the center of the image.

$$\delta_r = k_1 r^3 + k_2 r^5 + k_3 r^7 + \ldots + k_n r^{n+2} \tag{2.5}$$

Where, r is the distance from the center of the image, and $k1$, $k2$, $k3$, etc. are the radial distortion coefficients.

2. Tangential distortion arises from misalignment between the lens and the image sensor, which causes the image to be distorted. This can be mathematically expressed as:

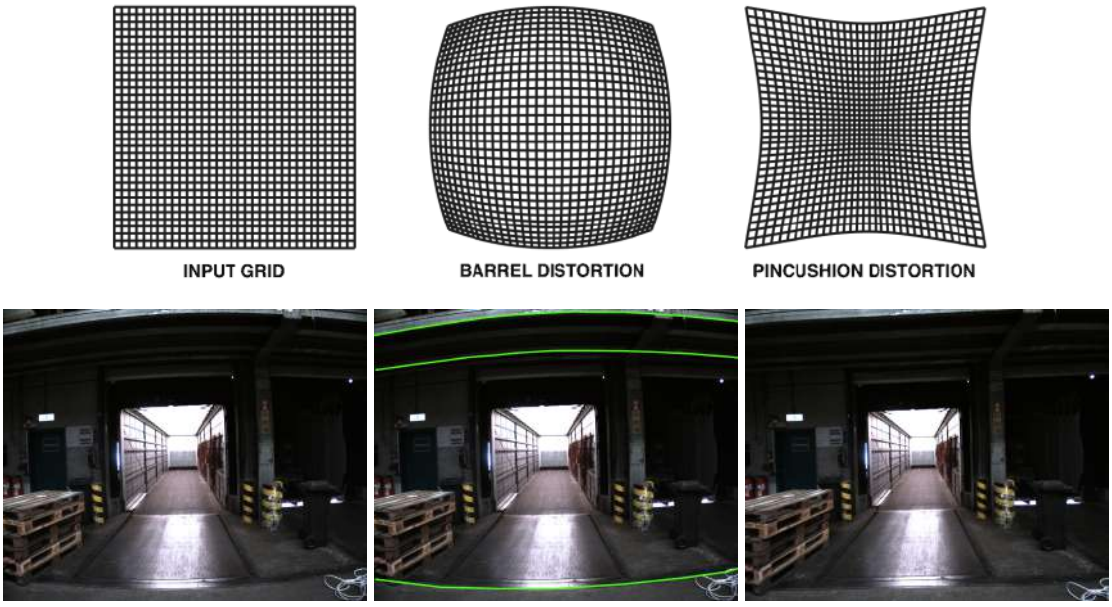
$$\delta x = 2p_1xy + p_2\left(r^2 + 2x^2\right) \tag{2.6}$$

$$\delta y = p_1\left(r^2 + 2y^2\right) + 2p_2xy \tag{2.7}$$

where x and y are the coordinates of the image, and p_1 and p_2 are the coefficients of tangential distortion. After merging the mathematical models for both types of distortion, they can be represented using 5 coefficients denoted as:

$$\text{Distortion Coefficients} = (k1, k2, p1, p2, k3). \tag{2.8}$$

The parameters $k1, k2, k3, p1,$ and $p2$ are the distortion coefficients that need to be estimated during the calibration process.



(a) Original image from camera (b) Distortion visualized (c) After correction

Figure 2.3: Types of radial distortion and visualization of distortion correction (adapted from [85]).

Camera calibration involves taking multiple images using a checkerboard (or other geometric) pattern with known dimensions to estimate the distortion coefficients and camera parameters. We can then undistort the original image and obtain a rectified image with straight lines (Figure 2.4). There are many known algorithms which use different approaches to calculate the in-

trinsic and extrinsic parameters; among these, the most prominent include: Zhang's method [86], Direct Linear Transform (DLT) [81], and Tsai's method [87].

(a) Detected corner points during calibration (b) Calibration image after undistortion

Figure 2.4: The calibration process involving a checkerboard pattern.

2.2 Camera Models

Camera models are mathematical abstractions that describe how cameras capture three-dimensional (3D) points in the world onto a two-dimensional (2D) image plane. Models such as the pinhole, fisheye, and omnidirectional models provide a framework for understanding and calibrating the transformation between 3D points of objects and their image representations [88]. The pinhole camera model assumes an idealized lens without distortions that projects 3D world points onto a 2D image plane linearly, making it well-suited for applications like structure-from-motion and stereo vision. In contrast, fisheye models capture hemispherical fields of view by introducing nonlinear distortions due to their wide-angle optics, which are advantageous for tasks such as surveillance, autonomous navigation, and mapping. Omnidirectional models extend this concept to a full 360-degree field of view, employing techniques like shaped mirrors or multiple cameras, making them ideal for multi-robot coordination and large-scale environment mapping [89, 90].

Although all these models project 3D points onto a 2D image plane, their formulations differ in how they handle distortions and wide-angle perspectives, aligning with the specific requirements of various robotic and computer vision applications. By incorporating intrinsic parameters such as focal length, lens distortion, and sensor alignment, along with extrinsic parameters such as camera position and orientation, these models enable precise scene understanding and 3D reconstruction, forming the foundation for effective robotic perception.

2.2.1 Pinhole Camera Model

The pinhole camera, or perspective projection, represents the simplest form of camera projection by assuming a single aperture through which light rays converge to form an image (Figure 2.5).

This model assumes an ideal camera without lens distortions, where light rays pass through a single point (the pinhole) and form an inverted projection on the image plane.
The relationship between 3D world coordinates and 2D image coordinates is described linearly, encapsulating intrinsic parameters such as focal length and principal point. Despite its simplicity, the pinhole model is widely used in commercial cameras along with a distortion model to compensate for optical distortion [91].
In the pinhole model, the image plane is situated in front of the camera, and 3D points (X, Y, Z)

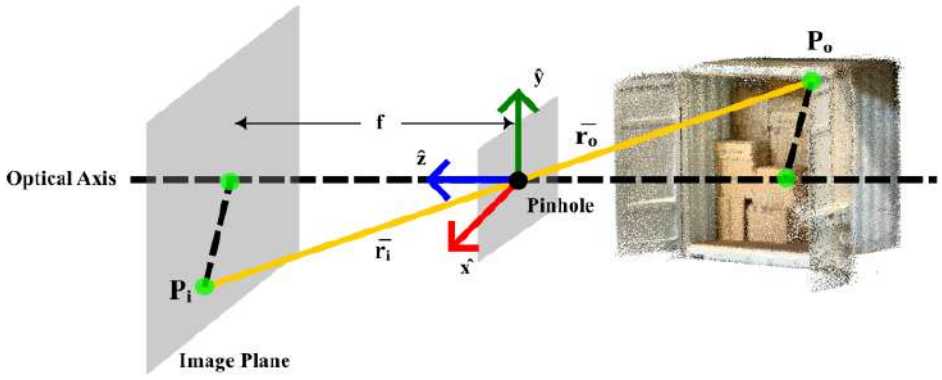


Figure 2.5: Visualization of the pinhole camera model, illustrating the projection of a 3D point in the world onto a 2D image plane. The optical axis, focal length, and coordinate transformations define the mapping between the scene and the captured image.

are projected onto the 2D image plane as (u, v) (Figure 2.5). The relationship is linear and follows the projection equation:

$$S \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K[R\ t] \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}, \tag{2.9}$$

where K is the intrinsic matrix that captures the focal lengths (f_x, f_y) and the camera centers (c_x, c_y), R and t, the rotation matrix and translation vector together, defining the extrinsic parameters:

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}. \tag{2.10}$$

However, we must make two assumptions about the pinhole camera to have this linear system. We assume the following:

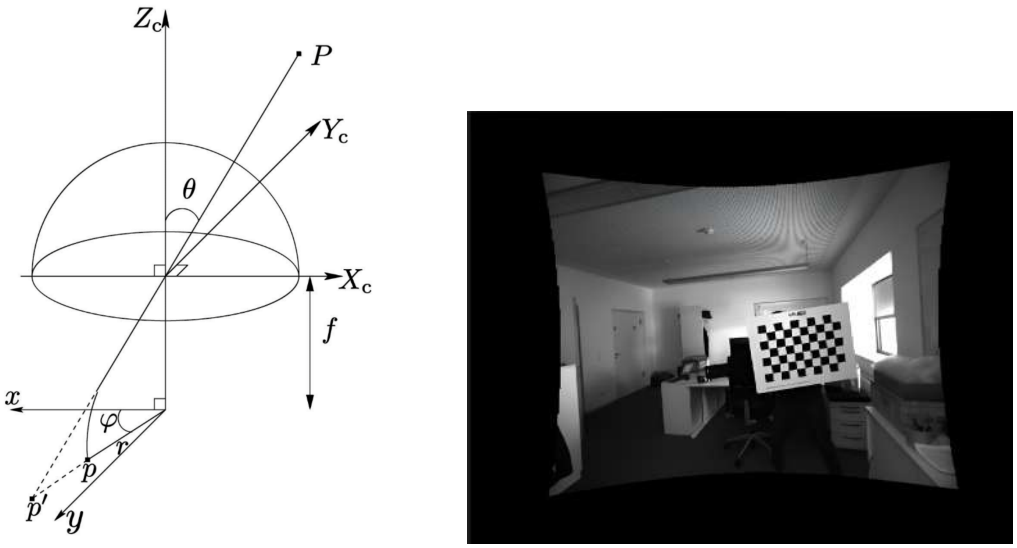
1. The lens does not introduce any distortion

2. The projection is perspective

In reality, cameras and lenses do not follow a perfect pinhole model, leading to more complex models such as fisheye and omnidirectional models. These models account for various distortions by adding terms for radial, tangential, and other distortions [88, 83].

2.2.2 Fisheye Camera Model (Kannala Brandt Model)

Fisheye cameras are an extension of pinhole cameras that use specialized wide-angle lenses to capture a panoramic or hemispherical field of view, typically ranging from 120° to 180°. Unlike pinhole cameras, which map 3D world points onto a 2D image plane linearly, fisheye cameras introduce significant nonlinear distortion due to the nature of their construction. Understanding the fisheye camera distortion model is essential for various applications in robotics such as image rectification, distortion correction (Figure 2.6b), and camera calibration [83].
The Kannala-Brandt model [92], introduced by Kannala and Brandt, is one of the most widely used fisheye camera models[3], particularly for field of view (FOVs) up to 115 °. This model is an extension of the ideal pinhole camera model, incorporating lens distortions in both the radial and the tangential directions. It generalizes the mapping of 3D world points to the 2D image plane by parameterizing the relationship between the incidence angle of light and its projection onto the image plane, allowing for flexible handling of different fisheye projection types. The



(a) Kannala-Brandt fisheye model [92] (b) Undistortion of a fisheye image

Figure 2.6: a) The Fisheye camera model and b) undistorted fisheye image.

Kannala-Brandt model (Figure 2.6a) characterizes distortion as a function of the incidence angle θ of light passing through the lens that results in a smoother distortion function easily modeled by a polynomial power series [91].
The perspective projection of a pinhole camera can be described by the following formula:

$$r = f \cdot tan\theta, \tag{2.11}$$

[3] also known as the OpenCV fisheye model

or,

$$\theta = arctan\frac{r}{f} \tag{2.12}$$

For an **equidistant projection**, θ is directly proportional to the radial distance r from the optical center and is given by:

$$\theta = \frac{r}{f}, \tag{2.13}$$

Where, r is the radial distance from the optical center and f is the focal length of the fisheye lens. In general, the radial distance r is modeled as a polynomial power series in terms of θ.

$$r(\theta) = k_1\theta + k_2\theta^3 + k_3\theta^5 + k_4\theta^7 + \ldots + k_n\theta^{2n-1} \tag{2.14}$$

Here, $r(\theta)$ is the distorted radial distance, θ is measured in radians, $k1, k2, ..., k_n$ are the distortion coefficients. This power series representation allows the model to flexibly approximate distortion for a wide range of lenses and FOVs.

Radial and Tangential Distortions

The equidistant projection model assumes that the rays of light passing through the lens and projected onto the image sensor form equal angles with the optical axis [83]. But the main advantage of the Kannala-Brandt model is that it can support different types of projection[4] by changing our formula for θ, which makes the distortion function smoother for wide-angle lenses.

Kannala-Brandt also characterizes other radial and tangential distortions by using additional parameters:

$$\delta_r = \left(l_1\theta + l_2\theta^3 + l_3\theta^5\right)\left(i_1\cos(\varphi) + i_2\sin(\varphi) + i_3\cos(2\varphi) + i_4\sin(2\varphi)\right) \tag{2.15}$$

$$\delta_t = \left(m_1\theta + m_2\theta^3 + m_3\theta^5\right)\left(j_1\cos(\varphi) + j_2\sin(\varphi) + j_3\cos(2\varphi) + j_4\sin(2\varphi)\right) \tag{2.16}$$

One distortion term acts in the radial direction (δ_r), and the other in the tangential direction (δ_t). Although the full Kannala-Brandt model requires up to 23 parameters to describe the radial and tangential distortions comprehensively, this level of complexity is often unnecessary for real-world applications. Instead, a reduced six-parameter model is commonly used. This reduced model retains the primary radial distortion coefficients (k_1, k_2, k_3) along with the basic intrinsic parameters (f, c_x, c_y), where c_x, c_y are the principal point offsets.

2.2.3 Omnidirectional Camera Model (Unified Tyler Model)

An omnidirectional camera model is a vision system that provides a 360-degree panoramic view of the scene with a field of view in the horizontal plane. These cameras can be classified into Catadioptric and Polydioptric cameras [88]. Catadioptric cameras combine a standard camera

[4] For example, equidistant, equisolid angle, stereographic, orthographic

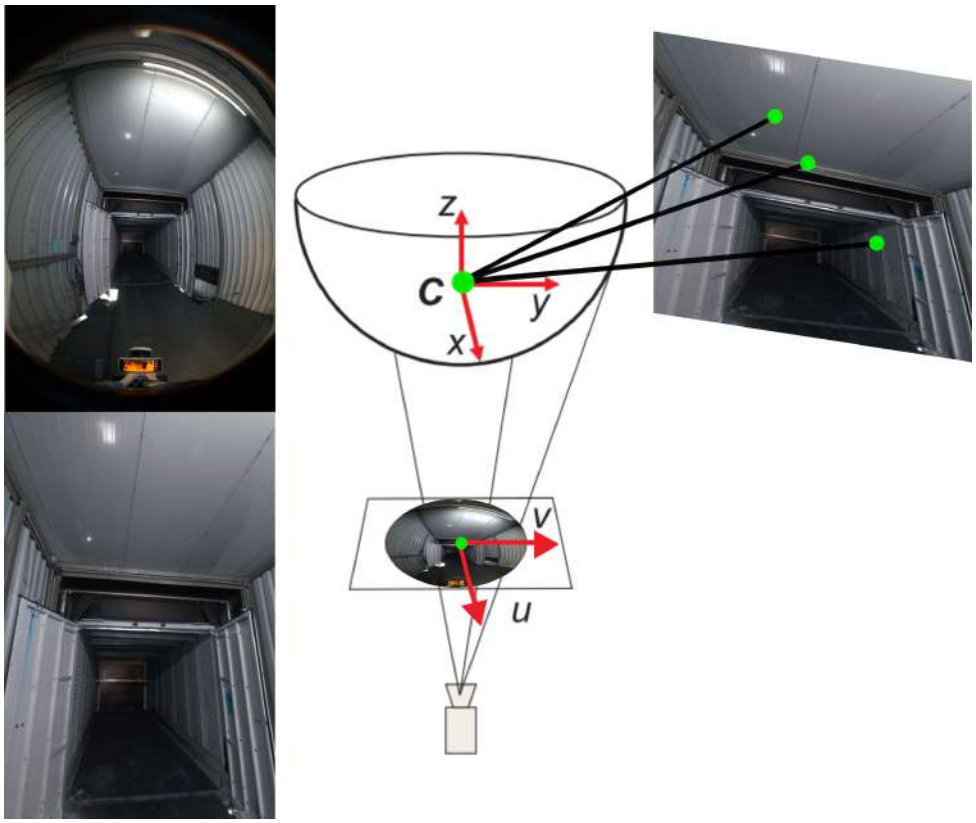


Figure 2.7: Unified Omnidirectional camera model (adapted from [93]).

with a shaped mirror which can be parabolic, hyperbolic, or elliptical. Polydioptric cameras, on the other hand, use multiple cameras with overlapping fields of view to provide a spherical field of view. Most modern polydioptric cameras for robotics often utilize multiple fisheye or dioptric cameras to model a unified omnidirectional camera model [93, 94].

The unified omnidirectional camera model (Figure 2.7) can describe catadioptric and dioptric camera systems and is particularly useful because it bridges the gap between fisheye and omnidirectional cameras, providing a flexible framework for calibration and image processing [91]. This unified approach simplifies the calibration process by reducing the mathematical formulation through the use of a Taylor polynomial whose coefficients and degrees are found through the calibration process [94]. The derivation can be outlined as follows: Consider a 3D point $X = [x, y, z]^T$ in the camera coordinate system. The model first projects this point onto a unit sphere centered at the origin, resulting in normalized coordinates. These are then mapped to the sensor plane using a parametrization $g(\rho)$, which describes the geometry of the optical system. The projection of X onto the normalized image plane $(u'', v'', g(\rho))$ is given by the following:

$$\begin{bmatrix} u'' \\ v'' \\ g(\rho) \end{bmatrix} = \frac{1}{\sqrt{x^2 + y^2 + z^2}} \begin{bmatrix} x \\ y \\ z \end{bmatrix}, \tag{2.17}$$

where, $\rho = \sqrt{u''^2 + v''^2}$ is the distance from the image center and $g(\rho)$ is a function that describes the shape of the mirror or lens that is typically approximated by a polynomial:

$$g(\rho) = a_0 + a_1\rho + a_2\rho^2 + a_3\rho^3 + \dots \tag{2.18}$$

The coefficients a_i are determined during the calibration process. Finally, the coordinates on the actual image plane (u, v) are obtained by applying the intrinsic camera matrix K:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \begin{bmatrix} u'' \\ v'' \\ 1 \end{bmatrix}, \tag{2.19}$$

$$K = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}. \tag{2.20}$$

This unified model is powerful because it can accurately represent the projection geometries of various catadioptric systems (e.g., parabolic, hyperbolic mirrors) as well as fisheye lenses. The flexibility of the polynomial $g(\rho)$ makes it adaptable to a wide range of omnidirectional imaging systems [93].

2.3 Depth Estimation from a Single Image

Estimating depth from images is a long-standing problem in computer vision. Although humans perceive depth through binocular vision, we also use monocular cues such as relative size, perspective projections, texture, and occlusion to perceive depth [95]. Recognizing a 3D scene's spatial structure, or 3D perception, is a fundamental problem in machine vision. Perspective projection is the process of mapping three-dimensional (3D) points onto a two-dimensional (2D) image plane based on the geometry of a camera's optical system. It ensures that the size of an object's image decreases as its distance from the camera increases, mimicking how human vision perceives depth [81]. A key consequence of perspective projection is the loss of depth information, as 3D points are flattened into 2D coordinates, making it challenging to infer spatial relationships directly from images without additional cues or models (Figure 2.8). Monocular depth estimation (MDE) aims to solve this inverse problem by reconstructing the lost third dimension, representing the distance between the image plane and the points in the 3D scene [96]. Various methods have been developed over the decades to solve this challenging problem, which has been the focus of intense research in the field. Historically, depth estimation relied on classical methods such as stereopsis inspired by human vision, structure-from-motion (SfM) [2], shape-from-shading (SfS) [3], and Visual SLAM (VSLAM) [97]; these techniques utilize multiple images or specialized sensors to capture depth information directly or indirectly. Although effective, these methods often require specialized hardware setups and suffer from limitations

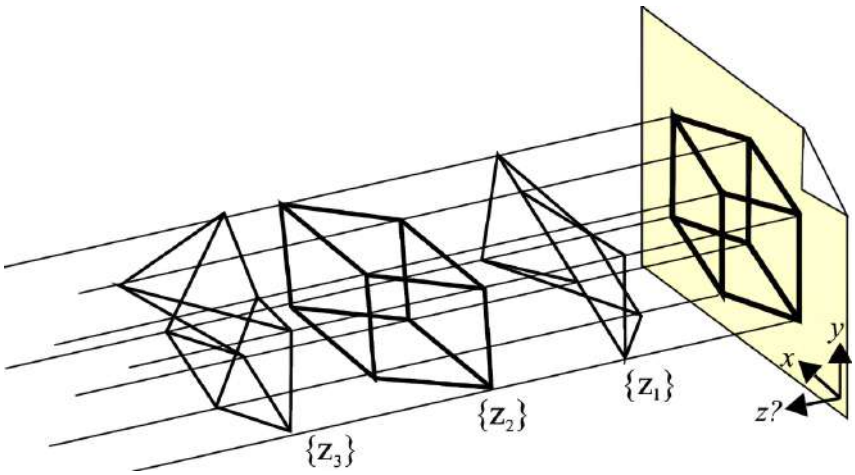


Figure 2.8: Estimating the 3D shape of 2D object is fundamentally ill-posed [96].

in scalability and cost-efficiency. In recent years, deep learning has emerged as a powerful tool for depth estimation in which networks trained on large enough datasets have shown remarkable improvements in accuracy and robustness. This shift towards a more data-driven approach for depth estimation has enhanced the performance of systems that use depth and expanded their applicability across domains [9, 43].

2.4 Supervised Learning for Depth

Supervised learning for depth estimation involves training a neural network model on pairs of images and corresponding depth maps through backpropagation. A loss function penalizes errors between predicted and ground truth depths, and the model converges when these errors are minimized, resulting in predicted depth values closely matching the ground truth [42].
Early works [42, 98] proposed a learning-based approach that utilized a multi-scale convolutional neural network based on a coarse-to-fine framework. In this framework, the coarse network learns the global depth structure of the entire image to produce a rough depth map, while the fine network refines this map by learning local features. These works also introduced a scale-invariant loss function for predicting depth maps from a single image. Since then, various techniques have been developed. These methods generally fall into two categories: formulating MDE as either a pixel-wise regression problem [42, 72] or a pixel-wise classification problem [14, 58, 99]. Regression methods predict continuous depth values, but are challenging to optimize, whereas classification methods predict discrete depth values, which are easier to optimize. A hybrid approach has also been proposed, reformulating the problem as per-pixel classification regression tasks using depth bins, significantly improving MDE precision [60]. Another effective method in supervised deep learning is the encoder-decoder architecture [100]. The encoder consists of convolutional and pooling layers that capture depth features, while the decoder regresses the estimated pixel-wise depth map, maintaining the same size as the input image. In addition, skip connections are used to preserve the features on each scale [64]. The

network is trained using specific depth loss functions such as the scale-invariant logarithmic loss [42] or Huber loss [101], and the model converges when the predicted depth maps meet the desired accuracy. Some works [14, 43] leverage the encoder-decoder structure along with a metric bin decoder [72] to calculate the per-pixel depth bin centers that are linearly combined to output the metric depth using powerful backbone transformer models such as BEiT384L [72,71]. While CNNs were effective when applied to depth estimation, they lacked the global receptive field required for robust depth estimation [102] which changed with the introduction of transformers for vision tasks [50, 71]. Vision Transformers (ViTs) substantially increased the performance of depth estimation models by leveraging self-attention mechanisms to capture global context more effectively. ViTs can model long-range dependencies within an image, which allows for better depth prediction as they can consider the entire image context rather than just local features [50]. This ability to understand global relationships within the image improves the precision of depth maps, particularly in complex cluttered scenes with varying object sizes and distances.

2.5 Vision Transformers

The Transformer architecture represents one of the most significant advancements in deep learning due to its ability to learn global dependencies and contextually understand data [51, 50]. Initially developed as a successor to Recurrent Neural Networks (RNNs) for sequence-to-sequence prediction tasks in natural language processing, Transformers have revolutionized various applications across domains with their self-attention mechanism and parallelism. The architecture enables simultaneous or multi-headed attention to different parts of the input sequence, enabling them to have a global receptive field and making them highly effective at learning compared to CNNs and RNNs.

ViTs adapt the Transformer architecture to images by pre-processing them as sequences of patches. Each image is divided into a fixed number of patches, flattened, and linearly projected onto a sequence of patches (Figure 2.9). These patches then undergo positional encoding to capture their spatial relationship in the image. They are finally converted into latent space representation by the Transformer encoder layers that utilize self-attention to capture the relationship between different patches, allowing the model to learn features both at the local and global levels [50].

A key advantage of the Vision Transformer compared to Convolutional Neural Networks is their ability to handle large-scale image data without the inherent inductive biases in CNNs [103]. Although CNNs excel at capturing local patterns through convolution, they struggle to comprehend long-range dependencies. ViTs can overcome this limitation by leveraging self-attention to model dependencies between different parts of an image. This results in improved performance in various vision tasks that require global comprehension of the scene, such as depth estimation [71].

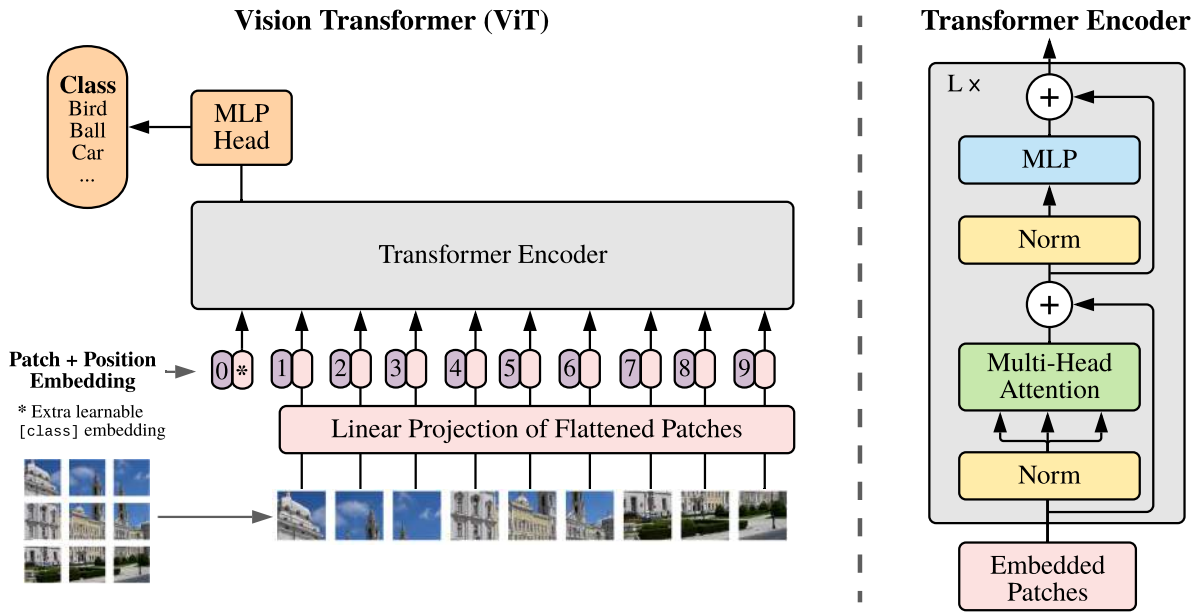


Figure 2.9: Architecture of a Vision Transformer model [50].

2.5.1 Image Tokenization

When using a Vision Transformer, we need to first convert an input image into corresponding tokens similar to transformers in Natural Language Processing (NLP); the simplest choice would be to tokenize each pixel; however, this would be memory inefficient; so the most common approach is to convert an input image into patches of the same size, typically 16x16 pixels. Suppose the images have dimension $x \in \mathbb{R}^{H \times W \times C}$ where H and W are the Height and Width of the image, and C is the number of channels (where typically, $C = 3$ for an RGB image). Each input image is split into non-overlapping patches of size $P \times P$ where (commonly, $P = 16$) and then flattened into a single-dimensional vector which gives the representation $x_p \in \mathbb{R}^{N \times P^2 C}$ where $N = \frac{HW}{P^2}$ is the total number of patches for one image [50].
Another approach to improving efficiency is to use hybrid ViT architectures that use a convolutional backbone and feed the input image first through a small CNN. This helps downsample the input image and create a feature map for input to the ViT, resulting in fewer tokens. This hybrid approach combines the local feature extraction capabilities of CNNs with the global context modeling capabilities of ViTs, offering improved performance.

2.5.2 Attention Mechanism

Attention, or self-attention, is the most crucial concept that enables transformers to handle sequential data and model dependencies between each element in the sequence and allows the model to weigh the importance of different image patches relative to each other. For NLP applications, attention is described as the relationship between words in a sentence; in vision, attention looks at the relationship between image patches in an input image.
Attention is computed using three vectors: the query, key, and value vectors, which form the

basis of the attention mechanism as a function that defines the mapping of the queries and the set of key-value pairs to the output, followed by a softmax operation to normalize these scores.
Single Headed Attention A single attention block (Figure 2.10a) takes in vector queries Q and keys K of each dimension d_k and the value V vector of dimension d_v. The dot product of the query and key vectors computes the attention weights between the queries and keys (cosine similarity). The dot product is then scaled by $\frac{1}{\sqrt{dk}}$ to avoid issues that may arise from large values of d_k in the dot product. In practice, attention is performed in parallel over a set of query, key, and value vectors by stacking into matrices Q, K, and V. This individual attention block forms the single-headed attention mechanism.

$$Attention(Q, K, V) = Softmax(\frac{QK^T}{\sqrt{d_k}})V \in \mathbb{R}^{N \times d_v} \tag{2.21}$$

Multi Headed Attention Multiple attention blocks can be linked to form a more significant network architecture where each head can capture different types of relationships and patterns between the different subspace representations of the input, giving a rich encoded representation of the original data. The query, key, and value are linearly projected h times with different learned linear projections, and attention is computed on each of these projections in parallel. These heads can be concatenated and linearly projected to produce a final multi-headed attention output (Figure 2.10).

$$MultiHead(Q, K, V) = Concat(head_1, ..., head_h)W^0 \tag{2.22}$$

$$\text{,where } head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

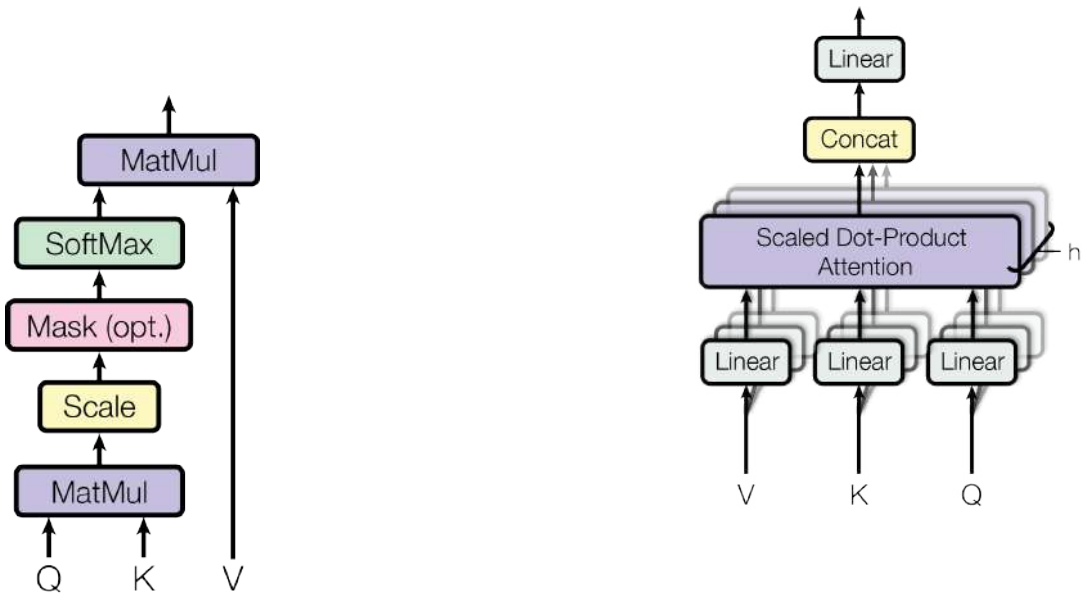
2.5.3 Positional Encoding

Transformers, due to their lack of recurrence or convolution, are incapable of learning information about the order of a set of tokens. Without positional embedding, transformers become invariant to token order. For images, this means that the patches of an image can be scrambled without impacting the predicted output, which is undesirable.
Positional encoding can be of two types: fixed or learned, based on the requirements.

$$P_E(Pos, 2i) = \sin \frac{Pos}{N^{\frac{2i}{D}}} \tag{2.23}$$

$$P_E(Pos, 2i+1) = \cos \frac{Pos}{N^{\frac{2i}{D}}}$$

Unlike the sinusoidal position encodings used in the original work for NLP tasks [51], ViTs typically use learnable position embeddings [50]. These are randomly initialized and updated during training, along with other model parameters. In [50], positional embeddings are applied to the images with ablation studies that showed variations with no position embedding, 1D posi-



(a) Scaled Dot-Product Attention (b) Multi-Headed Attention

Figure 2.10: Transformer Attention Blocks [50].

tion embedding, 2D position embedding, or relative position embedding. Models with a position embedding significantly outperform models without a position embedding. However, there is little difference between their different types of positional embeddings or between fixed and learnable embeddings. As stated in the original work [50], a position embedding is beneficial, though the exact embedding chosen is of little consequence.

2.6 Supervised Fine-Tuning of Foundation Depth Models

Previous depth prediction architectures utilized convolutional networks or hybrid designs such as ConvNeXt [104], which are typically structured as encoder-decoder systems. In these frameworks, the encoder employs pre-trained backbones [105–107] to extract hierarchical features from input images, whereas the decoder aggregates these features into depth predictions. Convolutional backbones progressively downsample inputs, enlarge the receptive field, and abstract high-level features. However, this downsampling sacrifices spatial resolution and fine-grained details, which are critical for accurate depth estimation, where recovering the lost granularity in the decoder remains challenging. Consequently, backbone selection profoundly impacts the model's performance, as demonstrated by architectures such as DPT (Dense Prediction Transformer) and BinsFormer [71, 99]. A key limitation in training transformer-based depth models is the scarcity of open-source RGB-D data, which constrains large-scale supervised training. To mitigate this, synthetic datasets [108, 109] and self-supervised methods have become instrumental for pretraining, with domain-specific real-world fine-tuning bridging the gap in practical applications. Aligning synthetic data with target depth distributions ensures that the models generalize effectively to real-world scenarios, motivating advances in tailored synthetic data generation. Modern approaches address these challenges by integrating foundation models that

are pre-trained on diverse vision tasks. For instance, Depth Anything [43, 15] utilizes the DPT architecture [71] for its decoder with DINOv2 [105] (Distillation with No Labels) for feature extraction as its encoder backbone, which is enhanced by self-supervised distillation to preserve multiscale contextual and high-resolution details (Figure 2.11). By initializing an MDE model with such robust backbones and pre-training on extensive datasets, the MDE model inherits rich representations of global context and local texture. Subsequent supervised fine-tuning of domain-specific metric-depth data further refines the accuracy, enabling adaptation to specific applications while maintaining the backbone's intrinsic understanding of visual hierarchies for depth. This two-stage paradigm, large-scale pretraining followed by targeted fine-tuning, balances generalizable feature learning and task-specific precision, ultimately enhancing reliability in real-world deployment.

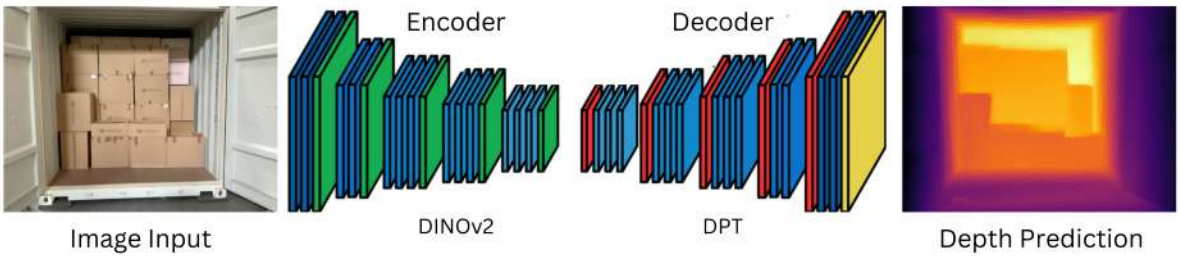


Figure 2.11: Encoder-Decoder architecture of Depth Anything V2 (adapted from [110]).

2.6.1 DINOv2

DINOv2 [105] represents the latest advancements in self-supervised learning in computer vision [111, 112] by building upon the advances from CNNs, ViTs, and self-supervised learning techniques. At its core, the DINO architecture utilizes a modified ViT that benefits from the global modeling capabilities of transformers and addresses its limitations by incorporating multi-scale feature learning and improved attention mechanisms that balance efficiency and accuracy. The architecture achieves robust performance by relying on large-scale self-supervised pretraining and is trained using self-distillation without labels, where a student network learns to match the output of a momentum-updated teacher network[5], enabling it to learn rich, semantically meaningful representations.

When used as an encoder, DINOv2 transforms an input image into a structured feature representation in a tokenized form. The DINOv2 encoder processes input images by dividing them into patches and encoding each patch into high-dimensional feature vectors using a ViT backbone. These tokens capture both global and local details, making DINOv2 well-suited for downstream tasks such as depth estimation, segmentation, and object recognition. The resulting feature maps can be decoded by transformer or convolutional decoders for dense prediction tasks. DINOv2's

[5] A momentum-updated teacher network in DINOv2 refers to a model whose parameters are gradually updated using an exponential moving average of the student network's parameters, providing a stable and high-quality target for the student to learn from during self-distillation.

ability to generate domain-agnostic, high-quality features with strong generalization properties makes it a robust encoder backbone for various vision applications [15, 105].

2.6.2 Dense Prediction Transformer (DPT)

DPT [71] is an encoder-decoder architecture that is tailored for dense prediction tasks. Dense prediction tasks involve making a prediction for every single pixel in an image, rather than for the whole image or few key features contained in the image. Depth estimation, among other vision tasks such as semantic segmentation and normal estimation, falls into the category of dense prediction tasks and requires models to output spatially structured information.

A major drawback of convolutional backbones for dense prediction tasks is that feature resolution and granularity are lost in deeper stages of the network and is hard to recover by the decoder [71, 103]. To overcome this, the DPT uses a ViT as the encoder backbone to process image patches into tokens. Unlike CNNs, which progressively downsample the input image to extract features at different scales, ViTs maintain constant resolution and global receptive fields across the layers, preserving fine-grained details. The DPT decoder (Figure 2.12) is what makes robust dense prediction possible. It utilizes a multi-scale fusion structure that assembles the set of tokens extracted from the ViT backbone and reconstructs 2D spatial details from the flattened token sequences from the encoder. The features are progressively fused into the final dense prediction using a three-stage reassemble operation to recover the input from the output tokens of arbitrary layers of the encoder. After multi-scale fusion, a final convolutional layer generates the dense prediction output. This architecture effectively bridges the global receptive field of transformers with the spatial precision of convolutional operations without sacrificing granularity and preserving structure, enabling accurate dense predictions.

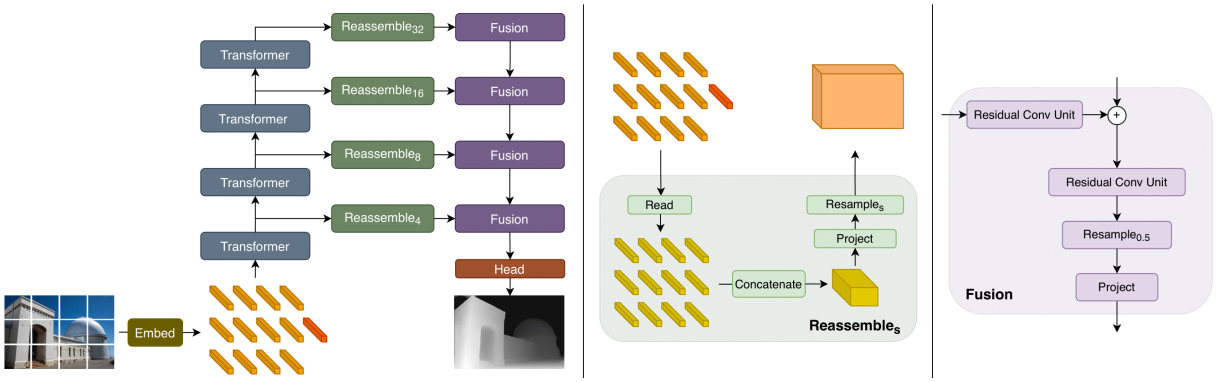


Figure 2.12: Overview of DPT Architecture with input image transformed into tokens (orange) and embedded positionally (red). The tokens are passed through multiple stages and reassembled at multiple resolutions (green). Fusion modules (purple) progressively fuse the representations for a final fine grained prediction [71].

2.7 Surface Reconstruction

Surface reconstruction is a fundamental problem in computer graphics and computational geometry [113, 114]. The goal is to create a continuous surface representation from a discrete set of sampled points. Given a set of n points $P = \{P_i \in \mathbb{R}^3 \mid i = 1, 2, 3.., n\}$, that are coordinates in 3D space which represent the surface of an object, the objective of surface reconstruction is to find a surface that approximates or interpolates these points to form a coherent surface regardless of noise, outliers, non-uniform point densities, and complex surface topology. The surface S is an orientable, continuous two-dimensional manifold within $\mathbb{R}^3$ ($S \subset R^3$) that may or may not have boundaries. Surfaces in computer graphics can be categorized as parametric or implicit. Parametric surfaces are represented by a continuous and bijective function $f : \Omega \hookrightarrow S^g$ that maps a parameter domain $\Omega \in \mathbb{R}^2$ to a 3D surface $S^g = f(\Omega) \in R^3$. Typically, the parameter domain Ω is divided into smaller sub-regions, each described by its own function and Ω is partitioned into triangles approximating S^g and can be efficiently stored as a triangle surface mesh $M = (\nu, \epsilon, \mathcal{F})$, where ν denotes the vertices of the triangles, ϵ their edges, and $\mathcal{F}$ their facets. Implicit surfaces, on the other hand, are defined by the level-set $\mathcal{C}$ of a scalar-valued function $F : \mathbb{R}^3 \hookrightarrow \mathbb{R}$:

$$\mathcal{S}_c^g = \left\{ \mathbf{x} \in \mathbb{R}^3 \mid F(\mathbf{x}) = \mathcal{C} \right\}. \tag{2.24}$$

The most common choices for F are Signed Distance Functions (SDF) or Occupancy Functions (OF) [115, 116]. An SDF gives the distance from a 3D point $\mathbf{x}$ to the closest part of the surface. Points to the interior of the surface are given negative values, the points on the surface are 0, and the points on the exterior are assigned positive values. An OF, in contrast, typically has a value of 1 inside the surface and 0 outside. The domains of implicit functions are split into sub-regions using Voxels, Octrees, or Tetrahedra, and most algorithms convert the computed implicit surface into parametric meshes using algorithms such as marching cubes as meshes are easier to handle for various applications [114]. The marching cubes algorithm [117] divides the volume into a grid of cubes and processes each cube to extract a polygonal mesh that approximates the isosurface (Figure 2.13). It is known for its efficiency and simplicity and is widely used in medical imaging. On the other hand, Poisson reconstruction [77] approaches the problem by solving a Poisson equation to produce a smooth, watertight surface that best fits the input points. The method is particularly effective in handling noisy data and generating high-quality surfaces.

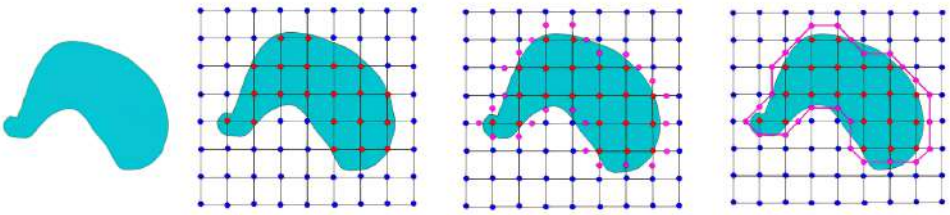


Figure 2.13: Illustration of the Marching Cubes algorithm in 2D [118].

2.7.1 Poisson surface reconstruction

Poisson surface reconstruction [77] is a method used to reconstruct smooth, continuous, watertight surfaces from a set of oriented points. It effectively handles noisy, incomplete data and works by solving a Poisson equation. Poisson's reconstruction is based on the observation that the inward-facing normal field of the boundary of a solid can be interpreted as the gradient of the solid's indicator function. Hence, given a set of oriented points and normal vectors $P = \{(p_i, n_i) \mid i = 1, 2, ..., n\}$ sampling the boundary where $p_i, n_i \in \mathbb{R}^3$, a watertight mesh can be obtained by approximating a surface that best fits these points and normal fields. The surface reconstruction problem is formulated as a Poisson equation where the core idea is to find a scalar function $\mathcal{X} : \mathbb{R}^3 \hookrightarrow \mathbb{R}$ such that the gradient of $\mathcal{X}$ matches the vector field defined by the input normals. The goal is to find the scalar function $\mathcal{X}$ whose gradient best approximates the vector field $\vec{V}$ ($\nabla\mathcal{X} = \vec{V}$). This can be formulated as a Poisson equation where Δ is the Laplacian operator, and ∇V is the divergence of the vector field $\vec{V}$.

$$\Delta\mathcal{X} = \nabla \cdot \nabla\mathcal{X} = \nabla \cdot \vec{V} \tag{2.25}$$

$$\min_{\chi} \int \|\nabla\chi(\mathbf{x}) - \vec{V}(\mathbf{x})\|_2^2 dx \tag{2.26}$$

The Poisson equation to be solved is then:

$$\Delta\mathcal{X} = \nabla \cdot \vec{V} \tag{2.27}$$

Solving the Poisson equation to obtain the indicator function $\mathcal{X}(x)$ is typically done by discretizing the equation and solving a linear system (Figure 2.14). This is often achieved using an adaptive octree for efficient space partitioning and representation. The linear system can be solved using numerical methods such as conjugate gradient or multigrid solvers, which are chosen for their efficiency in handling large, sparse systems. Once the indicator function is computed, the surface can be extracted as an isosurface, usually through the marching cubes algorithm. This results in a triangulated mesh representation of the reconstructed surface.

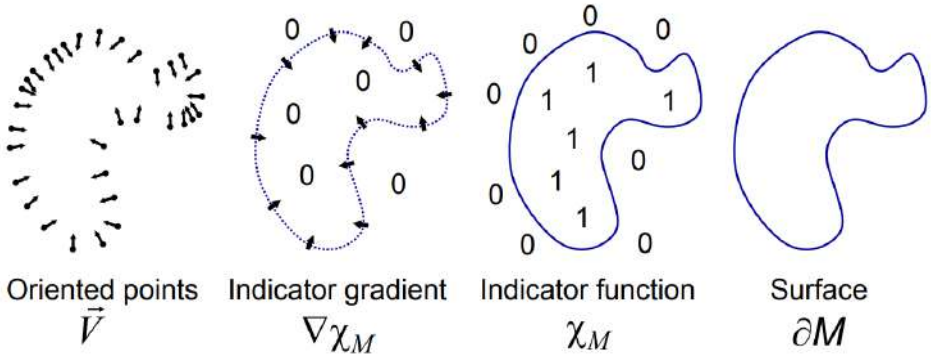


Figure 2.14: Formulation of indicator function for Poisson Reconstruction [119].

2.8 Metrics for Depth and Pointclouds

For 3D reconstruction, depth acts as an intermediary stage, providing information about the distance of each object in the scene to the camera and serving as an input to reconstruct points and surfaces. To effectively evaluate the performance of 3D reconstruction using MDE, it is crucial to examine the depth estimation process independently and assess the accuracy of the output from the monocular depth estimation model, as any discrepancies during this stage can significantly impact the overall quality of the reconstruction. Moreover, metrics for depth serve as a benchmark to compare different models and methodologies, guiding practitioners in developing more effective solutions. The metrics encompass various aspects, including accuracy, precision, completeness, and robustness to noise and lighting [120, 121].
In addition to depth metrics, evaluating the quality of the final point cloud derived from the depth data is crucial to assessing the reconstructed geometry and scale of the scene. This subsection will delve deeper into the key metrics chosen to evaluate the depth maps and the resulting reconstructed points, providing an understanding of how to evaluate and assess the overall 3D reconstruction algorithm.

2.8.1 Metrics for Depth

Root Mean Squared Error
The root mean squared error calculates the average of the squared differences between the predicted depth ($d_{i,j}$) and the ground truth ($d^*_{i,j}$). It measures the average difference between values predicted by a model and the actual values.

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i,j}\left|d_{i,j} - d^*_{i,j}\right|^2} \tag{2.28}$$

Mean Absolute Error
The mean absolute error calculates the average of the absolute magnitude difference between the predicted depth ($d_{i,j}$) and the ground truth ($d^*_{i,j}$).

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|d_{i,j} - d_{i,j^*}| \tag{2.29}$$

Absolute Relative Error
The absolute relative error is a metric useful for evaluating the accuracy of a measurement that calculates the absolute differences between the estimated ($d_{i,j}$) and ground truth depth, normalized by the ground truth depth ($d^*_{i,j}$).

$$\text{AbsRel} = \frac{1}{N}\sum_{i,j}\frac{\left|d_{i,j} - d^*_{i,j}\right|}{d^*_{i,j}} \tag{2.30}$$

Threshold Accuracy δ_1

The threshold accuracy δ_1 or accuracy with a threshold is a commonly used metric for depth estimation, which divides the error ratios δ_λ into intervals determined by a threshold λ. Typically, $\lambda = 1.25$ (δ_1) can be interpreted as the percentage of pixels where the depth deviation from the ground truth is less than 25%. A higher number indicates that fewer predicted pixels lie outside the specified threshold than the ground truth.

$$\delta_{1.25} = \frac{1}{N}\sum_{i=1}^{N}\left[\max\left\{\frac{d_i^*}{d_i}, \frac{d_i}{d_i^*}\right\} < \delta_{1.25}\right] \tag{2.31}$$

Structural Similarity Index

The Structural Similarity Index (SSIM) for depth is a metric used to assess the similarity between the predicted and ground truth depth maps. Rather than pixel-wise differences, it compares structural information, such as edges. SSIM allows us to quantify how well the structural details in the predicted depth map align with those in the ground truth and is calculated using the following equation [122].

$$\text{SSIM}(d_{\text{pred}}, d_{\text{true}}) = \frac{(2\mu_{d_{\text{pred}}}\mu_{d_{\text{true}}} + C_1)(2\sigma_{d_{\text{pred}},d_{\text{true}}} + C_2)}{(\mu_{d_{\text{pred}}}^2 + \mu_{d_{\text{true}}}^2 + C_1)(\sigma_{d_{\text{pred}}^2} + \sigma_{d_{\text{true}}^2} + C_2)} \tag{2.32}$$

Where, μ and σ are the mean and standard deviation of the original and predicted depth maps and C terms are small constants that avoid numerical instability.

2.8.2 Metrics for Pointcloud

Hausdorff's Distance

The Hausdorff Distance provides a stringent evaluation metric for comparing point clouds by measuring the greatest of distances between closest point pairs between the predicted and ground truth points. Unlike the Chamfer Distance, which considers average distances, the Hausdorff Distance identifies the worst-case deviation between the two point sets [123]. This metric is particularly useful for identifying outliers and ensuring the completeness of the reconstruction, as it is sensitive to any significant deviations between the predicted and ground truth clouds. A lower Hausdorff Distance indicates that even the most divergent points in the prediction are still reasonably close to the ground truth, suggesting a more reliable and accurate reconstruction across the entire point cloud. The Hausdorff Distance between two point sets P_1 and P_2 is defined as the maximum distance between any pairs of nearest neighbors between P_1 and P_2:

$$d_H(P_1, P_2) = \frac{1}{2}\max_{x\in P_1}|x - \text{NN}(x, P_2)| + \frac{1}{2}\max_{x'\in P_2}|x' - \text{NN}(x', P_1)| \tag{2.33}$$

Where $\text{NN}(x, P) = \text{argmin}_{x'\in P}\|x - x'\|$ represents the nearest neighbor function.

Density Aware Chamfer's Distance

Chamfer's Distance (CD) and Earth Mover's Distance (EMD) are widely used metrics for measuring the similarity between two point clouds. However, Chamfer's distance is insensitive to mismatched, locally dense points (Figure 2.15). While EMD only captures the global distribution between the point clouds and overlooks the local structures in the point cloud, density-aware Chamfer Distance (DCD) [124] builds upon the Chamfer Distance and considers both the overall distribution and the local density of the points, resulting in a more comprehensive evaluation (Figure 2.16).

The Chamfer's Distance between two point sets P_1 and P_2 is defined as:

$$d_{CD}(P_1, P_2) = \frac{1}{|P_1|} \sum_{x \in P_1} \min_{y \in P_2} ||x - y||_2 + \frac{1}{|P_2|} \sum_{y \in P_2} \min_{x \in P_1} ||y - x||_2 \tag{2.34}$$

Then, the Density Aware Chamfer's Distance can be defined as:

$$d_{DCD}\left(P_1, P_2\right) = \frac{1}{2}\left(\frac{1}{|P_1|} \sum_{x \in P_1}\left(1 - \frac{1}{n_{\hat{y}}} e^{-\alpha\|x-\hat{y}\|_2}\right) + \frac{1}{|P_2|} \sum_{y \in P_2}\left(1 - \frac{1}{n_{\hat{x}}} e^{-\alpha\|y-\hat{x}\|_2}\right)\right) \tag{2.35}$$

However, the equation above (Equation 2.35) only applies to when the point sets P_1 and P_2 contain the same number of points. When there is a mismatch in the number of points between the two point sets, a term is added to indicate the one-to-many mappings which also avoids division by small values $max(\frac{\eta}{\eta_{\hat{y}}}, 1)$, giving a final formulation (Equation 2.36) for the function as:

$$\begin{aligned} d_{DCD}\left(P_1, P_2\right) = & \frac{1}{2\left|P_1\right|} \sum_{x \in P_1}\left(1 - \frac{1}{\max\left(\eta / n_{\hat{y}}, 1\right)} e^{-\alpha\|x-\hat{y}\|_2}\right) \\ & + \frac{1}{2\left|P_2\right|} \sum_{y \in P_2}\left(1 - \frac{1}{\bar{\eta} \cdot n_{\hat{x}}} \sum_{\hat{x} \in N(y)_{\bar{\eta}}} e^{-\alpha\|y-\hat{x}\|_2}\right) \end{aligned} \tag{2.36}$$

Where $\hat{y} = \min_{y \in S_2} \|x - y\|_2,\ \hat{x} = \min_{x \in S_1} \|y - x\|_2$ and α denote a temperature scalar.

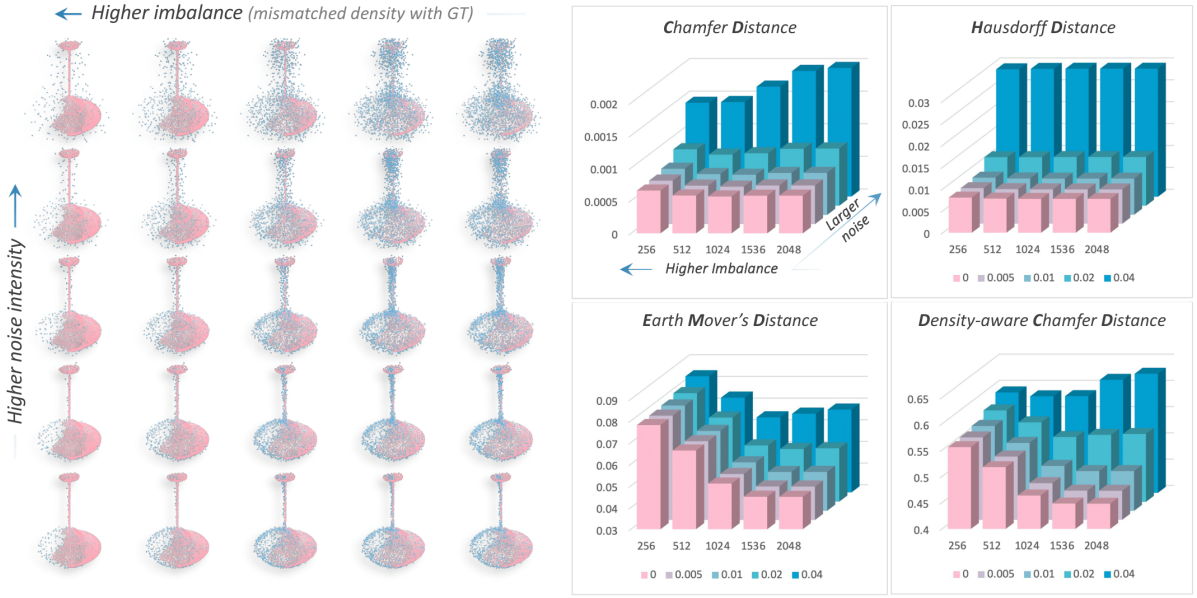


Figure 2.15: Comparison of point cloud metrics under varying noise intensity and density imbalance. The left panel illustrates examples with increasing noise and density mismatch. The right panel shows how the pointcloud metrics respond to such variations [124].

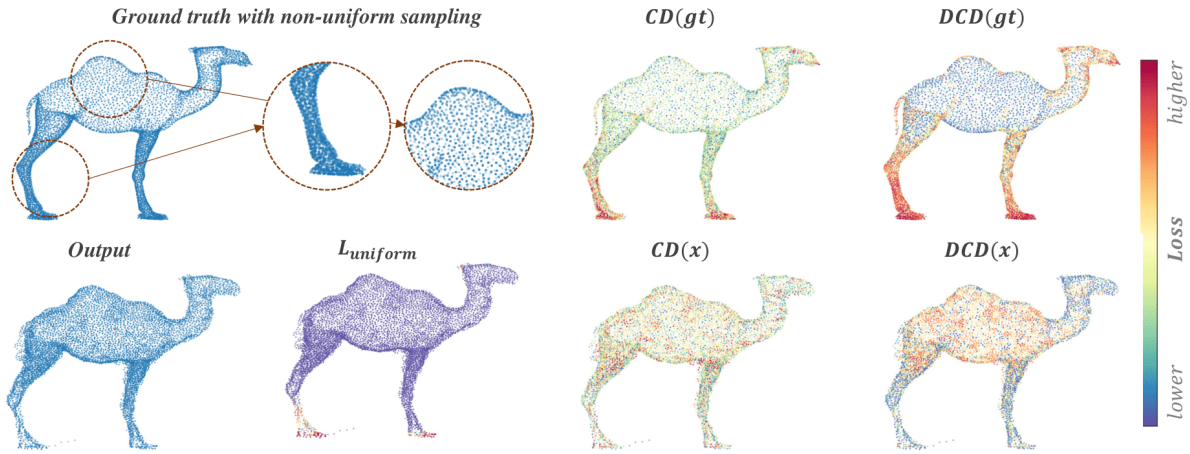


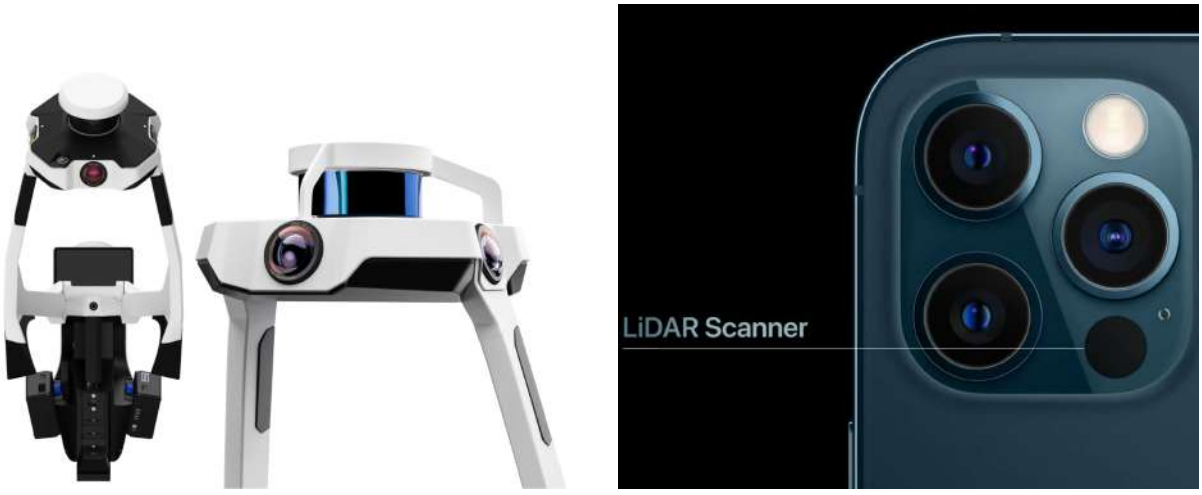
Figure 2.16: Visualization of the distance contributed by each point. DCD is better at capturing mismatched densities in local areas between pointclouds [124].

3 Dataset Generation

This section details the data collection and processing pipelines developed for the project. We describe the hardware setup and calibration procedures required to capture high-quality, aligned RGB-D data. The workflow for cleaning, processing, and refining collected multi-modal datasets comprising images, point clouds, and depth maps from diverse sources required consistent preprocessing to ensure compatibility. This step is critical for effectively evaluating and fine-tuning monocular depth estimation models, which are central to our 3D reconstruction application. Following this, we outline the steps in extracting raw data and the algorithms employed to process the raw data into usable form. Finally, we discuss selected public datasets used and augmentation techniques applied to enhance model robustness and improve fine-tuning outcomes.

3.1 Hardware Setup

To ensure comprehensive and accurate image, depth, and point cloud data of warehouse and container scenes required by our application, our data collection process employed three distinct devices: a high-precision NavVis VLX 3 scanner, two LiDAR-equipped consumer Apple devices (iPad Pro & iPhone 13 Pro) and a custom handheld "Flir-Livox" scanner comprising a Livox Horizon LiDAR and Flir BlackflyS RGB camera. Data was collected through multiple on-site visits to logistics warehouses in Duisburg and Dortmund, where detailed scans of container docks and loading areas were performed. Our approach yielded a rich multi-modal dataset comprising images, depth maps, pose data, and point clouds. This data was crucial for evaluating and fine-tuning depth estimation models for our application.



(a) NavVis VLX 3 Scanner (adapted from [125]) (b) LiDAR sensor on iPhone 13 Pro [126]

Figure 3.1: Comparison of NavVis VLX 3 Scanner and LiDAR sensor on iPhone.

3.1.1 NavVis VLX3

The NavVis VLX 3 is a state-of-the-art wearable dynamic mapping system designed for high-precision large-scale scanning applications (Figure 3.1a), such as the comprehensive documentation of warehouse and factory environments creating digital twins. This industrial-grade mobile mapping solution delivers highly accurate 3D maps by leveraging advanced technologies, including dual LiDAR sensors, integrated fisheye cameras, and incremental Simultaneous Localization and Mapping (SLAM) algorithms. The system achieves a point cloud accuracy of up to 5 mm and supports measurements over distances of up to 50 m [127], making it suitable for both indoor and outdoor use.

In the context of this project, we employed the NavVis VLX 3 to capture detailed spatial data within an indoor warehouse setting (Figure 3.2); this included multiple loading bays with cargo containers and cargo lying around in the staging areas. The device generates dense point cloud representations that are not only precise but also rich in detail. Its dual LiDAR configuration features two sensors with 32 beams each, collectively capable of capturing data at a rate of 1.28 million points per second (640,000 points per second per sensor) which ensures comprehensive coverage and accuracy even in complex environments.

Additionally, the NavVis VLX 3 incorporates four fixed-focus fisheye cameras with focal lengths of 3.3 mm, apertures of f/2.4, and an image resolution of 20 Megapixels (MP). These cameras capture images that are subsequently stitched into panoramic views using an omnidirectional camera model. To ensure compatibility with standard imaging workflows or mobile devices, these captured images must undergo post-processing steps such as being undistorted before being converted from the omnidirectional camera model to a pinhole camera model using specialized algorithms [91]. This combination of high-performance LiDAR sensors and advanced imaging capabilities made the NavVis VLX 3 a versatile tool for generating accurate and detailed digital twins of physical spaces [125].

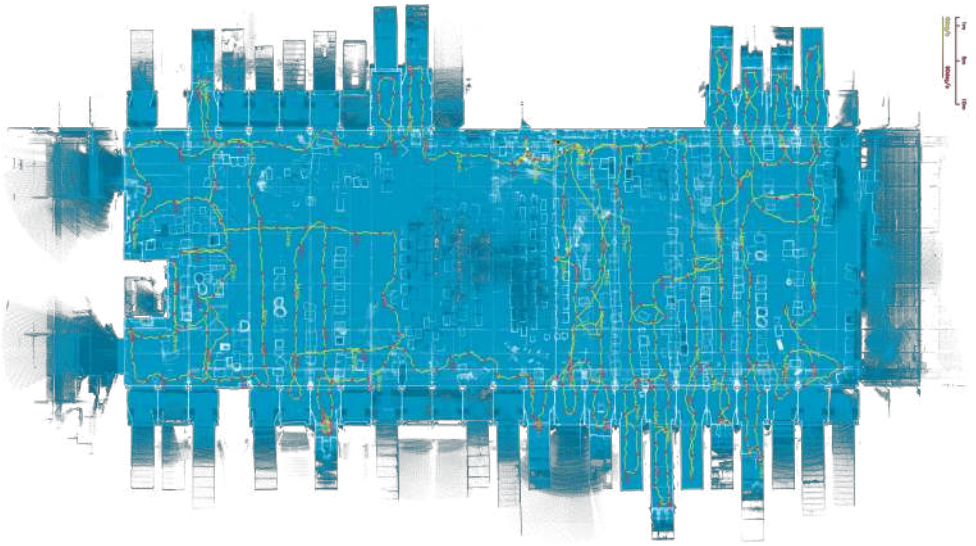


Figure 3.2: Scanning path through the warehouse using the NavVis VLX3.

3.1.2 Mobile LiDAR Devices

To complement the high precision data obtained from the NavVis VLX 3 scanner and Flir-Livox setup, we follow [128, 129] and utilize Apple devices equipped with LiDAR sensors (iPhone 13 pro and iPad pro). These devices offer portability and flexibility while closely resembling input data for our application, making them suitable for capturing data of container scenes and in confined spaces where the other devices were less practical. The LiDAR sensor (Figure 3.1b) on the iPhone 13 pro, located in the lower right corner of the camera cluster, operates on the principle of time-of-flight (ToF) and contains an emitter with 64 vertical cavity surface-emitting laser (VCSEL) cells, which are multiplexed by a 3 x 3 diffraction optical element (DOE) to generate 576 laser pulses [130]. These pulses reflect off object surfaces, and their time elapses are measured by a single-photon avalanche diode (SPAD) image sensor (Figure 3.3).
Both mobile devices we used featured a 12-megapixel RGB camera with a focal length equivalent to 26 mm on the iPad and iPhone.
The LiDAR sensor captures depth information by combining 576 laser points with RGB data from cameras to generate a depth map of 256x192 pixels at 60 Hz using a proprietary fusion algorithm. Apple's ARKit API (Application Programming Interface) provides access to the processed depth map, which can be captured using off-the-shelf apps such as 3DScannerApp[6] or Polycam[7], which utilize SLAM algorithms to capture data and allow for quick export of the raw data consisting of RGB images, depth maps, confidence maps, and point clouds for each scan. The inclusion of ARKit data was essential for simulating real-world data in logistics, where mo-

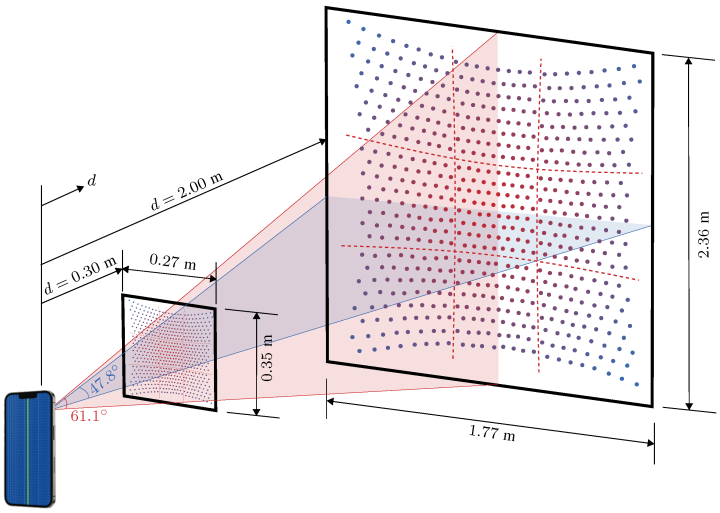
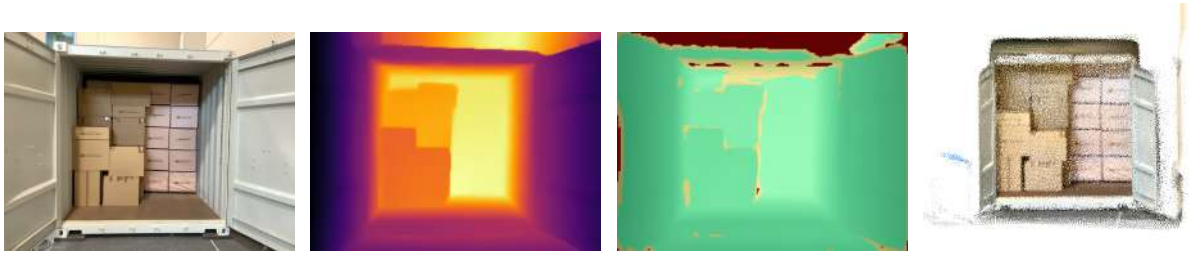


Figure 3.3: Distances and field of view necessary for reliable object measurement using the iPhone LiDAR system [130].

bile devices like smartphones and tablets are more likely to be deployed. The data served as an important testing ground for initially benchmarking various depth estimation models. However,

[6] https://3dscannerapp.com/
[7] https://poly.cam/

given the lower fidelity of ARKit-generated depth maps and their maximum distance limitation of 5 m, they represented a more challenging dataset for fine-tuning depth estimation models. Evaluating MDEs on the images from the mobile devices provided insights into how well they could generalize beyond training data and handle noise, close-range depths, and reduced resolution, which are common issues in real-world scenarios.
While the ARKit LiDAR data offers an accessible, low-cost approach to 3D data gathering, it is important to also recognize its limitations [131, 130, 128]. The sensor's 5-meter depth range means that depth values captured for objects farther away, such as the far ends of containers, are often inaccurate, as reflected in the confidence maps (Figure 3.4c). By combining high-precision industrial scans with consumer-grade data, we aim to curate a comprehensive mobile RGB-D dataset that can be used to benchmark and fine-tune monocular depth estimation for mobile devices.



(a) RGB Image (b) Depth Image (c) Depth Confidence (d) Pointcloud

Figure 3.4: ARKit Data Outputs (a) RGB image, (b) depth image, (c) depth confidence map, and (d) point cloud.

3.1.3 Flir-Livox Scanner Prototype

To overcome the limitations imposed by the NavVis VLX3 and Apple LiDAR Scanners, we developed a custom handheld scanner[8] that comprised a Livox Horizon LiDAR and BlackflyS camera (Figure 3.5a), powered by a portable battery pack (Figure 3.5b). This setup was designed to capture high-quality dense depth RGB-D data required for evaluation purposes. When activated, this system recorded a continuous stream of LiDAR point clouds and synchronized RGB images. It was lightweight and ergonomic enough to be operated by a single person, allowing flexible movement around the scene at various distances. The software for collecting data was integrated with the Robot Operating System (ROS), enabling seamless operation and efficient data acquisition in complex environments. Using this setup, we collected an extensive dataset from a large warehouse in Dortmund, including point clouds and images of various container scenes. In addition to the RGB camera, we also utilized two mobile phones (iPhone 12 and Google Pixel 7) to collect additional RGB images of the same scene using a tripod stand for consistently aligned shots.

[8]The author's involvement only extends to the software aspects of the device such as calibration, data extraction and alignment.

(a) Prototype handheld Flir-Livox scanner (b) Prototype Scanner with Battery Pack

Figure 3.5: Different views of the prototype handheld Flir-Livox scanner.

Livox Horizon LiDAR

The Livox Horizon is a high-performance LiDAR designed for precise 3D mapping in autonomous navigation applications. Its horizontal field of view spans 81.7°, while its vertical FoV is 25.1°. The device achieves a detection range of up to 260 m at 80% reflectivity with an angular resolution as fine as 0.2°. The Horizon operates using a non-repetitive "hourglass shaped" scanning pattern that captures 240,000 points per second to create a dense representation over time (Figure 3.6), making it particularly suitable for indoor mapping tasks where uniformity in point density is critical and point density can be controlled [127].

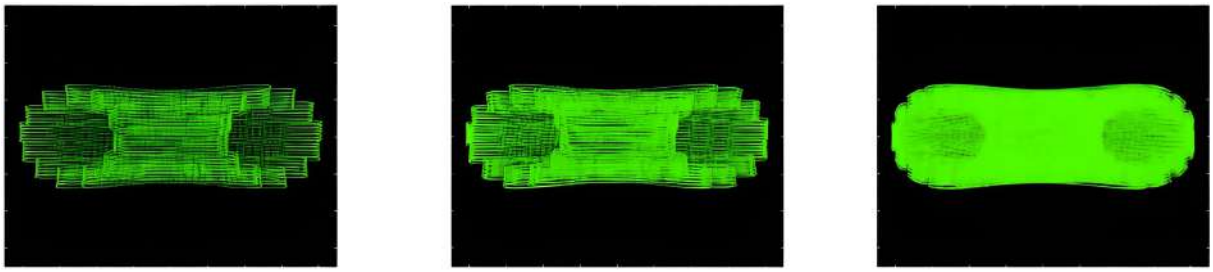


Figure 3.6: The accumulation of points over time in Livox Horizon [127].

Flir BlackflyS Camera

The Flir BlackflyS is an industrial-grade RGB camera equipped with Sony's IMX183 CMOS (Complementary Metal-Oxide-Semiconductor) sensor. It features a resolution of 1280 × 1024 pixels at 30 frames per second (FPS), ensuring color image capture for our RGB-D dataset [132]. A fisheye lens with variable focus and aperture was mounted on this camera (Figure 3.5) to maximize coverage within indoor settings while maintaining sharpness across varying depths of field. While we performed on-site calibration of the camera at the warehouse to ensure optimal gamma and focus adjustments aligned with the Livox LiDAR's scanning plane, the absence of photometric auto-adjustment features when operated via the ROS driver, combined with a fixed

focus setting, often resulted in dark and blurry images of container interiors where details at the far ends were not clearly visible. To address this limitation, we supplemented our setup by using two mobile phones to capture high-resolution RGB images.

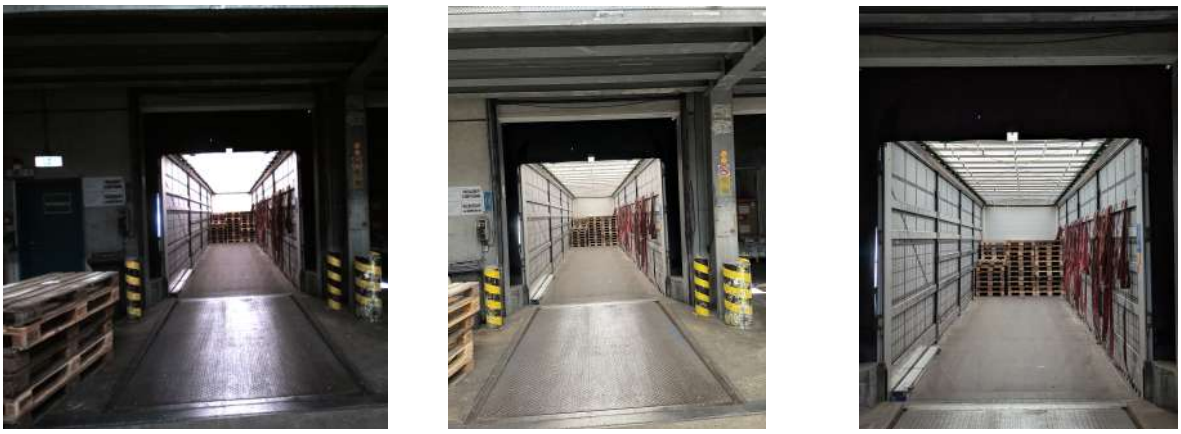
iPhone12 and Pixel 7 Devices

To complement our primary handheld scanner setup, we utilized two high-end smartphones to capture additional high-resolution RGB images of the scenes. Both devices (iPhone 12 and Google Pixel 7) were mounted next to the Flir camera during data collection to ensure consistent framing and alignment across all shots taken from fixed perspectives around each scene.

The iPhone 12 features a dual-camera system comprising a wide-angle lens with a 12 MP sensor (f/1.6 aperture) and an ultra-wide-angle lens with a 12 MP sensor (f/2.4 aperture). It comes with advanced computational photography algorithms that automatically enhance image quality by improving dynamic range, reducing noise, and retaining fine details even in challenging lighting conditions.

Similarly, the Google Pixel 7 is equipped with a dual-camera system that includes a wide-angle primary camera and telephoto lens with a resolution of 50 MP (f/1.9 aperture, 25mm equivalent focal length) paired with laser autofocus technology for precise focusing. The device also leverages advanced software enhancements like automatic high dynamic range and Pixel Shift for improved sharpness and color accuracy in low-light or high-contrast conditions. The telephoto capabilities of the Pixel 7 enabled us to take detailed close-up shots of the container interiors from approximately 5 m away.

By incorporating these additional devices into our data collection process, we captured complementary data that addressed limitations in low-light performance and mitigated some of the challenges posed by the Flir camera.



(a) Flir BlackflyS (b) iPhone 12 (c) Pixel 7

Figure 3.7: Comparison of RGB images captured by different devices.

3.2 Calibrations

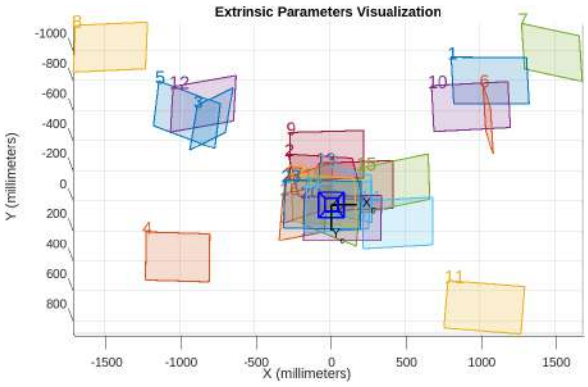
To effectively utilize the data collected by our multi-modal sensor setup, it was necessary to calibrate the individual camera parameters and perform external calibration to accurately align all RGB and projected Depth images.

The NavVis VLX3, due to its proprietary nature, came factory calibrated and had all the parameters and distortion coefficients given as calibration text files. The 3D data is automatically fused, processed, and uploaded to a cloud storage server and can be downloaded from the cloud.

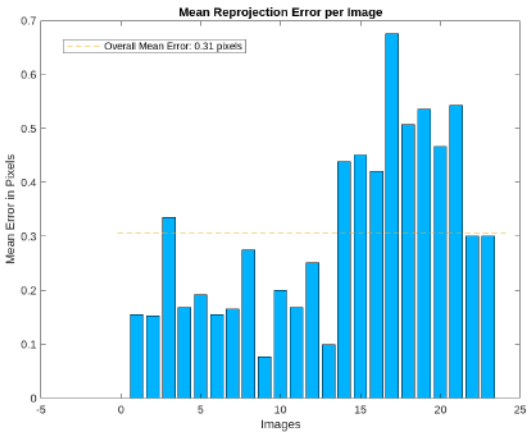
We began by calibrating all the mobile devices used in the project to determine their camera intrinsic parameters. This included the LiDAR-equipped Apple devices (iPad Pro 2020 and iPhone 13 Pro), the smartphones used for RGB capture (iPhone 12 and Pixel 7), and the Flir Blackfly S cameras. Additionally, we utilized apps such as Polycam and 3DScanner during data collection to extract detailed camera parameter information. These apps provided a **.json** file for each RGB-D pair, containing details such as camera intrinsics, blur coefficients, and relative extrinsic parameters.

We used MATLAB's Single Camera Calibrator app due to its ease of use, high accuracy, graphical refinement tools, and advanced fisheye distortion correction capabilities. The app works by analyzing multiple images of a checkerboard pattern captured from different angles using the target camera and estimates intrinsic parameters such as focal length, principal point, skew coefficient, radial distortion coefficients, and tangential distortion coefficients. Users can visualize reprojection errors graphically (Figure 3.8) within the app to iteratively refine calibration accuracy. For cameras with fisheye distortions, the app provides specific options to model wide-angle distortions effectively. Once calibrated, MATLAB outputs a detailed report of intrinsic parameters that can be directly integrated into downstream processing pipelines. We then performed intrinsic calibration (Figure 2.6b) of the Flir BlackflyS camera and extrinsic calibration to align the Flir camera with the Livox LiDAR for our prototype scanner. For intrinsic calibration, we used the ROS camera calibration package with a standard 64 mm square 7x10 checkerboard pattern. This package employs OpenCV's calibration method, which is based on Zhang's technique [86].

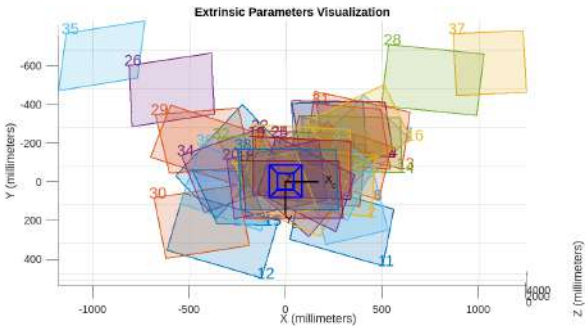
For the extrinsic calibration between the LiDAR and the camera, we follow the method outlined in [133]. The algorithm extracts edge features from both the LiDAR point cloud and the camera image, using depth-continuous edges for LiDAR and intensity edges for the camera. A cost function is formulated to quantify the alignment between these features. Through an iterative optimization process, the algorithm minimizes this cost, refining the rotation and translation parameters to achieve precise pixel-level alignment without requiring calibration targets (Figure 3.9a).



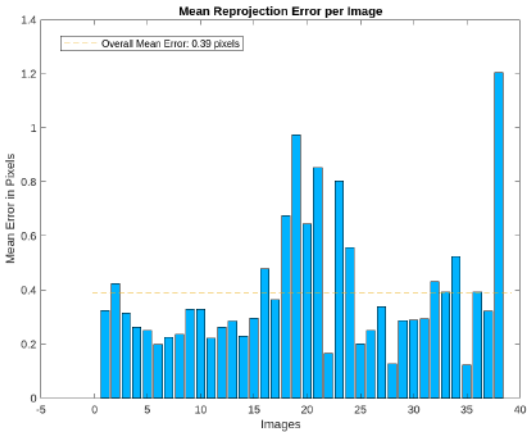
(a) iPhone 12 extrinsics vizualization



(b) iPhone 12 reprojection errors

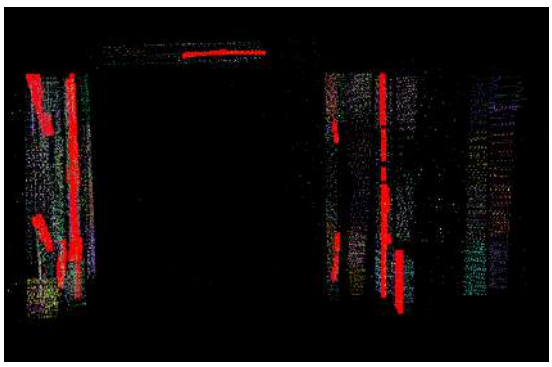


(c) Pixel 7 extrinsics vizualization

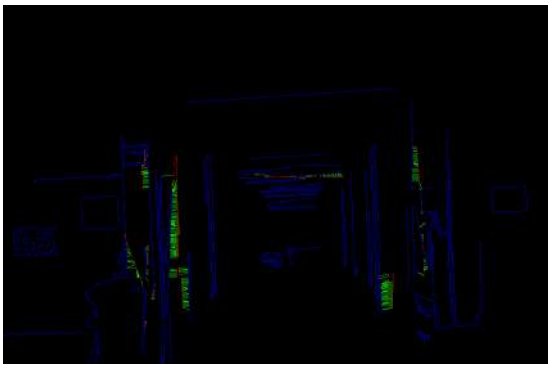


(d) Pixel 7 reprojection errors

Figure 3.8: Camera calibration results from MATLAB camera calibration app.



(a) 3D patch detection and fitting

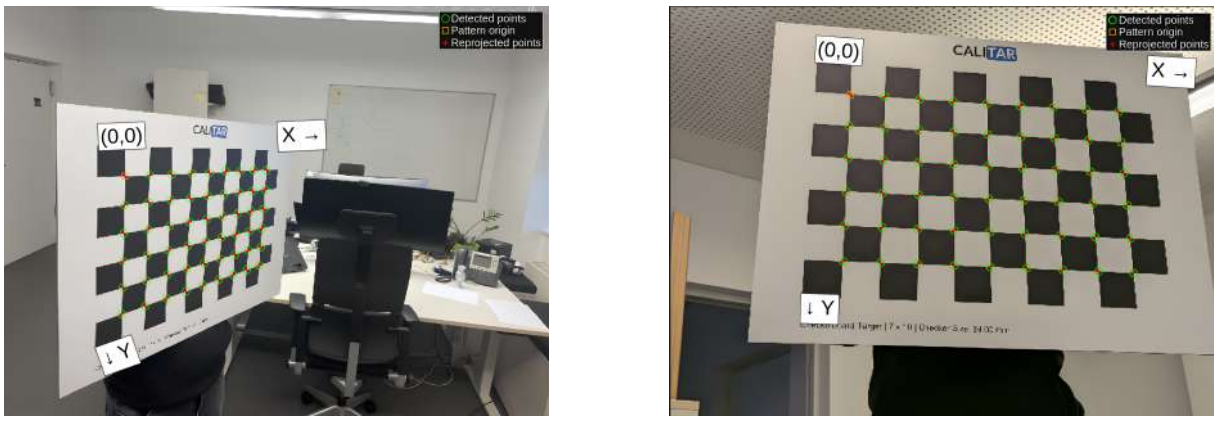


(b) Edge detection (blue) and residuals (green)

Figure 3.9: Extrinsic calibration procedure for LiDAR and Camera.

3.3 Data Collection & Raw Data Extraction

In order to obtain usable RGB-D data across our different sensor modalities, the data had to undergo various processing steps. Each device had its own data processing pipeline that dealt with device-specific issues such as sparse depth, low-resolution depth, and depth reprojection from point clouds. We will now go through each data pipeline individually and examine the



(a) Calibration for iPhone12 (b) Calibration for Pixel7

Figure 3.10: Camera intrinsic calibration with checkerboard.

collected data along with details about the quantity of collected data, image sizes, and formats.

3.3.1 NavVis Data

We utilized the NavVis scanner to scan a large warehouse in Duisburg consisting of four halls loaded with cargo containers. The NavVis scanner captures panoramic streams of images while fusing LiDAR data using GPS and SLAM-based methods to create a 3D map of the environment. The dataset generated from this process includes 2,108 RGB-D pairs, sampled across four different halls (Section 3.2).

Due to data-privacy measures implemented when collecting data in public spaces, the NavVis scanner automatically removes humans from the RGB images by inpainting them with the background. As a result, depth maps projected from scenes containing humans result in "ghost artifacts" (Figure 3.11), where the depth map retains information about human presence while the corresponding RGB image does not. Similarly, point clouds generated during SLAM-based mapping occasionally captured moving objects such as forklifts and humans but removed their presence during loop closure, creating similar artifacts in the depth maps.

The raw data collected using the NavVis scanner required extensive processing due to its complex camera setup and proprietary nature. The scanner uses four fisheye cameras and two LiDARs to capture a panoramic stream of images during the scanning process (Section 3.1.1). The output includes omnidirectional images, which need to be projected into a perspective pinhole camera model for compatibility with standard computer vision workflows. This transformation was achieved by using calibration code based on the omnidirectional toolbox [93, 91], along with an experimentally determined scaling factor that effectively zooms into the image to form perspective pinhole camera images with pinhole intrinsics. A scaling factor of 2 was chosen as it produced images with a good balance between perspective and distortion, zooming in on the image slightly.

To undistort fisheye images, a lookup table (LUT) is generated using Ocam calibration param-

eters. Each pixel in the pinhole image is back-projected into world coordinates using a unit spherical model and then remapped into the fisheye image using an inverse polynomial model to determine source pixels for each pinhole pixel. OpenCV remapping functions apply this LUT to generate undistorted perspective images, which are subsequently used to create depth maps by projecting LiDAR points onto them. A pipeline diagram illustrating these steps is shown in (Figure 3.12).

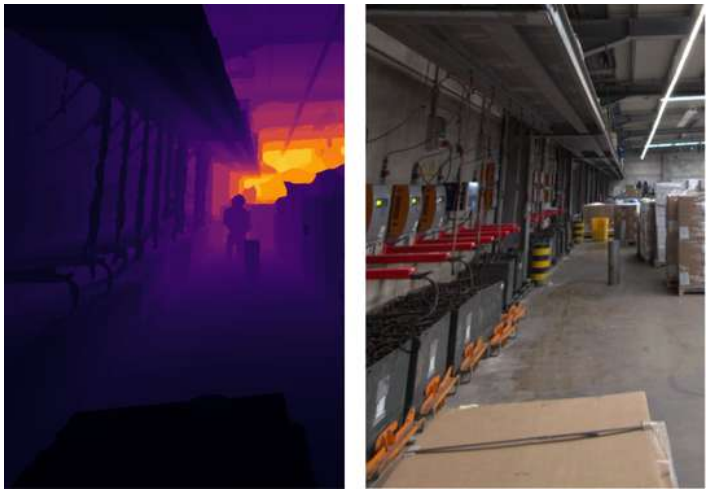


Figure 3.11: A ghost artifact formed by projecting a NavVis pointcloud.

3.3.2 ARKit Data

We used third-party iOS apps to capture images and depth maps using on-device LiDAR. The raw data was extracted directly from these apps in the form of zip files. Each captured scene included images, depth maps, camera parameters, a blur score, and relative position parameters provided by ARKit. The depth maps captured were, however, low resolution and needed to be upsampled in order to be of use [130]. For this, we developed a Joint Bilateral Upsampling algorithm that uses the high-resolution RGB image as a guide and upscales the low resolution 192 x 256 depth map to a resolution of 1024 x 768. We noticed no loss in quality from the original depth maps[9], comparing the upscaled depth map to the originals using common depth metrics such as mean absolute error (MAE) and root mean squared error (RMSE). Although some of the depth maps had sharpness issues around object edges, which is a common issue encountered[10] with such techniques [128].

Data was captured at Fraunhofer IML using the iPad Pro (2020), where an empty shipping container was systematically loaded with boxes in various configurations and scanned in a controlled setting. The resulting dataset, called the “ten-container-dataset, contains 787 RGB-D images capturing ten distinct container arrangements. Additional scans were conducted at lo-

[9] See appendix A

[10] Acknowledging this, we only use ARKit data for evaluation purposes and limit it’s contribution in our training data after careful post-processing.

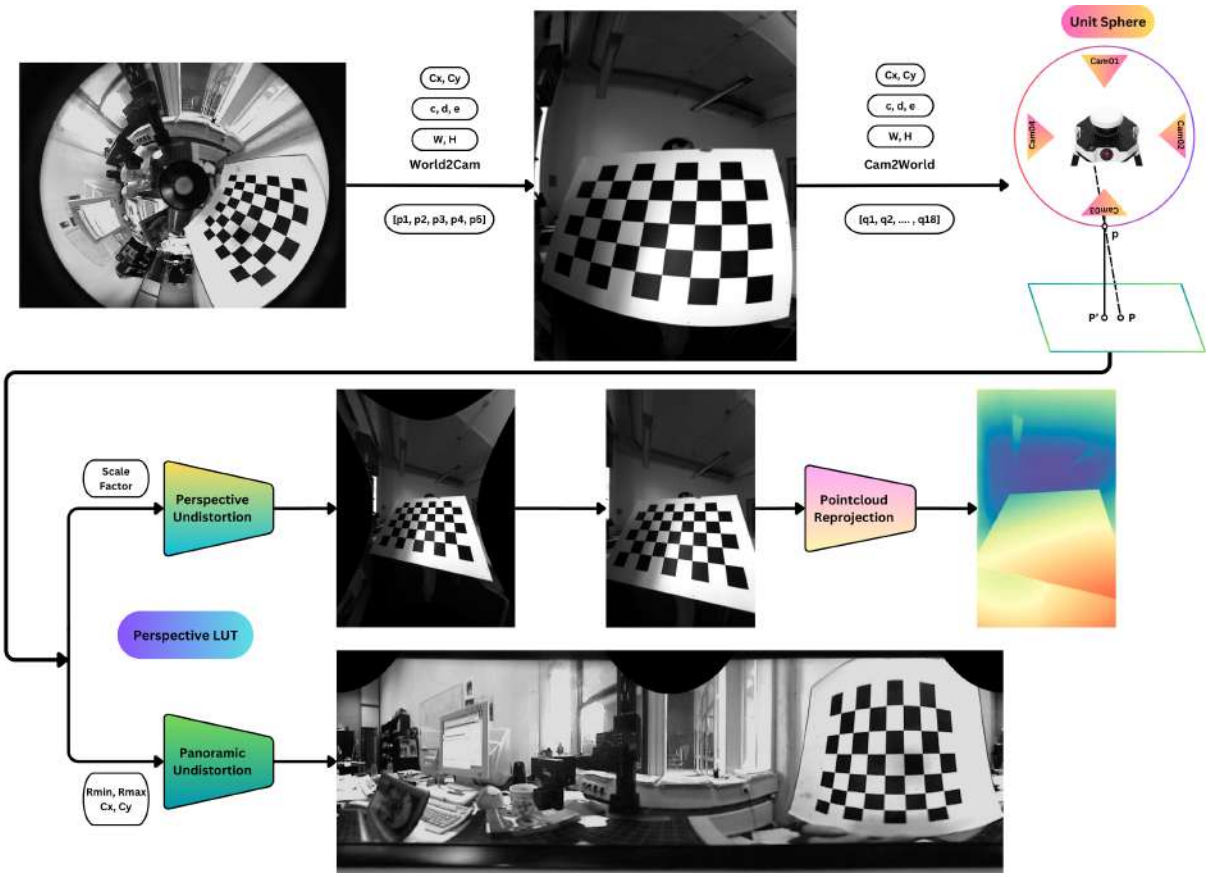


Figure 3.12: Pipeline for generating perspective images from the Omnidirectional camera model (images from [91]).

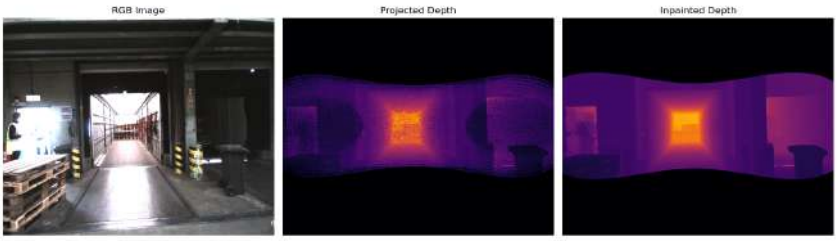
gistics warehouses in Duisburg and Dortmund using ARKit-enabled devices. In Duisburg, we utilized an iPhone 13 Pro with Polycam to collect 2,615 RGB-D pairs across various scenes. In Dortmund, we employed the iPad again with the 3DScannerApp to capture 700 RGB-D pairs from a warehouse environment. In total, during two separate visits to logistics warehouses in Duisburg and Dortmund, we scanned a total of 72 containers using both the iPad Pro and iPhone 13 Pro that altogether yielded a comprehensive dataset containing a total of 4,102 RGB-D images collected using ARKit-enabled devices. This dataset provided a diverse and flexible way for evaluating various monocular depth estimation models in real-world environments.

3.3.3 Flir-Livox Data

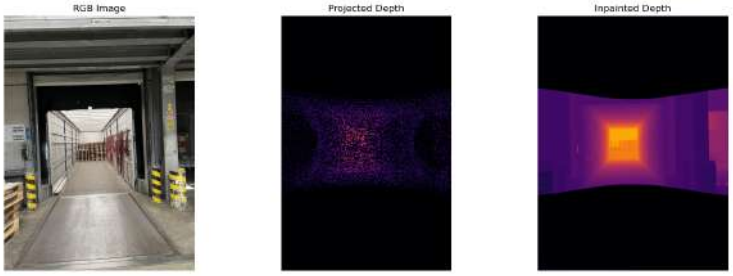
Using our handheld Flir-Livox scanning setup, data collection was efficient and streamlined. The LiDAR and camera were triggered using a launch file in ROS, which initiated synchronized streams of LiDAR point clouds and RGB images. These were saved as ROS Bag files on the device. Each recording lasted for a fixed time interval of approximately five seconds, capturing dense point clouds along with the corresponding RGB frames. The collected streams were then merged into dense point clouds representing the scanned environment.

We scanned 18 distinct container scenes using the Flir-Livox scanner. Combined with repro-

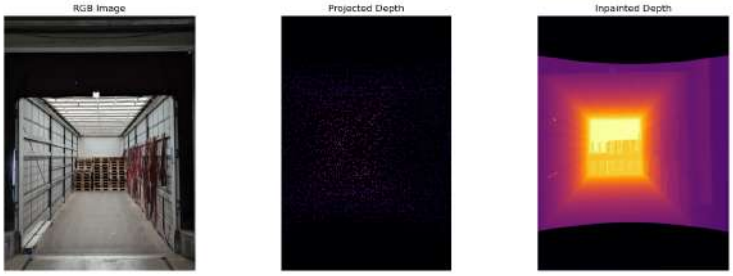
jected depth maps from RGB devices (Flir camera, iPhone 12, Pixel 7), this process yielded a total of 55 ground truth RGB-D pairs, including one extra pair acquired by re-scanning a container under an additional light source to test the performance of the Flir camera[11].



(a) Flir BlackflyS depth projection



(b) iPhone12 depth projection



(c) Pixel7 Depth Projection

Figure 3.13: Depth projection and inpainting results for Flir-Livox data.

3.4 Data Processing Pipelines

Processing raw data is a critical step in deep learning pipelines for robotics [134]. Proper processing ensures consistent formatting and quality across multi-modal sensor data and enables accurate downstream tasks. High-quality annotated data plays an important role in training and fine-tuning modern deep learning models for monocular depth estimation [43, 76, 75]. Although synthetic datasets have made significant strides in generating large-scale training examples for vision tasks such as segmentation, challenges remain when working with 3D data for tasks such as depth estimation and 3D reconstruction due to the persistent "sim-to-real" gap [135–137]. This gap often results from differences between simulated environments and real-world conditions, necessitating high-quality real-world datasets for fine-tuning models to achieve robust

[11] This additional scan resulted in one extra RGB-D pair being added to our dataset

performance across diverse scenarios [15].
In this section, we outline the steps required to process the data obtained through our various sensor modalities. We present three data pipelines; the first deals with processing raw data obtained using the NavVis VLX3 scanner. The raw data is first aligned to a global coordinate system and then reprojected to depth maps using the pinhole camera model. The second data pipeline addresses the challenge of upsampling low-resolution depth maps obtained by LiDAR-supported Apple devices as these consumer devices often produce incomplete depth information due to their limited range and resolution. The iOS applications used to collect data only offer up to 16-bit float values for depth, further degrading quality. In order to address these shortcomings, we upscale and inpaint the depth maps by utilizing Joint Bilateral Filtering (JBF) and Joint Bilateral Upsampling (JBU) algorithms[12] outlined in [138–140].
Finally, our third data pipeline outlines the methodology used to project, align, and enhance data collected using our prototype Flir-Livox scanner (Section 3.3.3).

3.4.1 Joint Bilateral Inpainting for NavVis Data

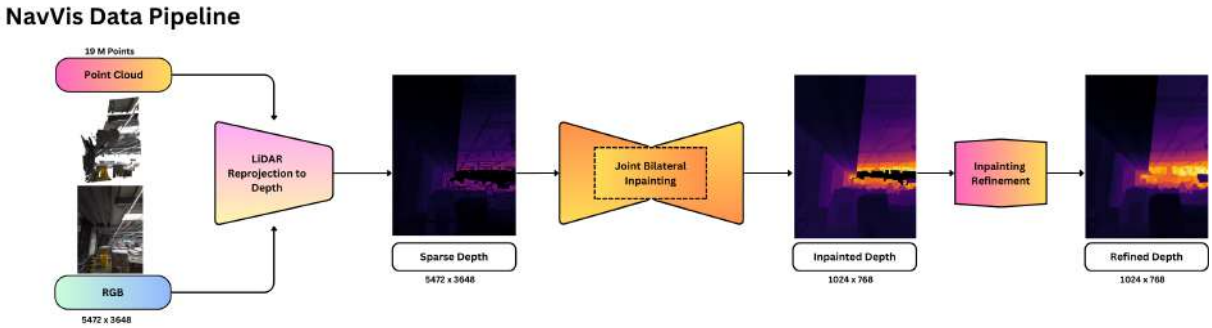


Figure 3.14: Data pipline from the NavVis scanner.

The NavVis scanner generates high-quality point cloud and image data by scanning approximately 2.56 million points per second and capturing panoramic images with an omnidirectional camera system utilizing four fisheye cameras. The pinhole images obtained from fisheye cameras retain a resolution of 5472 x 3648 pixels and are used to project LiDAR point clouds using a Z-buffering algorithm. The algorithm compares the depth of each new pixel to be projected with the depth values already stored in a buffer. If a pixel is closer to the viewer (with a smaller depth value), it replaces the existing pixel in both the color and depth buffer. This process ensures that only the closest objects to the viewer are visible, effectively solving the hidden surface problem and generating geometrically consistent depth maps [141].
Next, we inpaint the sparse depth map obtained through LiDAR projecting lidar points using a parallelized Joint Bilateral Filter that uses the high-resolution RGB image as a guide to inpaint

[12]While numerous deep learning based depth fusion/upsampling models exist in current literature, their effectiveness is often constrained by the need for re-training or fine-tuning on domain-specific datasets. Our proposed data pipeline, utilizing JBU/JBF, addresses this challenge by offering a computationally efficient and simple alternative.

the missing depth and ignores values greater than 90 m (max_depth = 90)[13]. The algorithm (Figure 3.14) is outlined below:

The algorithm utilizes a bilateral filter, which combines spatial proximity and photometric sim-

Algorithm 1 Joint Bilateral Filtering and inpainting.

Require: Sparse depth map D, RGB guide image I, mask M, parameters σ_s, σ_r, `max_depth`
Ensure: Dense depth map $\hat{D}$

Initialize $\hat{D} \leftarrow D$
for all pixels $p \in D$ **in parallel do**
  if $M(p) = 0$ **or** $D(p) >$ `max_depth` **then**
    $\hat{D}(p) \leftarrow 0$ ▷ Skip invalid/missing depth values
  else
    for all neighbors $q \in W$ (window around p) **do**
      Compute bilateral weight:
      $w(p,q) \leftarrow \exp\left(-\frac{\|p-q\|^2}{2\sigma_s^2}\right) \cdot \exp\left(-\frac{\|I(p)-I(q)\|^2}{2\sigma_r^2}\right)$
    end for
    Update depth value:
    $\hat{D}(p) \leftarrow \frac{\sum_{q\in W} w(p,q)\cdot D(q)}{\sum_{q\in W} w(p,q)}$
  end if
end for
return $\hat{D}$

ilarity to fill in the missing depth values in the sparse depth map. Spatial weighting ensures that only nearby pixels contribute significantly to the inpainting, while photometric similarity ensures the contributions come primarily from pixels belonging to the same object or surface. This dual weighting strategy preserves edges and prevents over-smoothing across depth boundaries, maintaining the geometric consistency of the objects in the scene [139] [140]. Invalid or extreme depth values are excluded using a binary mask and max_depth threshold, ensuring robustness against depth artifacts caused by incorrect projection. Parallelizing the algorithm using Joblib further enhances its efficiency, making it an excellent choice for processing large datasets on a single workstation[14]. The algorithm produces dense depth maps of size 1024 x 768, which can then be utilized for downstream tasks (Figure 3.15).

[13]The max depth value was chosen experimentally to ensure that the maximum number of depth points are projected while avoiding artifacts.

[14]The mean performance of the parallelized algorithm is reported in appendix A.

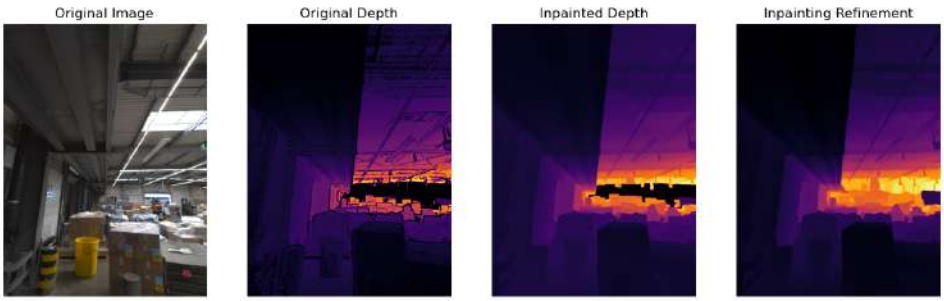


Figure 3.15: Steps in the NavVis data processing pipeline.

3.4.2 Joint Bilateral Upsampling for ARKit Data

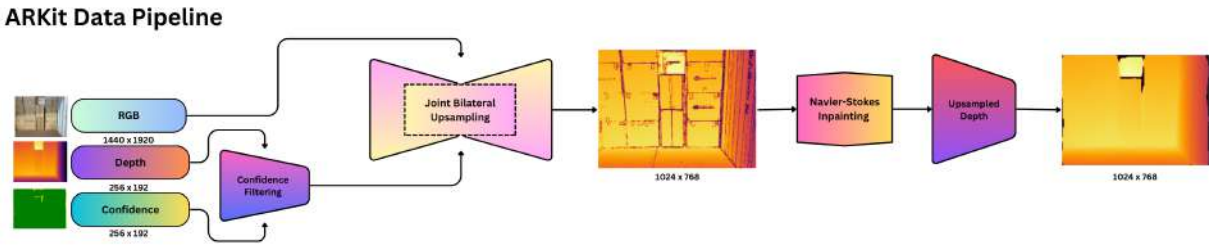


Figure 3.16: Upsampling pipeline for ARKit data.

In contrast to Joint Bilateral Filtering, Joint Bilateral Upsampling utilizes a high-resolution guidance image to steer the interpolation process. To enhance the resolution and quality of the low-resolution depth maps (256 x 192) obtained through our ARKit devices, we implemented a JBU method adapted from [138] using the corresponding high-resolution RGB image as inputs for guidance. The JBU algorithm takes a high-resolution RGB image and low-resolution depth map pair and uses spatial and range filters to enhance the depth resolution while preserving edges. The RGB image is resized to a resolution compatible with the upscaling factor, ensuring geometric consistency in the resulting upscaled depth maps.

We start the process (Figure 3.16) by filtering out the original low-resolution depth maps using confidence maps provided through ARKit and considering only medium and high confidences (Figure 3.4c, medium is yellow, high is green) for depth. We then use the resulting depth map along with the high-resolution RGB image to perform JBU. To manage computational time, we parallelize our JBU implementation using the Joblib library in Python, allowing each of the pixels in the high-resolution depth map to be processed independently across multiple CPU (Central Processing Unit) cores. This parallelization significantly reduces runtime from ≈ 78.0 seconds to ≈ 15 seconds, making the approach scalable for large amounts of data. Additionally, an optional refinement step of inpainting using the Navier-Stokes method [142] is performed (with a small radius, $r = 1$) to ensure that the depth map is geometrically complete. The pipeline

also ensures consistent formatting between depth and RGB data by resizing the RGB images to the target resolution of the depth maps using interpolation methods[15] in OpenCV, enabling color and geometric consistency with the upsampled depth map. The upsampled and confidence-refined depth maps and resized RGB images are saved in a high-resolution .jpg and .tiff format, making them suitable for downstream tasks such as 3D reconstruction for augmented reality applications, model training, mapping, or metric measurements. Figure 3.17 visualizes the

Algorithm 2 Parallel Joint Bilateral Upsampling

Require: High-Res color image C, Low-Res depth map D, upsampling factor f, spatial std σ_s, range std σ_r, window radius w

Ensure: High-Res depth map D'

Validate Input: Ensure C has 3 channels

Convert C and D to double precision

Get dimensions H, W from C

Initialize D' as an empty array of size (H, W)

for all pixels (i, j) in C **in parallel do**

Compute low-res coords: $i_d \leftarrow \frac{i}{f},\ j_d \leftarrow \frac{j}{f}$

Define window bounds:

$$i_{min} \leftarrow \max(\lceil i_d - w \rceil, 0),\ i_{max} \leftarrow \min(\lfloor i_d + w \rfloor, H/f - 1)$$

$$j_{min} \leftarrow \max(\lceil j_d - w \rceil, 0),\ j_{max} \leftarrow \min(\lfloor j_d + w \rfloor, W/f - 1)$$

Extract depth patch D_{patch} and color patch C_{patch} within bounds

Compute Gaussian range weights:

$$W_r \leftarrow \exp\left(-\frac{(C_{patch} - C[i,j])^2}{2\sigma_r^2}\right)$$

Compute Gaussian spatial weights:

$$W_s \leftarrow \exp\left(-\frac{\text{distance}^2}{2\sigma_s^2}\right)$$

Compute bilateral weights:

$$W_b \leftarrow (D_{patch} > 0) \cdot W_r \cdot W_s$$

Compute weighted depth value:

$$D'[i,j] \leftarrow \begin{cases} \frac{\sum W_b \cdot D_{patch}}{\sum W_b} & \text{if } \sum W_b > 0 \\ 0 & \text{otherwise} \end{cases}$$

end for

return D'

process, showing side-by-side comparisons of the original, upsampled, and inpainted depth

[15] To ensure the best results we use the methods $CV2.INTER\_LINEAR$ for upsampling and $CV2.INTER\_AREA$ for downsampling

maps. Although this approach showcases a robust and efficient method for enhancing ARKit depth data, it is important to acknowledge the limitations concerning depth accuracy as the LiDAR only supports a maximum measurement distance of 5 m. Any model trained on solely ARKit data may struggle to predict depth accurately beyond the range provided by ARKit.
This necessitated the need for a highly accurate data collection device to gather evaluation data for our application.

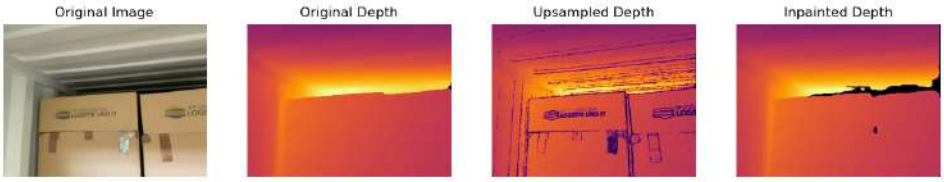


Figure 3.17: Steps in the upsampling process for ARKit data.

3.4.3 LiDAR Pointcloud Projection Flir-Livox Data

Our prototype Flir-Livox devices provided an easy way to acquire high-quality RGB-D data (Section 3.3.3). Depth maps were generated by projecting 3D points from the LiDAR onto RGB images captured by the Flir camera and mobile devices (iPhone 12 and Pixel 7) using calibration parameters and distortion coefficients obtained during calibration (Section 3.2). Depth values exceeding 40 m were discarded to maintain accuracy and to reduce points appearing from outside the container dock. To ensure alignment between the reprojected depth point clouds and ground truth LiDAR point clouds, we applied a Point-to-Point Iterative Closest Point (ICP) algorithm in Open3D that minimized alignment errors caused by sensor offsets or distortions. This step ensured that the point clouds could be compared with suitable point cloud metrics. The resulting depth maps and point clouds serve as ground truth for evaluating our monocular depth estimation models by comparing predicted depths against these LiDAR-acquired reconstructions of the 3D scene. The data processing required for the Flir-Livox device was minimal, as the Livox Horizon LiDAR captured enough points that, when projected to depth maps, only required simple inpainting using OpenCV's Navier-Stokes method [142] to generate dense depth maps. Additionally, the RGB images were processed with photometric enhancements to improve image quality[16].

[16]We performed a simple linear photometric adjustment consisting of contrast and brightness adjustments using OpenCV's $cv2.convertScaleAbs$ method.

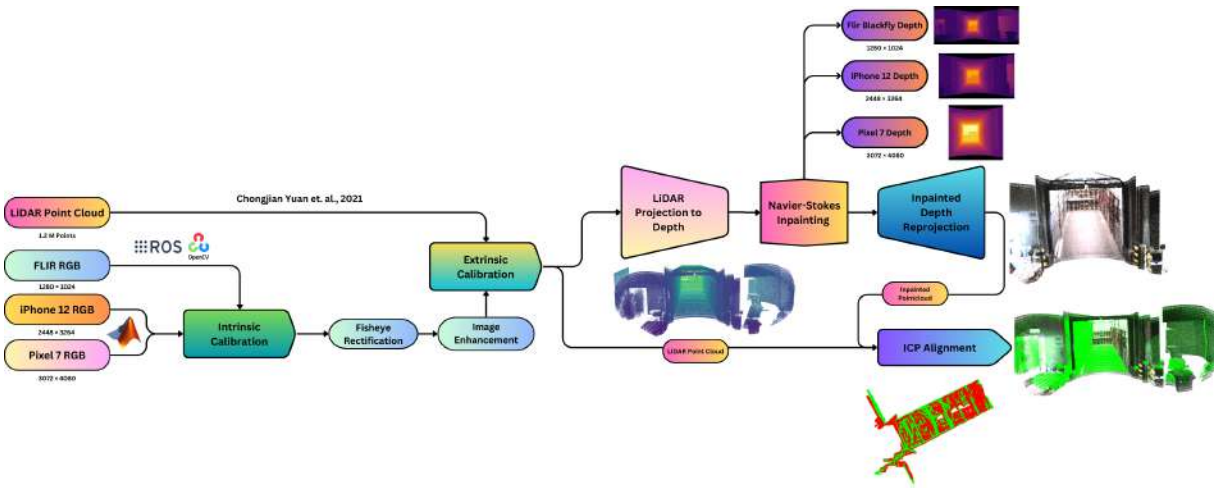


Figure 3.18: Data pipeline for the prototype Flir-Livox device.

3.4.4 Publicly Available Datasets

In order to curate a robust training dataset consisting of the depth ranges we require, we also utilized multiple publicly available metric RGB-D datasets commonly used for training depth estimation models consisting of indoor and outdoor scenes. We performed valid depth masking using the provided confidence masks and filtered the depths for depth ranges between 8 m and 200 m.

We utilize PhoneDepth, [143] which is designed for monocular depth estimation on mobile devices and includes 6,035 image sets (5,035 for training, 1,000 for validation) captured in diverse indoor and outdoor scenes. Data was collected using a rig with two smartphones (Huawei P40 Pro and Google Pixel 2) capturing RGB images and coarse depth maps, alongside a ZED stereo camera providing high-resolution depth maps. The reprojected depth maps cover a depth range of 0.2–10 m with an average depth estimation error of less than 0.2 m for objects within 8 m. These reprojected depth maps were used as ground truth for benchmarking and fine-tuning our model.

We also include other public datasets for our training mix such as DIODE (Depth in the Wild) [134] consisting of indoor and outdoor scenes, InSpaceType [144] which consists of indoor scenes categorized by the "space type" which includes multiple large spaces such as lounges, hallways, and libraries consisting of depths up to 20 m and selected scenes from DIML (Digital Image Media Laboratory) [145] which consist of urban outdoor scenes taken from a custom stereo camera with depths up to 80 m and indoor scenes of large spaces such as storage rooms which better resemble our logistic warehouse data. A detailed table of the utilized public datasets is given below (Table 3.2).

3.5 Data Augmentations

Data augmentation is an important step in generating data for training deep learning models, especially for depth estimation tasks. By artificially increasing the diversity of the training data,

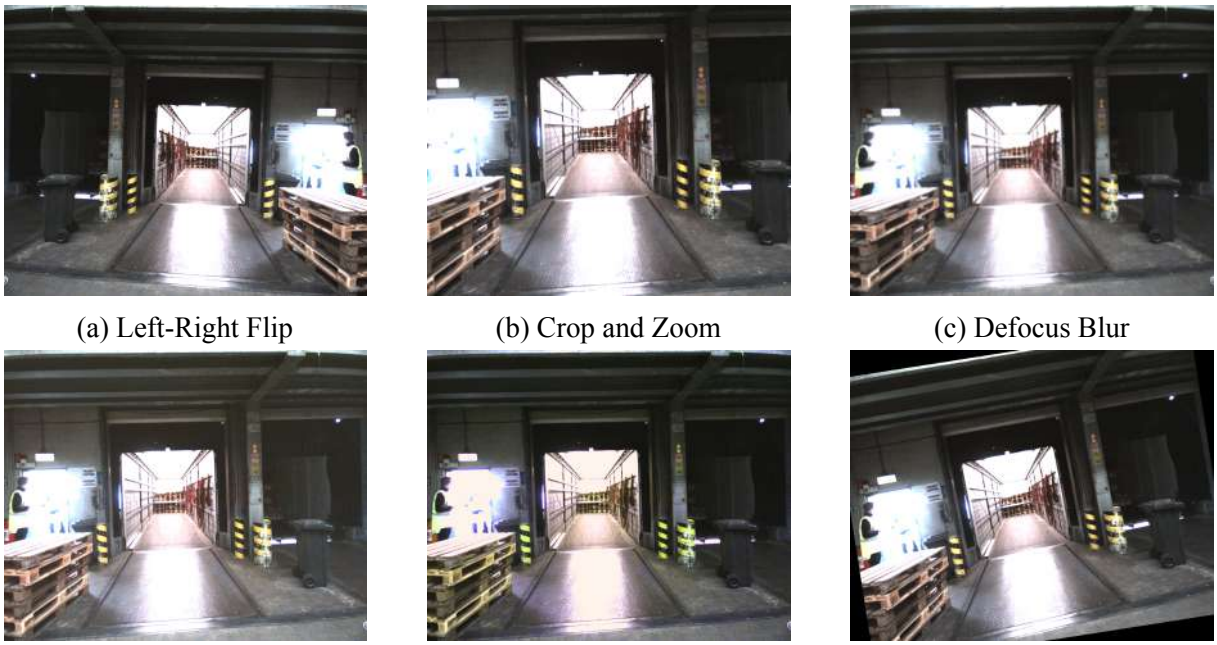
we ensure that the models generalize better and handle real-world variability. This is particularly significant in our application of monocular depth estimation for 3D reconstruction, where models must be robust to changes in lighting, color, noise, and geometric variations in the scene. As emphasized in [146], the role of augmentation in geometric tasks like depth estimation has traditionally been limited due to challenges such as reconstruction artifacts and consistency issues during training.

Monocular depth estimation models often rely on large and diverse datasets to avoid overfitting to a specific scene configuration, camera parameters, or lighting conditions. Our data augmentation module written in Python using the Albumentations library facilitates the augmentation and simulation of real-world errors such as variations in illumination, tilt or skew in photo taking, and perspective distortions. Our RGB-D augmentations also mitigate the challenge of our limited domain-specific data by artificially increasing their effective size and variability, ensuring that the models trained on the datasets will have better zero-shot transfer capabilities, as demonstrated in various benchmarks [134, 108, 43].

For depth estimation, augmentations must be designed carefully to preserve geometric consistency between the RGB images and their corresponding depth maps. Key considerations include ensuring that the transformations do not introduce geometric changes in depth or distort the effective focal length of the images. For generating the training dataset, we augment our data while maintaining alignment between RGB and depth data for geometric augmentations; the photometric augmentations are only applied to the RGB images. A summary (Table 3.1) of the augmentations we use and their effect on the images is shown below (Figure 3.19). Although it was possible to increase the quantity and variability of our collected datasets indefinitely for model training, practical constraints such as storage capacity and computational infrastructure necessitated a more pragmatic approach. We capped our total training dataset to roughly 70,000 RGB-D images (Table 3.2) and prioritized basic yet impactful augmentations. Photometric techniques, including random brightness, contrast, and gamma adjustments, were applied to replicate common artifacts like overexposure. Horizontal flipping ensured invariance for left-right depth maps, while random cropping and zooming into high-resolution depth maps obtained through NavVis simulated data captured from various mobile camera sensors.

Augmentation	**Type**	**Description**
Left-Right Flip	Geometric	Flip the data horizontally
Crop and Zoom	Geometric	Zoom and crop into the data simultaneously
Defocus Blur	Photometric	Blurs image to simulate shallow depth of field
Random Photometric	Photometric	Random brightness, contrast, and gamma adjustments
Color Jitter	Photometric	Random hue and saturation adjustments
Random Skew	Geometric	Rotate RGB-D images slightly ($\pm 15\,\mathrm{deg}$)

Table 3.1: Table of data augmentations used



(a) Left-Right Flip (b) Crop and Zoom (c) Defocus Blur

(d) Photometric Distortions (e) Color Jitter (f) Random Skew

Figure 3.19: Visualizations of utilized data augmentations for RGB-D data.

Dataset	Qty. RGB-D	Train	Test	Val
ARKit - Duisburg Warehouse	2615	2484	131	-
ARKit - Dortmund Warehouse	700	-	-	700
ARKit - Fraunhofer IML	787	-	-	787
NavVis Hall 1-4	2108	2003	105	-
NavVis Augmented	16864	16021	843	-
Flir-Livox Scan	55	-	-	55
DIODE Indoor	8899	8454	445	-
DIODE Outdoor	10603	10073	530	-
InSpaceType	12715	12080	635	-
DIML	5879	5585	294	-
PhoneDepth	11070	10516	554	-

Table 3.2: Overview of curated metric RGB-D data across sources.

4 3D Reconstruction from Monocular Depth Estimation

This chapter details the methodology employed for monocular depth estimation, focusing on its application to point cloud generation and mesh reconstruction of cargo containers. To estimate mesh volume, the proposed pipeline integrates single-view 3D reconstruction with surface reconstruction and volume estimation. A comprehensive evaluation using zero-shot benchmarking is presented to assess the performance of various models across diverse architectures and pre-trained checkpoints. These evaluations tested multiple depth and point cloud metrics to identify the most suitable model for our specific application. Furthermore, we explore the importance of camera parameters in enabling scale-aware reconstructions, followed by an in-depth analysis of the architecture and training methodology of the selected model.
We introduce a gate detection framework designed to effectively segment cargo containers from surrounding noise to address challenges associated with cluttered warehouse environments. This segmentation step ensures precise container reconstruction for accurate volume estimation. In addition, we also delve into the surface reconstruction problem, detailing post-processing techniques applied to refine the point clouds into watertight meshes suitable for downstream tasks. Lastly, we discuss the method used for calculating container volumes using the divergence theorem, providing an overview of the background of this essential component within our pipeline.

4.1 Zero-Shot Benchmarking

Zero-shot benchmarking involves evaluating a model on a task without any task-specific fine-tuning. In other words, the model is tested directly on domain-specific data, such as images of logistical container scenes, without retraining or making additional adjustments to adapt it to the target domain. This paradigm has gained prominence in recent research because it assesses a foundation model's ability to generalize and perform well on unseen or out-of-domain data that differs from its pre-training dataset. Zero-shot benchmarking also evaluates the efficiency of a model's inductive bias, which refers to its inherent assumptions about the data. A deep learning model's inductive biases help determine whether it is learning appropriate patterns for the task, enhancing its ability to generalize effectively.
Zero-shot benchmarking is particularly important for large pre-trained models in real-world scenarios, where retraining for every new task or domain is often impractical due to time, resources, and available data constraints [43, 147]. Zero-shot benchmarking was critical in evaluating the performance of state-of-the-art monocular depth estimation models on logistics data, enabling an objective comparison using selected metrics to identify suitable models for our application. Accurate depth estimation is essential, as errors propagate through the pipeline and negatively impact downstream tasks such as volume estimation. The primary goal of this benchmarking process was to assess how effectively various models predict the depth of cargo containers under challenging conditions. These challenges include occluded cargo, container depths up to 14 m from the camera, and poorly lit back regions. By evaluating multiple models on domain-specific

data, we aim to identify those capable of accurate metric depth prediction in out-of-distribution scenes while demonstrating robustness against clutter and noise typical of busy warehouse environments.

As our 3D reconstruction pipeline involves imaging and capturing a detailed representation of the cargo container interior and cargo, we first perform a preliminary benchmark in which we evaluate promising metric monocular depth estimation models from current literature using container scenes extracted from the larger dataset collected by the high precision NavVis VLX3 scanner and our prototype Flir-Livox device. We obtained 58 RGB-D container scenes from the NavVis scanner and 55 from the Flir-Livox device. Due to the lower fidelity of the Flir-Livox prototype, we use the data obtained only for preliminary benchmarks and to study the effect of image quality on depth predictions. We then prioritize the best-performing models with inherent robustness to unseen data for further fine-tuning.

4.1.1 Preliminary Benchmarking on ARKit Data

Our preliminary benchmark evaluates different models on our "ten-container-dataset" (ARKit Fraunhofer IML in Table3.2) that we collected using the iPad Pro, which contained 737 RGB-D images 3.3. The predicted depth maps are evaluated using four different metrics for depth on all images, and their mean values are logged using a benchmarking module in Python, which compares predicted and ground truth depth maps for each image and computes the metrics after masking valid depths 3.4. Additionally, inference time for each image is also noted for each model, as it is an important factor for the real-time application of our algorithm[17]. To narrow

Model Name	Encoder	Inference Time (s)	MAE	RMSE	ARE	δ1	License
ZeroDepth	ResNet18	0.8353	0.9790	1.0698	0.6794	0.3292	CC-BY-NC-4.0
UniDepthv1	ConvNext	**0.2185**	**0.4200**	**0.5049**	**0.2191**	**0.6132**	CC-BY-NC-4.0
UniDepthv2	ViTl	0.5493	**0.4042**	**0.4927**	**0.1944**	**0.6537**	CC-BY-NC-4.0
PackNet	PackNet	**0.0438**	0.9488	1.1667	0.5258	0.2643	CC-BY-NC-4.0
Metric3D	ViTs	**0.2841**	0.5312	**0.5930**	0.3103	0.5105	BSD-2
Metric3D	ViTl	**0.4180**	**0.5230**	0.7258	**0.2806**	**0.6464**	BSD-2
Metric3D	ViTG	1.0770	0.6535	0.8926	0.3667	0.5418	BSD-2
ZoeDepth	BEiT_{384-L}	0.5302	0.6109	0.6834	0.3618	0.4528	MIT
DepthAnythingv2	ViTs	**0.3152**	0.5431	**0.6272**	0.3015	0.5014	Apache-2.0
DepthAnythingv2	ViTb	1.0140	**0.4450**	**0.5339**	**0.2610**	**0.6255**	CC-BY-NC-4.0
DepthAnythingv2	ViTl	3.8914	**0.4980**	0.6654	**0.2893**	**0.6433**	CC-BY-NC-4.0
DepthPro	Dual ViT	2.2904	1.4773	2.0134	0.9326	0.5068	Copyright Apple Inc

Table 4.1: Metrics computed from benchmarks on different monocular depth estimation models.

down the search space of suitable models, we perform all preliminary benchmarking on the worst-case logistics data acquired by the Flir camera (Figure 3.7a), which contains lighting and sharpness issues and has a wide focal length causing the container and its content to look small and dark (See Section 3.1 for limitations). The preliminary benchmark (Table 4.1) tests various

[17] Evaluations were performed on an Nvidia RTX A2000 GPU with 5 Gigabytes of Video Random Access Memory (VRAM)

state-of-the-art models from current literature and their ability to generalize to the Flir-Livox data. Promising candidates for further study are highlighted in bold (Table 4.1).
The benchmark reveals that models utilizing ViT encoders outperform older ResNet-based architectures, CNNs [42], and BEiT [106] across most metrics. Notably, inference times also scale proportionally with encoder size. Models such as PackNet [68] and UniDepth [76] [104], utilizing convolutional backbones, demonstrate excellent real-time performance with low inference times at the cost of slightly reduced accuracy across depth estimation metrics. Interestingly, smaller ViTs (ViT-Small) encoder architectures exhibit competitive trade-offs between performance and efficiency. The ViTG (ViT-Giant) encoder utilized by Metric3D [74] surprisingly underperforms its ViTl (ViT-Large) and ViTb (ViT-Base) counterparts, suggesting possible overfitting. This observation aligns with prior findings that ViTs lack inductive biases inherent to CNNs and require large amounts of data to generalize; scaling up encoder size without scaling up the size of the training dataset causes instabilities in performance [50, 148]. In order to choose a model for further fine-tuning, we take into account the following considerations:

- Reasonable inference time ($\leq$ 1 second)
- Reasonable performance across depth metrics
- The availability of training code and methodology used
- Open-source license, non-restrictive for commercial use

Our selection process, though rigorous, led to the elimination of several strong candidates despite their promising performance. DepthPro [75], for instance, achieved excellent results across metrics but was excluded due to its lack of publicly available training code and restrictive licensing terms unsuitable for commercial use. Similarly, UniDepth [76] provided well-documented training resources and competitive metrics but fell short due to its non-commercial license restrictions. Metric3D [74] met most of our evaluation criteria; however, its reliance on extensive amounts of training data rendered it less suitable for fine-tuning within our constrained setup.
Ultimately, we chose to fine-tune DepthAnythingv2 [15] due to its reproducible training strategy and balanced trade-offs between accuracy and inference speed. Its open-source implementation[18] aligns with our project requirements for scalability and adaptability in real-world logistics applications.
While DepthAnythingv2 is the primary focus of our fine-tuning efforts, we continue to evaluate other models in subsequent benchmarking experiments. For this purpose, we retained UniDepth and DepthPro as additional baselines. Including more models in the evaluation was impractical for the scope of this project, as it would require significant time and is unlikely to provide additional meaningful insights on our application.

[18] Depth Anything V2 is built upon DINOv2 and DPT both of which are open-source.

4.1.2 Evaluating Model Performance on Flir-Livox Data

We first study the performance of UniDepth, DepthPro, and DepthAnything V2 (hypersim checkpoint) on our hardest evaluation dataset obtained from the Flir-Livox scans (Section 3.3.3) which contains images of container scenes obtained by the wide-angle Flir BlackflyS under low light conditions. The evaluation results (Table 4.2) reveal significant differences in performance across the models and encoder types. UniDepth with a ViTl encoder achieves the best performance across depth estimation metrics, followed closely by DepthPro and Depth Anything ViTs. Comparing the encoders of UniDepth and DepthAnything, we observe that the ViTs variant of Depth Anything outperforms that of UniDepth in depth and point cloud metrics. Meanwhile, the ViTl variant of UniDepth supersedes DepthAnything's metrics. This could be because we utilized the ViTl checkpoint from Depth Anything that was pre-trained only on Hypersim data, making it likely to overfit. In contrast, UniDepth ViTl is trained on roughly 3 million RGB-D samples across various datasets, including outdoor datasets collected for self-driving cars (e.g., Waymo). DepthPro, on the other hand, uses a custom Dual ViT encoder comprising a patch encoder and an image encoder that are jointly optimized during training, which achieves the best results overall for depth metrics and good results for point cloud metrics such as Hausdorff Distance[19]. This suggests that DepthPro reconstructs geometrically consistent depth with minimal outliers. However, the Density-Aware Chamfer Distance[20] for UniDepth ViTl and ViTs encoders performs the best, followed by Depth Anything ViTs.

Model Name	Encoder	MAE	ARE	RMSE	SSIM	Delta 1	Delta 2	Delta 3	Hausdorff Distance	CD-P	CD-T
DepthPro	Dual ViT	**2.1200**	**0.4032**	**2.6499**	**0.8833**	**0.3503**	**0.8057**	**0.9603**	**10.1444**	1.4776	7.2704
UniDepth	ViTs	2.7211	**0.4664**	3.8214	0.8695	**0.2428**	**0.7797**	0.9273	20.3074	**1.1020**	5.7497
UniDepth	ViTl	**1.5750**	**0.2956**	**2.1716**	**0.9076**	**0.5519**	**0.9267**	**0.9657**	**18.0723**	**0.9157**	**3.7475**
DepthAnything	ViTs	**2.4298**	0.4860	**2.7643**	**0.8723**	0.1885	0.7070	**0.9558**	19.53506	**1.1515**	**5.0511**
DepthAnything	ViTb	2.5243	0.5088	2.8350	0.8643	0.1593	0.6881	0.9527	19.0979	1.1653	**5.1148**
DepthAnything	ViTl	2.9065	0.5765	3.2505	0.8495	0.0968	0.5544	0.9455	**18.8397**	1.3177	6.06133

Table 4.2: Benchmark on Flir-Livox evaluation data.

4.2 Benchmarking Pretrained Checkpoints

In order to evaluate the performance of different pre-trained metric checkpoints released by Depth Anything V2, we measure the performances of the checkpoints trained on Hypersim (indoor dataset) and VKITTI (outdoor dataset) on our high-quality evaluation datasets obtained by the NavVis VLX 3 and Flir-Livox scanners. These evaluations aim to compare their generalization capabilities in diverse real-world settings and identify the most suitable checkpoint for further fine-tuning. The results highlight that the Hypersim pre-trained checkpoints significantly outperform those trained on VKITTI (Virtual KITTI) across all evaluation metrics for

[19]Hausdorff Distance measures the maximum distance between two point clouds meaning that a lower score means that its worst case pointwise error is minimal.

[20]A low CD-P indicates that majority of the predicted points are close to some part of the ground truth, reflecting good accuracy while low CD-T indicates that most parts of the ground truth have been reconstructed by predictions, reflecting good completeness.

both datasets. The difference in performance between checkpoints can be attributed to variations in dataset size and scene diversity during pretraining. The VKITTI dataset includes approximately 17,000 synthetic frames specifically created for multi-object tracking, segmentation, and optical flow tasks [149]. In contrast, the Hypersim dataset contains 77,400 high-quality synthetic frames of indoor scenes, featuring detailed per-pixel labels and corresponding ground truth geometry tailored for indoor scene understanding tasks [108].
For performance, on the Flir-Livox data[21], the ViTs variant pre-trained on both datasets outperforms the rest, suggesting that the ViTs variant does not overfit the limited training data provided, followed by ViTb and ViTl. This observation correlates with current literature on the effect of training ViT models on small datasets [150].
Similar strong trends can be observed for the VKITTI checkpoints on the NavVis data, where the ViTs encoder outperforms its counterparts. However, while similar, the performance of the Hypersim ViTs and ViTb checkpoints shows some arbitrary behavior. This suggests that neither model is strictly overfitting but has also not reached full convergence. Two other factors could be that the depth ranges in the Hypersim dataset do not closely resemble those of our container scenes, or there is a strong pre-trained bias on indoor scenes of living spaces, which may lead to arbitrary predictions on some scenes in our dataset.

Model Name	Encoder	MAE	ARE	RMSE	SSIM	Delta 1	Delta 2	Delta 3
VKITTI-DepthAnything	ViTs	**7.9610**	**1.5405**	**8.9165**	**0.6529**	**0.0193**	**0.0748**	**0.1793**
VKITTI-DepthAnything	ViTb	9.4905	1.8305	10.4586	0.6009	0.0055	0.0261	0.0777
VKITTI-DepthAnything	ViTl	12.189	2.3027	13.7270	0.5161	0.0023	0.0064	0.0184
Hypersim-DepthAnything	ViTs	**1.7338**	**0.3620**	**2.1300**	**0.8975**	**0.4491**	0.8966	**0.9632**
Hypersim-DepthAnything	ViTb	1.8652	0.4048	2.1975	0.8850	0.3308	**0.8985**	0.9615
Hypersim-DepthAnything	ViTl	2.3743	0.4990	2.7581	0.8616	0.2303	0.7416	0.9518

Table 4.3: Results for Flir-Livox data on Depthanythingv2 checkpoints.

Model Name	Encoder	MAE	ARE	RMSE	SSIM	Delta 1	Delta 2	Delta 3
VKITTI-DepthAnything	ViTs	**4.2434**	**1.2450**	**4.4789**	**0.7576**	**0.0896**	**0.2193**	**0.4567**
VKITTI-DepthAnything	ViTb	6.3394	1.8295	6.6147	0.6650	0.0073	0.0625	0.2379
VKITTI-DepthAnything	ViTl	7.9268	2.0693	8.6286	0.6074	0.0006	0.0110	0.0867
Hypersim-DepthAnything	ViTs	1.8323	**0.4610**	2.4987	0.9120	**0.3833**	0.7243	0.9362
Hypersim-DepthAnything	ViTb	**1.6666**	0.4640	**2.0860**	**0.9139**	0.3690	0.7461	0.9558
Hypersim-DepthAnything	ViTl	1.7693	0.4814	2.2268	0.9104	0.3148	**0.7503**	**0.9613**

Table 4.4: Results for NavVis data on Depthanythingv2 checkpoints.

4.3 Image Quality Performance

To test our hypothesis on the effect of image quality on the depth and point cloud metrics and compare the results obtained using the Flir camera from the previous tables, we conducted a

[21] All evaluations were performed on an Nvidia RTX 4070 GPU with 8 Gigabytes of VRAM

study using the VKITTI checkpoints for all available encoders on the dataset detailed in 3.3 using images obtained by two mobile devices (iPhone 12 & Pixel 7). The iPhone images were taken using the wide-angle primary lens, and the Pixel 7 images were acquired using its telephoto lens, effectively zooming in on the containers.
The results of the difference in image quality and zoom are apparent from the resulting data (Table 4.5). The Pixel 7 images result in significantly better performance across encoders. This performance difference can be attributed to several factors, including a higher effective resolution achieved by zooming on the container while cutting out background clutter (Figure 3.7. Moreover, telephoto lenses have a reduction in distortion when compared to wide-angle primary lenses. In contrast, the images taken from the iPhone tend to introduce additional noise and geometric distortions at greater distances, which likely detrimentally affect performance in monocular depth estimation.

Model Name	Encoder	Phone	MAE	ARE	RMSE	SSIM	Delta1	Delta2	Delta3
VKITTI-DepthAnything	ViTs	iPhone 12	11.1846	1.8487	12.5621	0.5860	0.0068	0.0355	0.0985
VKITTI-DepthAnything	ViTs	Pixel 7	**4.5533**	**0.6028**	**5.4498**	**0.8507**	**0.2080**	**0.5871**	**0.8384**
VKITTI-DepthAnything	ViTb	iPhone 12	13.0158	2.1253	14.6987	0.5373	0.0035	0.0127	0.0467
VKITTI-DepthAnything	ViTb	Pixel 7	**5.9940**	**0.7829**	**7.1665**	**0.8091**	**0.0833**	**0.3642**	**0.7387**
VKITTI-DepthAnything	ViTl	iPhone 12	15.0419	2.3742	17.7987	0.4949	0.0011	0.0079	0.0317
VKITTI-DepthAnything	ViTl	Pixel 7	**7.0525**	**0.8815**	**9.0421**	**0.7784**	**0.0823**	**0.3181**	**0.6769**

Table 4.5: Benchmark results for depth estimation on VKITTI dataset.

4.4 Scale Aware 3D Reconstruction

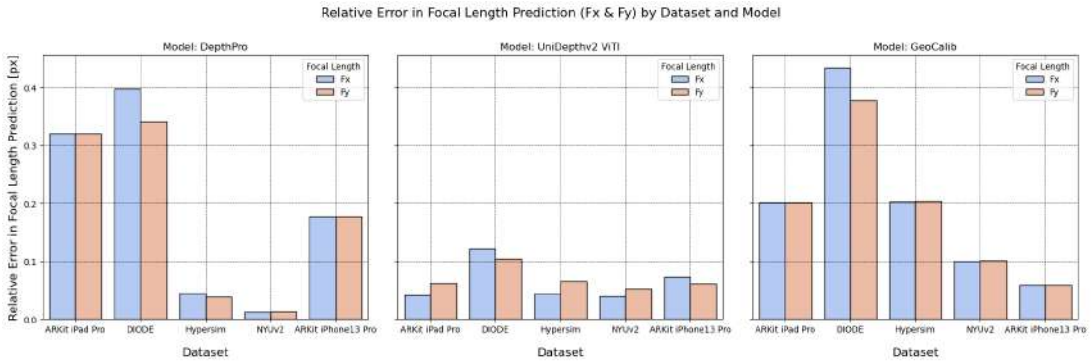
The significance of accurate camera parameters, particularly the focal length, in scale-aware 3D reconstruction, cannot be overstated. Camera parameters serve as an essential link in 3D scale-aware 3D reconstruction as they establish the geometric relationship between pixel measurements and metric dimensions in the real world. Without precise camera intrinsics, ambiguities, such as depth shift and focal length, propagate through the reconstruction pipeline during the conversion of depth maps into point clouds. These ambiguities lead to scale errors and misinterpretations of scene geometry (Figure 4.2).
Contemporary MDE models mitigate scale ambiguity by utilizing scale and shift invariant losses computed in logarithmic space (e.g., SiLog Loss) [42]. This invariance facilitates training by mitigating issues associated with the inherent ambiguity of single image depth estimation. However, while the scale ambiguity is less problematic, the shift ambiguity introduces distortions in the reconstructed 3D geometry. These distortions become particularly detrimental in recovering precise metric 3D objects and scene structure, especially when the camera intrinsics are unknown or inaccurately estimated at inference time, as the reprojection from depth to 3D space depends both on focal length (scene scale) and depth values [66].
In scale-aware 3D reconstructions, an unknown focal length merely enlarges or shrinks the reconstructed scene uniformly (Figure 4.2) while preserving the overall geometric structure. In

contrast, an unknown shift in estimated depth distorts the relative depth relationships, resulting in a non-uniform deformation of the 3D scene geometry. Modern MDE models have made progress in reducing the shift ambiguity through specialized loss functions that minimize error directly in 3D space [76] and by using other geometric constraints. However, the complete resolution of this ill-posed problem requires the use of external cues. Current methods utilize scale-invariant logarithmic losses that ignore absolute depth during training and emphasize learning relative depth gradients. As a result, the predicted depths remain ambiguous due to an unknown constant offset, and the model must learn to recover this unknown scale and shift offset during inference time for accurate 3D reconstruction [43, 15].

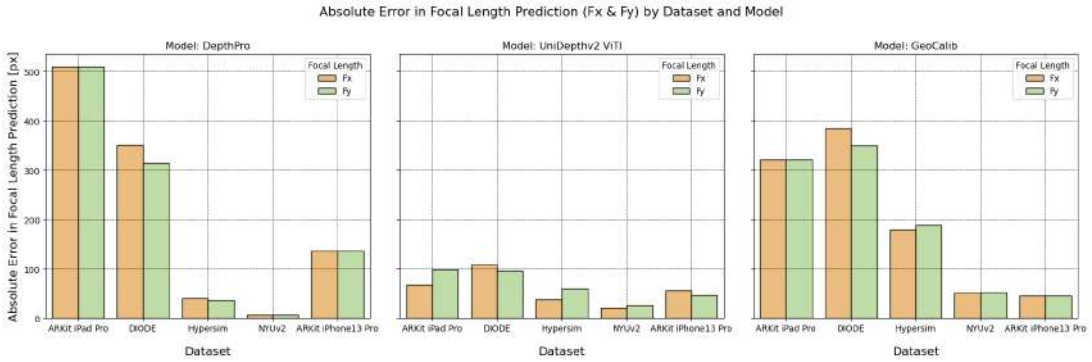
4.4.1 Study on Focal Length Estimators

One of the key factors in evaluating monocular depth estimation models is their ability to maintain accuracy across different camera parameters. Models trained on datasets with fixed intrinsic parameters may struggle when tested on images from new cameras, leading to significant drops in accuracy. To address this, we conducted a small study on the robustness of different focal length estimation methods employed by monocular depth estimation models for our application (Figure 4.1). Several promising approaches have been proposed to address these challenges. For instance, UniDepth [76] utilizes a self-promptable camera that estimates the focal length from a single image using a self-prompting camera module that learns a dense camera representation instead of relying on predefined intrinsics. DepthPro [75] tackles the focal length estimation problem by adding a dedicated convolutional focal length prediction head trained on a wide variety of images. This head predicts the horizontal field of view directly from the image. GeoCalib [151] estimates camera parameters such as focal length, lens distortion, and orientation, leveraging prior geometric cues like perspective fields using an end-to-end trained neural network.

To test the robustness of these methods, we compared their focal length predictions using absolute and relative errors on 700 images from our dataset and public datasets like NYUv2, Hypersim, and DIODE [152, 108, 134]. Furthermore, we benchmarked the models used in our evaluations with known camera intrinsics as input and without (the model estimates the focal length) and report the findings using point cloud and depth metrics.
Despite the significant advances made by these methods, it is important to acknowledge that they still exhibit shortcomings and limitations. We need to be confident in our reconstruction quality to reliably use reconstructed point clouds for downstream metric measurement tasks. Therefore, we use our pipeline with known camera intrinsics, as errors in 3D reconstruction can adversely affect the final volume measurements.
Our experimental results (Table 4.6) on the Flir-Livox dataset demonstrate the performance of the focal length estimators used in UniDepth and DepthPro. The results show that providing intrinsic parameters during 3D reconstruction significantly improves depth estimation and 3D



(a) Relative error in focal length prediction (Fx & Fy) across datasets and models.



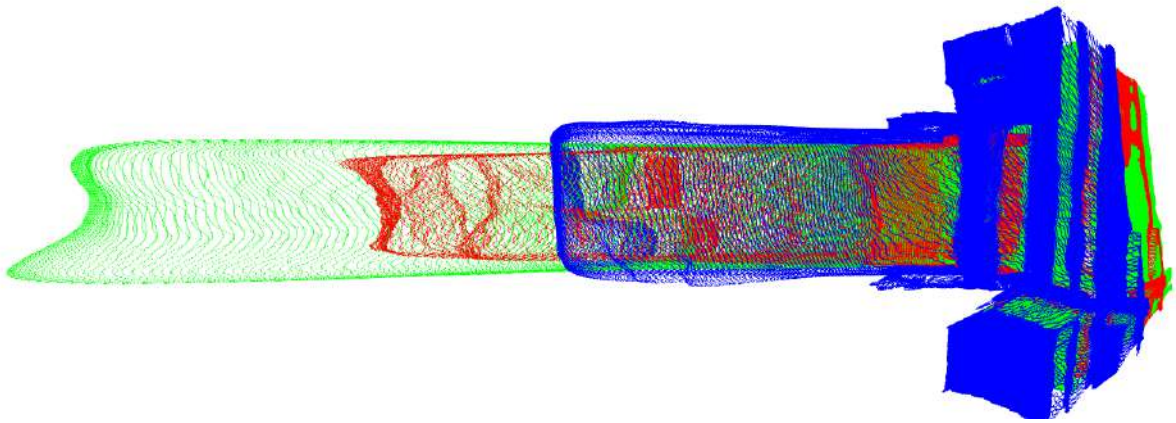
(b) Absolute error in focal length prediction (Fx & Fy) across datasets and models.

Figure 4.1: Comparison of focal length estimation errors for UniDepth, DepthPro & GeoCalib estimators.

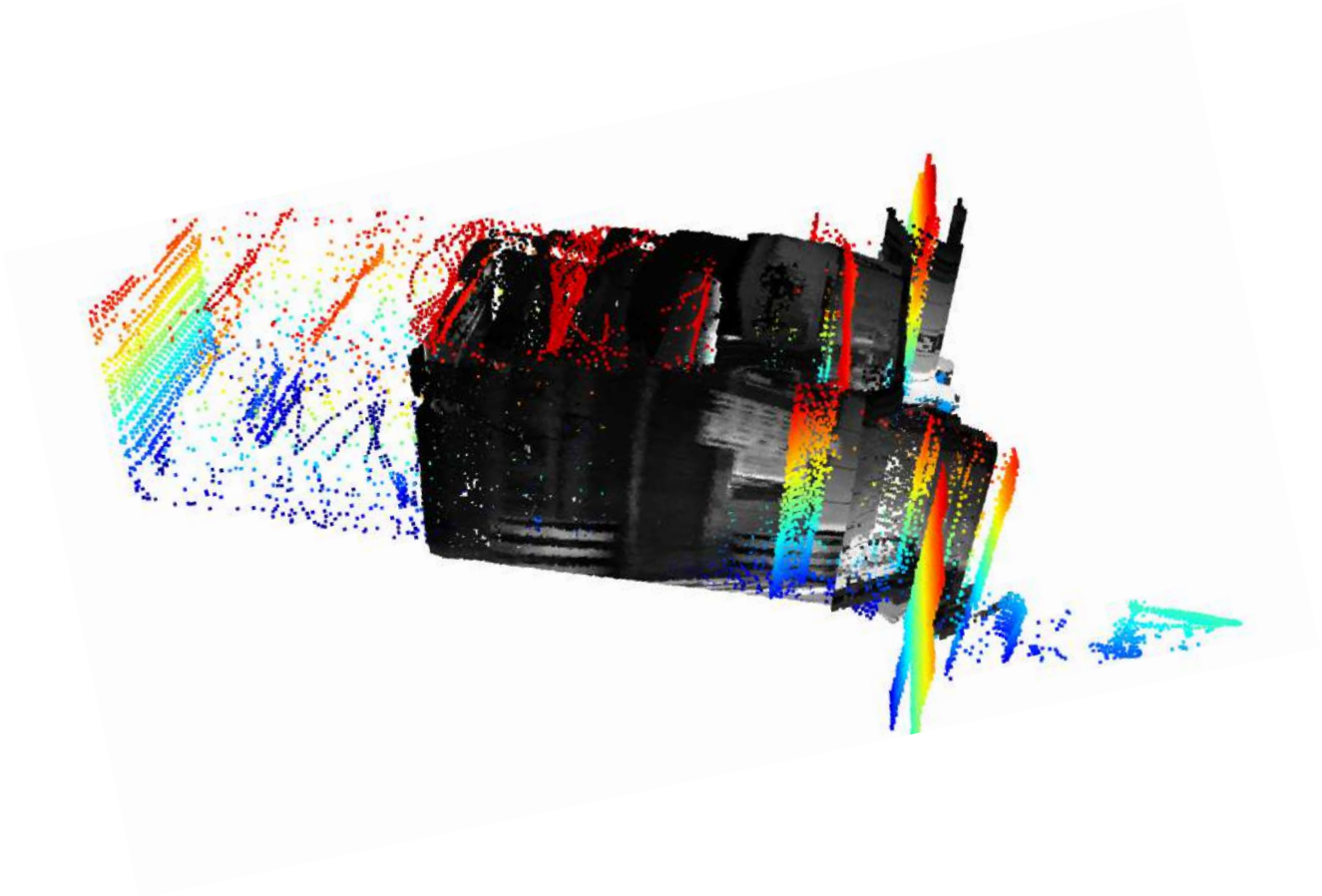
Metric/Model	MAE	ARE	RMSE	SSIM	Delta 1	Delta 2	Delta 3	Hausdorff Distance	CD-P	CD-T
DepthPro w/ Intrinsics	2.1200	0.4032	2.6499	0.8833	0.3503	0.8057	0.9603	10.1444	1.4776	7.2704
DepthPro w/o Intrinsics	3.8604	0.7362	4.5452	0.8095	0.1226	0.3673	0.7731	14.5365	2.1207	14.0176
Absolute Difference	**1.7404**	**0.3330**	**1.8953**	**0.0738**	**0.2276**	**0.4384**	**0.1871**	**4.3920**	**0.6431**	**6.7472**
UniDepth ViTs w/ Intrinsics	2.7211	0.4664	3.8214	0.8695	0.2428	0.7797	0.9273	20.3074	1.1020	5.7497
UniDepth ViTs w/o Intrinsics	6.3958	1.1560	7.4327	0.7204	0.0133	0.0578	0.3502	22.9902	1.2506	10.2869
Absolute Difference	**3.6747**	**0.6896**	**3.6112**	**0.1491**	**0.2295**	**0.7218**	**0.5771**	**2.6828**	**0.1485**	**4.5372**
UniDepth ViTl w/ Intrinsics	1.5750	0.2956	2.1716	0.9076	0.5519	0.9267	0.9657	18.0723	0.9157	3.7475
UniDepth ViTl w/o Intrinsics	3.4094	0.6349	4.0421	0.8400	0.0952	0.4915	0.9119	18.5290	0.8947	4.3897
Absolute Difference	**1.8343**	**0.3392**	**1.8705**	**0.0675**	**0.4566**	**0.4351**	**0.0537**	**0.4567**	**0.0210**	**0.6422**

Table 4.6: Performance of MDE models with and without Intrinsics on Flir-Livox data.

reconstruction performance. Models evaluated with provided intrinsics exhibited lower error metrics and better point cloud reconstructions in contrast to when the models predicted the intrinsics. Notably, the UniDepth ViTs variant showed the highest sensitivity to the absence of ground-truth intrinsics, with higher depth accuracy and point cloud quality degradations, as indicated by increased RMSE and worse Density Aware Chamfer and Hausdorff distances. In contrast, DepthPro, which integrates a dedicated focal length prediction head, maintained relatively robust performance even when intrinsic parameters were not provided. However, its performance is still degraded relative to when intrinsics were provided. These findings underscore the necessity of accurate camera intrinsics using current methods for precise 3D reconstruction, as the absence of this information diminishes the accuracy of the MDE models and propagates errors for downstream metric tasks such as volume estimation.



(a) Effect of varying focal lengths on 3D reconstruction ($f_{blue} > f_{red} > f_{green}$).



(b) Effect of a large focal length (f_{blue}) prediction on 3D point cloud (RGB colored) relative to ground truth (rainbow colored) demonstrated on a cargo container. Poor scene scales degrade the performance of metric estimations such as volume.

Figure 4.2: Significance of focal length in scale-aware reconstruction.

4.5 Closer look at Depth Anything V2

Depth Anything V2 [15], developed by researchers at TikTok, is a foundation model for monocular depth estimation that significantly advances the field by addressing the limitations of its predecessor, Depth Anything v1 [43]. It employs large-scale synthetic pre-training to generate accurate and robust pseudo-labels, which are used to train student models through a distillation framework [105, 15].

The model leverages a DINOv2 image encoder [105], which extracts high-quality hierarchical visual features. This makes it particularly well-suited for depth estimation tasks, which require both global scene understanding and fine-grained boundary predictions. The decoder employs a Dense Prediction Transformer (DPT) [71], which generates detailed depth maps that can be refined during metric fine-tuning. In this teacher-student training paradigm, the high-capacity teacher model, based on the DINOv2-Giant encoder, is pre-trained on synthetic datasets to produce precise pseudo-labels for large-scale real-world images.

Training models on synthetic data for monocular depth estimation is non-trivial due to its inherent limitations. Synthetic datasets often have restricted scene diversity, as they are generated using graphics engines consisting of pre-defined scene types, such as "living rooms" and "street views". Consequently, models trained on such data struggle to generalize to real-world scenarios characterized by clutter, occlusions, and randomness [15, 135]. Additionally, this "Sim-to-Real Gap" arises from the visual differences between synthetic and real images. Synthetic images are clean, structured, and noise-free, whereas real images exhibit sensor noise, reflections, lighting variations, and randomness, complicating the transfer of knowledge between domains [15]. Depth Anything V2 addresses these challenges through a multi-stage training

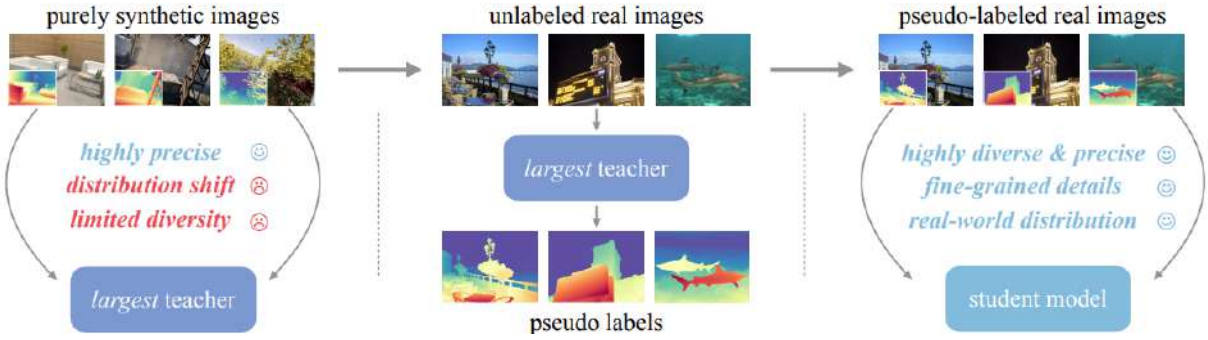


Figure 4.3: The framework employed in training Depth Anything V2 [15].

strategy (Figure 4.3) tailored for robust monocular depth estimation. In the **pre-training stage**, the DINOv2-Giant teacher model is trained on 595K high-quality synthetic images using scale- and shift-invariant loss $\mathcal{L}_{ssi}$ and gradient matching loss $\mathcal{L}_{gm}$ to ensure sharp and aligned depth predictions with fine-grained details. In the **pseudo-labeling** stage, the teacher model generates pseudo-depth labels, consisting of relative depth predictions, for 62M unlabeled real-world images, effectively reducing the Sim-to-Real gap. During the **distillation stage**, smaller student models are trained on this large-scale pseudo-labeled dataset, incorporating a feature alignment

loss to transfer semantic knowledge from the DINOv2 encoder. Finally, in the **fine-tuning stage**, these student models are adapted to domain-specific metric depth datasets, such as Hypersim and Virtual KITTI [149, 108], using a scale-invariant logarithmic loss $\mathcal{L}_{SiLog}$ to produce precise and scale-consistent depth predictions [15].
This multi-stage process, combining synthetic pre-training, large-scale pseudo-labeling, and domain-specific fine-tuning, enables Depth Anything to bridge the gap between synthetic and real-world data effectively, improving generalization and robustness across diverse domains.
In this work, we focus on the final stage of fine-tuning Depth Anything V2 for metric depth estimation using RGB-D data collected from various sources. Our goal is to enhance the model's performance on the domain-specific data and utilize it in the volume estimation pipeline. In the following sections, we delve deeper into our methodology, detailing the fine-tuning approach and training strategies employed to optimize Depth Anything V2 for our specific application.

4.6 Gate Detection

Gate detection refers to the task of detecting and segmenting the opening hatch or gate section of a cargo container, allowing our algorithm to separate the container from the surrounding scene (Figure 4.4a). Detecting container gates and floor areas is a fundamental step in the proposed volume estimation application as it improves the accuracy of container surface reconstruction and subsequent volume and floor area estimation. To achieve this, a RTMDet model [153] from the MMDetection framework is utilized. Our industrial partner trained the model on their proprietary dataset, which consisted of roughly 1300 annotated images taken from daily operations. However, as the data is not available to us, we use the trained weights of the model as is and report the model's accuracy for reference. The labels used to train the segmentation model include **container_gate** and **floor_area**, respectively. The evaluation results (Table 4.7) demonstrate

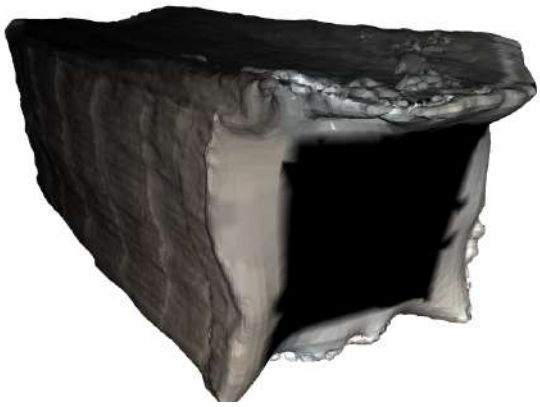
Class Label	TP	FP	FN	TN	Recall	Precision	F1 Score	Average Precision
container_gate	286	22	1	287	0.9965	0.9286	0.9613	0.9959
floor_area	201	48	28	229	0.8777	0.8072	0.8410	0.8319

Table 4.7: Evaluation results for RTMDet gate detection model.

the effectiveness of the RTMDet model in detecting and segmenting the container gates and floor area, achieving an F1 Score of 96.13% and an Average Precision (AP) of 99.59% validated on 596 images with segmented container gates and 506 floor areas. The model exhibited a remarkably high recall rate of 99.65%, indicating that almost all container gates in the evaluation dataset were successfully identified, with only one false negative recorded. However, the false positive rate was slightly higher, resulting in a precision of 92.86%. This suggests that while the model reliably detects container gates, occasional bad predictions may occur, potentially due to similarities between gate structures and other parts of the warehouse[22].

[22] See appendix A

(a) Output segmentation masks from gate detection.



(b) Gate closing and mesh reconstruction.

Figure 4.4: a) Gate detection on a container image produces gate (green) and floor masks (red). These masks are reprojected into 3D space and used to close the container for mesh reconstruction, eliminating background objects.

In contrast, the detection of floor areas presented greater challenges due to cargo clutter. The model achieves an F1 Score of 84.10% and an AP of 83.19%, reflecting moderate accuracy in this class. A recall of 87.77% indicates that most floor areas were successfully identified, although a precision of 80.72% highlights a higher incidence of false positives. This discrepancy could be attributed to the variability in the appearance of floor areas across different containers, such as differing textures, lighting conditions, or partial occlusions in the dataset.
Overall, the results indicate that the RTMDet model showcases robust performance for the task of container gate detection, making it of practical use in the pipeline. However, the performance for floor area detection suggests room for improvement. Future work could focus on refining the performance to include a broader range of floor area variations using augmentations or employing post-processing techniques to filter out false positives and enhance overall accuracy.

4.7 Watertight Surface Reconstruction

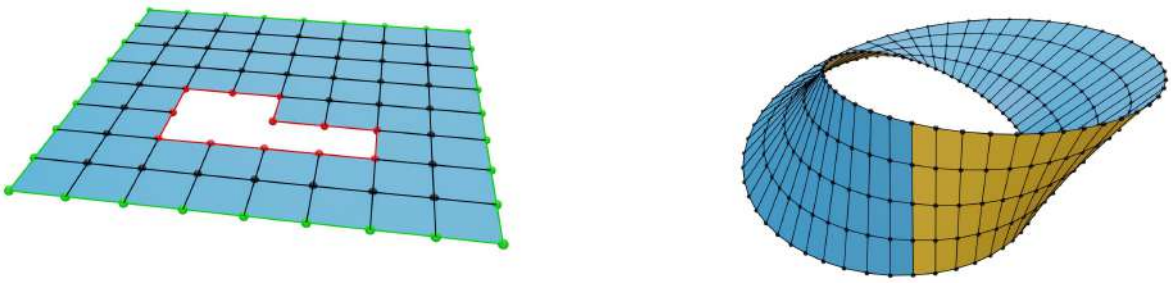
Like single-view depth estimation, reconstructing an object or surface mesh from a point cloud is a fundamental yet ill-posed problem in computer vision and graphics. A set of 3D points can have multiple continuous surface interpolations. Moreover, the point clouds can be noisy, non-uniform, and contain numerous outliers, adding to the task's complexity. 3D surfaces possess fundamental properties such as watertightness and orientability (Figure 4.6). A mesh is considered watertight when it forms a continuous, uninterrupted surface without holes enclosing a finite volume. On the other hand, a mesh is orientable if all its facets (edges, faces) can be categorized as pointing inwards or pointing outwards in a globally coherent way, establishing an interior and exterior with a surface at the boundary between the two [80]. For the application of metric volume estimation, mesh watertightness is a pre-requisite as the volume of a

(a) Reconstructed mesh without container isolation.

(b) Reconstructed mesh with container isolation.

Figure 4.5: Projected point cloud with and without gate detection. Gate detection leads to better mesh reconstruction and volume estimates.

non-watertight mesh cannot be reliably estimated. We begin our volume estimation pipeline with a point cloud obtained through depth estimation. The first step is to downsample the point cloud to reduce the number of points while preserving its overall structure. Next, we estimate the normal vectors for each point in the downsampled point cloud, which represent the surface orientation at each point. To improve accuracy, we perform normal correction to address noise



(a) A non-watertight surface.

(b) A non-orientable surface (Mobius strip).

Figure 4.6: Examples of non-watertight and non-orientable surfaces [80].

and outliers in the point cloud. This involves comparing each normal vector with the centroid of the point cloud and flipping it if it points in the wrong direction. This ensures a consistent outward orientation of the normals relative to the centroid (Figure 4.8).

Once the normals are corrected, we proceed with surface reconstruction using Poisson surface reconstruction, which generates a detailed 3D mesh from the point cloud (Figure 4.7). To ensure robustness, we filter out small disconnected components and retain only the largest connected mesh, which typically represents the main object of interest. Finally, we validate that the mesh is watertight and calculate its volume, accurately estimating the unutilized or free volume of the

container in cubic meters.

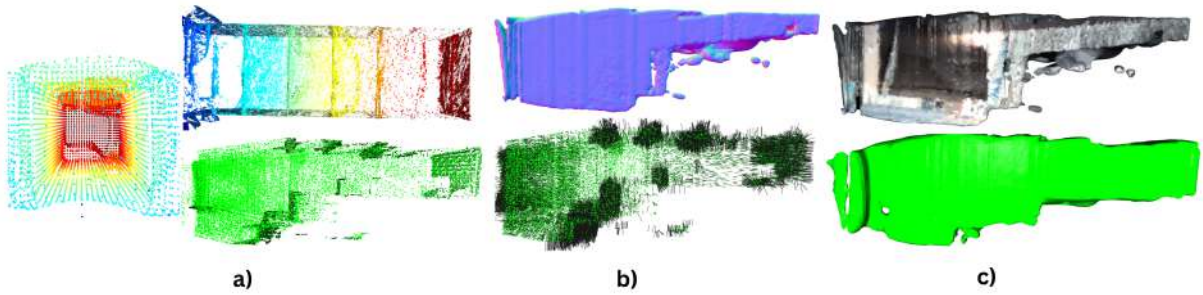
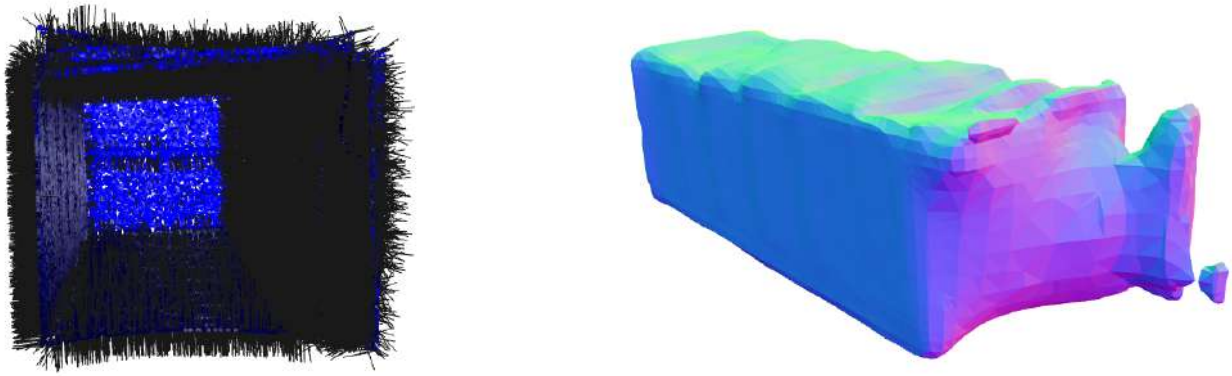


Figure 4.7: Steps involved in mesh reconstruction and volume estimation. a) Pointcloud down-sampling b) Normal estimation & correction c) Poisson mesh reconstruction.



(a) Corrected normals.

(b) Normal map of the reconstructed mesh.

Figure 4.8: Down-sampling and Normal correction improves computational efficiency and quality of the reconstructed mesh.

4.8 Estimating Volume

To calculate the volume of our watertight container mesh, we employ a technique known as the divergence theorem, which reduces volume integrals to surface integrals [154]. The divergence theorem allows for the efficient computation of geometric properties such as volume for polyhedral meshes. The meshes processed by Python libraries such as PyVista or Trimesh [155] are composed of triangular faces, making this approach advantageous for such mesh geometries. The volume V of a solid polyhedron can be expressed by integrating a constant function over the volume [154]:

$$V = \int_V P(x, y, z) dV \tag{4.1}$$

Using the Divergence Theorem, this integral can be reduced to a surface integral:

$$\int_V \nabla \cdot \mathbf{F} dV = \int_S \mathbf{N} \cdot \mathbf{F} dS \tag{4.2}$$

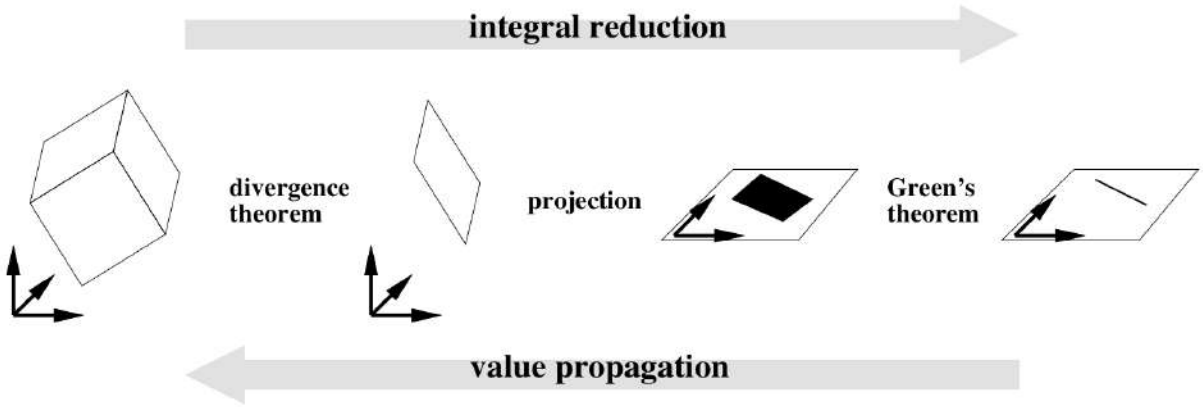


Figure 4.9: Visualization of volume reduction using the Divergence theorem [154].

Where S is the boundary of the polyhedron and dS is an infinitesimal measure of surface area. The function $\mathbf{F}(x, y, z)$ is chosen so that $\boldsymbol{\nabla} \cdot \mathbf{F} = p$. The vector $\mathbf{N}$ denotes outward-pointing, unit-length surface normals. The core idea of the numerical implementation of volume calculation can be summarized by reducing the volume integral to a surface integral and computing the surface integrals over all triangular faces in the mesh. For each mesh, the cross product of the edge vectors is computed to obtain the face normal N_f and the area of the triangle. The contribution of each triangle to the volume is proportional to the dot product of the triangle's normal and its geometric centroid. For a triangular face, the surface integral of the projection on the x-axis can be written as:

$$\int_F x dS = \frac{1}{6} (x_0 + x_1 + x_2) \cdot |\mathbf{E}_1 \times \mathbf{E}_2| \tag{4.3}$$

where x_0, x_1, x_2 are the x-coordinates of the triangle's vertices, and $\mathbf{E}_1 \times \mathbf{E}_2$ represents the area and orientation of the triangle. Summing the contributions from all faces yields the total volume of the polyhedron.

This implementation of volume computation for closed triangular meshes is highly efficient because it reduces the complexity of the problem (Figure 4.9) using vector calculus and avoids the need for complex volumetric discretizations. By exploiting the simplicity of triangular faces, integrals are computed easily both numerically and algebraically, resulting in a robust, scalable method for efficiently handling large meshes.

5 Results

This section presents the outcomes of the experiments and methodologies outlined in earlier chapters, providing quantitative and qualitative evaluations of the pre-trained and fine-tuned models. We begin by detailing the fine-tuning process, highlighting key considerations regarding the dataset, the hyperparameters used, and the training strategy employed. The loss function utilized to train the model is described, followed by a study on the effect of loss function choice on model performance for our dataset. The fine-tuned results are organized systematically and compared to pre-trained baselines to assess the model's accuracy and generalization on container scenes. Key metrics introduced earlier, such as MAE, RMSE, Density-Aware Chamfer Distance, and Hausdorff Distance, are utilized to quantitatively evaluate the quality of the reconstructed point clouds by comparing them with ground truth.
Finally, the performance of the end-to-end pipeline for volume estimation is evaluated, followed by a discussion of the results and sources of error.

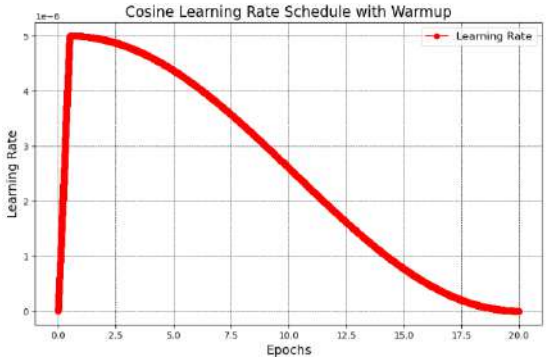
5.1 Fine-Tuning Details

To fine-tune Depth Anything V2 for our application, we curated a dataset of 70,753 RGB-D samples from multiple publicly available sources (Table 3.2). Data cleaning and preprocessing procedures (Section 3.4) ensured consistent training and testing splits. We selected indoor and outdoor scenes from the DIODE dataset by filtering invalid depth values using provided masks and discarding depth maps with more than 50% missing pixels. We also incorporated selected scenes from the InSpaceType dataset, focusing on large environments like hallways, libraries, and lounges, retaining only frames where the maximum depth exceeded 8 m. The NavVis dataset, originally comprising 2,108 warehouse scenes, was augmented as described in Chapter 3 to yield 16,864 samples. Additionally, we incorporated 11,070 RGB-D samples from PhoneDepth, 5,978 outdoor samples from DIML, and 2,615 upsampled, confidence-filtered images from an ARKit-collected dataset.
After preprocessing, the dataset was structured into JSON (JavaScript Object Notation) files and partitioned into training and testing sets with a 95:5 ratio, resulting in 67,215 and 3,538 RGB-D pairs, respectively. A custom dataloader helped load the data from these JSON files while applying training-time augmentations (color jitter and Gaussian noise, each with a 20% probability). Input images were resized to 518 × 518 to ensure compatibility with the Vision Transformer model. We employed two encoder configurations, ViTs and ViTb, for model training. While larger encoder architectures offer improved performance, hardware limitations and the moderate size of our training dataset motivated the adoption of compact models. This choice prioritizes computational efficiency while maintaining competitive accuracy and mitigating overfitting risks inherent to resource-constrained training scenarios. Both models were trained using a single NVIDIA GPU with 24 GB of VRAM. The batch sizes were set to 18 for the ViTs model and 8 for the ViTb model. The training process spanned 40 epochs and took approximately 2.5

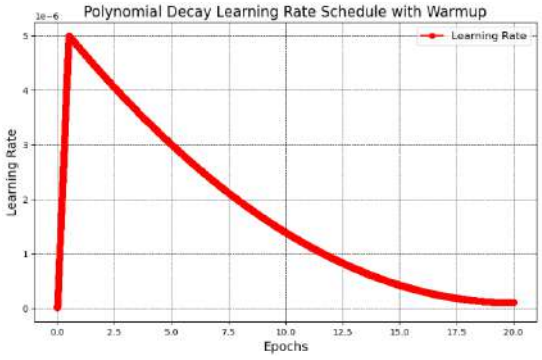
days. Checkpoints were saved periodically at five-epoch intervals to facilitate training recovery and analyze convergence.

5.1.1 Training Strategy and Loss functions

Our training strategy closely followed the methodology of Depth Anything V2. We randomly initialized the DPT decoder using the pre-trained VKITTI encoder weights from Depth Anything V2. We used the VKITTI checkpoint on the assumption that lighter pretraining (on less data) reduced the risk of the model being overly biased by the indoor Hypersim dataset. Moreover, we observed that models incorporating outdoor training datasets performed better in our initial benchmarks [76, 149] as our curated dataset underrepresented depth ranges greater than 25 m.
To efficiently fine-tune the model, we applied a differential learning rate strategy [150] in which the pre-trained encoder was updated conservatively with a base learning rate of $5e-6$. In contrast, the learning rate for the randomly initialized decoder was set at a $10 * base_{lr}$ of $5e-7$. This approach ensured that the encoder retained its learned representations, minimizing drastic weight changes while allowing the decoder to adapt to our domain-specific data quickly.
To optimize training stability and efficiency, we used the AdamW optimizer with $w_d = 0.01$, incorporating decoupled weight decay, preventing over-regularization of model parameters. We also implemented a cosine decay scheduler with a warm-up in place of the polynomial learning rate scheduler utilized in the original Depth Anything V2 training configuration. Rather than immediately employing the full learning rate, the warm-up phase gradually increases (Figure 5.1) the learning rate from a small initial value to the intended maximum, mitigating instability during early training. This approach is particularly advantageous for transformers, as the ViT encoder exhibits high sensitivity to early training dynamics [150]. By implementing a more gradual, extended warm-up for the encoder[23] and a shorter warm-up for the decoder, we aimed to facilitate stable fine-tuning on our constrained hardware setup.



(a) learning rate schedule with a cosine schedule



(b) learning rate schedule with a polynomial schedule

Figure 5.1: Comparison of learning rate schedulers for 20 epochs with a single warm-up epoch.

[23] Five epochs for ViTb, one epoch for ViTs

5.1.2 Loss Functions

Loss function design is crucial for monocular depth estimation as it directly influences the model's ability to learn meaningful representations and produce accurate predictions from the data. For our fine-tuning, we employed the Scale-Invariant Logarithmic (SiLog) loss (Equation 5.1), which has been widely used in monocular depth estimation [42, 15, 64, 72] due to its capacity to preserve relative depth relationships while remaining invariant to global scale shifts. SiLog loss operates in the logarithmic space, penalizing pixel-wise depth errors and global mean bias, ensuring the model learns a scale-consistent representation of depth. The loss function utilized to train the metric checkpoints of Depth Anything [15] uses a modified version of the original loss function presented in [42], which helps mitigate the impact of large loss values, resulting in more stable training. The loss function is given by:

$$L_{\text{SiLog}} = \sqrt{\frac{1}{N}\sum_i \left(\log d_i - \log \hat{d}_i\right)^2 - \lambda \left(\frac{1}{N}\sum_i \left(\log d_i - \log \hat{d}_i\right)\right)^2}, \tag{5.1}$$

where d_i is the ground truth depth for pixel I, $\hat{d}_i$ is the predicted depth for pixel I, N is the number of valid pixels, and λ is a weighting factor that controls the influence of the second term. The first term inside the square root penalizes pixel-wise logarithmic depth errors, and the second term, scaled by λ, accounts for the global scale bias, ensuring the loss remains invariant to uniform depth scaling.

Although the SiLog loss is effective for monocular depth estimation, it exhibits notable limitations when applied to the 3D reconstruction task. One of its primary constraints is its focus on global consistency rather than local structure. Depth maps optimized using SiLog loss alone can suffer from blurred edges and reduced fine-detail preservation, which can be important for fine-grained geometry reconstruction applications.

Models such as UniDepth and DepthPro use a combination of formulated loss functions at different stages during training to ensure robust learning of depth features. UniDepth employs a reformulated Mean Squared Error (MSE) loss in the final 3D output space (θ,ϕ,z_{log}), incorporating scale-invariant depth supervision. DepthPro, on the other hand, optimizes the canonical inverse depth using a combination of Mean Absolute Error (MAE), multi-scale gradient-based losses, and edge-aware terms to preserve fine-grained structural details [76, 75].

Alternative loss functions that address these limitations include the Huber loss, which blends quadratic (MSE) and linear (MAE) error terms to reduce sensitivity to outliers, and the structural similarity index (SSIM) loss, which penalizes deviations in luminance, contrast, and structural patterns to enforce local coherence in predicted outputs. Another effective strategy involves combining multiple loss functions or utilizing adaptive weighted strategies tailored to task-specific requirements, such as pairing global consistency terms (e.g., SiLog) with edge-sensitive losses to preserve structural fidelity in depth estimation tasks [122].

To test the influence of different loss functions, we prepared a small experiment involving the

SiLog loss utilized in Depth Anything V2, a scaled SiLog version from Pixelformer, and a custom weighted loss function utilizing the Huber, SSIM, and SiLog loss functions. We briefly train the ViTs DepthAnythingv2 VKITTI checkpoint on these three loss functions for 25 epochs and plot the training results for a comparative study on our dataset. (Figure 5.2) illustrates that the custom weighted loss function, which combines the Huber, SSIM, and SiLog losses using experimentally tuned weights formulated as:

$$\mathcal{L} = \alpha \mathcal{L}_{\text{SiLog}} + \beta \mathcal{L}_{\text{huber}} + \gamma \mathcal{L}_{\text{ssim}}\,, \tag{5.2}$$

where, $\alpha = 0.5, \beta = 0.6, \gamma = 0.8$.

The scaled SiLog loss [64] achieves lower error rates over the 25 training epochs when compared to the standard SiLog and weighted SiLog losses[24]. This loss function helps improve how quickly the model learns and makes its performance more consistent. It reduces errors measured by δ1, ARE, and RMSE. These results show that creating better loss functions is important for training depth estimation models effectively. Although our initial test with a scaled loss function looks promising, we decide to use the original loss function from the authors of Depth Anything V2. This choice allows us to fine-tune the model with our data, making it easier to compare our results fairly with those of pre-trained models.

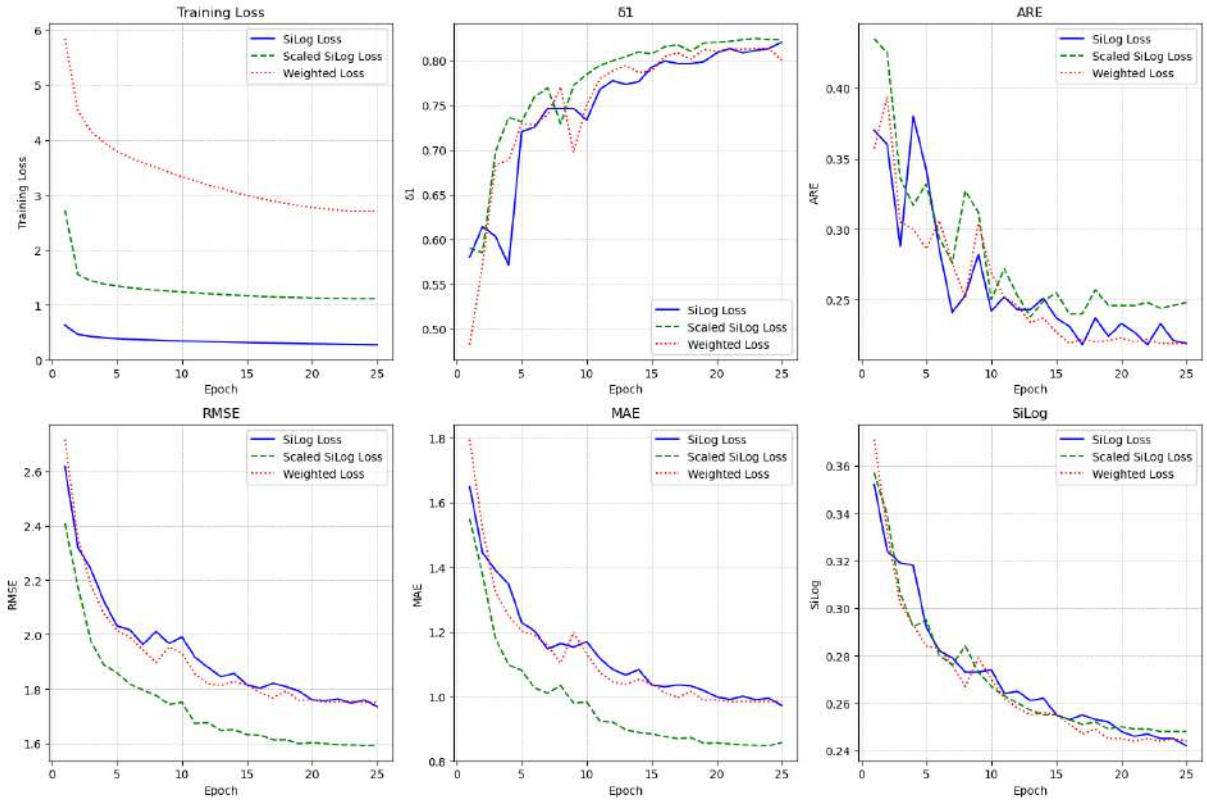


Figure 5.2: Comparision of three different loss function for training Depth Anything V2 on our dataset.

[24]We use $\mathcal{L}_{\text{Scaled SiLog}} = \alpha \cdot SiLog\ Loss$ where, $\alpha = 10$

Loss Function	MAE	ARE	RMSE	SSIM	Delta 1	Delta 2	Delta 3	Hausdorff	CD-P	CD-T
Scaled SiLog Loss	1.3852	0.3184	1.7182	0.9100	0.5194	0.8889	0.9795	21.6515	0.9145	3.7907
SiLog Loss	1.4993	0.3271	1.9440	0.9013	0.5017	0.8315	0.9589	22.5376	1.0075	4.6266
Weighted Loss	2.0059	0.4754	2.3368	0.8765	0.3171	0.7264	0.9103	22.3278	1.1308	5.2467

Table 5.1: Results for SiLog, Scaled SiLog and Weighted loss functions on Flir-Livox evaluation dataset.

5.1.3 Hyperparameters

Finally, the $max\ depth$ parameter is set to a value of 80 m, being an essential parameter in defining valid depth regions within the dataset. Specifically, it filters valid samples by enforcing the constraint $0 < depth \leq max\ depth$ during training and inference. This ensures that the depth values beyond the predefined threshold are excluded, maintaining consistency in depth estimation. Since Depth Anything employs a sigmoid activation function in its final layer, producing per-pixel outputs in the range $[0, 1]$, mapping these outputs to absolute depth values is done by multiplying the output with $max\ depth$ value, effectively scaling the depth predictions from 0 to 80 m. We found that the choice of $max\ depth$ significantly influences the scaling of the predicted depths and, consequently, the quality of downstream point cloud reconstruction. Optimizing $max\ depth$ during inference can lead to improved depth scaling and metric estimations. Fine-tuning this parameter based on dataset distribution and scene characteristics can further refine the model's depth prediction accuracy during inference. Table 5.2 shows a summary of used hyperparameters for each encoder.

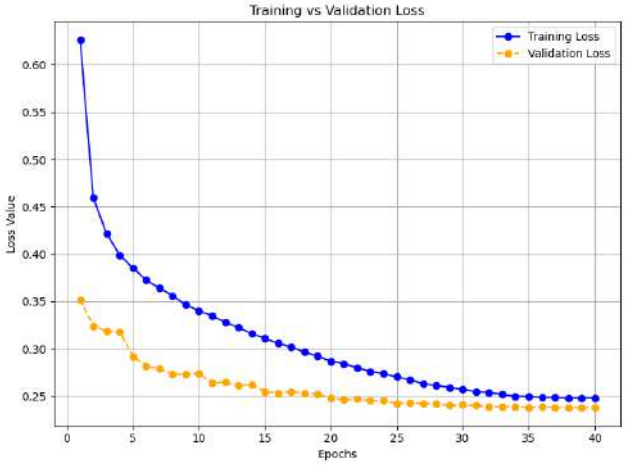
Hyperparameter	ViTs	ViTb
max_depth	80	80
batch_size	18	8
weight_decay	0.01	0.01
num_epochs	40	40
warmup_epochs	1	5
scheduler_rate	1	1
encoder_lr	5×10^{-6}	5×10^{-6}
decoder_lr	$10 \times$ base_lr	$10 \times$ base_lr

Table 5.2: Hyperparameters for trained encoder configurations.

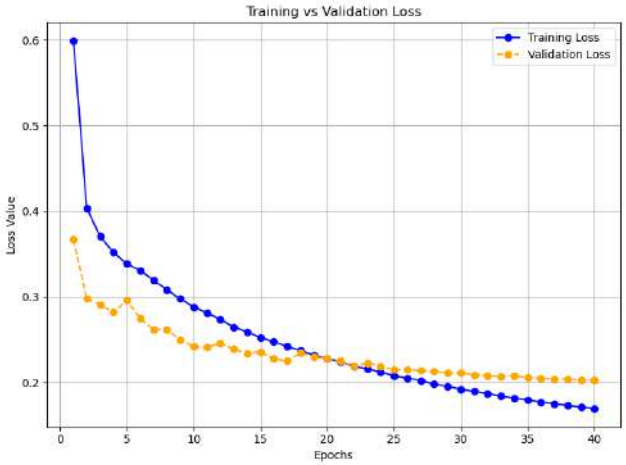
5.2 Training Results

The training and validation loss curves for the ViTs (Figure 5.4) and ViTb (Figure 5.5) encoders show stable convergence throughout the 40 epochs of training. The larger batch size of 18 for the ViTs likely contributes to the smoother learning curve and better generalization in comparison to ViTb with a batch size of 8. Both losses decrease monotonically, with the training loss declining steadily. The validation loss follows a similar trajectory, indicating effective generalization with minimal overfitting. Notably, the losses reduce in parallel until approximately epoch 20

for the ViTb encoder and until epoch 40 for the ViTs encoder, after which they stabilize asymptotically. This behavior suggests that the model capacities are well matched to the complexity of the dataset. In particular, the validation loss for the ViTs variant plateaus around epoch 30 while remaining below the training loss, further confirming robust generalization (Figure 5.3a). Conversely, the ViTb encoder exhibits some initial training instability, with the validation loss plateauing early. This early plateau may indicate slight overfitting, implying that the ViTb encoder could benefit from additional training data in future experiments (Figure 5.3b). Moreover, using a larger batch size with enhanced regularization techniques, such as gradient accumulation, may improve the training process's effectiveness and stability.



(a) ViTs shows signs of asymptotic convergence at the end of 40 epochs.



(b) ViTb shows minor instability in training, possibly due to the smaller batch size.

Figure 5.3: Training vs. Validation Loss for fine-tuning ViTs and ViTb on 70K samples. The plots indicate the progressive reduction in training and validation loss over 40 epochs. While both losses decrease, the validation loss plateaus around epoch 25, suggesting a point of diminishing generalization improvements.

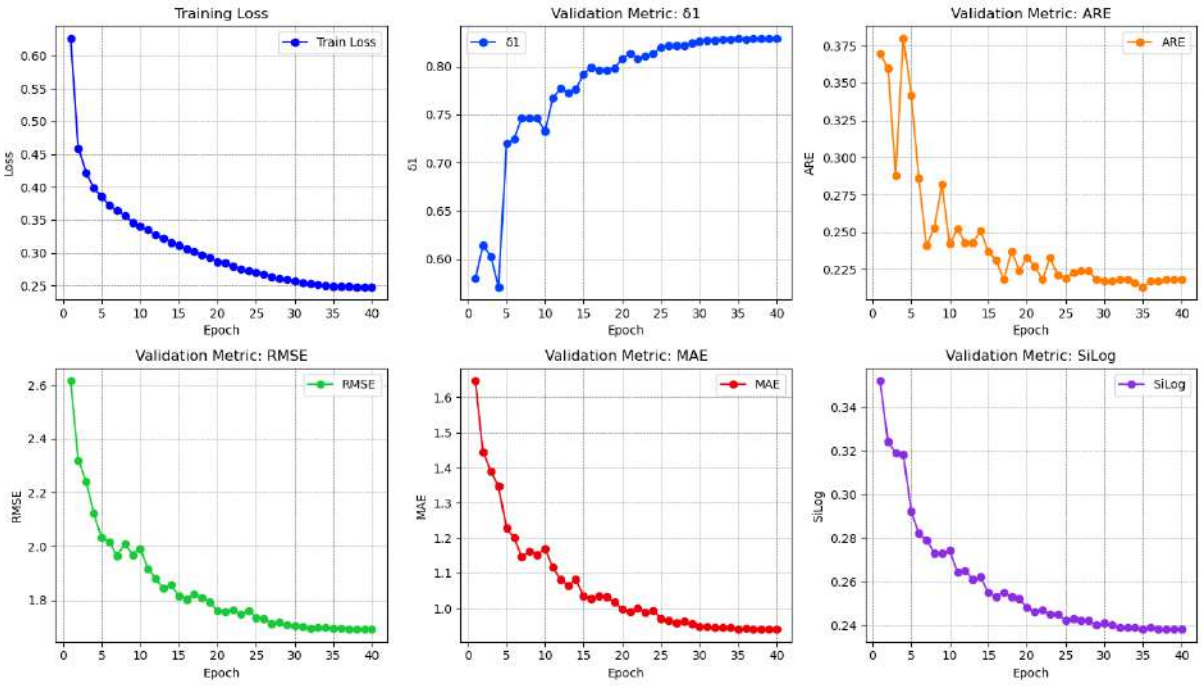


Figure 5.4: Training graphs for ViTs. The training loss reaches a stable value of 0.25, while metrics such as δ_1, MAE, RMSE and ARE converge to 0.83, 0.94, 1.70 and 0.20 respectively.

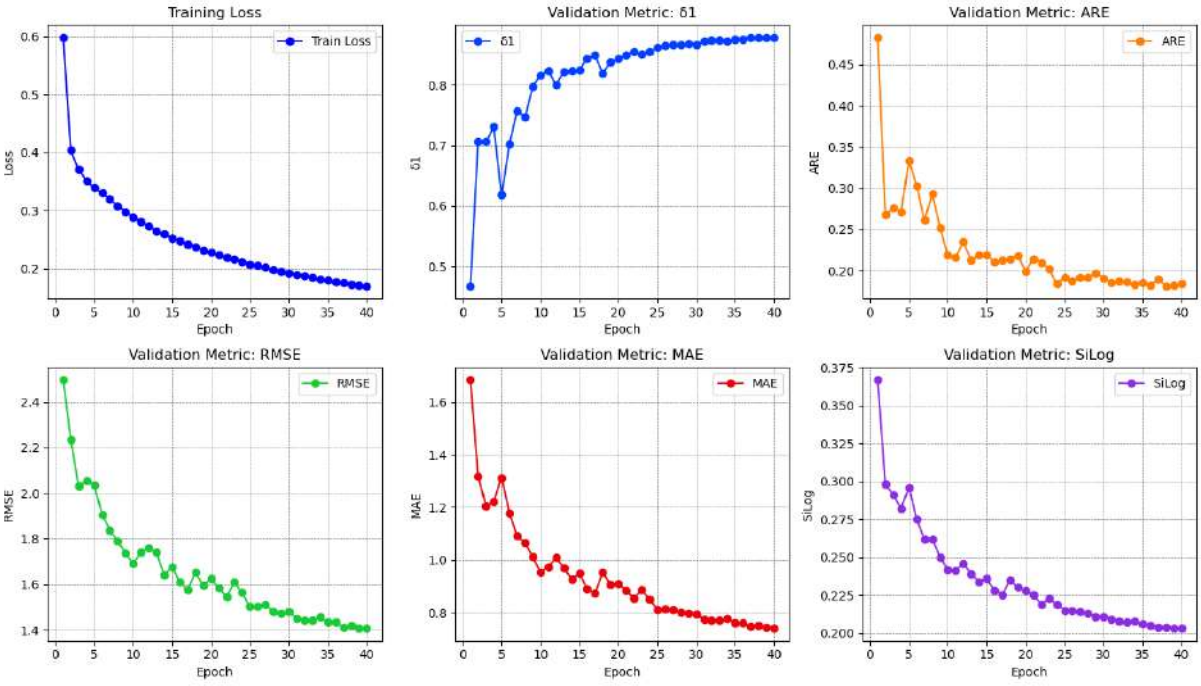


Figure 5.5: Training graphs for ViTb. The training loss reaches a stable value of 0.17, while metrics such as δ_1, MAE, RMSE and ARE converge to 0.88, 0.74, 1.40 and 0.18 respectively.

5.3 Fine-Tuned Evaluation

To assess the performance of the fine-tuned metric checkpoints relative to their pre-trained counterparts, we conducted evaluations on datasets captured using multiple imaging devices.

Model Name	Encoder	Camera Data	MAE	ARE	RMSE	SSIM	Delta1	Delta2	Delta3	Hausdorff Distance	CD-P	CD-T
VKITTI-FineTuned	ViTs	NavVis	1.5861	0.3407	2.0714	0.8947	0.4208	0.7265	0.8630	**5.0449**	**0.2501**	**1.2037**
VKITTI-FineTuned	ViTb	NavVis	**1.5404**	0.3535	**2.0464**	**0.9045**	0.4508	0.7484	**0.9027**	**3.7269**	**0.2007**	**0.4139**
VKITTI-FineTuned	ViTs	Flir	**1.3580**	**0.2907**	**1.8107**	**0.9093**	**0.5507**	**0.8748**	**0.9752**	22.2159	0.9727	4.2053
VKITTI-FineTuned	ViTb	Flir	**1.6063**	0.3698	**2.0063**	**0.8992**	0.4571	**0.8341**	**0.9660**	21.8124	1.0290	4.3490
VKITTI-FineTuned	ViTs	iPhone 12	3.5052	0.4012	4.2157	0.8294	0.0908	0.3747	0.7634	**11.8994**	0.8517	5.4661
VKITTI-FineTuned	ViTs	Pixel 7	2.1315	**0.2688**	3.4702	0.8497	**0.6226**	**0.8071**	0.8655	**12.5497**	0.6945	3.5808
VKITTI-FineTuned	ViTb	iPhone 12	3.5000	0.3888	4.3402	0.8356	0.1114	0.4561	0.7695	13.3673	1.0696	8.9981
VKITTI-FineTuned	ViTb	Pixel 7	2.2027	**0.2783**	3.7083	0.8422	**0.6222**	0.8133	0.8654	**12.3580**	**0.5377**	**2.3733**

Table 5.3: Fine-Tuned results for ViTs and ViTb Depth Anything V2 checkpoints across container evaluation dataset.

Model Name	Encoder	Camera Data	MAE	ARE	RMSE	SSIM	Delta1	Delta2	Delta3	Hausdorff Distance	CD-P	CD-T
VKITTI-PreTrained	ViTs	NavVis	**5.5151**	**1.6591**	**6.4980**	**0.6899**	**0.0935**	**0.2367**	**0.4026**	**19.7265**	**1.8734**	**20.9835**
VKITTI-PreTrained	ViTb	NavVis	6.9110	2.0920	8.1760	0.6381	0.0682	0.1896	0.3320	27.4053	2.3676	33.7774
VKITTI-PreTrained	ViTs	Flir	7.9610	1.5405	8.9166	0.6530	0.0193	0.0749	0.1794	28.3377	3.4667	37.0451
VKITTI-PreTrained	ViTb	Flir	9.4905	1.8305	10.4586	0.6010	0.0056	0.0262	0.0778	28.9488	3.9732	46.4711
VKITTI-PreTrained	ViTs	iPhone 12	11.1846	1.8488	12.5621	0.5860	0.0068	0.0356	0.0986	25.8745	3.1221	47.2991
VKITTI-PreTrained	ViTs	Pixel 7	**4.5533**	**0.6028**	**5.4499**	**0.8507**	**0.2080**	**0.5872**	**0.8384**	**13.1207**	**1.6615**	**21.0210**
VKITTI-PreTrained	ViTb	iPhone 12	13.0159	2.1253	14.6988	0.5373	0.0036	0.0127	0.0467	36.0758	3.6582	59.8198
VKITTI-PreTrained	ViTb	Pixel 7	**5.9940**	**0.7830**	**7.1666**	**0.8092**	**0.0833**	**0.3642**	**0.7387**	**16.1672**	**1.9781**	**27.1347**

Table 5.4: Pretrained baselines for Depth Anything V2 ViTs and ViTb checkpoints across our container evaluation dataset.

This dataset includes images from the FLIR Blackfly camera, NavVis scanner, and mobile devices like the iPhone 12 and Pixel 7. We developed a custom data loader to manage image retrieval while automatically extracting known focal lengths from each device. This facilitated consistent computation of depth and point cloud metrics, as detailed in Section 2.8. The dataset utilized in this evaluation comprises 19 container images obtained from the Flir-Livox scanner, 58 images from the NavVis scanner, and 36 images from each mobile device. It is important to note that the NavVis images were part of the training dataset, which could introduce a bias. To mitigate this, we augmented the NavVis images by projecting the omnidirectional images at varying scale factors, effectively modifying their perceived focal lengths. This adjustment significantly impacted performance metrics, demonstrating the sensitivity of monocular depth estimation to focal length variations. Conversely, the Flir and mobile camera images remained unaltered, as they were entirely out-of-domain samples relative to the training dataset.

Following the methodology described (Section 4), we evaluated the models using the constructed evaluation dataset. After masking invalid depth values, we compared depth predictions against the ground truth. Our analysis reveals substantial improvements across all device categories when comparing the fine-tuned models against their pre-trained baselines (Tables 5.3, 5.4). For in-domain data, the NavVis dataset exhibits marked improvements in point cloud metrics, affirming the effectiveness of fine-tuning in adapting to domain-specific characteristics. This metric difference suggests that the model successfully leverages in-domain knowledge to refine its predictions, enhancing geometric consistency. Performance improvements were also evident for out-of-domain data captured using the Flir camera and mobile devices. Among these, the ViTs model demonstrated superior generalization on the Flir dataset compared to its pre-trained counterpart, followed by the ViTb model. However, the point cloud metrics for ViTb did not show substantial improvements, likely due to the lower image quality

and the limited size of the fine-tuning dataset, potentially leading to slight overfitting [150].
A noteworthy observation is the performance discrepancy between the Pixel 7 and iPhone 12 datasets. The Pixel 7 consistently outperformed the iPhone 12 in pre-trained and fine-tuned evaluations. This outcome is unexpected, given that the training dataset contained approximately 2615 images captured using an iPhone device with ARKit. The minimal difference between the ViTs and ViTb models on Pixel 7 data further underscores the importance of input image quality in zero-shot monocular depth estimation for out-of-domain data.

Regarding point cloud metrics, ViTb outperforms ViTs in fine-tuned models, indicating its capacity to capture spatial depth relationships more effectively. However, ViTs demonstrate superior generalization across various datasets, suggesting they retain broader applicability across different imaging conditions. An important finding is that while ViTs surpass ViTb in depth metrics, ViTb achieves better point cloud metric performance. This discrepancy highlights a critical limitation in relying solely on depth metrics as a proxy for monocular depth estimation in 3D reconstruction. High-quality depth predictions do not necessarily translate to geometrically consistent reconstructions, reinforcing the necessity of evaluating 3D consistency beyond pixel-wise depth accuracy.

5.4 Results on Volume Estimation

Based on Poisson surface reconstruction and the divergence theorem, our volume estimation method demonstrated high efficiency when applied to the LiDAR point clouds obtained from NavVis scanners. While collecting data, we manually measured the containers and calculated their volumes based on these measurements. Therefore, the manually obtained volume serves as the true ground truth for the container. To quantitatively assess the measurement accuracy of our mesh reconstruction, we also employed our volume estimation algorithm to calculate the volume of the container algorithmically. We found a mean absolute error of **1.7255 cubic meters** between the ground truth and the estimated volume, which provides a reference for further studies.
We benchmarked the volume estimation algorithm on NavVis data due to its precise point clouds and high-quality RGB-D data from the point cloud projections. We extracted 55 images of containers from the NavVis dataset. We tested our pipeline on these 55 containers to evaluate the performance of each pre-trained model checkpoint from the VKITTI and Hypersim datasets. Since we only had the resources to train the ViTs and ViTb variants, we compared the fine-tuned versions against other variants of the same encoder type.
The volume results presented in Table 5.5 and Figures 5.8, 5.7 provide valuable insights into the performance of the different model encoders in predicting container volumes. The results are evaluated by calculating the mean error between each container's predicted, algorithmic, and true volume. The results are then averaged out to give a concise result table. However, it's worth noting that our end-to-end pipeline works (Figure 5.5,) best on containers smaller than

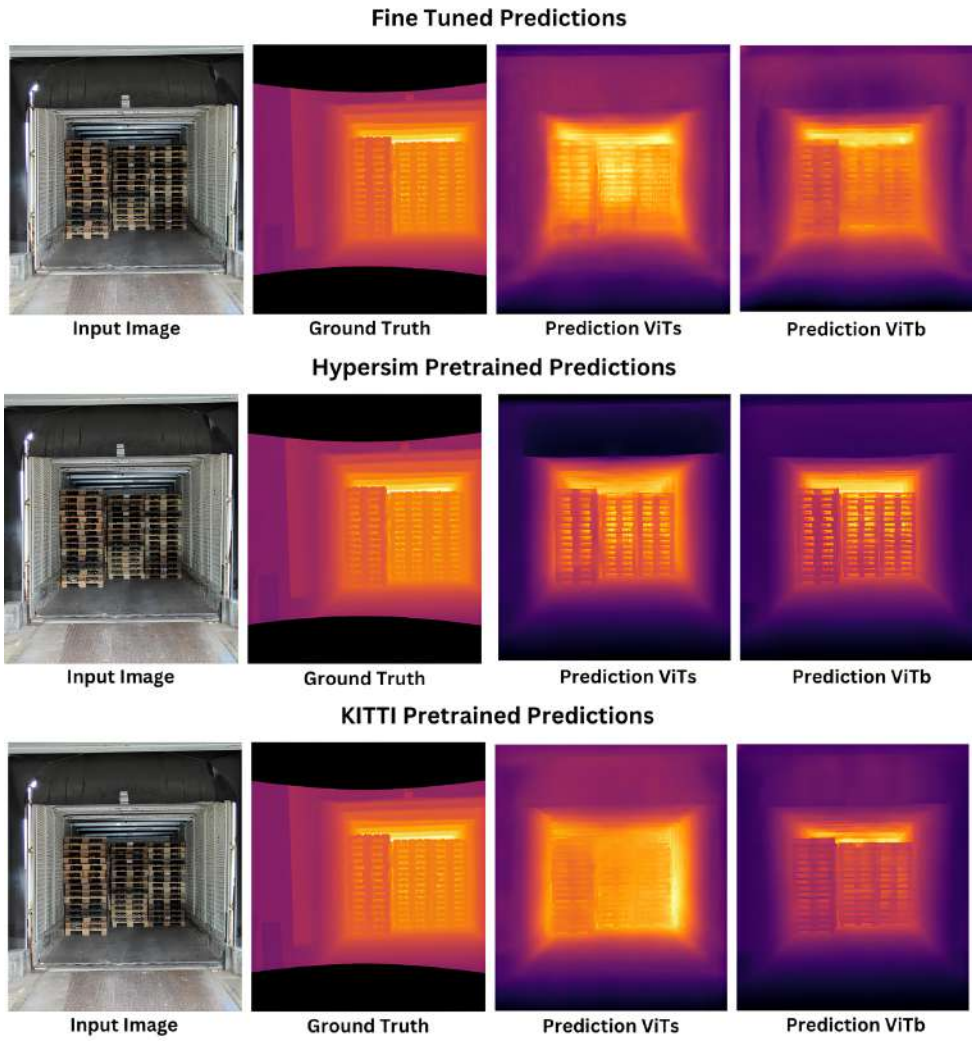


Figure 5.6: Difference in depth map quality between checkpoints for ViTb encoder.

40 cubic meters, as very long containers tend to get darker toward the end, making it hard to obtain quality images. The results are averaged across all encoder types, so they offer a fair relative comparison between the model encoders.

The ViTb encoder consistently outperforms the ViTs encoder in terms of absolute error for algorithmic and manual ground truth measurements. For instance, the VKITTI-Finetuned model with a ViTb encoder achieves an absolute algorithmic error of 17.07 cubic meters and 15.10 cubic meters for the manual measurement, compared to 20.44 and 18.31 for the ViT encoder. This suggests that the larger ViTb encoder is effective in capturing better spatial relationships. The pre-trained ViTl encoder from both VKITTI and Hypersim datasets shows the best performance among all encoders, even without fine-tuning. This suggests that larger parameter models result in more precise volume predictions. Fine-tuned models generally outperform their pre-trained counterparts, particularly for the ViTb encoder type, achieving slightly better accuracy and lower absolute errors.
However, the Hypersim-PreTrained model with ViTb also performs well, indicating that the

dataset's quality plays a significant role in achieving accurate volume estimations. Evidence from the Depth Anything V2 paper [15] supports the idea that depth estimation models perform better when trained on large-scale synthetic datasets. This approach improves transfer from simulations to real-world applications and enhances domain adaptation. Additionally, these models benefit from having a sufficiently large encoder size and implementing an appropriate training strategy.

Looking at error trends in Figure 5.7, we observe that the Mean Absolute Error (MAE) generally increases as ground truth volume increases (container size increases). The MAE is relatively low for smaller containers (around 20-30 cubic meters), typically below 20 cubic meters. However, as the container size grows beyond 40 cubic meters, the MAE rises significantly and shows variance, suggesting that the algorithm may not be suitable for larger or empty containers. The graph also indicates that the errors are more spread apart and unstable for larger containers. This instability could be attributed to factors like lighting conditions, image resolution, and the complexity of imaging very long containers, which become problematic on cameras with a short focal length.

Depth estimation is a critical step in the pipeline, and inaccuracies or inconsistencies in depth measurements can lead to decreased performance later in the process. The graph illustrating the MAE across different container sizes emphasizes this issue, particularly for larger containers where errors become more pronounced and unstable. This performance degradation in larger volumes is likely a direct result of inaccuracies in depth estimation, which tend to compound as container size increases.

To ensure reliable performance across all container sizes, it is essential to explore outlier rejection and failure detection methods[25]. These approaches can help identify and mitigate errors in depth estimation, especially under challenging conditions such as low-resolution imaging or poor lighting, which are more common with larger containers. Moreover, enhancing the robustness of depth estimation through improved data quality, data-model scaling, and optimized loss functions could further minimize errors and ensure more consistent performance throughout the pipeline. By addressing these challenges, the system can potentially achieve more reliable volume estimation, even for larger and more complex containers.

5.5 Results on Real-Time Performance

The real-time performance evaluation (Figure 5.9) of the pipeline reveals a trade-off between computational efficiency and accuracy, particularly in the depth estimation stage. The ViTs encoder completes depth estimation in 0.5 seconds, while the larger ViTb encoder takes 1.2 seconds, reflecting its increased parameter count. However, ViTb's higher-quality depth estimates lead to more geometrically consistent point clouds, resulting in slightly faster mesh reconstruction times (2.94 seconds compared to 3.2 seconds for ViTs), suggesting that inaccurate depth

[25] See appendix A

Model Name	Encoder	Ground Truth Volume	Predicted Volume	Absolute Error-Algorithmic	Absolute Error-Manual
VKITTI-FineTuned	ViTs	Manual: 33.7000 Algorithmic: 35.4255	15.4999	20.4488	18.3216
VKITTI-PreTrained	ViTs	Manual: 33.7000 Algorithmic: 35.4255	11.3465	24.0789	22.3534
Hypersim-PreTrained	ViTs	Manual: 33.7000 Algorithmic: 35.4255	17.1677	18.2577	16.5322
VKITTI-FineTuned	ViTb	Manual: 33.7000 Algorithmic: 35.4255	18.8110	**17.0713**	**15.1025**
VKITTI-PreTrained	ViTb	Manual: 33.7000 Algorithmic: 35.4255	20.0296	17.2211	15.6976
Hypersim-PreTrained	ViTb	Manual: 33.7000 Algorithmic: 35.4255	20.8792	**16.7953**	**15.0532**
VKITTI-PreTrained	ViTl	Manual: 33.7000 Algorithmic: 35.4255	22.7810	**14.9821**	**13.4757**
Hypersim-PreTrained	ViTl	Manual: 33.7000 Algorithmic: 35.4255	20.4971	14.9283	20.4971

Table 5.5: Results on volume estimation for NavVis container data.

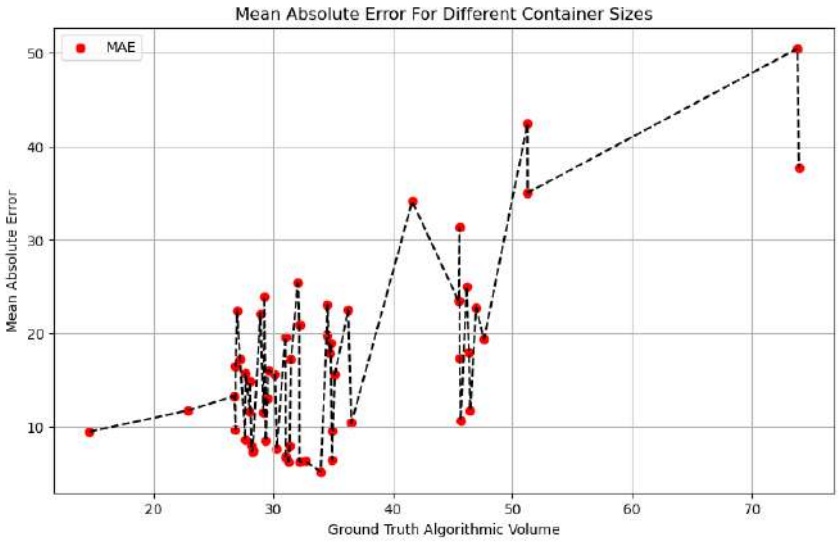


Figure 5.7: Mean Absolute Error in volume across all ViTb checkpoints for 55 containers.

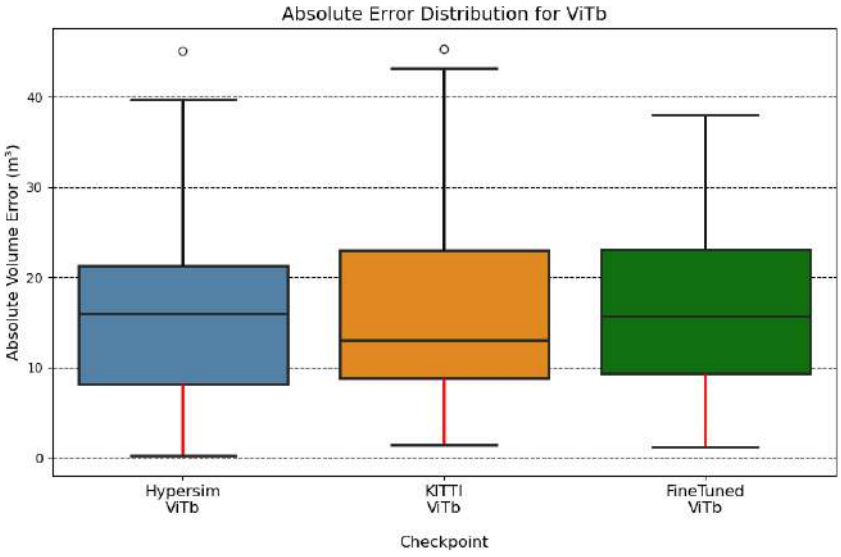


Figure 5.8: Absolute error in volume for all ViTb checkpoints across containers.

estimates from ViTs may introduce spatial inconsistencies, increasing processing time in later stages. Point cloud processing and standard estimation remain consistent across both encoders, taking approximately 0.52–0.54 seconds, indicating that these stages are less affected by encoder size. With point cloud downsampling by a factor of 10, the entire pipeline completes in roughly 5 seconds, making it suitable for real-time applications (Figure 5.10).

Although the performance shows promise, further optimizations are required to improve efficiency and achieve high accuracy. Parallel processing techniques could accelerate computationally intensive stages, such as depth estimation and mesh reconstruction, while adaptive down sampling strategies may better preserve geometric details. Additionally, robust outlier rejection methods could mitigate failure cases caused by inaccurate depth estimates, particularly in the mesh reconstruction stage. These improvements would strengthen the pipeline's real-time capabilities, ensuring reliable performance across diverse scenarios. Overall, the results demonstrate that the pipeline balances speed and accuracy, with ViTb offering superior depth estimation quality and ViTs providing a faster alternative for time-sensitive applications.

As we move toward a future where spatial computing technologies continue to evolve and play a pivotal role across various applications, integrating deep learning-based depth estimation with cost-effective consumer-grade LiDAR sensors offers a compelling roadmap to achieving robust precision compared to learning-based approaches alone. In addition, various visual SLAM algorithms developed for augmented reality and robotics can be utilized to enhance the accuracy of 3D reconstruction on edge devices.

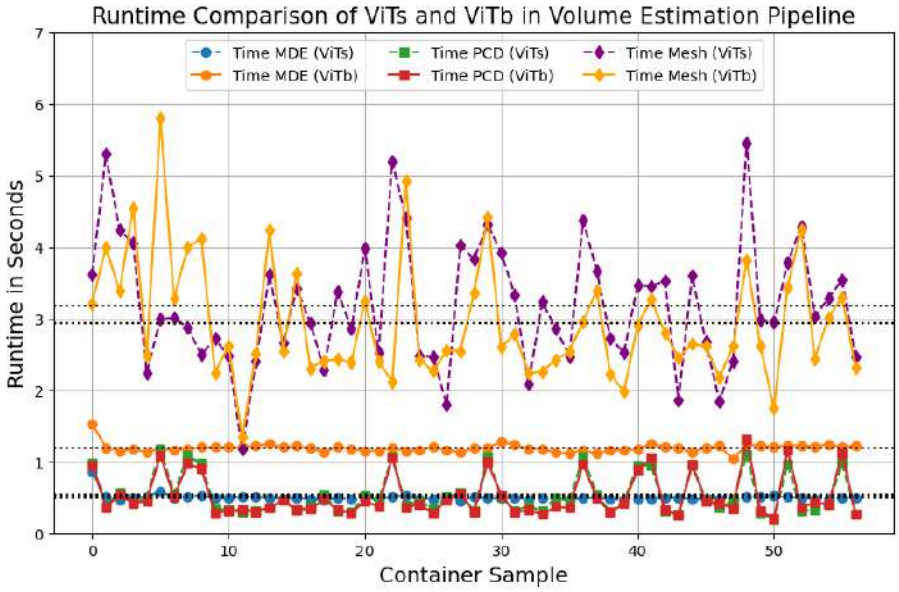


Figure 5.9: Comparison of runtime performance between ViTs and ViTb encoders across different stages (MDE, PCD, and Mesh) in the volume estimation pipeline. The plot shows the runtime for each container sample, with dashed horizontal black lines indicating the mean runtime for each stage.

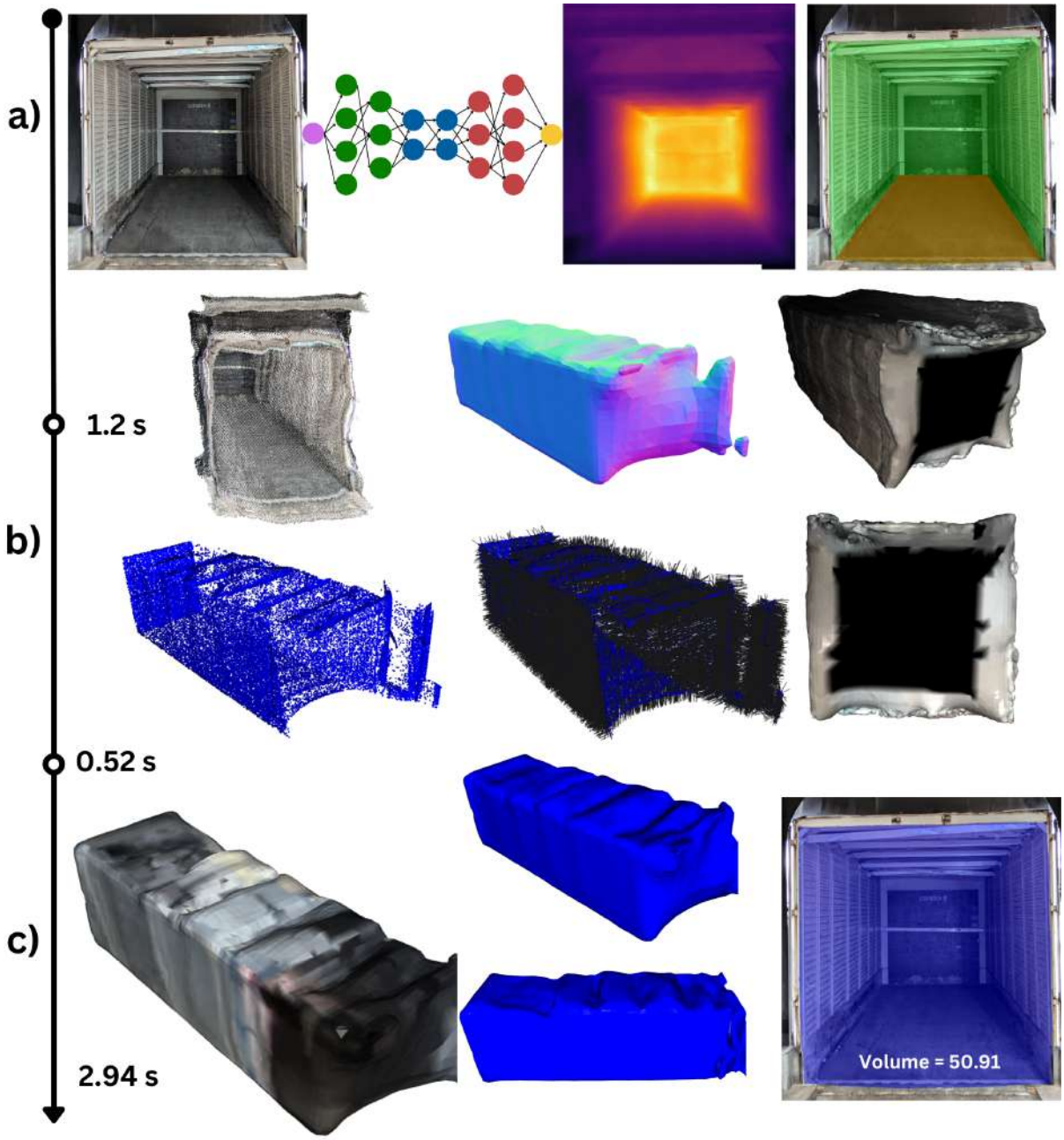


Figure 5.10: End-to-End volume estimation pipeline using ViTb. (a) Single-image depth map generation via neural network is fused with gate detection to isolate container mesh (≈1.2 s). (b) Mesh processing: downsampling, normal correction, and Poisson reconstruction (≈0.52 s). (c) Closed mesh volume and floor area calculation (≈2.94 s), yielding an estimated volume of 50.91 cubic meters.

6 Conclusions

We have witnessed rapid computer vision and deep learning advances in the past few years. These technologies are anticipated to improve further as hardware capabilities and data availability grow in combination with enhanced model architectures. The role of simulator and synthetic data generation pipelines have been instrumental for advancements in robotics and computer vision research, enabling applications like the novel work presented in this thesis to make metric measurements using state-of-the-art models in monocular depth estimation. It evaluated the end-to-end pipeline in various stages, focusing on its performance across different datasets, encoders, and real-time capabilities. The key findings highlight the importance of encoder size, data quality, and computational efficiency in achieving accurate and reliable results. The ViTb and ViTl encoders consistently outperformed the smaller ViTs encoder for depth estimation and volume reconstruction tasks, demonstrating their ability to capture local and global spatial information. However, this comes at the cost of increased computational complexity, particularly in the depth estimation stage, where ViTb takes 1.2 seconds compared to 0.5 seconds for ViTs. Despite this trade-off, the higher-quality depth estimates from ViTb lead to more geometrically consistent point clouds, resulting in faster mesh reconstruction times and improved overall accuracy.
Our developed pipeline achieves real-time performance, completing the entire process in approximately 5 seconds with point cloud downsampling, proving its practicality for applications on mobile devices in busy warehouses.

However, challenges remain, particularly in handling larger containers and mitigating errors caused by inaccurate depth estimates. Future work should focus on scaling up model training by generating large-scale synthetic datasets of warehouse scenes using high-fidelity simulators such as the Nvidia Isaac Sim. Further optimizations to the pipeline through parallel processing, adaptive downsampling, and robust outlier rejection methods would enhance efficiency and reliability. Additionally, exploring alternative loss functions and larger datasets could improve generalization across diverse imaging conditions. In conclusion, this project demonstrates the feasibility of real-time volume estimation using monocular depth estimation. From the model sizes we fine-tuned, ViTb offers superior accuracy, and ViTs provide a faster alternative for time-sensitive applications. These insights pave the way for further research in robotics and computer vision enabling more efficient perception-based algorithms in real-world scenarios, especially on edge devices such as mobile devices and mixed-reality wearable technologies.

References

[1] R. Szeliski, *Computer Vision: Algorithms and Applications*. Texts in Computer Science, Cham: Springer International Publishing, 2022.

[2] O. Ozyesil, V. Voroninski, R. Basri, and A. Singer, "A survey of structure from motion," 2017.

[3] F. Courteille, A. Crouzil, J.-D. Durou, and P. Gurdjos, "3d-spline reconstruction using shape from shading: Spline from shading," *Image and Vision Computing*, vol. 26, no. 4, pp. 466–479, 2008.

[4] A. Baudron, Z. W. Wang, O. Cossairt, and A. K. Katsaggelos, "E3d: Event-based 3d shape reconstruction," 2020.

[5] W. Agnew, C. Xie, A. Walsman, O. Murad, C. Wang, P. M. Domingos, and S. Srinivasa, "Amodal 3D Reconstruction for Robotic Manipulation via Stability and Connectivity," Sept. 2020.

[6] N. Simon and A. Majumdar, "MonoNav: MAV Navigation via Monocular Depth Estimation and Reconstruction," in *Symposium on Experimental Robotics (ISER)*, 2023.

[7] X. Dong, M. A. Garratt, S. G. Anavatti, and H. A. Abbass, "Towards real-time monocular depth estimation for robotics: A survey," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 10, pp. 16940–16961, 2022.

[8] R. Baskar, P. M. Krishnammal, A. Aeron, S. S. Ali, T. T. Leonid, and M. R. Arun, "3d image reconstruction and processing for augmented and virtual reality applications: A computer generated environment," in *2023 International Conference on Communication, Security and Artificial Intelligence (ICCSAI)*, p. 866–870, IEEE, Nov. 2023.

[9] J. Sun, Y. Xie, L. Chen, X. Zhou, and H. Bao, "Neuralrecon: Real-time coherent 3d reconstruction from monocular video," 2021.

[10] B. Ahmad, P. A. Floor, I. Farup, and C. F. Andersen, "Single-image-based 3d reconstruction of endoscopic images," *Journal of Imaging*, vol. 10, p. 82, Mar. 2024.

[11] C. Xu, Z. Liu, Y. Liu, Y. Dou, J. Wu, J. Wang, M. Wang, D. Shen, and Z. Cui, "Teethdreamer: 3d teeth reconstruction from five intra-oral photographs," 2024.

[12] C. Marín-Buzón, A. Pérez-Romero, J. L. López-Castro, I. Ben Jerbania, and F. Manzano-Agugliaro, "Photogrammetry as a new scientific tool in archaeology: Worldwide research trends," *Sustainability*, vol. 13, p. 5319, May 2021.

[13] G. Guidi, M. Russo, and D. Angheleddu, “3d survey and virtual reconstruction of archeological sites,” *Digital Applications in Archaeology and Cultural Heritage*, vol. 1, no. 2, p. 55–69, 2014.

[14] S. F. Bhat, R. Birkl, D. Wofk, P. Wonka, and M. Müller, “Zoedepth: Zero-shot transfer by combining relative and metric depth,” 2023.

[15] L. Yang, B. Kang, Z. Huang, Z. Zhao, X. Xu, J. Feng, and H. Zhao, “Depth anything v2,” 2024.

[16] B. Maettig, F. Hering, and M. Doeltgen, “Development of an intuitive, visual packaging assistant,” in *Advances in Human Factors and Systems Interaction* (I. L. Nunes, ed.), (Cham), pp. 19–25, Springer International Publishing, 2019.

[17] R. N. Boute and M. Udenio, *AI in Logistics and Supply Chain Management*, pp. 49–65. Cham: Springer International Publishing, 2023.

[18] A. Jugović, “Economic impact of container-loading problem,” *Transactions on Maritime Science*, 2020.

[19] J. Xue and K. Lai, “Effective methods for a container packing operation,” *Mathematical and Computer Modelling*, vol. 25, no. 2, pp. 75–84, 1997.

[20] W. Chen, Y. Men, N. Fuster, C. Osorio, and A. A. Juan, “Artificial intelligence in logistics optimization with sustainable criteria: A review,” *Sustainability*, vol. 16, no. 21, 2024.

[21] K. Tateno, F. Tombari, I. Laina, and N. Navab, “CNN-SLAM: Real-time dense monocular SLAM with learned depth prediction,” Apr. 2017. arXiv:1704.03489.

[22] Z. Zhang, H. Jiang, and H. Singh, “NeuFlow: Real-time, High-accuracy Optical Flow Estimation on Robots Using Edge Devices,” Mar. 2024. arXiv:2403.10425.

[23] B. Yu, J. Wu, and M. J. Islam, “UDepth: Fast Monocular Depth Estimation for Visually-guided Underwater Robots,” Feb. 2023. arXiv:2209.12358.

[24] R. Mur-Artal and J. D. Tardós, “ORB-SLAM2: an open-source SLAM system for monocular, stereo and RGB-D cameras,” *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255–1262, 2017.

[25] M. Maimone, Y. Cheng, and L. Matthies, “Two years of visual odometry on the mars exploration rovers,” *Journal of Field Robotics*, vol. 24, no. 3, pp. 169–186, 2007.

[26] Y. Ze, Z. Chen, W. Wang, T. Chen, X. He, Y. Yuan, X. B. Peng, and J. Wu, “Generalizable humanoid manipulation with improved 3d diffusion policies,” *arXiv preprint arXiv:2410.10803*, 2024.

[27] H. Yin, X. Xu, S. Lu, X. Chen, R. Xiong, S. Shen, C. Stachniss, and Y. Wang, "A survey on global lidar localization: Challenges, advances and open problems," 2024.

[28] B. G. Greenland, J. Pinskier, X. Wang, D. Nguyen, G. Shi, T. Bandyopadhyay, J. J. Chung, and D. Howard, "Sograb: A visual method for soft grasping benchmarking and evaluation," 2024.

[29] A. Bonci, P. D. Cen Cheng, M. Indri, G. Nabissi, and F. Sibona, "Human-robot perception in industrial environments: A survey," *Sensors*, vol. 21, no. 5, 2021.

[30] H. D. Herath, S. Kodagoda, and G. Dissanayake, *Stereo Vision Based SLAM: Issues and Solutions*. ITECH, Jan. 2007.

[31] S.-W. Yang and C.-C. Wang, "On solving mirror reflection in lidar sensing," *IEEE/ASME Transactions on Mechatronics*, vol. 16, no. 2, pp. 255–265, 2011.

[32] A. Gawel, T. Cieslewski, R. Dubé, M. Bosse, R. Siegwart, and J. Nieto, "Structure-based vision-laser matching," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 182–188, 2016.

[33] G. Fahim, K. Amin, and S. Zarif, "Single-view 3d reconstruction: A survey of deep learning methods," *Computers Graphics*, vol. 94, pp. 164–190, 2021.

[34] H. Luo, J. Zhang, X. Liu, L. Zhang, and J. Liu, "Large-Scale 3D Reconstruction from Multi-View Imagery: A Comprehensive Review," *Remote Sensing*, vol. 16, p. 773, Feb. 2024.

[35] F. Rong, D. Xie, W. Zhu, H. Shang, and L. Song, "A survey of multi view stereo," in *2021 International Conference on Networking Systems of AI (INSAI)*, pp. 129–135, 2021.

[36] Y. Yao, Z. Luo, S. Li, T. Fang, and L. Quan, "Mvsnet: Depth inference for unstructured multi-view stereo," 2018.

[37] S. Wang, V. Leroy, Y. Cabon, B. Chidlovskii, and J. Revaud, "Dust3r: Geometric 3d vision made easy," 2024.

[38] C. B. Choy, D. Xu, J. Gwak, K. Chen, and S. Savarese, "3d-r2n2: A unified approach for single and multi-view 3d object reconstruction," 2016.

[39] K. Malek and F. Moreu, "Methodology to deploy cnn-based computer vision models on immersive wearable devices," 2024.

[40] C. Feng, C. Zhang, Z. Chen, W. Hu, and L. Ge, "Real-time monocular depth estimation on embedded systems," 2024.

[41] L. Zhou, G. Wu, Y. Zuo, X. Chen, and H. Hu, “A comprehensive review of vision-based 3d reconstruction methods,” *Sensors*, vol. 24, p. 2314, 2024.

[42] D. Eigen, C. Puhrsch, and R. Fergus, “Depth map prediction from a single image using a multi-scale deep network,” 2014.

[43] L. Yang, B. Kang, Z. Huang, X. Xu, J. Feng, and H. Zhao, “Depth anything: Unleashing the power of large-scale unlabeled data,” 2024.

[44] J. Cao, R. Huang, Z. Ye, Y. Xu, and X. Tong, “Generative 3d reconstruction of martian surfaces using monocular images,” *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 2024.

[45] J. Spencer, C. S. Qian, M. Trescakova, *et al.*, “The second monocular depth estimation challenge,” 2023.

[46] V. Guizilini, I. Vasiljevic, D. Chen, R. Ambrus, and A. Gaidon, “Towards zero-shot scale-aware monocular depth estimation,” 2023.

[47] D. Hoiem, A. A. Efros, and M. Hebert, “Recovering surface layout from an image,” *International Journal of Computer Vision*, vol. 75, pp. 151–172, 2007.

[48] A. Saxena, M. Sun, and A. Y. Ng, “Learning 3-d scene structure from a single still image,” in *2007 IEEE 11th International Conference on Computer Vision*, pp. 1–8, 2007.

[49] R. Guo, C. Zou, and D. Hoiem, “Predicting complete 3d models of indoor scenes,” *ArXiv*, vol. abs/1504.02437, 2015.

[50] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, “An image is worth 16x16 words: Transformers for image recognition at scale,” 2021.

[51] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is all you need,” 2023.

[52] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, “Nerf: Representing scenes as neural radiance fields for view synthesis,” 2020.

[53] T. Müller, A. Evans, C. Schied, and A. Keller, “Instant neural graphics primitives with a multiresolution hash encoding,” *ACM Transactions on Graphics*, vol. 41, p. 1–15, July 2022.

[54] W. Zhang, Y.-S. Liu, and Z. Han, “Neural signed distance function inference through splatting 3d gaussians pulled on zero-level set,” 2024.

[55] S. Szymanowicz, C. Rupprecht, and A. Vedaldi, “Splatter image: Ultra-fast single-view 3d reconstruction,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10208–10217, June 2024.

[56] J. Spencer, F. Tosi, M. Poggi, *et al.*, “The third monocular depth estimation challenge,” 2024.

[57] A. Ignatov, G. Malivenko, R. Timofte, *et al.*, “Efficient single-image depth estimation on mobile devices, mobile ai aim 2022 challenge: Report,” 2022.

[58] S. Farooq Bhat, I. Alhashim, and P. Wonka, “Adabins: Depth estimation using adaptive bins,” in *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, p. 4008–4017, IEEE, June 2021.

[59] B. Ke, A. Obukhov, S. Huang, N. Metzger, R. C. Daudt, and K. Schindler, “Repurposing diffusion-based image generators for monocular depth estimation,” 2024.

[60] H. Fu, M. Gong, C. Wang, K. Batmanghelich, and D. Tao, “Deep ordinal regression network for monocular depth estimation,” 2018.

[61] Z. Li and N. Snavely, “Megadepth: Learning single-view depth prediction from internet photos,” in *Computer Vision and Pattern Recognition (CVPR)*, 2018.

[62] B. Gabdullin, N. Konovalova, N. Patakin, D. Senushkin, and A. Konushin, “Depthart: Monocular depth estimation as autoregressive refinement task,” 2024.

[63] A. Agarwal and C. Arora, “Depthformer: Multiscale vision transformer for monocular depth estimation with global local information fusion,” in *2022 IEEE International Conference on Image Processing (ICIP)*, pp. 3873–3877, 2022.

[64] A. Agarwal and C. Arora, “Attention attention everywhere: Monocular depth prediction with skip attention,” 2022.

[65] S. Shao, Z. Pei, X. Wu, Z. Liu, W. Chen, and Z. Li, “Iebins: Iterative elastic bins for monocular depth estimation,” 2023.

[66] W. Yin, J. Zhang, O. Wang, S. Niklaus, L. Mai, S. Chen, and C. Shen, “Learning to recover 3d scene shape from a single image,” 2020.

[67] J. Liu, Q. Li, R. Cao, W. Tang, and G. Qiu, “A contextual conditional random field network for monocular depth estimation,” *Image and Vision Computing*, vol. 98, p. 103922, 2020.

[68] V. Guizilini, R. Ambrus, S. Pillai, A. Raventos, and A. Gaidon, “3d packing for self-supervised monocular depth estimation,” 2020.

[69] Z. Li, S. F. Bhat, and P. Wonka, "Patchfusion: An end-to-end tile-based framework for high-resolution monocular metric depth estimation," 2023.

[70] Y. Zheng and C. Cao, "Monocular depth estimation with multiscale feature fusion networks," in *2022 2nd International Conference on Algorithms, High Performance Computing and Artificial Intelligence (AHPCAI)*, pp. 742–746, 2022.

[71] R. Ranftl, A. Bochkovskiy, and V. Koltun, "Vision transformers for dense prediction," 2021.

[72] R. Ranftl, K. Lasinger, D. Hafner, K. Schindler, and V. Koltun, "Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer," 2020.

[73] W. Yin, C. Zhang, H. Chen, Z. Cai, G. Yu, K. Wang, X. Chen, and C. Shen, "Metric3d: Towards zero-shot metric 3d prediction from a single image," 2023.

[74] M. Hu, W. Yin, C. Zhang, Z. Cai, X. Long, H. Chen, K. Wang, G. Yu, C. Shen, and S. Shen, "Metric3d v2: A versatile monocular geometric foundation model for zero-shot metric depth and surface normal estimation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 46, p. 10579–10596, Dec. 2024.

[75] A. Bochkovskii, A. Delaunoy, H. Germain, M. Santos, Y. Zhou, S. R. Richter, and V. Koltun, "Depth pro: Sharp monocular metric depth in less than a second," 2024.

[76] L. Piccinelli, Y.-H. Yang, C. Sakaridis, M. Segu, S. Li, L. V. Gool, and F. Yu, "Unidepth: Universal monocular metric depth estimation," 2024.

[77] M. Kazhdan, M. Bolitho, and H. Hoppe, "Poisson surface reconstruction," in *Proceedings of the Fourth Eurographics Symposium on Geometry Processing*, SGP '06, (Goslar, DEU), p. 61–70, Eurographics Association, 2006.

[78] R. Hanocka, G. Metzer, R. Giryes, and D. Cohen-Or, "Point2mesh: a self-prior for deformable meshes," *ACM Transactions on Graphics*, vol. 39, Aug. 2020.

[79] J. Huang, Z. Gojcic, M. Atzmon, O. Litany, S. Fidler, and F. Williams, "Neural kernel surface reconstruction," 2023.

[80] R. Sulzer, R. Marlet, B. Vallet, and L. Landrieu, "A survey and benchmark of automatic surface reconstruction from point clouds," 2024.

[81] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. New York, NY, USA: Cambridge University Press, 2 ed., 2003.

[82] Supertek Module, "Understanding lens focal length: A guide to choosing the right lens for your camera." `https://tinyurl.com/uchvb9mb`, N/A. [Online; Accessed: October 12, 2024.

[83] J. Xu, D.-W. Han, K. Li, J.-J. Li, and Z.-Y. Ma, "A comprehensive overview of fish-eye camera distortion correction methods," 2024.

[84] D. Brown, "Close-range camera calibration," in *Photogrammetric Engineering, 37, 855-866*, 1971.

[85] Stack Exchange Community, "Manual lens distortion: continuously stretching image." `https://tinyurl.com/4t3kse8x`, Feb 2021. [Online; Accessed: October 12, 2024].

[86] Z. Zhang, "A flexible new technique for camera calibration," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, p. 1330–1334, Nov. 2000.

[87] B. K. P. Horn and Copright, "Tsai's camera calibration method revisited," in *MIT Press*, 2003.

[88] K. Ikeuchi, ed., *Computer Vision, A Reference Guide*. Springer, 2014.

[89] G. Pudics, M. Z. Szabó-Resch, and Z. Vámossy, "Safe robot navigation using an omnidirectional camera," in *2015 16th IEEE International Symposium on Computational Intelligence and Informatics (CINTI)*, pp. 227–231, 2015.

[90] D. Scaramuzza, A. Martinelli, and R. Y. Siegwart, "A flexible technique for accurate omnidirectional camera calibration and structure from motion," *Fourth IEEE International Conference on Computer Vision Systems (ICVS'06)*, pp. 45–45, 2006.

[91] D. Scaramuzza, A. Martinelli, and R. Siegwart, "A Toolbox for Easily Calibrating Omnidirectional Cameras," *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2006.

[92] J. Kannala and S. Brandt, "A generic camera calibration method for fish-eye lenses," in *Proceedings of the Pattern Recognition, 17th International Conference on (ICPR'04) Volume 1 - Volume 01*, ICPR '04, (USA), p. 10–13, IEEE Computer Society, 2004.

[93] D. Scaramuzza, "Omnidirectional camera," in *Computer Vision: A Reference Guide* (K. Ikeuchi, ed.), pp. 85–88, Springer US, 2014.

[94] D. Scaramuzza, *Omnidirectional vision: From calibration to root motion estimation*. PhD thesis, ETH Zurich, 2007.

[95] N. Qian, "Binocular disparity and the perception of depth," *Neuron*, vol. 18, p. 359–368, Mar. 1997.

[96] M. Masry and H. Lipson, "A sketch-based interface for iterative design and analysis of 3d objects," in *ACM SIGGRAPH 2007 Courses*, SIGGRAPH '07, (New York, NY, USA), p. 31–es, Association for Computing Machinery, 2007.

[97] C. Campos, R. Elvira, J. J. G. Rodriguez, J. M. M. Montiel, and J. D. Tardos, "Orb-slam3: An accurate open-source library for visual, visual–inertial, and multimap slam," *IEEE Transactions on Robotics*, vol. 37, p. 1874–1890, Dec. 2021.

[98] W. Chen, Z. Fu, D. Yang, and J. Deng, "Single-image depth perception in the wild," *ArXiv*, vol. abs/1604.03901, 2016.

[99] Z. Li, X. Wang, X. Liu, and J. Jiang, "Binsformer: Revisiting adaptive bins for monocular depth estimation," 2022.

[100] H. Basak, S. Ghosal, M. Sarkar, M. Das, and S. Chattopadhyay, "Monocular depth estimation using encoder-decoder architecture and transfer learning from single rgb image," in *2020 IEEE 7th Uttar Pradesh Section International Conference on Electrical, Electronics and Computer Engineering (UPCON)*, pp. 1–6, 2020.

[101] G. Irie, T. Kawanishi, and K. Kashino, "Robust learning for deep monocular depth estimation," in *2019 IEEE International Conference on Image Processing (ICIP)*, pp. 964–968, 2019.

[102] S. Stabinger, D. Peer, and A. Rodríguez-Sánchez, "Arguments for the unsuitability of convolutional neural networks for non-local tasks," *Neural Networks*, vol. 142, pp. 171–179, 2021.

[103] Z. Wang and L. Wu, "Theoretical analysis of inductive biases in deep convolutional networks," 2024.

[104] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie, "A convnet for the 2020s," 2022.

[105] M. Oquab, T. Darcet, T. Moutakanni, *et al.*, "Dinov2: Learning robust visual features without supervision," 2024.

[106] H. Bao, L. Dong, S. Piao, and F. Wei, "Beit: Bert pre-training of image transformers," 2022.

[107] Z. Liu, H. Hu, Y. Lin, Z. Yao, Z. Xie, Y. Wei, J. Ning, Y. Cao, Z. Zhang, L. Dong, F. Wei, and B. Guo, "Swin transformer v2: Scaling up capacity and resolution," 2022.

[108] M. Roberts, J. Ramapuram, A. Ranjan, A. Kumar, M. A. Bautista, N. Paczan, R. Webb, and J. M. Susskind, "Hypersim: A photorealistic synthetic dataset for holistic indoor scene understanding," 2021.

[109] A. Rajpal, N. Cheema, K. Illgner-Fehns, P. Slusallek, and S. Jaiswal, "High-resolution synthetic rgb-d datasets for monocular depth estimation," 2023.

[110] A. Syed and B. T. Morris, "Sseg-lstm: Semantic scene segmentation for trajectory prediction," *2019 IEEE Intelligent Vehicles Symposium (IV)*, pp. 2504–2509, 2019.

[111] M. Poddar, A. Mishra, M. Kewlani, and H. Pei, "Self-supervised learning based depth estimation from monocular images," 2023.

[112] P. Goyal, Q. Duval, I. Seessel, M. Caron, I. Misra, L. Sagun, A. Joulin, and P. Bojanowski, "Vision models are more robust and fair when pretrained on uncurated images without supervision," *ArXiv*, vol. abs/2202.08360, 2022.

[113] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, "Surface reconstruction from unorganized points," in *Proceedings of the 19th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '92, (New York, NY, USA), p. 71–78, Association for Computing Machinery, 1992.

[114] Z. Huang, Y. Wen, Z. Wang, J. Ren, and K. Jia, "Surface reconstruction from point clouds: A survey and a benchmark," 2022.

[115] J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove, "Deepsdf: Learning continuous signed distance functions for shape representation," 2019.

[116] M. Jia and M. Kyan, "Learning occupancy function from point clouds for surface reconstruction," 2020.

[117] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3d surface construction algorithm," *SIGGRAPH Comput. Graph.*, vol. 21, p. 163–169, Aug. 1987.

[118] B. Anderson, "Marching cubes." `https://www.cs.carleton.edu/cs_comps/0405/shape/marching_cubes.html`, 2017. [Online; accessed 10-Oct-2023].

[119] S. Kim, S. Rhee, and T. Kim, "Digital surface model interpolation based on 3d mesh models," *Remote Sensing*, vol. 11, p. 24, Dec. 2018.

[120] C. Zhao, Q. Sun, C. Zhang, Y. Tang, and F. Qian, "Monocular depth estimation based on deep learning: An overview," *Science China Technological Sciences*, vol. 63, p. 1612–1627, June 2020.

[121] N. Padkan, P. Trybala, R. Battisti, F. Remondino, and C. Bergeret, "Evaluating monocular depth estimation methods," *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. XLVIII-1/W3-2023, p. 137–144, Oct. 2023.

[122] M. Hafeez, M. Madden, G. Sistu, and I. Ullah, "Depth estimation using weighted-loss and transfer learning," in *Proceedings of the 19th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*, p. 780–787, SCITEPRESS - Science and Technology Publications, 2024.

[123] F. Mémoli and G. Sapiro, “Comparing point clouds,” in *Proceedings of the 2004 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, SGP ’04, (New York, NY, USA), p. 32–40, Association for Computing Machinery, 2004.

[124] T. Wu, L. Pan, J. Zhang, T. Wang, Z. Liu, and D. Lin, “Density-aware chamfer distance as a comprehensive metric for point cloud completion,” in *Proceedings of the 35th International Conference on Neural Information Processing Systems*, NIPS ’21, (Red Hook, NY, USA), Curran Associates Inc., 2021.

[125] NavVis, “Navvis vlx 3 specifications,” 2023. Accessed: February 03, 2025.

[126] G. Staff, “Apple iphone 12 pro: What is lidar?,” 2023. Accessed: February 03, 2025.

[127] Livox Technology Company Limited, *Livox Horizon User Manual v1.0*, 2019. Accessed: 02.02.2025.

[128] G. Baruch, Z. Chen, A. Dehghan, T. Dimry, Y. Feigin, P. Fu, T. Gebauer, B. Joffe, D. Kurz, A. Schwartz, and E. Shulman, “Arkitscenes: A diverse real-world dataset for 3d indoor scene understanding using mobile rgb-d data,” 2022.

[129] K. Li, J.-W. Bian, R. Castle, P. H. S. Torr, and V. A. Prisacariu, “Mobilebrick: Building lego for 3d reconstruction on mobile devices,” 2023.

[130] G. R. Tondo, C. Riley, and G. Morgenthal, “Characterization of the iphone lidar-based sensing system for vibration measurement and modal analysis,” *Sensors*, vol. 23, p. 7832, Sept. 2023.

[131] A. A.-W. Hazem M. Abdel-Majeed, Ibrahim F. Shaker and A. A. D. I. Awad, “Indoor mapping accuracy comparison between the apple devices’ lidar sensor and terrestrial laser scanner,” *HBRC Journal*, vol. 20, no. 1, pp. 915–931, 2024.

[132] Teledyne FLIR, “Blackfly s usb3 camera,” 2024. Accessed: 02.02.2025.

[133] C. Yuan, X. Liu, X. Hong, and F. Zhang, “Pixel-level extrinsic self calibration of high resolution lidar and camera in targetless environments,” 2021.

[134] I. Vasiljevic, N. Kolkin, S. Zhang, R. Luo, H. Wang, F. Z. Dai, A. F. Daniele, M. Mostajabi, S. Basart, M. R. Walter, and G. Shakhnarovich, “DIODE: A Dense Indoor and Outdoor DEpth Dataset,” *CoRR*, vol. abs/1908.00463, 2019.

[135] S. Höfer, K. Bekris, A. Handa, J. C. Gamboa, M. Mozifian, F. Golemo, C. Atkeson, D. Fox, K. Goldberg, J. Leonard, C. Karen Liu, J. Peters, S. Song, P. Welinder, and M. White, “Sim2real in robotics and automation: Applications and challenges,” *IEEE Transactions on Automation Science and Engineering*, vol. 18, no. 2, pp. 398–400, 2021.

[136] J.-F. Cai, Z. Chen, X.-M. Wu, J.-J. Jiang, Y.-L. Wei, and W.-S. Zheng, "Real-to-sim grasp: Rethinking the gap between simulation and real world in grasp detection," in *8th Annual Conference on Robot Learning*, 2024.

[137] E. Bonetto, C. Xu, and A. Ahmad, "Grade: Generating realistic and dynamic environments for robotics research with isaac sim," 2024.

[138] J. Kopf, M. F. Cohen, D. Lischinski, and M. Uyttendaele, "Joint bilateral upsampling," *ACM Trans. Graph.*, vol. 26, p. 96–es, July 2007.

[139] K. He, J. Sun, and X. Tang, "Guided image filtering," in *Proceedings of the 11th European Conference on Computer Vision: Part I*, ECCV'10, (Berlin, Heidelberg), p. 1–14, Springer-Verlag, 2010.

[140] F. Durand and J. Dorsey, "Fast bilateral filtering for the display of high-dynamic-range images," *ACM Trans. Graph.*, vol. 21, p. 257–266, July 2002.

[141] T. Akenine-Mller, E. Haines, and N. Hoffman, *Real-Time Rendering, Fourth Edition*. USA: A. K. Peters, Ltd., 4th ed., 2018.

[142] M. Bertalmío, A. Bertozzi, and G. Sapiro, "Navier-stokes, fluid dynamics, and image and video inpainting," *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, vol. 1, pp. I–I, 2001.

[143] F. Tapia Benavides, A. Ignatov, and R. Timofte, "Phonedepth: A dataset for monocular depth estimation on mobile devices," in *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 3048–3055, 2022.

[144] C.-Y. Wu, Q. Gao, C.-C. Hsu, T.-L. Wu, J.-W. Chen, and U. Neumann, "Inspacetype: Dataset and benchmark for reconsidering cross-space type performance in indoor monocular depth," 2024.

[145] J. Cho, D. Min, Y. Kim, and K. Sohn, "Diml/cvl rgb-d dataset: 2m rgb-d images of natural indoor and outdoor scenes," 2021.

[146] Y. Wu, T. Y. Liu, H. Park, S. Soatto, D. Lao, and A. Wong, *AugUndo: Scaling Up Augmentations for Monocular Depth Completion and Estimation*, p. 274–293. Springer Nature Switzerland, Oct. 2024.

[147] Y. Ge, G. Xu, Z. Zhao, L. Sun, Z. Huang, Y. Sun, H. Chen, and C. Shen, "Geobench: Benchmarking and analyzing monocular geometry estimation models," 2024.

[148] X. Zhai, A. Kolesnikov, N. Houlsby, and L. Beyer, "Scaling vision transformers," 2022.

[149] A. Gaidon, Q. Wang, Y. Cabon, and E. Vig, “Virtual worlds as proxy for multi-object tracking analysis,” 2016.

[150] A. Steiner, A. Kolesnikov, X. Zhai, R. Wightman, J. Uszkoreit, and L. Beyer, “How to train your vit? data, augmentation, and regularization in vision transformers,” 2022.

[151] A. Veicht, P.-E. Sarlin, P. Lindenberger, and M. Pollefeys, “Geocalib: Learning single-image calibration with geometric optimization,” 2024.

[152] P. K. Nathan Silberman, Derek Hoiem and R. Fergus, “Indoor segmentation and support inference from rgbd images,” in *ECCV*, 2012.

[153] C. Lyu, W. Zhang, H. Huang, Y. Zhou, Y. Wang, Y. Liu, S. Zhang, and K. Chen, “Rtmdet: An empirical study of designing real-time object detectors,” 2022.

[154] B. Mirtich, “Fast and accurate computation of polyhedral mass properties,” *J. Graphics, GPU, & Game Tools*, vol. 1, pp. 31–50, 1996.

[155] Dawson-Haggerty *et al.*, “trimesh: A python library for 3d geometry processing.” [Online; Accessed: October 12, 2024].

List of Figures

List of Tables

List of Abbreviations

SfM Structure-from-Motion

SfS Shape-from-Shading

MVS Multi-View Stereo

ViTs Vision Transformers

DLT Direct Linear Transform

FOVs Fields of View

MDE Monocular Depth Estimation

VSLAM Visual Simultaneous Localization and Mapping

RNNs Recurrent Neural Networks

CNNs Convolutional Neural Networks

NLP Natural Language Processing

DINOv2 Distillation with No Labels v2

DPT Dense Prediction Transformer

SDF Signed Distance Functions

OF Occupancy Functions

SSIM Structural Similarity Index

HD Hausdorff Distance

CD Chamfer's Distance

EMD Earth Mover's Distance

DCD Density-Aware Chamfer Distance

SLAM Simultaneous Localization and Mapping

MP Megapixels

VCSEL Vertical Cavity Surface Emitting Laser

SPAD Single Photon Avalanche Diode

ROS Robot Operating System

FPS Frames Per Second

LUT Lookup Table

MAE Mean Absolute Error

RMSE Root Mean Squared Error

JBF Joint Bilateral Filtering

JBU Joint Bilateral Upsampling

ICP Point-to-Point Iterative Closest Point

DIODE Depth in the Wild

VRAM Video Random Access Memory

Hypersim Indoor Dataset (Hypersim)

VKITTI Virtual KITTI (also called VKITTI2)

AP Average Precision

ViT Vision Transformer

SiLog Scale-Invariant Logarithmic

MSE Mean Squared Error

ToF Time-of-Flight

NKSR Neural Kernel Surface Reconstruction

A Appendix

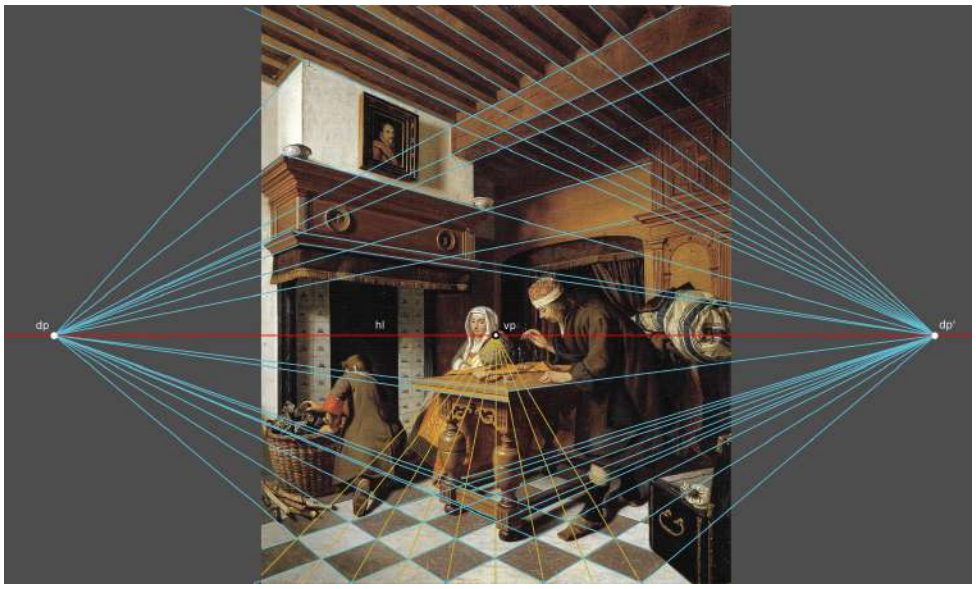


Figure A.1: Perspective reconstruction of Cornelis de Man's "The Goldweigher" (1621–1706). The painting shows lines leading up to the subject using vanishing points and the horizon, This highlights the importance of multi-view geometry and vanishing lines for depth perception (Source)

Figure A.2: Caravaggio's "The Calling of St Matthew" (1600) Illustrates the technique of "chiaroscuro," which translates to "light" and "dark," illustrating how the way light and shadows appear on objects can give us clues about their 3D shape (Source)

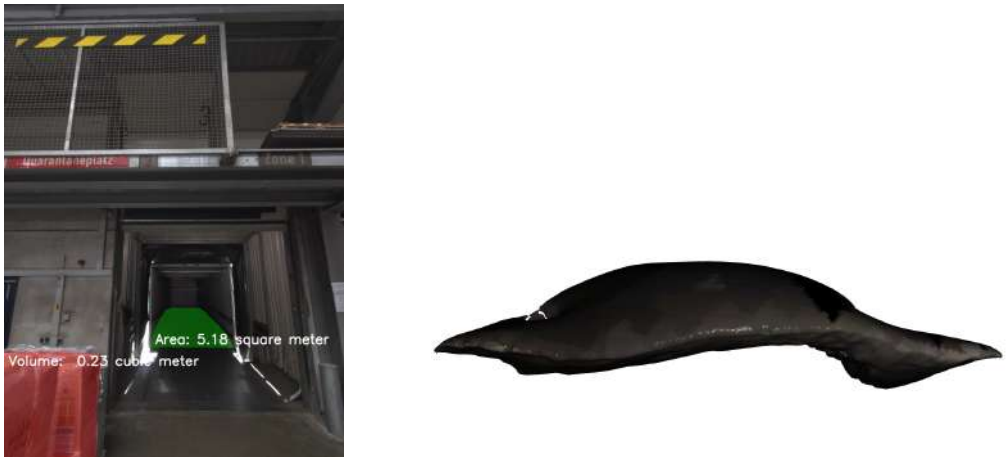
A.1 Quality of Processed Data

To ensure and quantify the loss of quality for the depth upsampling pipeline for our ARKit LiDAR data, we compare the upsampled depth maps to their original counterparts on the selected depth metrics and report the results for different upscaling factors of 2x, 4x, and 8x. The parallelized upsampling algorithm took roughly 15.2 seconds per image for upscaling a raw depth map of resolution 256x192 by 4x on an Intel Ultra 9 185H. It is evident from the table that the

Depth Images	Upsampling Factor	MAE	RMSE	ARE	δ1
300	2	0.0005809	0.0035621	0.0002034	0.9999970
300	4	0.0006076	0.0036131	0.0002121	0.9999970
300	8	0.00058402	0.0031349	0.0001837	0.9999985

Table A.1: Error Metrics for Depth Upsampling

upsampling algorithm has a minimal effect on the depth maps and maintains the original values with an error rate in the order of 10^{-3} for MAE and RMSE, and 10^{-6} for ARE. The threshold accuracy, δ1, indicates that over 99.9998% of upsampled depth values are within the given threshold. This demonstrates the effectiveness of our depth upsampling pipeline in preserving the integrity of the original low-resolution ARKit LiDAR depth maps. While these errors are reasonable for our task involving scenes with clear edges and for moderate upsampling factors, tasks involving augmented reality SLAM-based algorithms may require more robust methods [128].



(a) Edge case caused by occlusion causing bad gate detection.

(b) Poorly reconstructed mesh (with a UFO shape) due to geometrical inconsistencies in depth map.

Figure A.3: Visualization of failure cases.

Eidesstattliche Versicherung
(Affidavit)

Nedungadi, Ashwin Suresh
Name, Vorname
(Last name, first name)

230397
Matrikelnr.
(Enrollment number)

Ich versichere hiermit an Eides statt, dass ich die vorliegende Bachelorarbeit/Masterarbeit* mit dem folgenden Titel selbstständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

I declare in lieu of oath that I have completed the present Bachelor's/Master's* thesis with the following title independently and without any unauthorized assistance. I have not used any other sources or aids than the ones listed and have documented quotations and paraphrases as such. The thesis in its current or similar version has not been submitted to an auditing institution.

Titel der Bachelor-/Masterarbeit*:
(Title of the Bachelor's/ Master's* thesis):

Master Thesis

A Monocular Vision Architecture for Free Volume Estimation in Shipping Containers

*Nichtzutreffendes bitte streichen
(Please choose the appropriate)

05.02.2025
Ort, Datum
(Place, date)

A Nedungadi
Unterschrift
(Signature)

Belehrung:
Wer vorsätzlich gegen eine die Täuschung über Prüfungsleistungen betreffende Regelung einer Hochschulprüfungsordnung verstößt, handelt ordnungswidrig. Die Ordnungswidrigkeit kann mit einer Geldbuße von bis zu 50.000,00 € geahndet werden. Zuständige Verwaltungsbehörde für die Verfolgung und Ahndung von Ordnungswidrigkeiten ist der Kanzler/die Kanzlerin der Technischen Universität Dortmund. Im Falle eines mehrfachen oder sonstigen schwerwiegenden Täuschungsversuches kann der Prüfling zudem exmatrikuliert werden. (§ 63 Abs. 5 Hochschulgesetz - HG -).

Die Abgabe einer falschen Versicherung an Eides statt wird mit Freiheitsstrafe bis zu 3 Jahren oder mit Geldstrafe bestraft.

Die Technische Universität Dortmund wird gfls. elektronische Vergleichswerkzeuge (wie z.B. die Software „turnitin") zur Überprüfung von Ordnungswidrigkeiten in Prüfungsverfahren nutzen.

Die oben stehende Belehrung habe ich zur Kenntnis genommen:

Official notification:
Any person who intentionally breaches any regulation of university examination regulations relating to deception in examination performance is acting improperly. This offense can be punished with a fine of up to €50,000.00. The competent administrative authority for the pursuit and prosecution of offenses of this type is the chancellor of TU Dortmund University. In the case of multiple or other serious attempts at deception, the examinee can also be unenrolled, section 63, subsection 5 of the North Rhine-Westphalia Higher Education Act (*Hochschulgesetz*).

The submission of a false affidavit will be punished with a prison sentence of up to three years or a fine.

As may be necessary, TU Dortmund will make use of electronic plagiarism-prevention tools (e.g. the "turnitin" service) in order to monitor violations during the examination procedures.

I have taken note of the above official notification:**

05.02.2025
Ort, Datum
(Place, date)

A Nedungadi
Unterschrift
(Signature)

**Please be aware that solely the German version of the affidavit ("Eidesstattliche Versicherung") for the Bachelor's/ Master's thesis is the official and legally binding version.