

UNIVERSITÀ  
DEGLI STUDI  
DI PADOVA



DEPARTEMENT OF INFORMATION ENGINEERING  
MASTER DEGREE IN COMPUTER ENGINEERING

# Monocular Depth Estimation And Collision Prediction For Quadrotors

## Supervisor

Prof. Alessandro Sperduti

## Student

Niko Picello

## Assistant Supervisors

Alessandro Saviolo, Prof. Giuseppe Loianno

ACADEMIC YEAR 2021-2024

Graduation date 16/04/2024



*To everybody I know, if you were part of this journey, thank you.*





# Abstract

This thesis presents a novel approach to nonlinear model predictive control for collision avoidance flight using quadrotors. We use a deep neural network to estimate depth information from monocular image sequences. The network is an encoder-decoder architecture that leverages the weights of a state-of-the-art foundation vision model to encode the input images into dense embeddings, and then a trained-from-scratch decoder head to predict a lightweight depth map tailored for obstacle avoidance. The network decoder is trained on simulated custom-generated depth images due to the lack of sufficient open-sourced datasets tailored for obstacle avoidance tasks. The output depth map is embedded in the model predictive control state and mapped through the camera's intrinsic parameters to the three-dimensional space. At every optimization iteration of the controller, the projected points are translated and rotated based on the system dynamics of the quadrotor. Then, the points are projected back to the image plane along with the size of the robot at each point's depth. The closest point that falls within the robot's size reprojection provides the information on the minimum distance to collide. We map this distance to a collision probability by first converting it into time to collision and then applying a complementary sigmoid function. The overall framework only requires monocular images and inertial measurement unit information to control the robot to navigate a three-dimensional environment without colliding with obstacles. The framework is optimized to run at the control frequency of 100Hz, hence enabling real-time control of the quadrotor. We make a comprehensive analysis of the state-of-the-art approaches for monocular depth estimation and qualitatively demonstrate the limitations of such methods for real-time collision-free navigation. We validate the proposed framework with a statistical analysis in Gazebo simulations, demonstrating its efficacy in cluttered environments.



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Monocular Depth Estimation</b>	<b>3</b>
2.1	Fundamentals and Challenges . . . . .	3
2.2	Deep Learning Models . . . . .	5
2.2.1	DORN - Deep Ordinal Regression Network for Monocular Depth Estimation . . . . .	6
2.2.2	MiDaS - Towards Robust Monocular Depth Estimation: Mixing Datasets for Zero-shot Cross-dataset Transfer . . . . .	8
2.2.3	ZoeDepth - Zero-shot Transfer by Combining Relative and Metric Depth . . . . .	9
2.2.4	Depth Anything - Unleashing the Power of Large-Scale Unlabeled Data . . . . .	10
2.2.5	Our model . . . . .	11
2.3	Data Collection and Augmentation . . . . .	14
2.3.1	Gazebo-Based Data Collection . . . . .	14
2.3.2	Incorporating Complex Environments . . . . .	17
2.3.3	Data Augmentation Techniques . . . . .	17
2.3.4	NYU Depth V2 . . . . .	19
2.4	Results . . . . .	21
<b>3</b>	<b>Learning Deep Collision Probabilities</b>	<b>35</b>
3.1	Monocular Depth Estimation For Collision Avoidance . . . . .	35
3.2	Forecasting Collision Probabilities . . . . .	36
<b>4</b>	<b>Collision-Free Model Predictive Control</b>	<b>39</b>
4.1	The Essence of NMPC for Quadrotor Control . . . . .	39
4.2	Incorporating Collision Probabilities into NMPC . . . . .	40
4.2.1	Formulation of the Optimal Control Problem . . . . .	40
4.2.2	Quadrotor Dynamics Model . . . . .	41
4.2.3	State's propagation . . . . .	43

4.3	Results . . . . .	43
<b>5</b>	<b>Conclusions</b>	<b>49</b>
<b>6</b>	<b>Appendix</b>	<b>51</b>
6.1	Data Collection . . . . .	51
6.2	Foundation Vision models . . . . .	53
6.2.1	SAM . . . . .	53
6.2.2	MobileSAM . . . . .	55
6.2.3	DepthSAM . . . . .	56
6.3	Background . . . . .	57
6.4	CNN . . . . .	57
6.4.1	Convolutional layer . . . . .	59
6.4.2	Pooling . . . . .	62
6.4.3	Batch normalization . . . . .	63
6.5	Transformers . . . . .	64
6.5.1	Architecture . . . . .	64
6.5.2	The attention mechanism . . . . .	64
6.5.3	Foundation Model . . . . .	65
	<b>Bibliography</b>	<b>67</b>

# List of Figures

2.1	Deep monocular depth estimation architecture proposed by [17]. . . . .	5
2.2	DORN: Deep Ordinal Regression Network for Monocular Depth Estimation [22].	6
2.3	ZoeDepth: Zero-shot Transfer by Combining Relative and Metric Depth [24]. .	10
2.4	Depth Anything: Unleashing the Power of Large-Scale Unlabeled Data [27]. .	10
2.5	MobileNetV2’s bottleneck block . . . . .	12
2.6	Our proposed Depth Estimation Model. . . . .	13
2.7	Gazebo-based simulator developed for this thesis for collecting the data with high-fidelity dynamics and millimeter-accurate depth ground-truth estimations.	16
2.8	Processed sample from NYU Depth V2: The bounding box in green on the RGB images represent the collision box of the drone. Depth images have been colorized for visualization. . . . .	20
2.9	Generalization capability of the Depth Anything model on 3 scenes. The bottom right image is from NYU Depth V2, and it is resized for better visualization. . .	21
2.10	MonoNav example for depth estimation and point cloud reconstruction. Image taken from [31] . . . . .	22
2.11	Fitting curve in the hospital environment. . . . .	23
2.12	Prediction and Ground truth taken from the hospital environment. . . . .	24
2.13	Fine grained and Coarse grained depth maps on NYU Depth V2. . . . .	25
2.14	Ablation of the decoder architecture. . . . .	26
2.15	Ablation of the optimal decoder w.r.t. the history. . . . .	27
2.16	Ablation of the optimal decoder w.r.t. the time stride. . . . .	28
2.17	Ablation of the optimal decoder w.r.t. the number of bins. . . . .	29
2.18	Ablation of the optimal decoder w.r.t. the learning rate. . . . .	30
2.19	Predictions of the model for images from our laboratory. No ground truth depth map is available for ablation. . . . .	32
2.20	Coarse Depth Map predictions on NYU Depth V2. From left to right: input RGB image, target depth map, and prediction. . . . .	33

2.21	Coarse Depth Map predictions on the hospital environment. From left to right: input RGB image, target depth map, and prediction. . . . .	34
3.1	Fixed binning with 10 bins spacing the range [0.01, 10.0] . . . . .	36
3.2	Fixed binning with 32 bins spacing the range [0.01, 10.0] . . . . .	36
4.1	Top-down view of the robot’s odometry with respect to the position command .	45
4.2	Views demonstrating how the drone avoids a corner by expanding its trajectory.	45
4.3	Ground truth and depth predictions on the corner demonstrating how the drone avoids a corner by expanding its trajectory. . . . .	46
4.4	Red: actual odometry. Blue: inteded position command. Green: collision probability. . . . .	47
6.1	RGB image and depth map captured with L515 LiDAR camera . . . . .	52
6.2	The RealSense L515 LiDAR camera . . . . .	53
6.3	Overview of Segment Anything Model . . . . .	54
6.4	Semi-Coupled Knowledge Distillation . . . . .	56
6.5	Decoupled Knowledge Distillation . . . . .	56
6.6	Functioning of the convolution operation . . . . .	60
6.7	Depth-wise convolution workflow. The color scheme is just for representation, depth-wise convolution can be applied to whatever grid-type input having $N$ channels. . . . .	61
6.8	Dilated convolution applied to a $3 \times 3$ kernel, with dilation $d = 1,2$ . . . . .	62
6.9	Max Pooling and Average Pooling applied with a kernel of size $2 \times 2$ . . . . .	62

# Chapter 1

## Introduction

Unmanned Aerial Vehicles (UAVs), including quadrotors, play an increasingly crucial role in a variety of applications, ranging from logistics and search and rescue to reconnaissance missions, thanks to their simplicity in design, agility, low cost, and ability to hover in place and move in 3D [1]. Their importance has dramatically increased amidst global crises like the COVID-19 pandemic and ongoing conflicts, highlighting their versatility and broad utility in critical scenarios. However, the autonomous operation of UAVs in complex and dynamic environments is constrained by their limited capability for following the desired path while ensuring collision-free navigation.

Traditionally, obstacle avoidance requires accurately *localizing the robot, mapping the environment, planning collision-free trajectories*, and finally *controlling it* to follow these desired paths. This sequence must be continuously executed during flight to adjust for localization errors, mapping inaccuracies, or changes due to dynamic obstacles. The complexity of these tasks, given the high-dimensional nature of the data involved, imposes substantial computational demands in terms of processing speed and memory usage. Furthermore, the sequential design of the traditional autonomy stack can lead to compounded errors, where a failure in one module compromises the overall performance. These limitations call for novel approaches that tightly couple perception, planning, and control. Such approaches would enable the controller to quickly respond directly to raw sensory inputs without the need for complex processing and intermediate steps. This direct reaction capability is crucial for enhancing UAV autonomy, particularly in environments where rapid adaptability and precise control are required [2].

Motivated by the above, in this work we develop a novel obstacle avoidance scheme that learns to enable collision-free flight in complex environments, while solely relying on a history of real-time RGB image observations and inertial measurement unit (IMU) reading.

We have developed a deep collision encoder-decoder network that leverages a prominent foundation model known for *relative depth estimation*, titled **Depth Anything** [3]. This model

is frozen within our training loop to serve as a robust feature extractor. Each image in the sequence is processed independently, allowing Depth Anything to compute feature maps in parallel for each frame. The extracted features are then fed into our decoder network, which outputs an **absolute coarse depth map** for the latest frame in the sequence. Importantly, we do not require the entire history of depth maps, but rather only the one corresponding to the current timestamp. Our approach is inspired by traditional stereo vision techniques, which utilize triangulation across multiple views of a scene to estimate depth. By adopting this paradigm and processing distinctive perspectives from successive timestamps, we effectively reconstruct a precise coarse depth map of the robot’s view, enabling our model to operate at high frequencies in real-time.

Furthermore, we have designed a Non-Linear Model Predictive Control (**NMPC**) system that utilizes these depth maps to determine the optimal trajectory for obstacle avoidance. In our framework, depth maps are converted to point clouds—3D volumes that contain the coordinates of points in space. Leveraging the predictive capabilities of NMPC, we unroll these future depth maps, or “future point clouds,” over a horizon of  $N$  timestamps, the approach will be further explained in 4.2.3. Subsequently, we select the optimal trajectory that minimizes the cost function, which includes both reaching the desired destination and minimizing collision risks. The first action in the predicted sequence is executed, while the remaining actions are discarded, consistent with standard MPC-integrated pipeline practices.

As the optimization loop continues, a new set of images is processed by the network, and the cycle commences anew, maintaining the same rigorous methodology throughout each iteration. This setup forms a critical part of our introduction, showcasing our model’s capability to integrate advanced depth perception with dynamic control strategies for real-time navigation.

The proposed approach boasts a set of contributions:

- It allows aerial robots to fly through diverse cluttered environments without being limited by a particular task and associated environment assumptions or knowledge.
- Through its modular architecture, it can be exploited by both traditional controllers using Euclidean dynamics and vision-based controllers that operate directly in the sensor space.
- it demonstrates the capability to process high-dimensional sensory data at high frequencies, enabling real-time navigation and decision-making, which is crucial for operations in complex and dynamic settings.



# Chapter 2

## Monocular Depth Estimation

Monocular Depth Estimation (MDE) is a pivotal area in the intersection of computer vision and robotics. It involves deducing the three-dimensional structure of an environment from a single two-dimensional image. This task, inherent with complexities due to the loss of spatial information during image capture, presents considerable theoretical and practical challenges. This chapter explores the foundational principles of MDE, highlighting its importance, the inherent challenges in depth prediction from a single viewpoint, and the necessity of addressing these challenges to push forward the realms of technology and robotics. Moreover, it provides a comprehensive review of the state-of-the-art approaches in MDE, including qualitative benchmarks across various models on tailored datasets, and unveils a novel neural network architecture designed specifically for this thesis. Additional background details about the methodologies introduced in this chapter are reported in Appendix 6.3.

### 2.1 Fundamentals and Challenges

The path to accurate monocular depth estimation is filled with challenges, each contributing to the overall complexity of accurately predicting depth from single images:

- **Ambiguity in Depth Perception:** The primary challenge stems from the inherent limitation of single images in providing detailed distance information. This limitation leads to ambiguity in distinguishing between objects at different distances using only monocular cues. This phenomenon mirrors the human capability to gauge depth with one eye, reliant on accumulated experience and interaction with the environment. This parallel suggests promising avenues for data-driven approaches that emulate this iterative learning process [4].
- **Variability in Environmental Conditions:** Environmental factors such as lighting,

weather, and seasonal changes significantly affect the visual appearance of scenes, introducing variability that complicates depth estimation. Algorithms must exhibit robustness to these conditions to ensure reliability in real-world applications [5].

- **Texture and Pattern Complexity:** The presence of repeating patterns or environments with minimal texture can mislead depth estimation algorithms, resulting in inaccurate depth cues. This challenge underscores the difficulty of consistently localizing robots in environments lacking distinct features [6].
- **Dynamic and Cluttered Scenes:** Scenes characterized by moving objects or clutter demand algorithms that are not only robust but also adaptable, capable of adjusting depth estimates in response to dynamic changes within the environment [7].

Addressing these challenges requires a confluence of advanced computational techniques and innovative methodological approaches:

- **Deep Learning Models:** Deep neural networks, especially Convolutional Neural Networks (CNNs), have demonstrated success in MDE by learning complex depth representations from extensive data. These models typically consist of convolutional layers, pooling layers, fully connected layers, and activation functions, enabling the extraction of spatial features from input images and the generation of depth maps [8]. Representative CNNs include AlexNet [9], VGG [10], GoogLeNet [11], ResNet [12], DenseNet [13], and some lightweight networks, such as MobileNet [14], ShuffleNet [15], and GhostNet [16], each of which is used as the backbone of the existing CNN-based depth estimation network.
- **Data Collection and Augmentation:** Enhancing the robustness of depth estimation models against environmental variability calls for comprehensive data augmentation strategies and training on diverse datasets. This approach aims to ensure model generalization across a wide array of scenes and lighting conditions, bolstering their applicability in real-world scenarios.

The resolution of monocular depth estimation challenges is critical for propelling advancements in computer vision and robotics. Precise depth prediction from a single image can significantly elevate the autonomy and safety of robots and autonomous vehicles, facilitating their navigation and interaction within their environments. Furthermore, enhancements in depth estimation methodologies contribute to progress in related domains, such as 3D modeling, virtual reality, and augmented reality, by providing a nuanced and accurate representation of spatial scene properties.

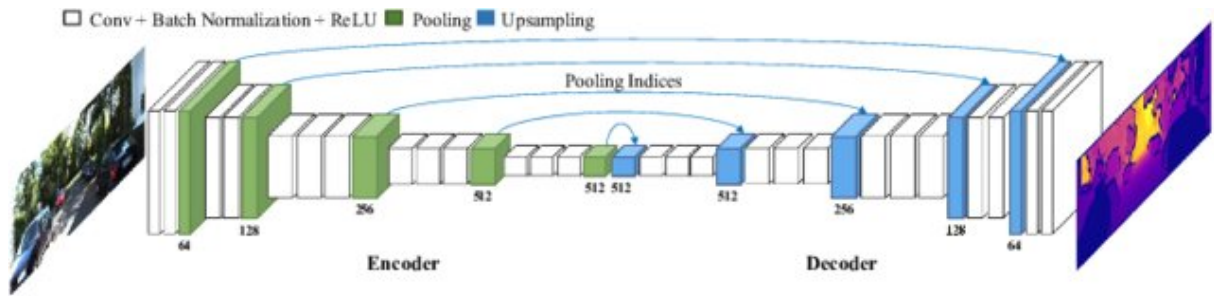


Figure 2.1: Deep monocular depth estimation architecture proposed by [17].

## 2.2 Deep Learning Models

Deep neural networks have played an important role in various areas with their powerful feature learning ability. Monocular depth estimation-based deep learning was first proposed by [18] in 2014. It was a coarse-to-fine framework, where the coarse network learned the global depth on the entire image to obtain a rough depth map and the fine network learned the local features to refine the depth map. Since then, many researchers have carried out deep learning methods for monocular depth estimation [19]–[21].

The state-of-the-art framework of monocular depth estimation based on deep learning is an encoder-decoder network, with the RGB image input and depth map output (Figure 2.1). The encoder network consists of convolution and pooling layers to capture the depth features, and the decoder network includes deconvolution layers to regress the estimated pixel-level depth map, with the same size as the input. Additionally, in order to preserve the features of each scale, the corresponding layers of encoder and decoder are concatenated with skip-connections. The entire network is constrained and trained by the depth loss functions and converges when the desired depth map is generated.

Deep learning methods for monocular depth estimation often utilize gradient descent to train deep neural networks, and obtain a local minimum finally. The best local minimum depends on initialization and specific parameter settings. In the initialization process, it is generally necessary to resize the image to meet the needs of network learning. In addition, it also need to set the initial learning rate, optimizer parameters, batch size and mini-batch size, to learn and save image features. The commonly used learning method is stochastic gradient descent, and the optimizer is Adam. When the gradient no longer changes and the loss function becomes stable, the network converges.

Compared with traditional methods, deep learning methods for monocular depth estimation construct the multi-layer neural network to learn deep features, which has higher accuracy. When there is small occlusion in the monocular image or part of the ground-truth depth is missing, the deep learning methods can still estimate the depth of the scene, and have low errors;

when there is large occlusion in presence in the scene or there is no ground-truth depth, deep learning methods can learn the depth of the scene by adding network constraints. In short, deep learning methods for monocular depth estimation have shown strong robustness.

## 2.2.1 DORN - Deep Ordinal Regression Network for Monocular Depth Estimation

In the exploration of monocular depth estimation, a critical aspect of computer vision that enables machines to perceive the world in three dimensions from a single image, the **Deep Ordinal Regression Network (DORN)** [22] emerges as a groundbreaking approach. Traditional depth estimation methods have primarily focused on regression techniques, directly predicting the distance of objects from the camera. However, these methods often struggle with the inherent ambiguity of depth information in single images, particularly for objects at varying distances.

DORN innovatively addresses this challenge by adopting a deep ordinal regression framework: it combines a dense feature extractor and a scene understanding module to regress multi-channel dense ordinal depth maps. The first component has a standard configuration for a Deep Convolutional Neural Network (DCNN), in which the final downsampling layers are omitted so not to reduce the spatial dimension of the output. The second component comprises 3 components:

- **ASPP** blocks - which exploit **dilated convolution** (see Appendix 6.4.1) to enlarge the field-of-view of filters without decreasing the spatial resolution nor increasing the number of parameters.
- **cross-channel** learner - that can learn complex cross-channel interactions.
- **full-image** encoder - which captures global contextual information and clarifies local confusion in depth estimation.

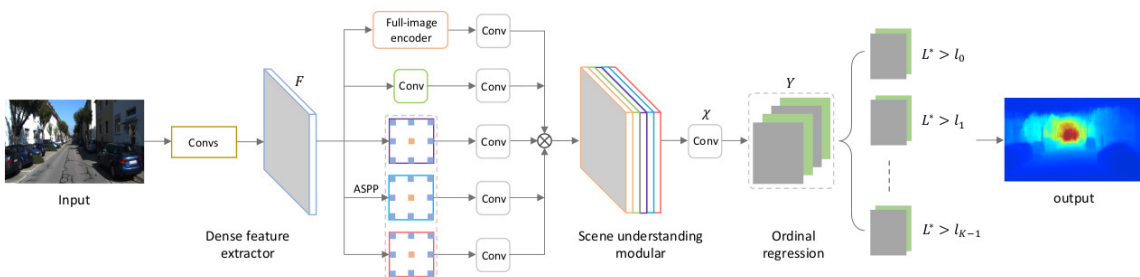


Figure 2.2: DORN: Deep Ordinal Regression Network for Monocular Depth Estimation [22].

After that, the features are concatenated and stacked together, and two other convolutional layers transform these features into a multi-channel dense ordinal depth map: instead of predicting precise distances, DORN indeed categorizes depth into discrete levels. This methodology is rooted in the understanding that depth perception becomes increasingly uncertain with distance. To this end, DORN employs a strategy known as *spacing-increasing discretization (SID)*, which adjusts the spacing between depth categories based on their distance, allowing for greater discrepancies in depth prediction at farther ranges without compromising overall model accuracy.

With the predicted discrete depth values at hand, the learning problem is transformed into a multi-class classification problem, and adopts *softmax* regression loss to optimize the network. Despite that, typical multi-class classification problems ignore the ordinal correlation between the labels, while here the discrete depth targets are highly correlated since they form a well-ordered set. Therefore, the authors transform the depth estimation task into an **ordinal regression problem**, i.e., one where the target variable has a natural, ordered relationship between its categories, but the distances between these categories are not known or necessarily consistent, and create an ordinal loss function to train the neural network.

A key strength of DORN lies in its architectural design, which is specifically tailored to overcome common obstacles in depth estimation. Many existing methods produce low-resolution depth maps, necessitating complex techniques like skip-connections to achieve higher resolution. DORN circumvents these challenges through its dilated convolutions and specialized multi-scale feature learner, enabling the direct prediction of high-resolution depth maps. This streamlined architecture not only enhances the model’s efficiency but also its effectiveness in capturing both the fine details and broader context of a scene.

Tested across several benchmark datasets, including KITTI, ScanNet, Make3D, and NYU Depth v2, DORN has demonstrated superior performance compared to previous depth estimation methods. Its ability to accurately estimate depth across a wide range of distances showcases the potential of ordinal regression in advancing the field of monocular depth estimation.

The introduction of DORN marks a significant advancement in the quest for more accurate and reliable depth estimation from single images. By redefining the problem within an ordinal regression framework and leveraging innovative architectural features, DORN sets a new benchmark for future research in the field. Its success underscores the importance of adapting and refining computational models to better mimic the nuanced ways in which humans perceive depth, opening new avenues for the development of more sophisticated and accurate depth estimation techniques in computer vision.

## 2.2.2 MiDaS - Towards Robust Monocular Depth Estimation: Mixing Datasets for Zero-shot Cross-dataset Transfer

Building on the foundation laid by previous advancements in monocular depth estimation, including the notable DORN, MiDaS [23] introduces a novel strategy that significantly enhances model *robustness* and *generalization* across diverse environments. This approach, characterized by the innovative use of multiple datasets with potentially incompatible annotations, represents a leap towards solving the depth estimation challenge in a broader range of settings.

The essence of MiDaS lies in its ability to mix data from various sources during training by employing a robust training objective that remains **invariant** to differences in depth **range** and **scale**. This method allows for the effective combination of distinct datasets, addressing the critical need for models to perform reliably across varied scenes without direct exposure during training, a concept referred to as zero-shot cross-dataset transfer.

The *Shift-Scale Invariant loss* for a single sample is defined as

$$\mathcal{L}_{ssi}(\hat{d}, \hat{d}^*) = \frac{1}{2M} \sum_{i=1}^M \rho(\hat{d} - \hat{d}^*), \quad (2.1)$$

where  $M$  denote the number of pixels in the image, and

$$\hat{d} = sd + t, \quad (2.2)$$

$$\hat{d}^* = d^*, \quad (2.3)$$

$$(s, t) = \arg \min_{s, t} \sum_{i=1}^M (sd_i + t - d_i^*)^2. \quad (2.4)$$

The authors also defined the *gradient matching term* as:

$$\mathcal{L}_{reg}(\hat{d}, \hat{d}^*) = \frac{1}{M} \sum_{k=1}^K \sum_{i=1}^M (|\nabla_x R_i^k| + |\nabla_y R_i^k|). \quad (2.5)$$

Here,  $R_i = \hat{d}_i - \hat{d}_i^*$ , and  $k$  denotes the scale at which the difference of disparity maps is computed (in MiDaS, the authors used  $k = 4$ ).

Ultimately, the final loss is computed as:

$$\mathcal{L}_l = \frac{1}{N_l} \sum_{n=1}^{N_l} \left[ \mathcal{L}_{ssi}(\hat{d}^n, (\hat{d}^*)^n) + \alpha \mathcal{L}_{reg}(\hat{d}^n, (\hat{d}^*)^n) \right], \quad (2.6)$$

with  $N_l$  being the training set size and  $\alpha = 0.5$

### 2.2.3 ZoeDepth - Zero-shot Transfer by Combining Relative and Metric Depth

ZoeDepth [24] introduces a unique approach that leverages a two-stage training framework, initially harnessing a wide array of datasets to master relative depth estimation, followed by fine-tuning with metric depth data. This strategy ensures that ZoeDepth not only excels in generalization across various scenes but also retains the precision of metric scale depth estimation. The flagship model, ZoeD-M12-NK, exemplifies the power of this methodology. It undergoes pre-training on 12 datasets for relative depth, followed by meticulous fine-tuning on both indoor (NYU Depth v2) and outdoor (KITTI) datasets for metric depth, using a specialized lightweight head for each domain.

The architecture proposed for ZoeDepth has a typical encoder-decoder structure (**DPT**), where the encoder is however replaced with a more recent transformer-based backbone. Despite that the core innovation lies in the metric bins module, a novel component designed to adaptively predict depth across multiple scales, that is attached on top of the aforementioned backbone of the model.

Inspired by older binning mechanisms such as **AdaBins** [25], and **LocalBins** [26], the **MetricBins** module takes multi-scale features from the decoder and predicts the bin centers that are going to be used for depth estimation. Differently from its predecessors, instead of starting with a small amount of bin centers, and splitting them later, the new module predicts all the bin centers at once and then adjust them in the successive decoder layers. Through its ingenious bin adjustment design, ZoeDepth significantly enhances depth prediction accuracy, especially in challenging environments.

During inference, ZoeDepth intelligently routes each input image to the appropriate head, based on a latent classifier, ensuring optimal depth estimation across varying domains.

Empirical results underscore ZoeDepth’s exceptional capabilities. Without pre-training, ZoeDepth already improves upon the state of the art on the NYU Depth v2 dataset. When leveraging pre-training across 12 datasets and fine-tuning on NYU Depth v2, it achieves an unprecedented 21% improvement in terms of relative absolute error (REL). Moreover, ZoeD-M12-NK marks a milestone as the first model to train jointly on multiple datasets without significant performance degradation, showcasing unparalleled zero-shot generalization across eight unseen datasets.

ZoeDepth not only pushes the boundaries of monocular depth estimation but also introduces a versatile framework that can be adapted to various domains, from indoor to outdoor, real to simulated environments. This breakthrough paves the way for a new generation of depth estimation models, capable of operating with unmatched accuracy and generalizability.

ZoeDepth’s methodology aligns closely with the principles underpinning MiDaS. Both Mi-

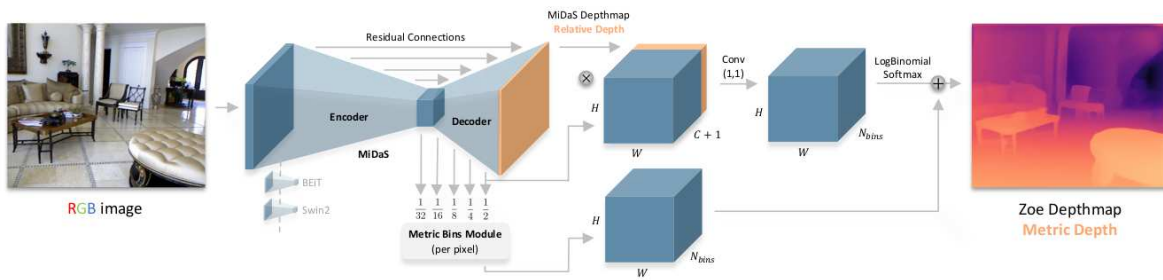


Figure 2.3: ZoeDepth: Zero-shot Transfer by Combining Relative and Metric Depth [24].

DaS and ZoeDepth underscore the importance of leveraging diverse data sources to train depth estimation models. MiDaS’s exploration of 3D films as a novel data source and its robust training objectives complement ZoeDepth’s strategy of using a vast and varied dataset collection for initial training followed by domain-specific fine-tuning. Together, these methodologies highlight a pivotal shift in the field towards models that are not only highly accurate but also remarkably adaptable to new and unseen environments.

In summary, MiDaS and ZoeDepth collectively represent the forefront of research in monocular depth estimation. Their shared emphasis on dataset diversity, robust training techniques, and the goal of achieving high generalization across datasets showcases a unified direction towards creating more reliable and versatile depth estimation models.

## 2.2.4 Depth Anything - Unleashing the Power of Large-Scale Unlabeled Data

Concluding our exploration of state-of-the-art models in monocular depth estimation, the debut of Depth Anything [27] signifies a notable advancement in **relative depth** regression. Diverging from traditional reliance on labeled datasets, Depth Anything innovates by harnessing a vast collection of unlabeled images to bolster its generalization capacity across varied environments.

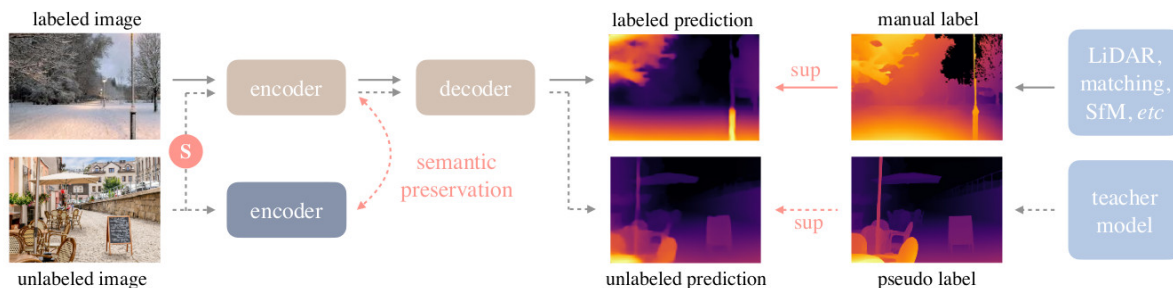


Figure 2.4: Depth Anything: Unleashing the Power of Large-Scale Unlabeled Data [27].



Depth Anything exploits the wealth and variety of **unlabeled data**, shifting from conventional methods that necessitate expensive and time-consuming collection of labeled data. Through the creation of a data engine capable of automatically annotating these unlabeled images, the model broadens its learning beyond the confines of labeled datasets. This approach not only mitigates the challenges associated with data scarcity and diversity but also clears the path for more scalable and robust depth estimation solutions.

By adopting a more demanding optimization goal and auxiliary supervision to impose rich semantic priors from pretrained encoders, Depth Anything demonstrates remarkable zero-shot generalization capabilities. Depth Anything is meticulously trained in a two-stage process: initially, a model is trained with labeled images, then this model acts as a mentor to generate labels for the unlabeled data, which trains the actual student model. Mathematically, the labeled and unlabeled datasets are denoted as  $D^l = (x_i, d_i)_{i=1}^M$  and  $D^u = u_i_{i=1}^N$  respectively. The objective is to develop a teacher model  $T$  from  $D^l$ , then utilize  $T$  to assign pseudo-labels to  $D^u$ , and subsequently train the student model  $S$  with both labeled and pseudo-labeled data.

The methodology for training the teacher model mirrors that used for MiDaS; depth maps are transformed into **disparity maps** and normalized within the range  $[0 - 1]$ . Furthermore, to facilitate multi-dataset joint training, the Shift-Scale Invariant loss as outlined in 2.1 is employed. Despite utilizing fewer labeled images than MiDaS for initial training, the subsequent inclusion of unlabeled images in the second training phase introduces the required diversification to enhance the model’s generalization capability and robustness. This enhancement is evident in its superior performance across various public datasets and real-world scenarios, thereby redefining benchmarks in the domain.

Moreover, Depth Anything underscores the untapped potential of large-scale unlabeled data in advancing monocular depth estimation. Its innovative application of such data significantly diminishes the generalization error, facilitating precise model performance in any context. This accomplishment highlights the model’s versatility and resilience, establishing it as a fundamental tool for future endeavors in computer vision and robotics. Particularly, Depth Anything’s efficacy in generating high-quality results on simulated data minimizes the sim-to-real gap between training and deployment phases.

## 2.2.5 Our model

Leveraging the impressive capabilities of Depth Anything for accurate relative depth estimation, our final model architecture is derived from this robust foundation. Depth Anything serves as the cornerstone for feature extraction, upon which we architect a decoder network tasked with the computation of absolute depth values. Drawing inspiration from the pioneering work of MobileNetV2 [28] our decoder is modeled after its architectural principles, with a particular

focus on the innovative **bottleneck block** utilized in MobileNetV2 (see fig. 2.5).

This specialized bottleneck block adapts its behavior based on the setting of the *stride* parameter, manifesting in two distinct operational modes:

- With a *stride* of 1, the bottleneck block incorporates a residual connection, effectively adding the input directly to the bottleneck's output. This addition necessitates the input's adjustment in terms of both shape and channel count to align with the output's dimensions.

- Conversely, a *stride* of 2 shifts the bottleneck block's operation to process the input independently, foregoing residual connections. This approach results in a reduction of the input's dimensions by half due to the increased stride, rendering the application of residual connections infeasible.

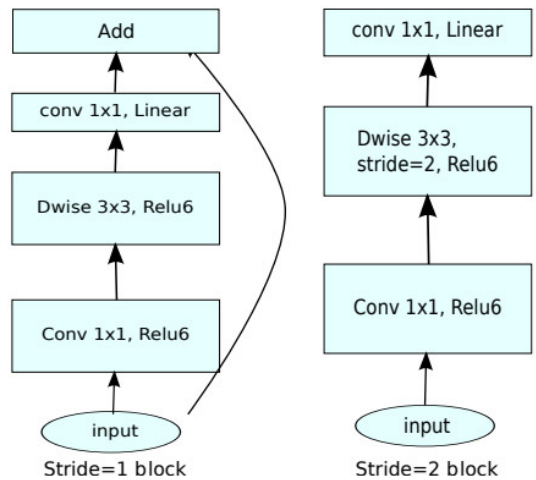


Figure 2.5: MobileNetV2's bottleneck block

A critical element of MobileNetV2's bottleneck block is its reliance on **depth-wise convolution** (see Appendix 6.4.1), as opposed to conventional convolution methods. This choice facilitates the construction of a more profound architectural depth while simultaneously curtailing the total parameter count, thereby enhancing the model's efficiency without compromising its depth estimation performance.

As mentioned, our Depth Anything model plays a crucial role in feature extraction: it extracts four sets of features from each image, each set having increased dimensionality. These are then reshaped and processed differently to infer the relative depth map. In our work, we are not focused on the model's final prediction. Instead, we utilize each set of features individually as input to our decoder. We discovered that the features set closest to the model's final layers yields the best results. Thanks to the **residual connections** between the cascading sets of features, the final feature map integrates information from the preceding sets as well, significantly enhancing the model's learning capability.

Figure 2.6 presents the complete architecture of our network. Initially, the input image is processed by the Depth Anything model to extract a feature map. This feature map is then fed into the decoder, which consists of a series of convolutional layers, batch normalization, and ReLU activation functions. The decoder's output is a coarse-grained bin volume. This volume

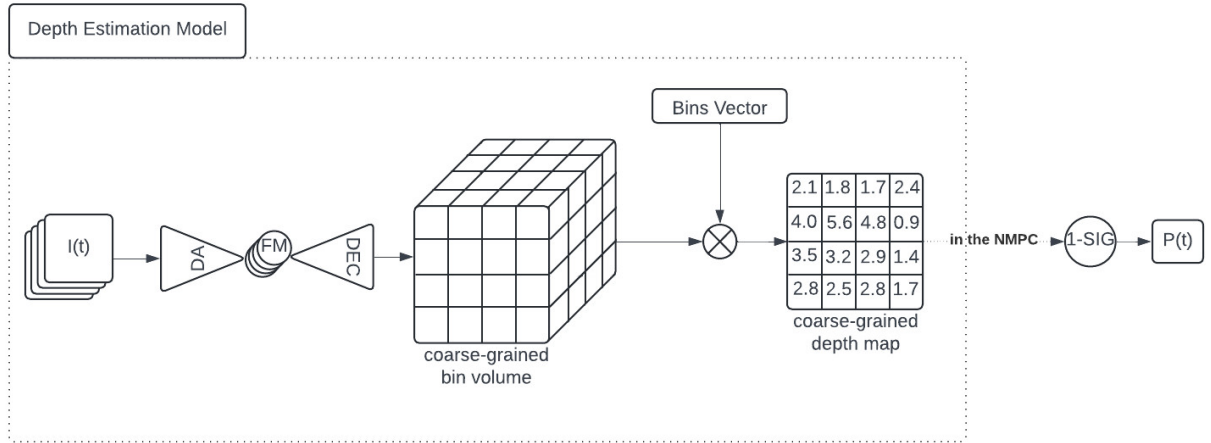


Figure 2.6: Our proposed Depth Estimation Model.

features a markedly reduced width and height, while its third dimension corresponds to the total number of bins used to discretize the depth range.

The volume is then processed with a **softmax** function on the *bin* dimension, so to retrieve what can be claimed to be the probability of each bin to correspond to the actual distance of the nearest element in the region captured by the coarse block. Following this, a matrix-vector multiplication occurs between the bin volume and the bin vector, which defines the depth intervals. This operation produces a single-channel, coarse-grained depth map. For each segment, this map indicates the nearest distance to any object within the corresponding region of the original RGB image, offering a simplified yet effective depth estimation.

In our work, this **depth estimation network** is integrated within the Model Predictive Control (**MPC**), that predicts the sequences of possible future states, composed of *linear velocity*, *orientation*, *position*, and *angular velocity*, and the action that the robot should take, here the 4 motors speed, in Rounds Per Minute (**RPM**). In this setup, the **coarse-grained depth map**, reprojected as a flatten 3D *point cloud*, is also added to the drone's current state.

In the horizon of actions predicted by the NMPC, only the first action is executed, and the rest are disregarded, continuing the process with the next history of RGB images.

## 2.3 Data Collection and Augmentation

In the pursuit of refining monocular depth estimation for quadrotor navigation and obstacle avoidance, the choice and creation of datasets play a critical role. Among the existing datasets, NYU Depth Dataset V2 (NYU v2) and the KITTI dataset stand as benchmarks in the field. NYU v2, with its focus on indoor environments, provides high-resolution images and depth maps captured within various indoor settings. The KITTI dataset, on the other hand, is tailored for outdoor scenarios, primarily targeting autonomous driving applications with its collection of images and depth information captured in urban and rural environments. Despite their extensive use and contributions to advancements in MDE, these datasets present certain limitations, particularly when applied to the specific demands of quadrotor navigation.

The primary challenge with existing datasets like NYU v2 and KITTI lies in their static nature and the absence of dynamic interactions and robot dynamics. While they offer a snapshot of the environment, they lack data on how the environment or object positions change in response to the robot’s movements. More critically, these datasets do not capture the quadrotor’s dynamics, such as the effects of motor speeds and thrust variations, which are essential for understanding and predicting the quadrotor’s behavior and interactions with its surroundings.

Furthermore, the scenarios depicted in NYU v2 and KITTI are limited to what the sensors have captured, without the possibility of exploring unseen or hypothetical situations. This limitation restricts the depth estimation models’ ability to generalize across varied and novel environments, an essential capability for quadrotors operating in diverse or unknown territories.

To address these challenges and limitations, we embarked on the collection of a custom dataset, specifically designed to meet the unique requirements of quadrotor navigation and obstacle avoidance. Our dataset aims to incorporate the dynamic aspects of quadrotor flight, including the influence of motor speeds and the resultant changes in the environment from the quadrotor’s perspective.

### 2.3.1 Gazebo-Based Data Collection

To construct a dataset tailored to the intricate requirements of quadrotor navigation, we utilized a cutting-edge simulation framework that intricately models both the dynamics and the visual aspects of quadrotor flight. This framework integrates Gazebo, a comprehensive and widely adopted robotics simulator, with ROS2 (Robot Operating System version 2) and PX4 1.14 simulation. The combination of these tools forms a potent simulation environment that can accurately represent the physical world and the quadrotor’s interaction with it.

Gazebo provides a rich set of features, including high-quality rendering of environments and physics-based simulation of light, shadows, and textures, which are essential for creating

realistic visual data for depth estimation. The incorporation of ROS2 facilitates seamless and efficient communication within the simulator, enabling the orchestration of complex scenarios and the collection of data from various sensors simulated within the environment.

The use of PX4 1.14 simulation is particularly crucial. PX4 is among the leading open-source flight control software for drones and other unmanned vehicles. Its simulation within Gazebo allows for the accurate replication of quadrotor dynamics, including the effects of propeller speed, tilt angles, and external forces such as wind. This level of detail ensures that the collected data reflects the nuances of real-world quadrotor flight, providing a solid foundation for training depth estimation models that can predict and react to dynamic changes in the environment.

Our simulation setup goes beyond static environments to include dynamic elements that a quadrotor might encounter during flight. This includes moving obstacles, variable wind conditions, and changes in lighting. Such dynamic simulations are crucial for developing models capable of understanding and reacting to changes in real-time, a critical requirement for autonomous navigation and obstacle avoidance.

To further enhance the realism and efficacy of the simulation, we meticulously modeled our quadrotor based on specific configurations and capabilities. Our simulated quadrotor, mirroring its real-world counterpart, has a mass of 1.3kg. It is equipped with four motors that provide the necessary lift and maneuverability across various simulated environments.

For processing and capturing visual data within the simulation, we emulate the capabilities of an NVIDIA Jetson Xavier NX and an Arducam IMX477 camera, operating at resolutions of  $544 \times 960$ . This setup allows for the collection of high-quality visual data, essential for depth estimation tasks. The simulation operates in real-time, with the control system processing inertial measurements at 100Hz and the perception framework processing camera frames at 60Hz, ensuring that the simulation accurately reflects the dynamics and responsiveness of actual quadrotor flight.

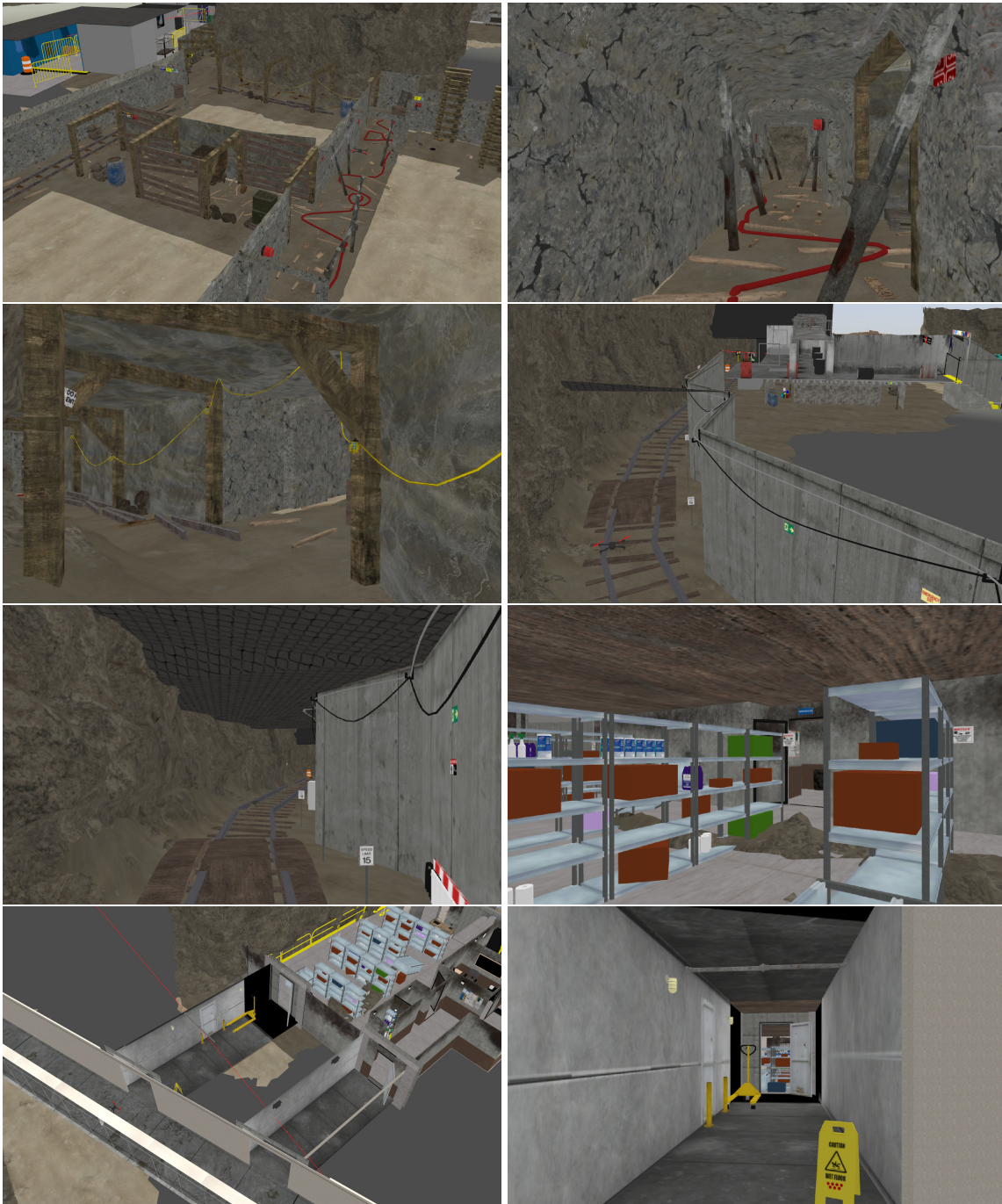


Figure 2.7: Gazebo-based simulator developed for this thesis for collecting the data with high-fidelity dynamics and millimeter-accurate depth ground-truth estimations.

### 2.3.2 Incorporating Complex Environments

To further enhance the realism and applicability of our dataset, we incorporated detailed models of environments from the DARPA Subterranean Challenge into the Gazebo simulator (Figure 2.7). These environments are characterized by their complexity and variety, featuring narrow tunnels, large open spaces, and intricate urban underground structures. The high-fidelity modeling of these environments ensures that the dataset covers a wide range of scenarios and obstacles, from tight spaces that require precise navigation to large, open areas where depth perception becomes critical for identifying distant obstacles.

### 2.3.3 Data Augmentation Techniques

To maximize the utility and realism of our dataset for monocular depth estimation (MDE) and obstacle avoidance tasks, we extended our data augmentation techniques by incorporating various transformations from the *torchvision* library [29]. These augmentations simulate a range of visual conditions and distortions that a quadrotor might encounter, further enhancing the robustness and adaptability of the depth estimation models developed from our dataset.

The augmentation techniques we employed are as follows:

- **Gaussian Blur:** This augmentation applies a Gaussian blur with a kernel size of 3 and a sigma range of 0.1 to 2.0. This simulation can represent the effect of rapid motion or atmospheric conditions like fog or smoke that reduce visibility, teaching the model to infer depth information from less distinct visual cues.
- **Random Erasing:** We use Random Erasing with a probability of 1.0, targeting a small portion of the image for erasure with a scale range of 0.001 to 0.01 and a ratio range of 0.3 to 3.3, filling the erased area with a value of 0. This technique mimics the occurrence of sensor occlusions or dead pixels, encouraging the model to make depth estimations based on the surrounding context.
- **Color Jitter:** Variations in lighting and color are introduced through Color Jitter, adjusting the brightness, contrast, saturation, and hue by up to 20%, 20%, 20%, and 10% respectively. This augmentation ensures that the model can reliably estimate depth in varying lighting conditions and environments with different color profiles.
- **Grayscale Conversion:** By converting color images to grayscale (while retaining three output channels), we simulate the loss of color information, which can occur in low-light conditions or with certain sensor limitations. This forces the model to rely on texture and shape rather than color cues for depth estimation.








Name	Augmentation
Original	
Gaussian blur	
Erasing	
Color jitter	
Grayscale	

Table 2.1: Augmentation techniques used to increase the dataset size; for the sake of visualization, the images are not normalized.



These augmentations are composed of the original transformation applied to the dataset, ensuring that the base image processing remains consistent across all augmented data. By employing these sophisticated data augmentation techniques from torchvision, we significantly enhance the dataset’s ability to train monocular depth estimation models capable of performing under a wide range of operational conditions, mirroring the complexities of real-world autonomous quadrotor navigation; see table 2.1.

### 2.3.4 NYU Depth V2

The NYU Depth Dataset V2 [30], an advancement in computer vision research, offers a comprehensive collection of depth data suitable for various applications including 3D reconstruction, indoor navigation, and object recognition. Developed by the Computer Science and Engineering Department at New York University, this dataset comprises RGB and depth images meticulously captured with a Microsoft Kinect sensor across multiple indoor scenes. Notably, it includes over 1,500 sequences divided into 464 distinct scenes, providing a rich resource for training and evaluating depth estimation algorithms. The diversity of the environments captured, ranging from residential spaces to office settings, ensures its applicability in developing versatile and robust computer vision models. The dataset also comes equipped with labeled objects and segmentation masks, which we didn’t really use.

Initially, the dataset was accessible directly from its official webpage. Later, it was incorporated into the *Hugging Face* dataset library, necessitating its conversion to a format compatible with our requirements. Specifically, this involved re-projecting the drone’s position within the depth images. This step was critical for determining a collision probability for the moving vehicle. However, it’s important to note that the original dataset did not employ a drone for data capture. Furthermore, even if drone-captured data were available, the flight dynamics would significantly differ from the scenarios we are modeling. Consequently, we leveraged this dataset primarily to expand our sample pool, thereby enhancing the training process of our Depth Estimation Model. Figure 2.8 displays some samples from the dataset. Here, the drone is re-projected into the RGB images using depth map data. It’s important to note, however, that during training, the bounding box indicating the drone’s position is not actually present. Instead, it is utilized to determine the minimum distance from the drone to any obstacle within the robot’s field of vision.



Figure 2.8: Processed sample from NYU Depth V2: The bounding box in green on the RGB images represent the collision box of the drone. Depth images have been colorized for visualization.

## 2.4 Results

We distinguish the results obtained through our model between fine-grained and coarse-grained. Our initial goal was to predict dense depth maps from RGB images. When examining *relative* depth estimation, Depth Anything performs exceptionally well. Thanks to its extensive training dataset, it delivers remarkable outcomes on new, unseen data, positioning it as an ideal choice for relative depth estimation tasks. As demonstrated in Figure 2.9, Depth Anything showcases its capability to effectively generalize across both real-world and simulated data. This versatility not only highlights the model’s robustness but also opens up opportunities to leverage simulated data for training subsequent models with relative depth inputs.

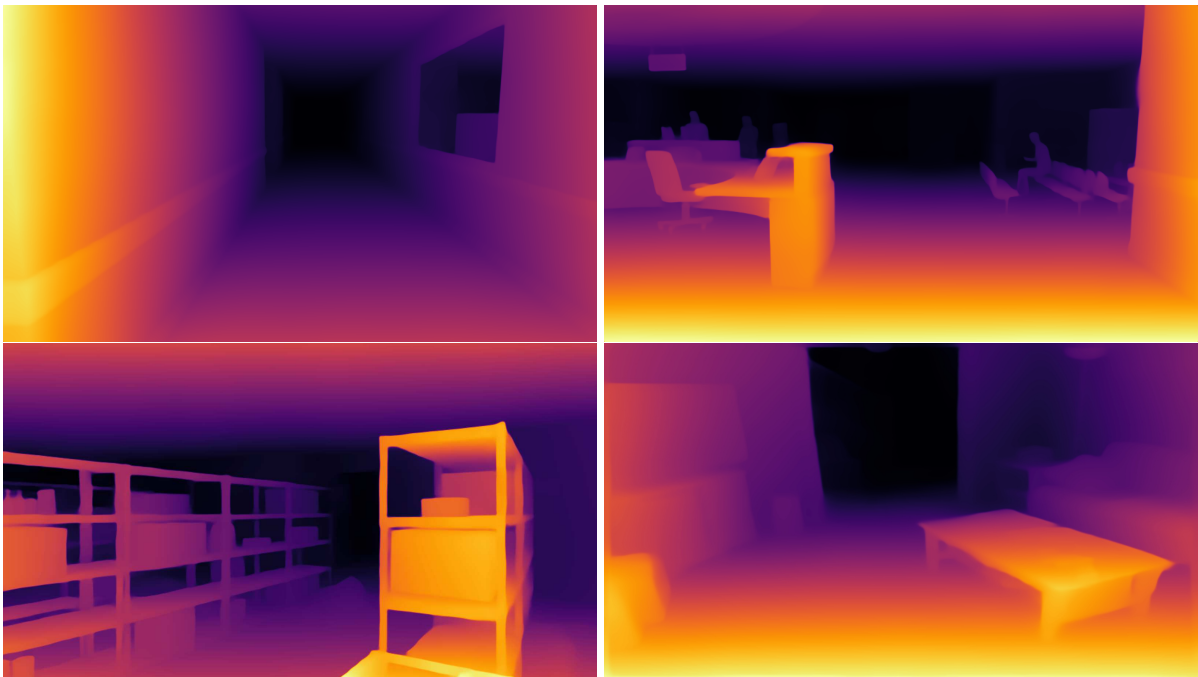


Figure 2.9: Generalization capability of the Depth Anything model on 3 scenes. The bottom right image is from NYU Depth V2, and it is resized for better visualization.

Despite our efforts to utilize Depth Anything for estimating the collision probability of the flying drone, the outcomes were not as favorable as anticipated. The model encountered difficulties in accurately learning the conversion from relative depth to the minimum distance to an obstacle within the drone’s reprojection. Initially, we believed that extracting distance information for a single point within the bounding box from the depth map would be straightforward. However, this task proved to be more challenging than expected, leading to unsatisfactory final estimations. Specifically, with a significant number of samples where the minimum depth within the robot’s reprojection reached its maximum (i.e., 10m), our model tended to minimize the overall error. This approach resulted in accurate predictions for this particular set of samples but introduced larger errors for other inputs. Despite our efforts to address the issue of an **im-**

**balanced** dataset through techniques like data augmentation and loss weighting (for example, assigning higher weights to rarer samples and reducing the weight for more common ones), the model still fell short in accurately predicting distances across all images: it struggled to decrease the error margin for data that was less densely represented.

Consequently, we shifted our focus to *absolute* depth estimation, where **ZoeDepth** stands out as the leading model. It demonstrates impressive performance even on unseen data, showing a strong ability to generalize across various environments, both real and simulated. However, the main challenge with **ZoeDepth** lies in its inference time. Previous studies, such as [31], attempted to apply this model for Monocular Depth Estimation and processed the resulting depth map to reconstruct the environment surrounding the robot. The significant drawback of this method is the computational load associated with depth estimation and reconstruction. These processes must be carried out off-board, introducing delays between the robot and the computing workstation. Despite these efforts, the system operates at approximately 4-5Hz, which does not meet our desired performance criteria. In order to have an end-to-end framework for autonomous navigation, we don't want to do reconstruction nor planning outside of the controller. Instead, what we want is to have a drone that, given a RGB image, understands where it has to go to reach its final goal.

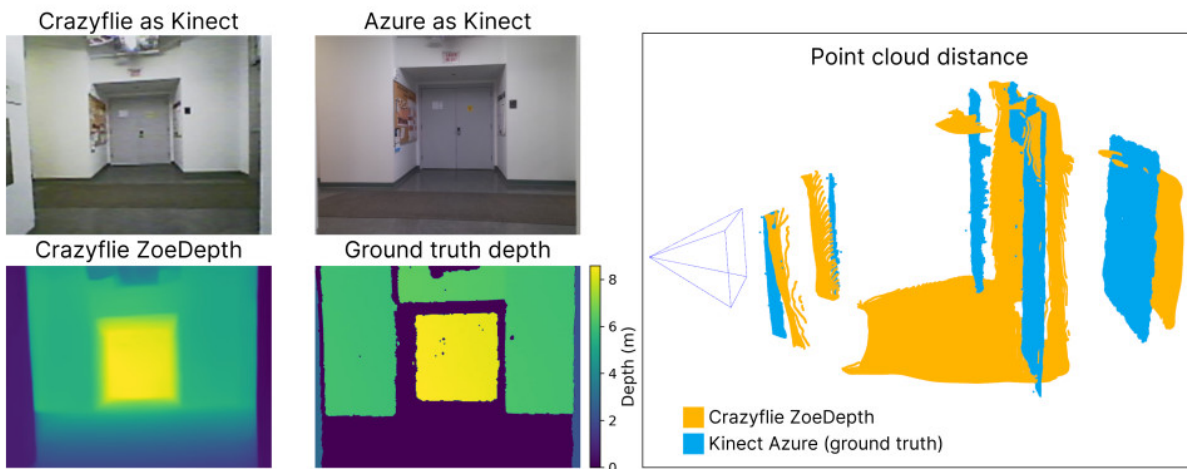


Figure 2.10: MonoNav example for depth estimation and point cloud reconstruction. Image taken from [31]

In our search for effective absolute depth estimation solutions, we explored the metric variant of Depth Anything. This version integrates the ZoeDepth stack atop the DepthAnything module for relative depth estimation. The authors of ZoeDepth suggest that the metric heads, responsible for inferring absolute depth, comprise only 1% of the total parameters of the backbone, indicating that choosing a different backbone could enhance inference speed. Despite these claims of efficiency, the model remained too slow for integration within our operational framework. This particular metric variant was built on the largest of the Depth Anything mod-

els, **DepthAnything-L**, which contains approximately 335.3 million parameters. As a result, it imposes substantial computational demands that our real-time depth prediction system cannot accommodate.

We thus need a faster and accurate enough model, so to be able to exploit the retrieved depth maps in a real-time deployment, and our model comes at hand. Using the smallest of the Depth Anything model (**DepthAnything-S**, with 24.8M parameters), we managed to build a depth estimation network that work at a higher inference speed w.r.t. to the former approaches. Specifically, our network2.2.5, optimized with TensorRT, manages to work at 73Hz, allowing us to integrate it within our framework. Figure 2.12 shows the qualitative results achieved on a sequence of images in the *hospital* environment. Here, the goal is to show that we can fit the training data and generalize on unseen test data. Despite that, the two sets of images are captured within the same environment, hence we don't expect them to be too different from the training pool of samples. Future works aim at expanding both the training and testing data with all the environments we collected in order to prove the generalization capability of our model. In figure 2.11, the curves illustrate the discrepancies between the predicted depths and the actual ground truths. Despite occasional spikes, the errors for the **minimum** and **mean** predictions mostly remain below 0.8 meters, which could be considered sufficiently accurate for applications such as drone navigation within an environment. Conversely, the errors associated with **maximum** depth predictions are significantly larger. This could be attributed to the increased complexity of estimating distances for farther points in an image, a challenge that mirrors human visual perception. For nearby objects, humans can gauge distances with reasonable accuracy. However, estimating the distance to more distant objects is inherently more difficult, and precision diminishes. It appears the depth estimation model follows a similar pattern, excelling at closer range predictions but struggling with accuracy at greater distances.

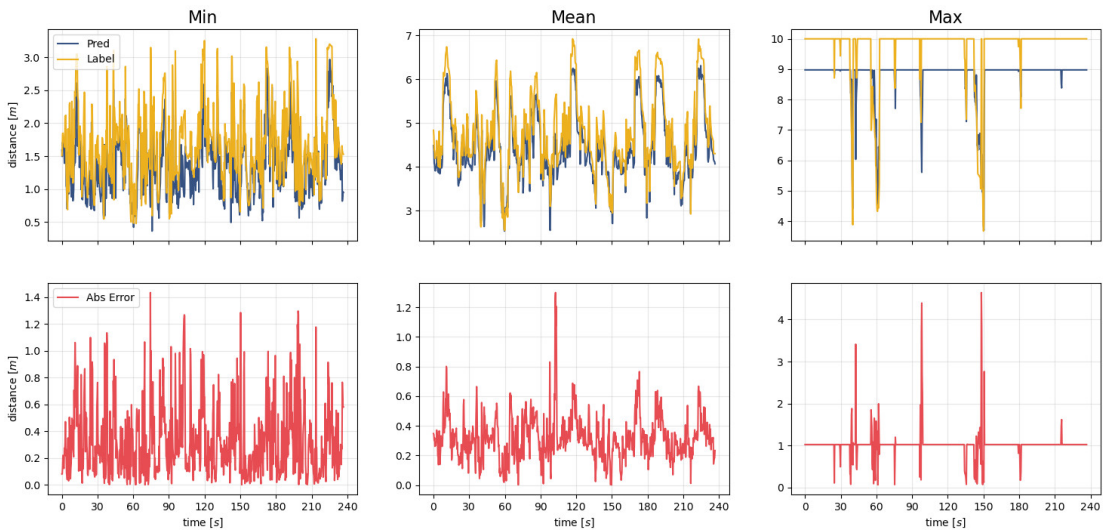


Figure 2.11: Fitting curve in the hospital environment.



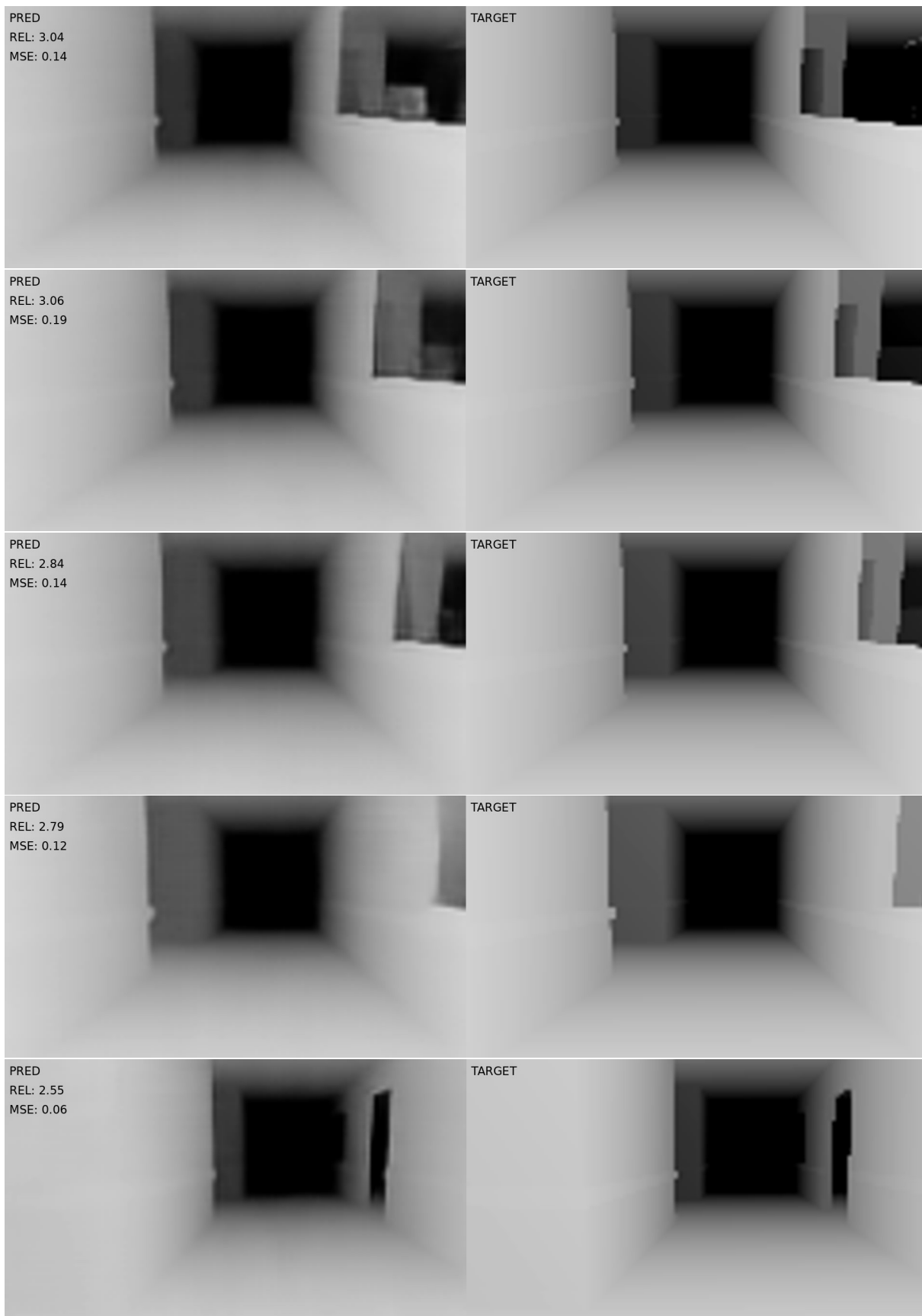


Figure 2.12: Prediction and Ground truth taken from the hospital environment.

For coarse-grained depth estimation the model is trained to predict a  $7 \times 4$  depth matrix, which serves as a sparse approximation of the complete depth map. This reduction in resolution is crucial for mitigating the computational burden posed to the Model Predictive Controller (MPC). A fine-grained depth map would be prohibitively dense for the MPC to process in a timely manner, hence a more coarse representation is adopted for efficiency.

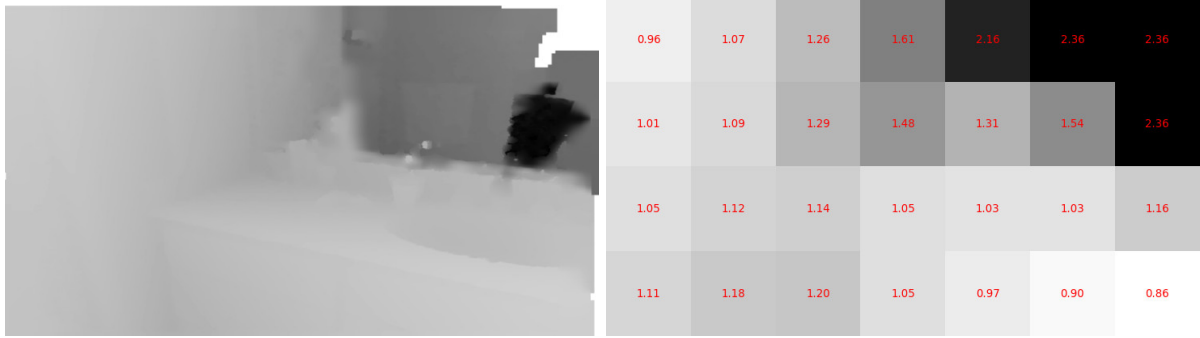


Figure 2.13: Fine grained and Coarse grained depth maps on NYU Depth V2.

Our underlying assumption posits that dense pixel-level depth information is not essential for the drone’s navigation. Rather, a sparser set of depth data should suffice for safe flight. Analogous to human navigation, the drone’s primary requirement is to discern the proximity to the nearest obstacle to avoid collisions, rather than a detailed distance measure for every visible point. Therefore, our model focuses on identifying the most critical depth information to inform obstacle avoidance strategies.

Finally, Figures 2.14, 2.15, 2.16, 2.17, 2.18 present the training curves for the models discussed previously. We conducted training and evaluation of various models utilizing the *High Performance Computer (HPC)* facilities at NYU, which enabled us to leverage NVIDIA A100 GPUs across 16 parallel processes simultaneously for periods of 18 hours each. Given that numerous training attempts were discontinued throughout the project, we focus on showcasing the final, most relevant outcomes.

Figure 2.14 show the performance of the 3 decoders implemented to infer the depth map from the features extracted by Depth Anything, while table 2.2 shows the specifics of the decoders.

<b>Model</b>	<b>Number of Parameters</b>	<b>Inference Speed</b>
Vanilla	96.1 million	0.0984 ms
With Pool	70.6 million	0.0318 ms
DepthWise	9.34 million	0.0148 ms

Table 2.2: Ablation of different decoders based on parameters and speed

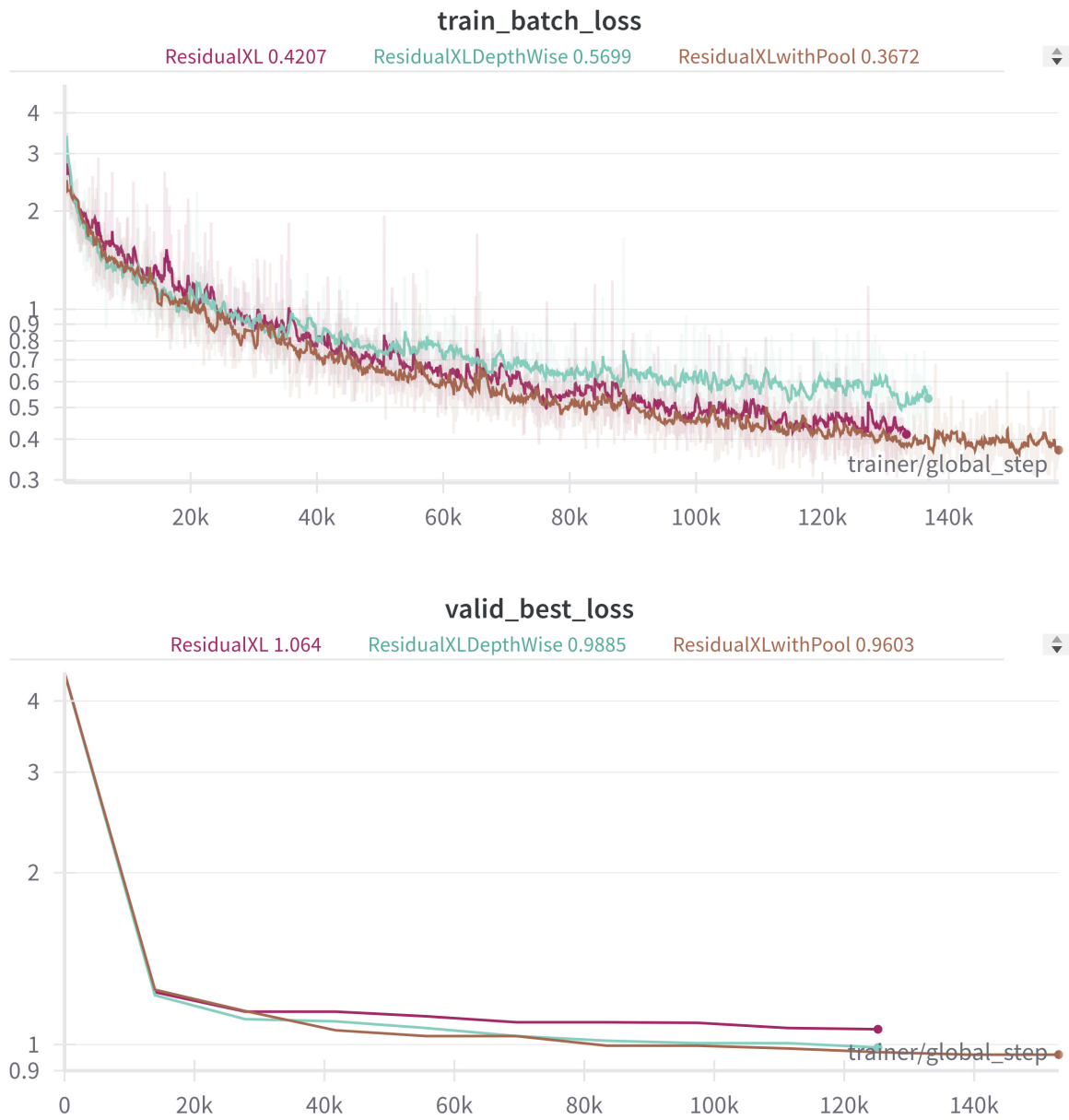


Figure 2.14: Ablation of the decoder architecture.



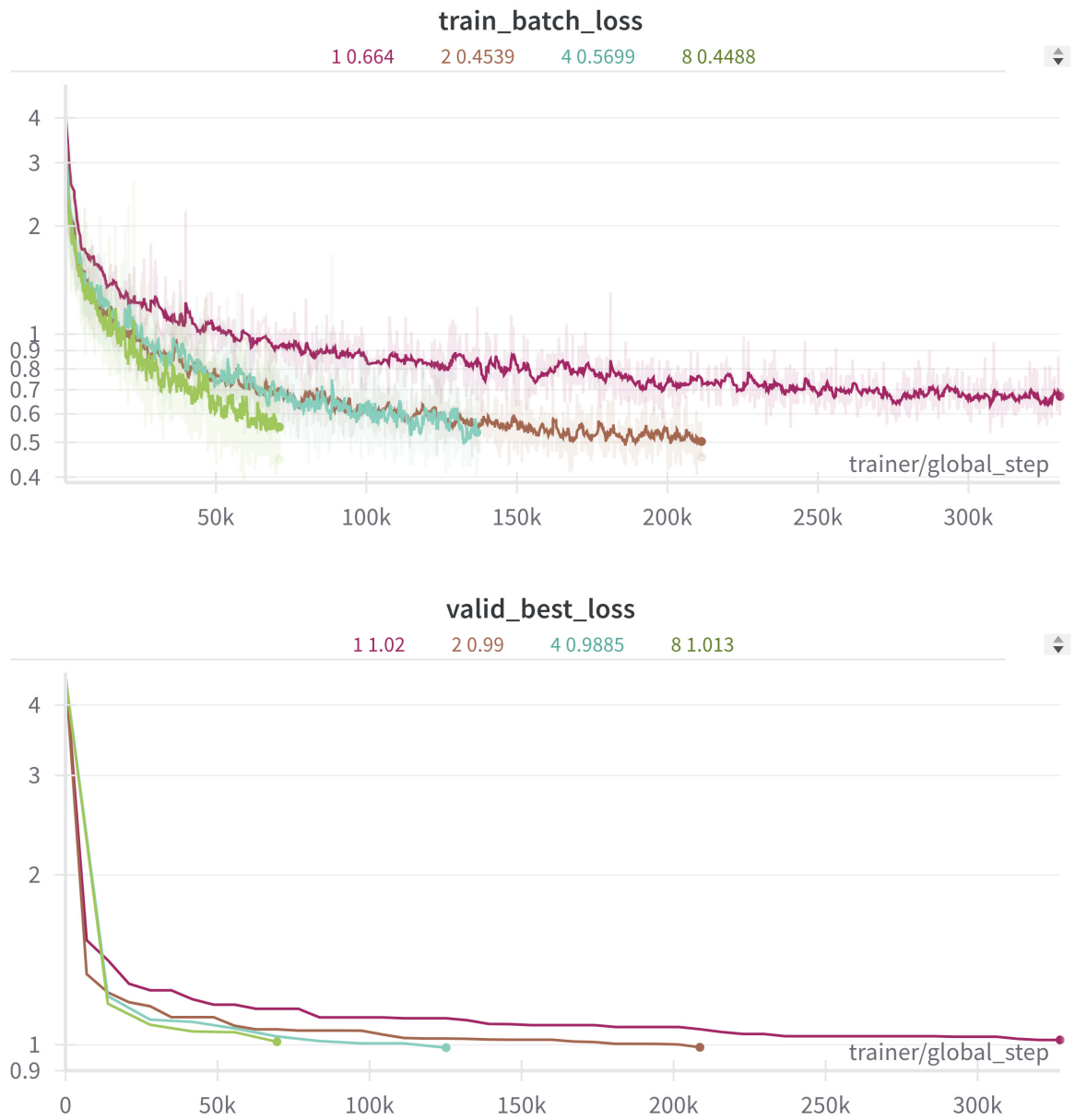


Figure 2.15: Ablation of the optimal decoder w.r.t. the history.

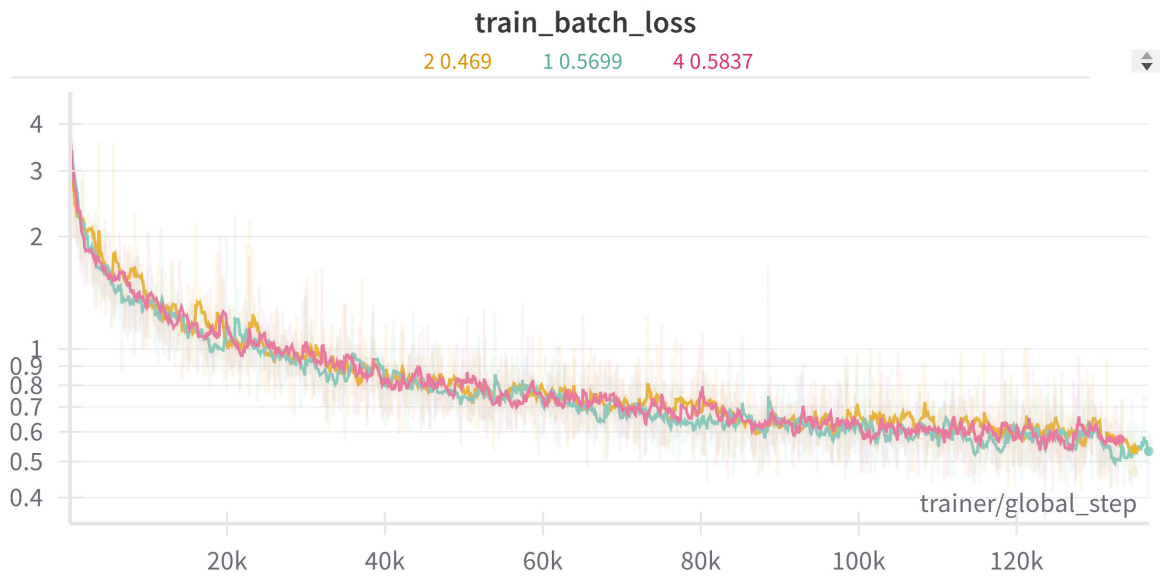


Figure 2.16: Ablation of the optimal decoder w.r.t. the time stride.

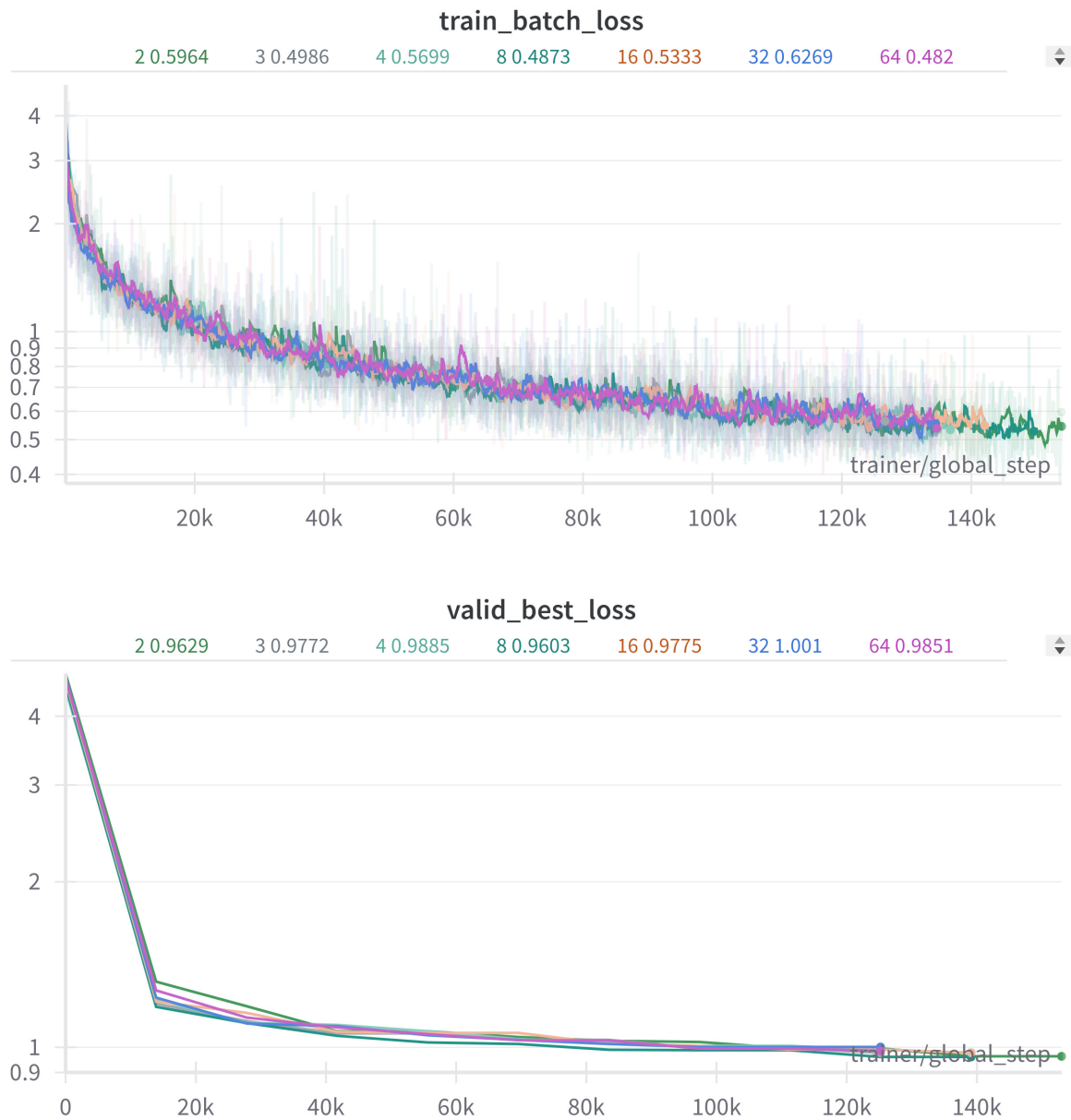


Figure 2.17: Ablation of the optimal decoder w.r.t. the number of bins.

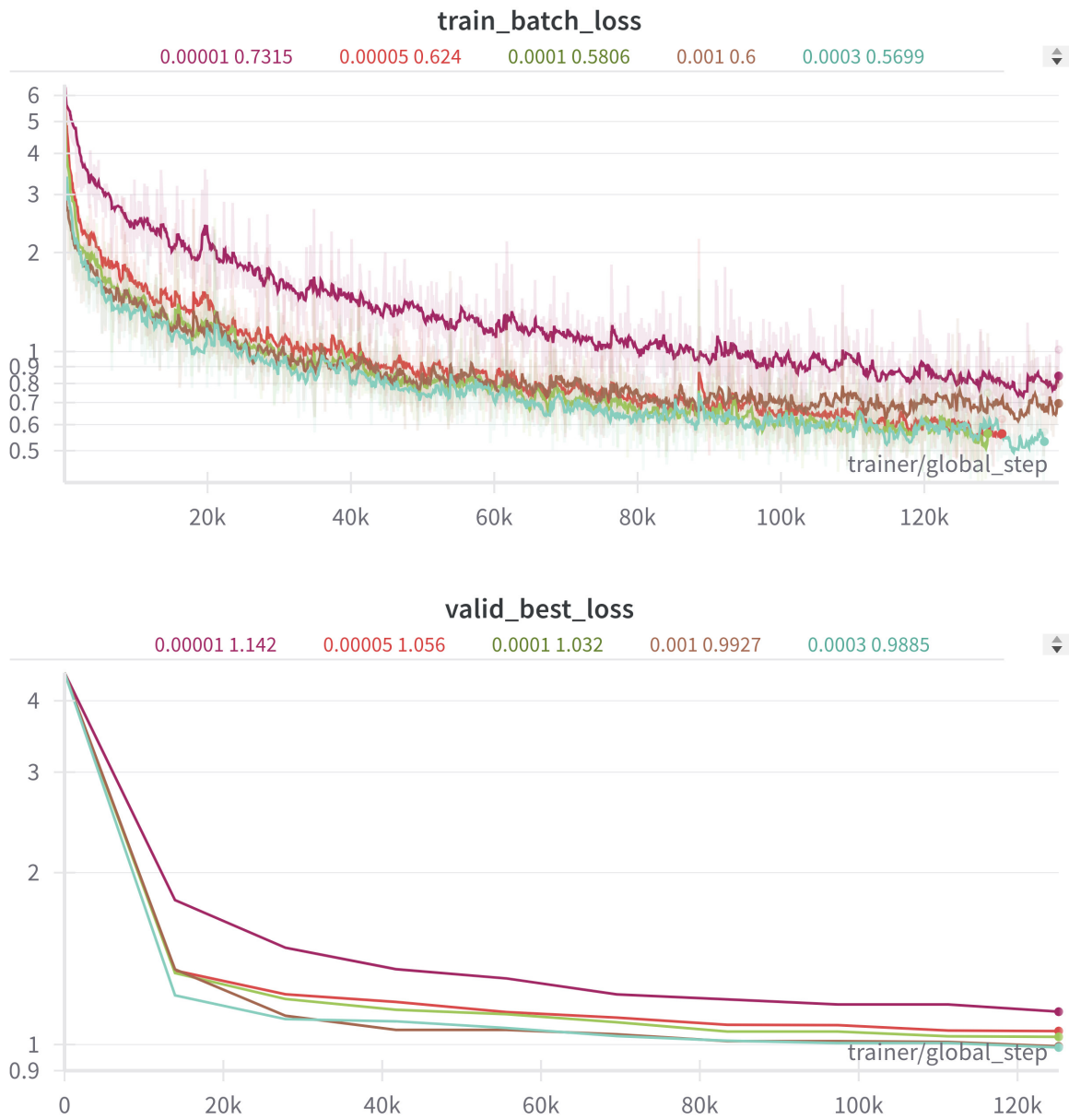


Figure 2.18: Ablation of the optimal decoder w.r.t. the learning rate.

Model	History	Time stride	# bins	Learning rate	Train loss	Test loss
Vanilla	4	1	4	0.0003	0.4207	1.064
With Pool	4	1	4	0.0003	0.3672	0.9603
DepthWise	1	1	4	0.0003	0.664	1.02
DepthWise	2	1	4	0.0003	0.4539	0.99
DepthWise	4	1	4	0.0003	0.5699	0.9885
DepthWise	8	1	4	0.0003	0.4488	1.013
DepthWise	4	2	4	0.0003	0.469	1.009
DepthWise	4	4	4	0.0003	0.5837	0.9873
DepthWise	4	1	2	0.0003	0.5964	0.9629
DepthWise	4	1	3	0.0003	0.4986	0.9772
<b>DepthWise</b>	<b>4</b>	<b>1</b>	<b>8</b>	<b>0.0003</b>	<b>0.4873</b>	<b>0.9603</b>
DepthWise	4	1	16	0.0003	0.5333	0.9775
DepthWise	4	1	32	0.0003	0.6269	1.001
DepthWise	4	1	64	0.0003	0.482	0.9851
DepthWise	4	1	4	0.00001	0.7315	1.142
DepthWise	4	1	4	0.00005	0.624	1.056
DepthWise	4	1	4	0.0001	0.5806	1.032
DepthWise	4	1	4	0.001	0.6	0.9927

Table 2.3: Ablation of the various model trained during our work.

Figure 2.19 illustrates the predictive capability of our model with images captured within our laboratory. Without access to a high-quality depth-sensing device, we lack precise ground truth data; however, the qualitative assessment of the model’s output is quite promising. The model displays a competent level of depth estimation: for example, the area within the drone in the first image is appropriately identified as being further away, while the door is rightly estimated as closer to the camera with depth values diminishing along the walls.

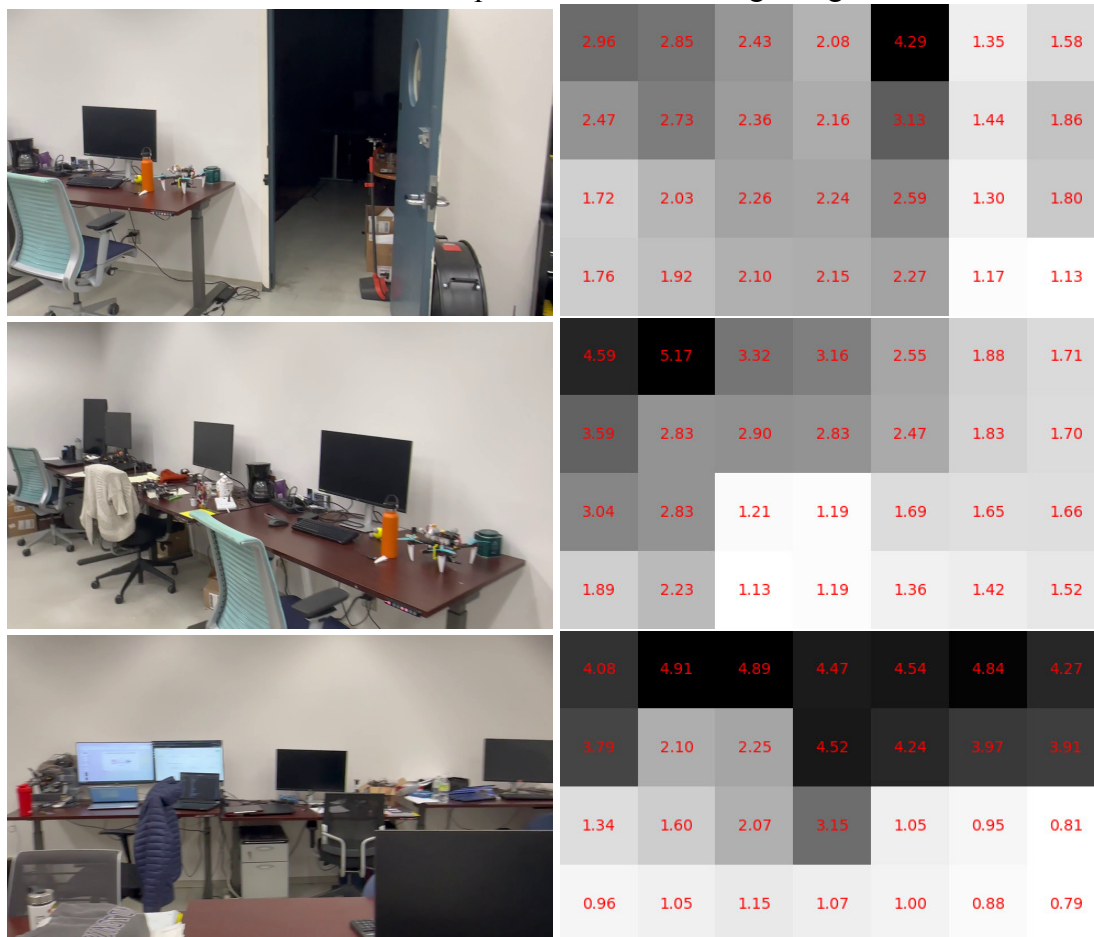


Figure 2.19: Predictions of the model for images from our laboratory. No ground truth depth map is available for ablation.

In the second image, the region towards the upper left is anticipated to be the most distant from the camera, whereas the chair is predicted to be much nearer. The model also correctly infers that the distance increases progressively along the table’s length and the adjacent wall. Finally, for the third image, the model estimates the monitor at the bottom right to be about 70cm from the camera, while the opposite wall shows greater distances, reaching a maximum at the top of the image where it is furthest from the camera.

Figure 2.20 illustrates our model’s performance on the NYU Depth V2 dataset. There are evident discrepancies between the model’s predictions and the actual ground truth. For instance, in the first image, the model successfully identifies the open door and estimates the distance to the wall behind it. However, it misses the closer collision distance presented by the door itself. Additionally, the model displays consistent errors, with a minimum deviation of about 50cm, often misjudging closer objects as being farther away.

The second set of images highlights a weakness of our model: its inability to accurately estimate depth on reflective surfaces. The mirror in the left portion of the room, for example, is inaccurately estimated to be around 3 meters away, when it should be much nearer to the sensor. The third image, however, shows a prediction that aligns more logically with the scene when

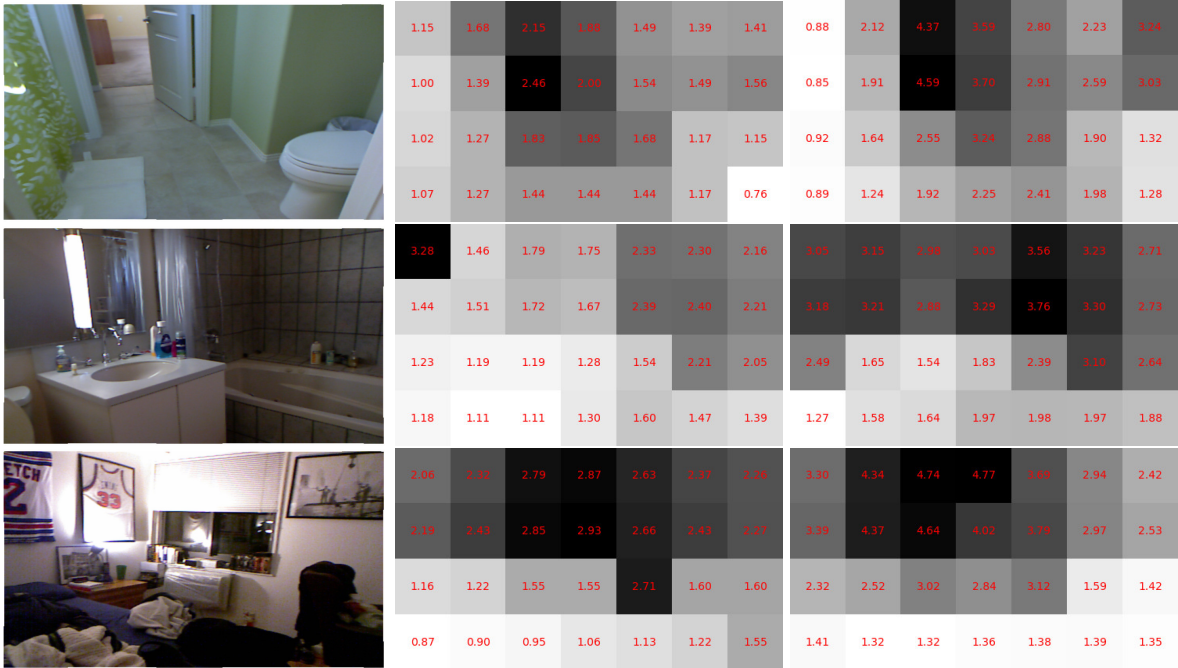


Figure 2.20: Coarse Depth Map predictions on NYU Depth V2. From left to right: input RGB image, target depth map, and prediction.

compared to the RGB image. The model’s estimation of the bed at 1.4 meters appears more credible than the ground truth’s 0.67 meters. Yet, it overestimates the distance to the far corner of the room, suggesting it is around 5 meters away, which seems improbable for a standard-sized bedroom, especially considering the bed is the only object between the camera and the corner.

It’s important to note that the environments in which we trained our model were less cluttered compared to those in the NYU Depth V2 dataset. Typically, our training scenes are indoor but with larger areas. We believe that with more varied data in our training set, the model will be able to generate high-quality predictions for a wide variety of environments it may encounter.

Finally, the predictions on the test set from the *hospital* environment are displayed in figure 2.21. When compared to the NYU Depth V2 dataset, the predictions here align closely with the ground truths, demonstrating the model’s ability to effectively estimate depth maps for unseen samples. It is crucial to acknowledge, however, that these samples originate from the same type of environment upon which the model was trained. This similarity in distribution between the training samples and the test set likely contributes to the enhanced performance observed in these predictions.

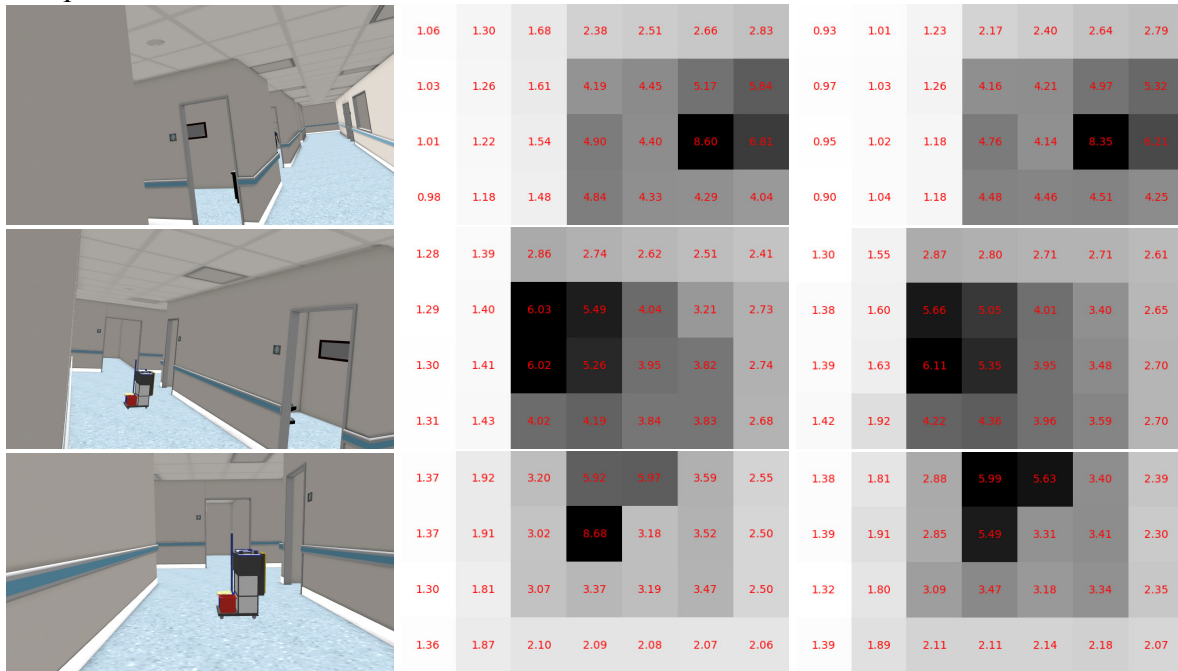


Figure 2.21: Coarse Depth Map predictions on the hospital environment. From left to right: input RGB image, target depth map, and prediction.



# Chapter 3

## Learning Deep Collision Probabilities

This chapter delves into the critical aspects of enhancing robotic autonomy and safety through the precise estimation of collision probabilities using monocular depth estimation. The capacity to accurately predict and avoid potential collisions in real-time is paramount for robots, especially those operating in dynamic and unpredictable environments. Our methodology is inspired by DORN, where the concept of binning plays a central role in refining depth estimation accuracy, particularly for objects close to the robot. Unlike the Adaptive Binning (AdaBins) approach, which adapts bin boundaries for a global perspective on depth distribution, our strategy is meticulously designed for the specific exigencies of robotics and obstacle avoidance. The primary focus is on resolving fine-grained distances near the robot, which are crucial for immediate navigation and safety concerns, rather than distant object detection.

To achieve this, we implement a fixed binning scheme predicated on an exponential model, enhancing bin frequency for closer distances while distributing bins more sparsely as depth increases. This configuration ensures elevated resolution and sensitivity in proximity to the robot, facilitating the timely identification and circumvention of obstacles. Such a strategic choice underscores our commitment to prioritizing near-field precision, aligning seamlessly with the overarching requirements of robotic navigation and obstacle avoidance.

### 3.1 Monocular Depth Estimation For Collision Avoidance

Building upon the foundational principles set by DORN, our approach capitalizes on binning to amplify the precision of depth estimation. This is particularly vital for understanding the spatial relationship between the robot and its immediate surroundings. Our divergence from AdaBins, characterized by its dynamic adjustment of bin boundaries, stems from our focused interest in the intricacies of near-field distances. By adhering to a fixed binning scheme informed by an exponential model, we ensure a granular focus on proximal spaces—where the risk of collision

looms largest—and a de-emphasis on distant objects, which bear less immediate relevance to the robot’s navigational safety.

Figures 3.1 and 3.2 illustrate the adaptability of our binning strategy across different ranges, based on the number of bins. Incorporating more bins allows for a finer-grained representation of depth, enhancing the precision of our analysis. However, this advantage comes with a trade-off: increasing the number of bins also amplifies the complexity of the algorithm. This complexity arises from the expanded 3D volume that represents pixel distances, which is a critical factor in the eventual computation of depth. Therefore, while a higher bin count can improve depth resolution, it also necessitates careful consideration of the algorithm’s efficiency and computational demands.



Figure 3.1: Fixed binning with 10 bins spacing the range [0.01, 10.0]



Figure 3.2: Fixed binning with 32 bins spacing the range [0.01, 10.0]

## 3.2 Forecasting Collision Probabilities

In our pursuit to quantify collision probabilities, we introduce the collision probability as a function of the time-to-collide ( $TTC$ ), which quantifies the duration before a robot collides with an obstacle if it maintains its current trajectory and velocity. Formally, for each discrete time step  $t$ ,  $TTC$  is computed as

$$TTC_t = \frac{d_t}{v_t}, \tag{3.1}$$

where  $d_t$  denotes the minimal distance to the nearest obstacle detected by the robot’s camera, and  $v_t$  is the robot’s velocity. To model the relationship between  $TTC_t$  and the collision probability  $c_t$ , we employ a sigmoid function. This choice is based on the sigmoid’s inherent characteristic of facilitating a smooth probabilistic transition from values nearing 0 to those approaching 1. This smooth transition mirrors real-world dynamics, where the proximity of an obstacle incre-

mentally escalates the likelihood of a collision. The probabilistic model is expressed as

$$c_t = 1 - [1 + k \exp((TTC_t - TTC_0))]^{-1}. \quad (3.2)$$

The parameter  $k$  is a positive scalar that governs the sigmoid curve's steepness, thereby modulating the model's responsiveness to variations in  $TTC$ . This modeling approach enables a rigorous mathematical representation of the collision probability, facilitating its integration into the robot's decision-making algorithms. By adjusting the parameters  $TTC_0$  and  $k$ , the model can be calibrated to different robot dynamics and operational environments, enhancing its applicability across various robotic applications.

Determining  $TTC$  necessitates accurate estimations of velocity and the closest distance to potential obstacles. Velocity estimation is straightforward to measure through conventional state estimation. Conversely, calculating the minimal distance to obstacles presents a more complex challenge, as the camera's field of view includes obstacles of different importance to collision risk. For instance, obstacles that are far away or on the edge of the camera's view might not pose a risk if the robot moves straight ahead, as it will likely avoid them naturally. On the other hand, obstacles directly in the robot's path are a significant collision risk, as the robot will eventually hit them if it continues forward. To address this, we focus on obstacles that appear in front of the robot, as predicted by projecting the robot's size onto the camera's view.

The underactuated dynamics of a quadrotor, which moves forward by tilting (pitching), affect how obstacles are seen on the camera. This pitching motion shifts the position of obstacles in the camera's view. To consider the effect of pitching, we adjust the current depth image by the robot's pitch angle. After this adjustment, we project the robot's shape (simplified to a rectangle for ease) into the center of this modified image. The minimum distance is then defined as the space between the closest obstacle within this projected rectangle and the robot.



# Chapter 4

## Collision-Free Model Predictive Control

The advanced maneuverability of quadrotors, coupled with their growing deployment in complex and dynamic environments, necessitates the development of sophisticated control strategies that ensure safety without sacrificing performance. Among these strategies, Collision-Free Model Predictive Control (CFMPC) stands out for its ability to dynamically integrate obstacle avoidance into the trajectory planning process. This chapter delves into the intricacies of embedding collision probabilities within a Nonlinear Model Predictive Control (NMPC) framework tailored for quadrotor control. Through this approach, we aim to strike an optimal balance between adherence to the desired trajectory and the imperative of avoiding obstacles, thereby paving the way for more resilient and adaptive quadrotor navigation.

### 4.1 The Essence of NMPC for Quadrotor Control

At the core of NMPC lies the principle of optimizing a control action over a finite prediction horizon, based on a model of the system dynamics and constraints. For quadrotors, which are highly agile yet inherently unstable vehicles, NMPC offers a framework for real-time trajectory optimization that accounts for the system's nonlinearities and input constraints. This control strategy is particularly advantageous for quadrotors due to its ability to accommodate the complex dynamics and underactuated nature of these aerial vehicles, enabling precise navigation and maneuverability even in tightly constrained spaces.

The application of NMPC in quadrotor control involves forecasting the system's future states over a predefined horizon based on a mathematical model of the quadrotor dynamics. The optimization process seeks to minimize a cost function that typically encompasses deviations from a reference trajectory, control effort, and, in the context of CFMPC, penalties associated with the probability of collision. This predictive capability allows the controller to anticipate and mitigate potential collisions well in advance, ensuring smoother and safer flight trajectories.

## 4.2 Incorporating Collision Probabilities into NMPC

To embed collision avoidance directly into the NMPC framework, we introduce a novel cost function component that explicitly accounts for collision probabilities. This approach enables the controller to evaluate potential future trajectories not only in terms of their adherence to the desired path but also their safety with respect to obstacles. By integrating the collision probability as a quadratic term in the cost function, we provide the NMPC optimizer with a quantitative measure of collision risk that influences the selection of optimal control inputs.

### 4.2.1 Formulation of the Optimal Control Problem

The Optimal Control Problem (OCP) for CFMPC is formulated with multiple shooting steps over a prediction horizon of  $N$ , aiming to minimize a cost function composed of three key terms: the tracking error for both states and inputs, and the square of collision probabilities at each step. Mathematically, the OCP is expressed as follows:

$$\min_{\substack{\mathbf{x}_0, \dots, \mathbf{x}_N \\ \mathbf{u}_0, \dots, \mathbf{u}_{N-1} \\ c_0, \dots, c_N}} \sum_{i=0}^N \tilde{\mathbf{x}}_i^\top \mathbf{Q}_x \tilde{\mathbf{x}}_i + \sum_{i=0}^{N-1} \tilde{\mathbf{u}}_i^\top \mathbf{Q}_u \tilde{\mathbf{u}}_i + \sum_{i=0}^N c_i^2 \quad (4.1)$$

$$\begin{aligned} \text{s.t.} \quad & \mathbf{x}_{i+1} = h(\mathbf{x}_i, \mathbf{u}_i), & i = 0, \dots, N-1 \\ & c_i = f_{\text{col}}(\mathbf{r}_i), & i = 0, \dots, N \\ & \mathbf{r}_{i+1} = f_{\text{trans}}(\mathbf{r}_i, \mathbf{u}_i), & i = 0, \dots, N-1 \\ & \mathbf{r}_0 = f_{\text{rep}}(\mathbf{I}_0) \\ & \mathbf{x}_0 = \hat{\mathbf{x}}_0 \\ & g(\mathbf{x}_i, \mathbf{u}_i) \leq 0 \end{aligned}$$

where  $\mathbf{Q}_x$  and  $\mathbf{Q}_u$  are positive semi-definite diagonal weight matrices, while  $\tilde{\mathbf{x}}_i = \mathbf{x}_{\text{des},i} - \mathbf{x}_i$ , and  $\tilde{\mathbf{u}}_i = \mathbf{u}_{\text{des},i} - \mathbf{u}_i$  are the errors between the desired state and input and the actual state and input. Therefore, the cost function calculates the discrepancy between the predicted and reference states over the time horizon, using multiple reference points. The system dynamics are represented by  $h(\mathbf{x}_i, \mathbf{u}_i)$  and the initial state is constrained to the current estimate  $\hat{\mathbf{x}}_0$ . The problem is further constrained by state and input constraints  $g(\mathbf{x}_i, \mathbf{u}_i) \leq 0$  which comprise actuator constraints [32].

In our implementation, we formulate the NMPC OCP in eq. (??) with  $N = 20$  shooting steps, covering the evolution of the system over 1 s. The optimization is solved using sequential quadratic programming and a real-time iteration scheme [33] with its implementation in the

acados package [34]. The Quadratic Programming (QP) subproblems are obtained using the Gauss-Newton Hessian approximation and regularized with a Levenberg-Marquardt regularization term to improve the controller robustness. The QPs are solved using the high-performance interior-point method in HPIPM [35] with full condensing and the basic linear algebra library for embedded optimization BLASFEO [36].

The OCP in eq. (??) is weighted by the definite diagonal weight matrices

$$\begin{aligned}\mathbf{Q}_p &= \text{diag}(200, 200, 300) , \\ \mathbf{Q}_v &= \text{diag}(10, 10, 10) , \\ \mathbf{Q}_q &= \text{diag}(150, 150, 200, 1) , \\ \mathbf{Q}_\omega &= \text{diag}(10, 10, 10) , \\ \mathbf{Q}_u &= \text{diag}(5, 5, 5, 5) , \\ \mathbf{Q}_c &= \text{diag}(1) ,\end{aligned}$$

where  $\text{diag}$  denotes a diagonal matrix and is formulated to respect the state and input constraints

$$\begin{aligned}-20 &\leq \mathbf{v}_i \leq 20 & \forall i \in [0, 2] \text{ [m s}^{-1}\text{]}, \\ -3.14 &\leq \boldsymbol{\omega}_i \leq 3.14 & \forall i \in [0, 2] \text{ [rad s}^{-1}\text{]}, \\ 0.01 &\leq \boldsymbol{\Omega}_i \leq 1.124 & \forall i \in [0, 3] \text{ [N]}.\end{aligned}$$

The desired control inputs  $\mathbf{u}_{\text{des},i}$  can be obtained from the flat outputs of a differential-flatness planner [37], [38]. The planner, leveraging the property of differential flatness, designs optimal trajectories in the reduced space of flat outputs. These trajectories are then transformed into the full state and input space due to the unique mapping with respect to the flat outputs, resulting in the desired control inputs.

## 4.2.2 Quadrotor Dynamics Model

We introduce the dynamics model of the quadrotor's system. Table 4.1 lists the relevant variables used in the paper. Nominal methods model the quadrotor's system dynamics by using nonlinear ordinary differential equations. Specifically, consider the quadrotor system modeled by the state vector  $\mathbf{x} = \left[ \mathbf{p}^\top \mathbf{v}^\top \mathbf{q}^\top \boldsymbol{\omega}^\top \right]^\top$  and the control input  $\mathbf{u}$ , then the quadrotor's nominal

$\mathcal{I}, \mathcal{B}$	inertial, body frame
$m$	mass of quadrotor in $\mathcal{I}$
$\mathbf{p} \in \mathbb{R}^3$	position of quadrotor in $\mathcal{I}$
$\mathbf{v} \in \mathbb{R}^3$	linear velocity of quadrotor in $\mathcal{I}$
$\mathbf{q} \in \mathbb{R}^4$	orientation of quadrotor with respect to $\mathcal{I}$
$\boldsymbol{\omega} \in \mathbb{R}^3$	angular velocity of quadrotor in $\mathcal{B}$
$\mathbf{u} \in \mathbb{R}^4$	motor commands generated by quadrotor's controller
$\dot{\mathbf{v}} \in \mathbb{R}^3$	linear acceleration of quadrotor in $\mathcal{B}$
$\dot{\boldsymbol{\omega}} \in \mathbb{R}^3$	angular acceleration of quadrotor in $\mathcal{B}$
$f \in \mathbb{R}$	total thrust of quadrotor
$\boldsymbol{\tau} \in \mathbb{R}^3$	torque of quadrotor in $\mathcal{B}$
$\mathbf{J} \in \mathbb{R}^{3 \times 3}$	diagonal moment of inertia matrix of quadrotor
$k_f$	rotor thrust constant
$k_\tau$	rotor torque constant
$l$	length of the quadrotor arm
$g$	gravity constant
$\odot$	quaternion-vector product

Table 4.1: Notation table.

dynamics evolve as follows

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{\mathbf{p}} \\ \dot{\mathbf{v}} \\ \dot{\mathbf{q}} \\ \dot{\boldsymbol{\omega}} \end{bmatrix} = \begin{bmatrix} \mathbf{v} \\ \frac{1}{m}(\mathbf{q} \odot f) + \mathbf{g} \\ \frac{1}{2}(\mathbf{q} \odot \boldsymbol{\omega}) \\ \mathbf{J}^{-1}(\boldsymbol{\tau} - \boldsymbol{\omega} \times \mathbf{J}\boldsymbol{\omega}) \end{bmatrix} = h(\mathbf{x}, \mathbf{u}), \quad (4.2)$$

where  $\mathbf{J} = \text{diag}(J_{xx}, J_{yy}, J_{zz})$  is the diagonal moment of inertia matrix,  $\mathbf{g} = [0 \ 0 \ -g]^\top$  is the gravity vector, and the collective thrust  $f$  and torque  $\boldsymbol{\tau}$  of the quadrotor are defined as

$$f = k_f \sum_{i=0}^3 u_i^2, \quad \boldsymbol{\tau} = \begin{bmatrix} k_f l (u_0^2 + u_1^2 - u_2^2 - u_3^2) \\ k_f l (-u_0^2 + u_1^2 + u_2^2 - u_3^2) \\ k_\tau (u_0^2 - u_1^2 + u_2^2 - u_3^2) \end{bmatrix}. \quad (4.3)$$

The parameters  $J_{xx}, J_{yy}, J_{zz}, m, k_f, k_\tau, l$  are related to the physical system and strictly define the nominal model  $h$ . Accurately identifying their values is key for guaranteeing high-performance flight control while using nominal dynamics. However, precisely modeling the system's parameters is very difficult due to the nonlinearity of external effects that make the estimation process difficult.



### 4.2.3 State’s propagation

The classic NMPC (Nonlinear Model Predictive Control) employs a dynamic model of the robot, which dictates that given a specific state and action, it can predict the next state. This state includes vital information such as position, linear velocity, orientation, and angular velocity, while the actions are defined by the **RPMs** of the robot’s four motors. Each time the NMPC resolves an optimization problem, it recursively utilizes the dynamics model, performing a roll-out over the horizon to predict how a given action at a current state will affect the state at time  $t + 1$ .

Leveraging this capability inherent to MPC, which is essentially provided ‘for free’, we utilize the velocity predicted by the MPC for a future action at time  $t + 1$ . We then take our depth map at time  $t$ , and perform a **roto-translation** (using quaternions for rotation and velocity for translation), thereby updating our depth map - which, at this stage, could be considered a point cloud. Essentially, we deploy a model that converts the quaternion into a rotation matrix, applies this matrix to all points in the point cloud that represents our current state, and calculates the new point cloud configuration.

Based on this transformed point cloud, both the robot’s size and the 3D points from the point cloud are reprojected into 2D. This projection allows us to identify the point within it that is closest to the quadrotor (if present). The collision probability is then computed based on the time-to-collide, which is derived from the minimum distance to the quadrotor and the predicted velocity. The computed **TTC** is in fact forwarded to a translated sigmoid 4.4 that shrink the value into the range  $[0,1]$ , making it an ideal representation of the collision probability.

$$c_t = 1 - \frac{1}{1 + 10^{(-TTC)}} \quad (4.4)$$

This approach not only enhances the precision of our trajectory predictions but also significantly improves the safety measures by enabling real-time, dynamic adjustments to the flight path based on the predicted and updated spatial configurations.

## 4.3 Results

In this study, we present a novel framework that effectively integrates the local planning capabilities of Nonlinear Model Predictive Control (NMPC) with advanced Deep Learning techniques. Specifically, we employ a Deep Neural Network to analyze RGB images for real-time collision avoidance, demonstrating proficiency in generating depth maps from these images. Furthermore, the predicted information from the NMPC are then exploited to propagate the current state of the robot into an horizon in the future, as described in 4.2.3.

During each optimization cycle of the NMPC, a **history of images** serves as the input, allowing the model to predict depth maps for the most recent frame at time step  $t$ . The NMPC then uses this information to forecast potential velocities over a future horizon of  $N$  steps.

The system features a forward-facing RGB camera securely mounted on the vehicle, ensuring the robot's projection within the depth map remains centered and dynamically adjusts with the drone's **pitch**. As the quadrotor reaches its peak velocity, the camera tilts downward, moving its projection above the center of the image. Conversely, as the drone decelerates and pitches backward, the camera angles upwards. In each scenario, the drone's projection is shifted vertically to continually align with its actual trajectory and effectively manage collision avoidance with imminent front-facing obstacles.

We specifically focus on the effectiveness of our NMPC framework in modifying the drone's trajectory when encountering obstacles. Figure 4.1 illustrates the *actual odometry* of the drone, shown in green, alongside the *intended position command* in red. Flags within the image mark waypoints that delineate the planned trajectory, which is dynamically adjusted to avoid any obstacles in the path. Figure 4.2 further demonstrates the drone's ability to navigate around a corner by subtly shifting its path to ensure a safe passage. The trajectory adjustments are smooth, and while they are not dramatic, they are essential. Without our framework, the MPC would simply adhere to the original red trajectory, potentially leading to collisions. Additionally, figure 4.4 presents variations in the trajectory across different axes, along with the collision probability at each timestamp. The first three graphs display the odometry in red and the position command in blue. A separate graph in green indicates the collision probability. It is noteworthy that the drone deviates from the intended path, widening its route to maintain a safer distance from obstacles as the collision probability increases, particularly near the corner.

Our innovative Collision-Free Model Predictive Control (CFMPC) framework constitutes a significant advancement in quadrotor control, particularly for navigation in complex environments. By integrating collision probabilities into the NMPC's cost function, we not only enhance the quadrotor's safety but also retain the flexibility and efficiency of trajectory planning. This approach showcases the crucial role of predictive modeling and optimization in achieving secure and autonomous drone flight, illustrating the potential of combining advanced control theories to address the multifaceted challenges in modern robotics.

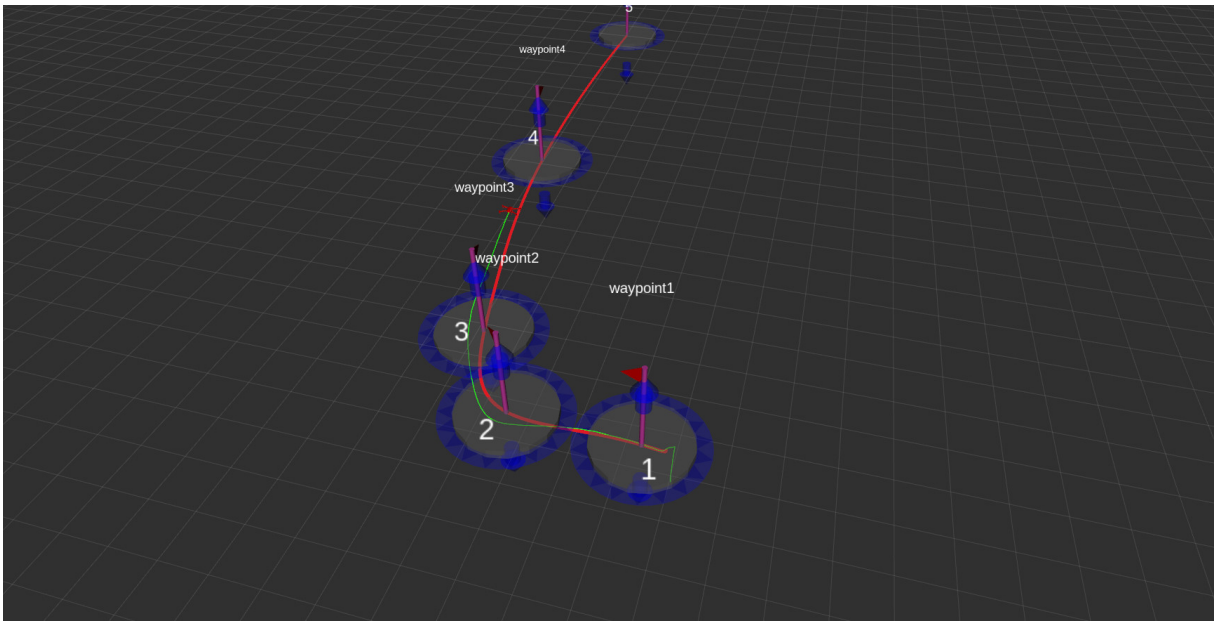


Figure 4.1: Top-down view of the robot's odometry with respect to the position command

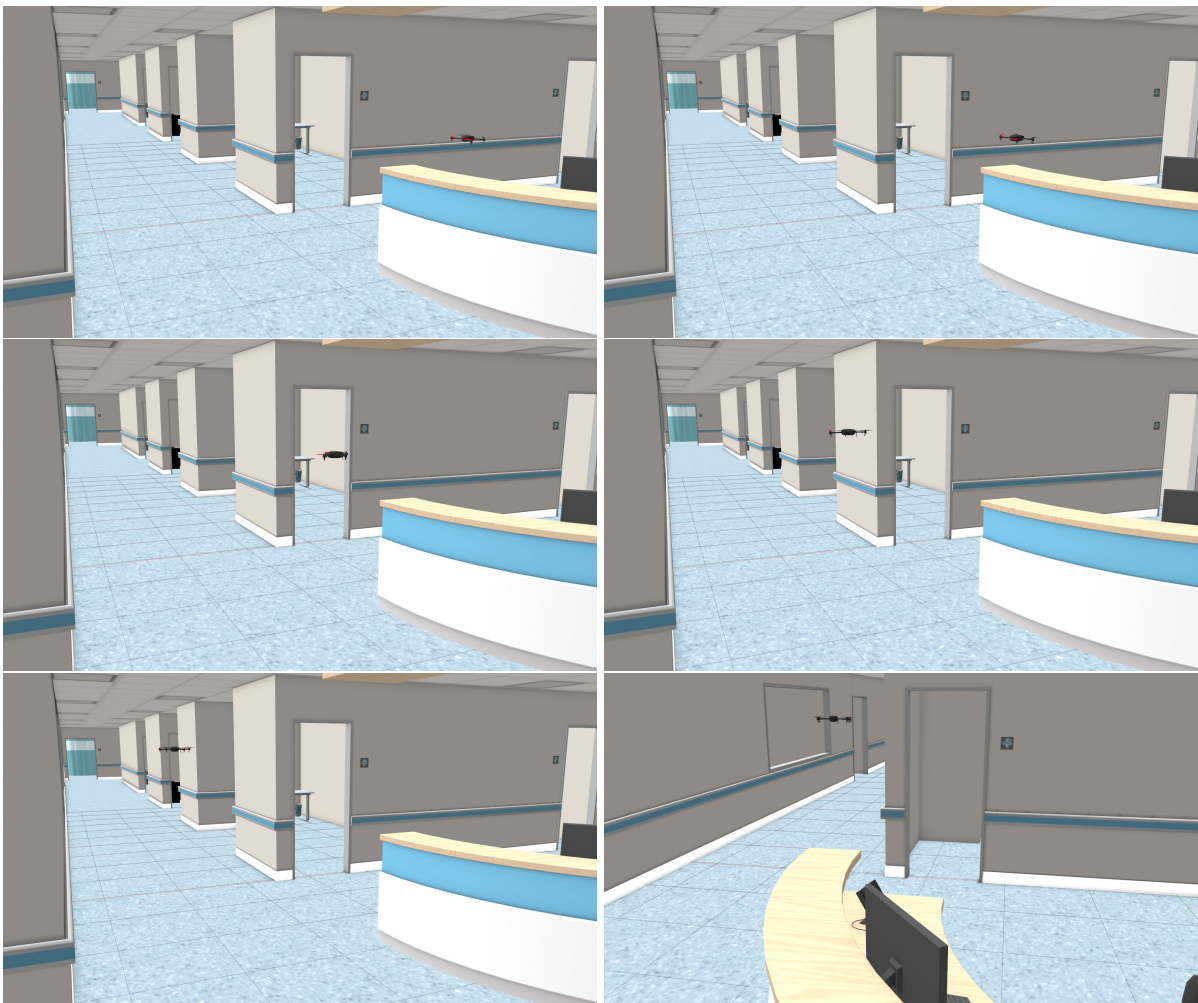


Figure 4.2: Views demonstrating how the drone avoids a corner by expanding its trajectory.

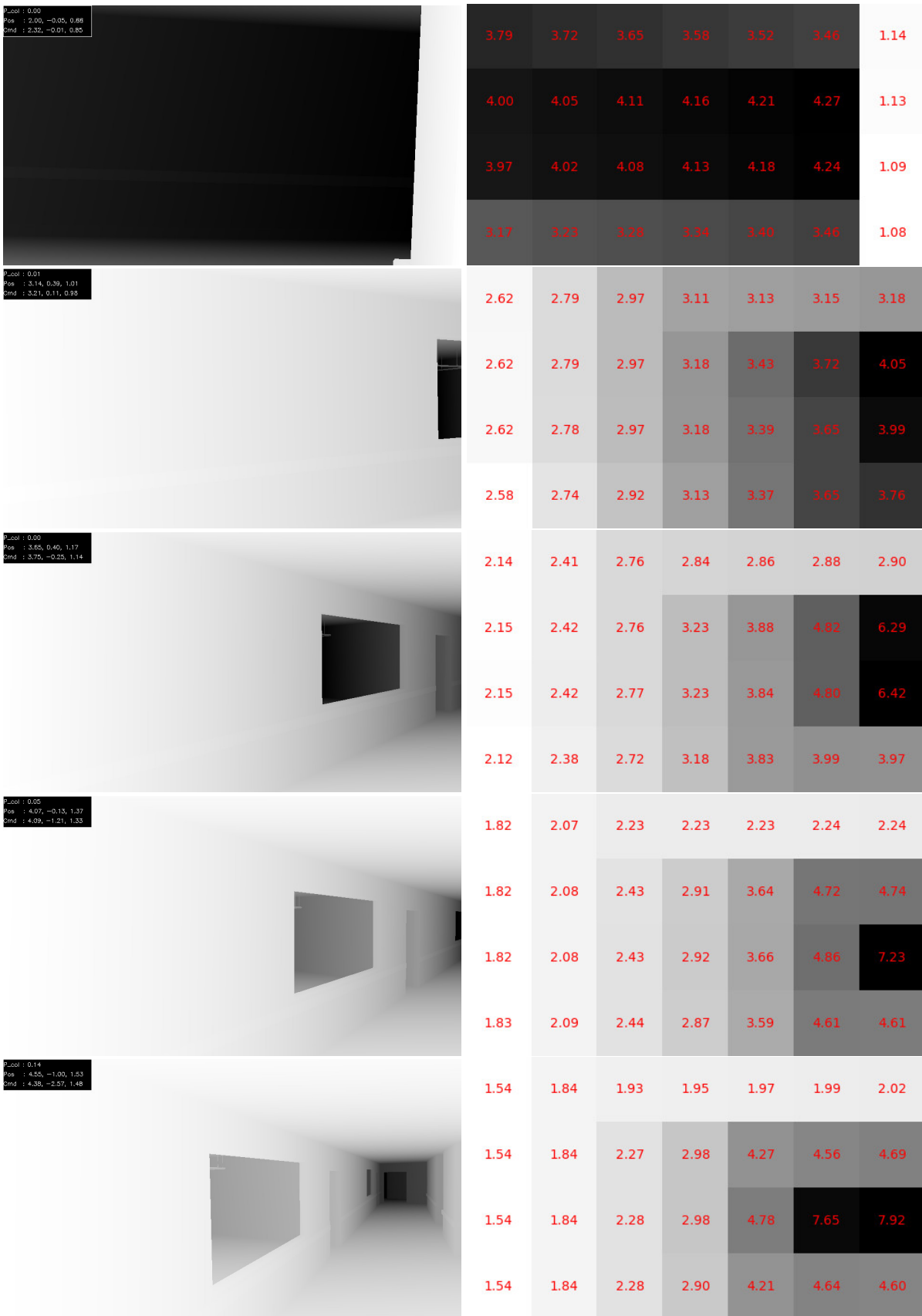


Figure 4.3: Ground truth and depth predictions on the corner demonstrating how the drone avoids a corner by expanding its trajectory.

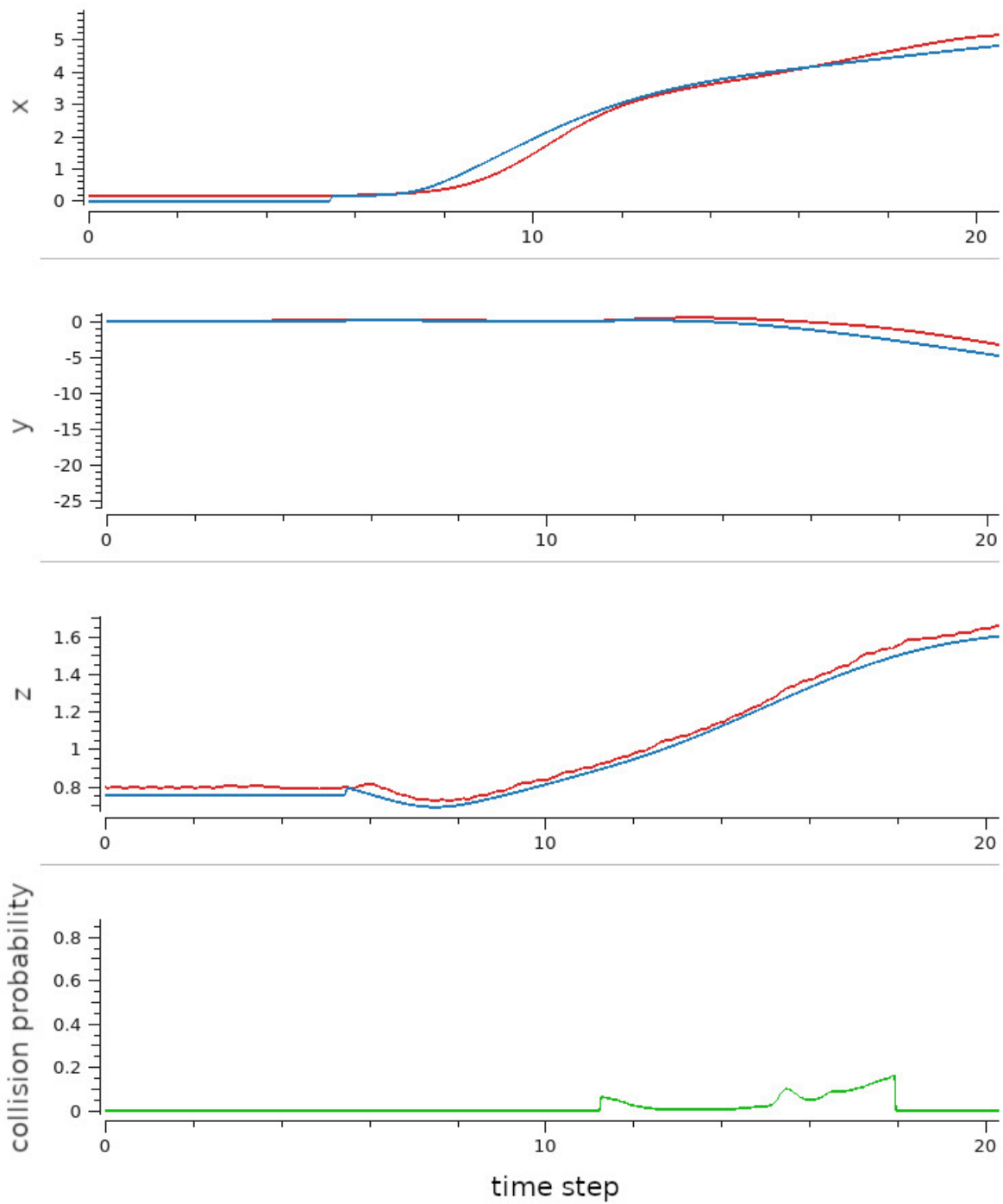


Figure 4.4: Red: actual odometry. Blue: intended position command. Green: collision probability.



# Chapter 5

## Conclusions

This work represents a significant progression in monocular depth estimation and end-to-end collision avoidance without preliminary planning. Initially, our focus was on depth estimation, which we successfully achieved using the model detailed in 2.2.5. This model enables depth reconstruction from sequences of RGB images. As demonstrated by the ablation studies in 2.4, utilizing image sequences enhances performance compared to using single images. This approach draws inspiration from *stereo vision* techniques, which employ triangulation from multiple viewpoints to estimate depth. Despite variations in camera sensor positioning during capture—a consideration crucial in stereo imaging—our network effectively leverages the increased data from different scene views (captured at distinct times) to refine object distance estimations.

We believe there is considerable scope to enhance our model. Future work involves expanding the dataset used for training, validating, and testing our model, thus maximizing the architecture’s capabilities and enabling better generalization across diverse environments, both simulated and real. Furthermore, by employing the Depth Anything feature extractor, we have minimized the **sim-to-real** gap between simulated and real images. This allows for training with simulated data, which is significantly easier to obtain compared to real-world data. Real-world data collection presents challenges, especially due to sensor noise, which is difficult to mitigate in lightweight drones that cannot accommodate heavier, more sophisticated devices.

Operating at a frame rate of 100Hz, the framework can reconstruct depth maps in real-time, marking a substantial enhancement in the field that can be utilized by any chosen controller.

Regarding Model Predictive Control, our initial strategy was to deploy a simple Recurrent Neural Network (**RNN**) that could predict future states of the drone (i.e., depth, position, linear and angular velocities, and orientation) based on current states and control actions (i.e., motor speeds), and calculate collision probabilities. However, integrating such an architecture within the ACADOS framework proved exceedingly complex. We then experimented with a simpler

Multi-Layer Perceptron (**MLP**), but the results were suboptimal. This method's shortcomings are currently under investigation, with remedial measures planned for future research.

To address these challenges, we adopted a more straightforward approach by harnessing velocity data from the Inertial Measurement Unit (**IMU**) and using MPC to predict these measurements. This mathematical formulation of the problem enabled us to determine new states of the robot and calculate the probability of future collisions, thereby identifying an optimal trajectory for collision-free navigation.

The successful implementation of this model signifies a step forward in autonomous systems, bridging the gap between theoretical research and practical applications. Our results underline the robustness of combining classical control methods with modern machine learning techniques to solve real-world problems. This synergy not only enhances the performance of autonomous systems but also opens up new avenues for future innovations.

In conclusion, our study highlights the transformative potential of integrated approaches in robotics, paving the way for more sophisticated and reliable autonomous navigation solutions. As we continue to refine these technologies, the prospects for their application across various industries become increasingly promising, heralding a new era of efficiency and safety in autonomous operations.



# Chapter 6

## Appendix

### 6.1 Data Collection

The foundation of our project was initially laid on the concept of collecting depth information directly from real-world environments. This approach aimed at capturing detailed spatial information essential for accurate monocular depth estimation and effective obstacle avoidance.

For our initial experiments, we selected the Intel RealSense L515 LiDAR camera, a forefront device in depth sensing. The L515's efficient LiDAR sensor not only captures precise depth information but is also characterized by its lightweight and compact design, making it ideally suited for drone integration and usage in small-sized embedded robots, offering a practical solution for real-world data collection. It is engineered to discern depth information accurately up to a distance of 9 meters, albeit with performance significantly influenced by environmental factors such as lighting, occlusion, and object surface properties.

A notable feature of the L515 is its capability for real-time 3D image reconstruction. Utilizing LiDAR technology that deploys millions of laser pulses, it can reconstruct complex real-world scenes. Nonetheless, this sophisticated process tends to introduce substantial noise into the depth maps, posing challenges in environments not tailored to minimize such drawbacks.

Adjusting the LiDAR Camera's parameters did yield some improvement in the depth map quality. However, the outcomes were not sufficiently reliable for incorporation into our framework. The primary issue encountered was the excessive noise within the depth maps, rendering them unsuitable for direct application as model inputs or for training a neural network for depth estimation. Despite extensive efforts to optimize the camera's performance, the resulting depth maps consistently contained large amounts of missing information, particularly problematic in areas with dark or reflective surfaces.

Figure 6.1 showcase the severity of these issues, with many objects poorly represented in the depth map. The grainy and noisy sections indicate pixels for which the sensor failed to

accurately determine distances. Additionally, the field of view (FoV) discrepancy between the RGB and depth maps further complicates data utilization. Objects that are clearly visible in the RGB scene appear more occluded in the depth map, necessitating adjustments for effective use of the captured data.

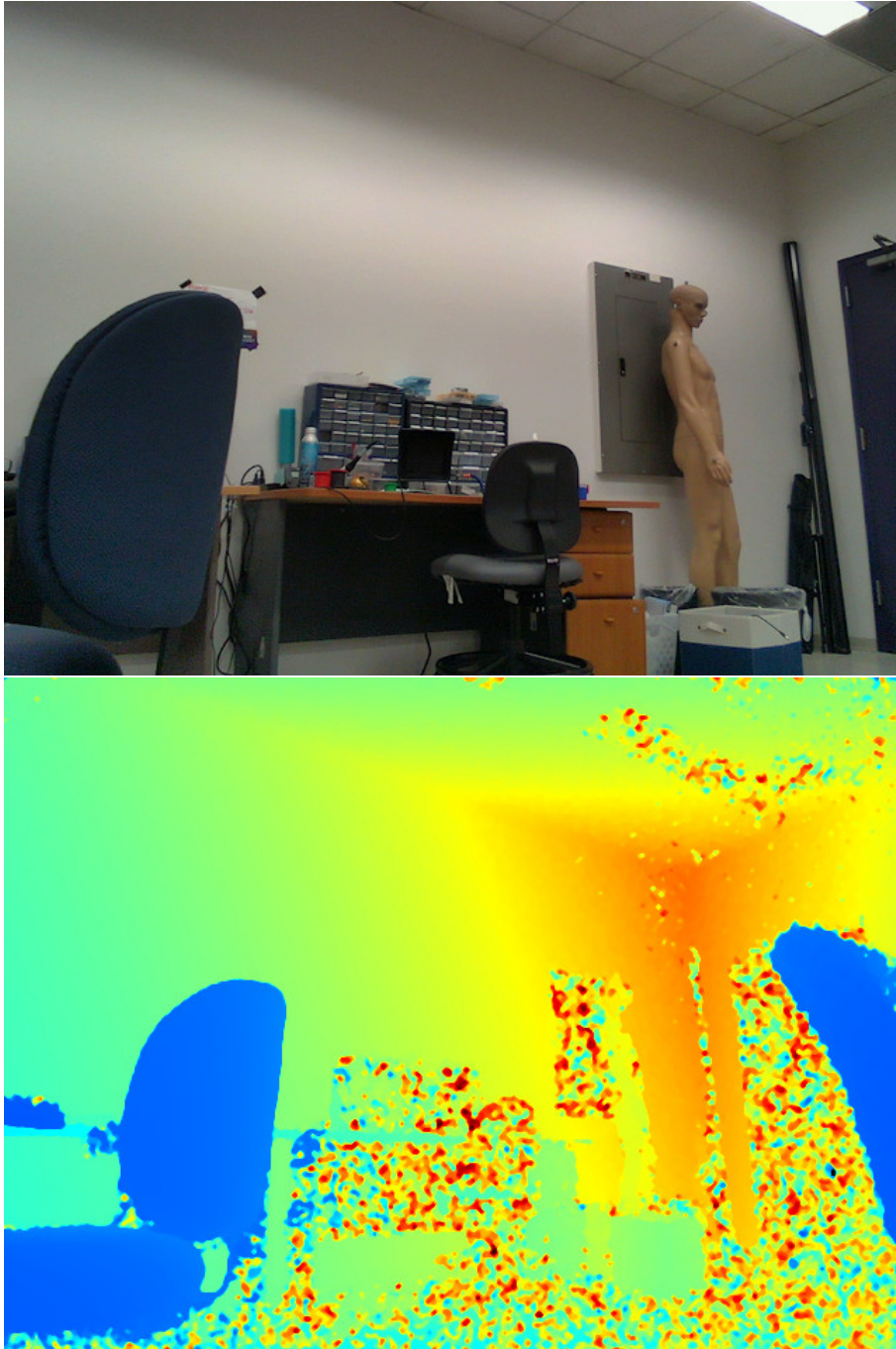


Figure 6.1: RGB image and depth map captured with L515 LiDAR camera



Figure 6.2: The RealSense L515 LiDAR camera

## 6.2 Foundation Vision models

Before diving into the framework’s implementation, it’s crucial to comprehend the distinction between absolute and relative depths.

- **Absolute Depth:** This type of depth map includes the actual distance value of each pixel in an image, often expressed in meters. Thus, each element is associated with a distance value that accurately defines how far that element is from the camera.
- **Relative Depth:** Unlike absolute depth, relative depths do not provide a numeric distance value to an object from the camera. Instead, they can be envisioned as a heatmap indicating which elements are closer or farther away, offering a qualitative assessment of element proximity within a scene.

Initially, our objective was to leverage absolute depth for predicting collision probabilities in our quadrotor. We then decided to employ a pretrained model as the backbone of our network. In this context, MobileSAM [39] emerged as a suitable choice. This model is designed to emulate the foundational capabilities of SAM [40], while offering enhanced inference speed and comparable performance.

### 6.2.1 SAM

The **Segment Anything Model** (SAM) stands out as a foundational model that has been trained on approximately 11 million images and 1 billion segmentation masks. According to the authors,

these images and masks cover a diverse range of geographic and economic backgrounds across different countries, thereby reducing the biases typically introduced into the network. SAM uniquely allows users to specify a prompt to control the segmentation process, enabling selective focus on certain elements while ignoring others.

Like many foundational models, SAM leverages the transformer architecture, employing a variant known as the **Vision Transformer**(ViT) for processing visual data. Also, all its variant share a similar structure made of 3 components:

**Image Encoder** : [41] showed that *Masked Auto-Encoders* are scalable self-supervised learners, hence the authors of SAM used a MAE pre-trained Vision Transformer slightly modified to be able to process high resolution inputs as the encoder of the model.

**Prompt Encoder** : based on the type of prompt there is a different kind of encoding: *sparse* prompts, such as points and boxes are represented through *positional encoding*, text-like prompts are embedded through an off-the-shelf encoder from CLIP [42], *dense* prompts, like masks, are encoded using convolution and then summed element-wise with the image embedding.

**Mask Decoder** : finally, the mask decoder maps the image embedding, prompt embedding, and an output token to a mask. This decoder exploits a variant of the decoder block of the traditional Transformer, followed by a dynamic mask prediction head. This uses prompt self-attention and cross-attention in two directions to update all the embeddings. Then, the image embeddings are upsampled and a MLP maps the output token to a dynamic linear classifier, which ultimately predicts the mask foreground probability at each image location.

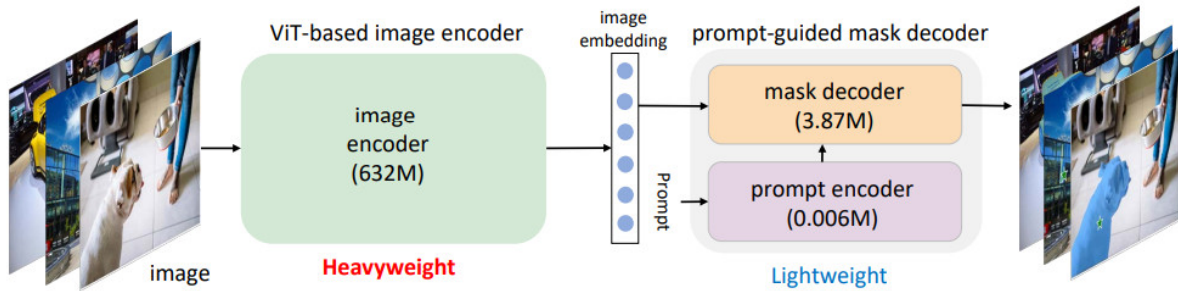


Figure 6.3: Overview of Segment Anything Model

Despite the great work on the architecture, the main contribution introduced with the **Segment Anything Model** is the way it is trained and the data used to perform the training itself. The data-engine has 3 stages:

1. **Assisted-manual stage** : In this stage, a team of professional annotators labeled masks in the images by setting foreground and background objects in the images. So at the very beginning SAM has been trained using public segmentation datasets, then using the newly annotated images. Throughout this stage the team collected 4.3M masks from 120K

images.

2. **Semi-automatic stage** : Here, the authors aimed at increasing the diversity of masks. Images were segmented with the most confident masks, and then the users were asked to manually segment the other unannotated objects. With this stage they managed to increase the number of masks of 5.9M over 180K images.
3. **Fully-automatic stage** : Eventually, the model has been enhanced so much it could predict good masks consistently. In this final stage, authors prompted the model with a 32x32 grid of points, and for each point the model had to predict the masks that may correspond to valid objects. Using IoU and thresholding they selected only *stable* and *confident* masks. At the end of the process the dataset contained 1.1B masks from 11M images.

This structured, incremental learning process enabled SAM to be trained on an unprecedented volume of images and masks, achieving unparalleled generalization across a diverse variety of images.

### 6.2.2 MobileSAM

The ability of SAM to perform amazingly on zero-shot transfer segmentation, and its high versatility among numerous vision applications, made it the baseline for many complex tasks. The main issue with this model is its inference time, which despite being fast enough for real-time browser-web applications, it remains unfeasible to use it on devices with limited resources, and where the time constraint is even harder.

In [39] the authors focused on developing a faster version of SAM, containing a fraction of the parameters of the original one, and gaining the knowledge from the heavy image encoder (ViT-H) to a lightweight transformer encoder, which outputs would be directly compatible with the mask decoder of the original model.

The authors introduce a novel approach to training a streamlined version of SAM, identifying the core challenge in *Knowledge Distillation* (KD) as the intertwined optimization of the image encoder and mask decoder. In traditional KD, the performance of one component is deeply interlinked with the other, leading to a situation where the efficacy of either the image encoder or mask decoder can be compromised by the limitations of its counterpart. To address this, the authors propose a method termed *semi-coupled distillation*. This technique involves freezing the mask decoder and the prompt encoder during the initial phase of training to prevent their performance from being adversely affected by an underperforming image encoder. This approach effectively separates the KD process into two distinct phases: distillation of the image encoder, followed by optional fine-tuning of the mask decoder, as illustrated in fig. 6.4.

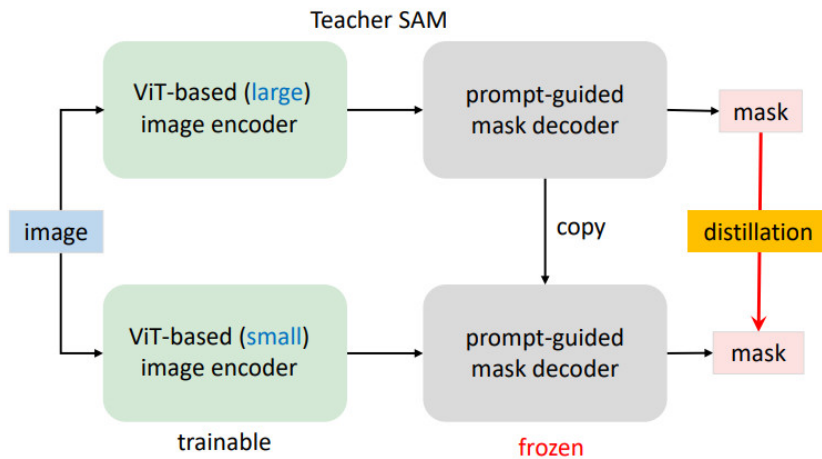


Figure 6.4: Semi-Coupled Knowledge Distillation

Despite these advancements, the authors acknowledge that optimization through this technique remains complex. A significant factor contributing to this complexity is the variability introduced by the randomness in prompt selection, which impacts the mask decoder’s performance. To simplify the distillation process further, they introduce *Decoupled Distillation*. This strategy focuses on distilling the smaller image encoder directly from the large Vision Transformer (ViT-H) used in the original SAM, bypassing the complexities associated with the mask decoder. This method is depicted in fig. 6.5, offering a more straightforward approach to knowledge transfer without engaging the composite decoder mechanism.

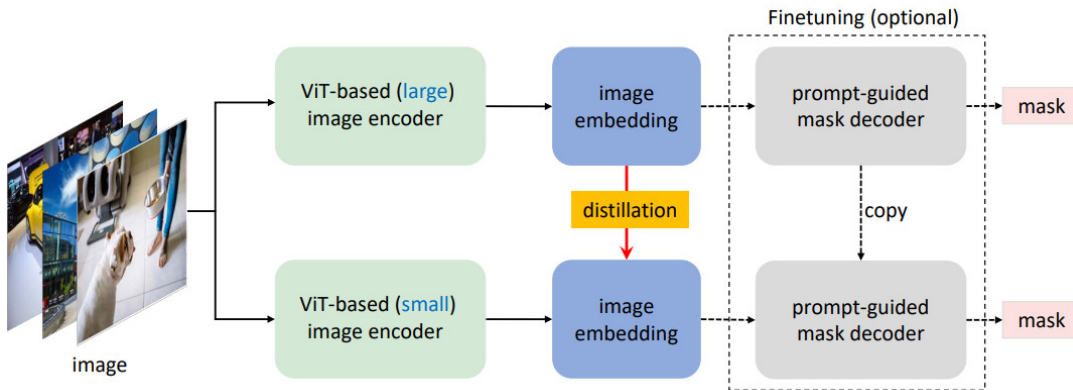


Figure 6.5: Decoupled Knowledge Distillation

### 6.2.3 DepthSAM

Our initial goal was to implement a model that could leverage the knowledge of MobileSAM, distilled from SAM, and adapt it to perform the Monocular Depth Estimation task.

We used MobileSAM for feature extraction, with feature maps having 256 channels and reduced width and height. MobileSAM’s weights are available for different sizes of input images,

hence we used resolution of  $(384 \times 384)$  to reduce the computational demand of the model, despite it being able to work with high-resolution input of size  $(1024 \times 1024)$ .

We tried both to fine-tune the network and to train it from scratch, as well as freezing it completely. At the beginning our idea was that fine-tuning the encoder could have compromised the quality of the feature maps that the model had achieved by mimicking its parent model SAM. We then discovered that, fine-tuning the model was the best option, and achieved better results with respect to the other counterparts. Despite that we couldn't really exploit this architecture too much: after several tries aimed at reducing the computational demand of the model, we couldn't deploy a model that worked real-time. Our *DepthSAM* model performed predictions on histories of images at a frequency rate of 17Hz, inevitably leading us to discard the architecture and to search for more lightweight and faster alternatives.

## 6.3 Background

It is helpful to clarify some key components that underpin our framework. We'll start by exploring essential concepts, from the workings of Convolutional Neural Networks (CNNs) and the definition of foundation models, to the intricacies of Nonlinear Model Predictive Control (NMPC). Beyond these, we'll also touch upon other elements that are crucial to our work. This approach ensures a well-rounded understanding of the foundation upon which our project is built, setting the stage for the detailed exploration that follows.

## 6.4 CNN

**Convolutional Neural Networks** are a specialized type of Neural Network invented for dealing with grid-type data, like images, sequences, or whatever input that can have a grid topology. Proposed in the 1989 by Yann LeCun [43], their name derives from the **convolution** operation they exploit to process the input data. Long story short, CNNs are neural networks that use convolution instead of matrix multiplication in one or more of their layers.

In its most conventional form, the mathematical concept of convolution can be expressed by the following formula

$$(x \star w)(t) = \int_{-\infty}^{\infty} x(\tau)w(t - \tau)d\tau \quad (6.1)$$

where  $x$  is usually referred to as the **input**,  $w$  is the **kernel**, and the output of the operation is called **feature map**. Despite this definition, when working on a computer the data we are dealing with are non-continuous, and such a formulation become useless. For discrete data (i.e.,



the one fed to a whatsoever neural network), the convolution operation is defined as

$$(x \star w)(t) = \sum_{\tau=-\infty}^{\infty} x(\tau)w(t - \tau) \quad (6.2)$$

In machine learning, the input for a CNN is often a multidimensional array, thus the convolution operation need to be adapted to process this kind of data. For instance, when dealing with two-dimensional image  $I$ , we also want (often, but not always) to use a two-dimensional kernel  $K$ , and the formulation is, given the commutative property of the convolution that allows to define the equality among the sums,

$$(I \star K)(i, j) = \sum_m \sum_n I(m, n)K(i - m, j - n) = \sum_m \sum_n I(i - m, j - n)K(m, n) \quad (6.3)$$

In traditional mathematical terms, convolution involves flipping one of the functions before performing the sliding dot product. However, in the context of CNNs, the convolution operation is more akin to another operation called **cross-correlation**, which is actually the one that most library refer to when talking about convolution. The difference is that in the latter the flipping step is typically omitted for simplicity and efficiency, since the learning filters (i.e., the kernels) can adapt regardless of the transposition. Thus, the convolution operation in modern machine learning algorithms is implemented in the following way

$$(I \star K)(i, j) = \sum_m \sum_n I(i + m, j + n)K(m, n). \quad (6.4)$$

In the context of neural networks, the convolution operation has been widely used with great success because it leverages 3 important paradigms:

- **Sparse interactions:** Achieved by utilizing a kernel size smaller than the input size. Traditional neural networks connect each input element to every output through a dense matrix of weights, leading to dense interactions where each output unit interacts with every input unit. Convolutional layers, in contrast, restrict the interaction of each output unit to a subset of input units within the kernel's footprint, denoted as  $k$ . This design choice results in a model that requires fewer parameters and less computational effort. It also reduces the risk of overfitting by limiting the complexity of the model, especially when the input size is very large (e.g., images) and a dense layer would result in an enormous number of parameters. Moreover, by capturing localized features within small regions of the input, convolutional layers facilitate a hierarchical feature learning process, which is particularly effective for tasks involving spatial data, such as image and sequence processing.
- **Parameter sharing:** This paradigm is implemented by utilizing the same kernel, with an



identical set of weights, to process every position of the input. Unlike in dense layers, where each input element is multiplied by a unique weight, leading to a distinct parameter for each connection, convolutional layers employ a singular kernel across the entire input. This approach means that the network learns a single set of parameters, applicable universally across the input field. Parameter sharing dramatically reduces the model's complexity and the number of parameters, enhancing the network's ability to generalize across different regions of the input. This is particularly advantageous for processing data where similar patterns can occur at any location.

- **Equivariance:** A function exhibits equivariance to translation when a shift in the input results in a corresponding shift in the output. Convolutional layers inherently possess this property due to the nature of the convolution operation combined with parameter sharing. This means that if, for example, an object within an image or a pattern within time-series data is moved spatially or temporally, the convolutional layer will produce an output where the detection of this object or pattern is shifted by the same amount. Such behavior ensures that the network's understanding and processing of features is consistent regardless of their position in the input space, which is crucial for effectively handling spatially and temporally varying data. Despite that, convolution is not equivariant to other transformations, such as changes in scale or rotation, for which other mechanisms are necessary to handle them.

### 6.4.1 Convolutional layer

A fundamental layer within a convolutional neural network (CNN) unfolds through a meticulously orchestrated two-stage process, designed to distill and refine the rich, complex patterns embedded within the input data. At its core, the first stage encompasses the actual **convolution operation**, where a linear function is adeptly applied in a sliding-window manner across the input. This strategic operation is pivotal for extracting spatial hierarchies of features, enabling the gradual construction of a profound understanding of the data's structural intricacies through successive layers. Following this, the second stage transitions the linear outputs through the crucible of a non-linear activation function such as ReLU (Rectified Linear Unit), ELU (Exponential Linear Unit), or GELU (Gaussian Error Linear Unit), among a diverse array. This infusion of non-linearity is critical: it gives the model the capability to apprehend and articulate more complex patterns within the data it encounters during the training process.

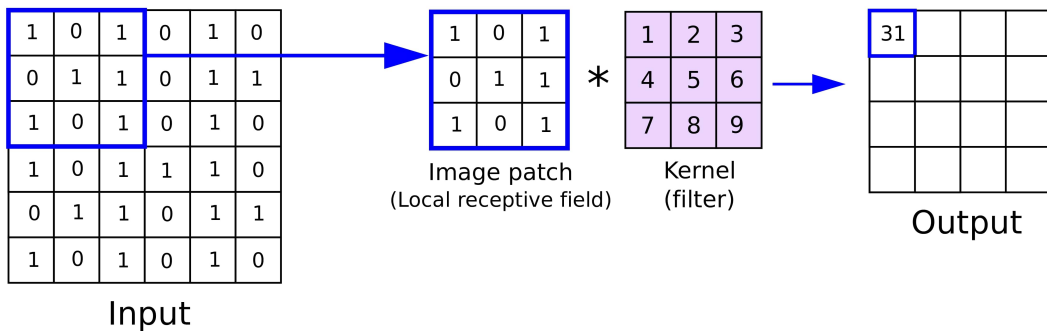


Figure 6.6: Functioning of the convolution operation

### Depth-wise convolution

Within the expansive realm of convolutional operations, depth-wise convolution emerges as a nuanced variant that fundamentally reimagines the convolution process. Contrary to the conventional approach where filters traverse and integrate features across all input channels, depth-wise convolution 6.7 adopts a more granular perspective. Here, each filter is dedicated to a single channel, operating independently. This tailored approach dramatically diminishes the computational load and the parameter count, ushering in a new era of efficiency without sacrificing the depth of feature extraction. Post convolution, these singular channel convolutions coalesce through a subsequent pointwise convolution phase, which amalgamates the independently processed channels into a unified feature map. This ingenious combination of depth-wise and pointwise operations crystallizes into a powerful mechanism for preserving the architectural depth and the nuanced detection of features, all the while operating within a markedly reduced computational framework. Depth-wise convolution, thus, represents a pivotal innovation in the design of lightweight, yet profoundly capable, neural architectures, especially suited for environments where computational resources are at a premium.

### Dilated convolution

Dilated convolution 6.8 is another useful twist on the regular convolution process used in deep learning, especially when you want your model to have a wider view of the input without increasing the size of the model too much. Imagine it like zooming out to see more of a picture without having to step back. In dilated convolution, the filter is spread out so it covers more area of the input with the same number of calculations. This is like poking holes in the filter to skip some parts of the input, making the filter's reach wider without needing more parameters. This method is really helpful for tasks where understanding the broader context or bigger picture is important, like in analyzing sound or predicting future parts of a sequence. By using dilated con-

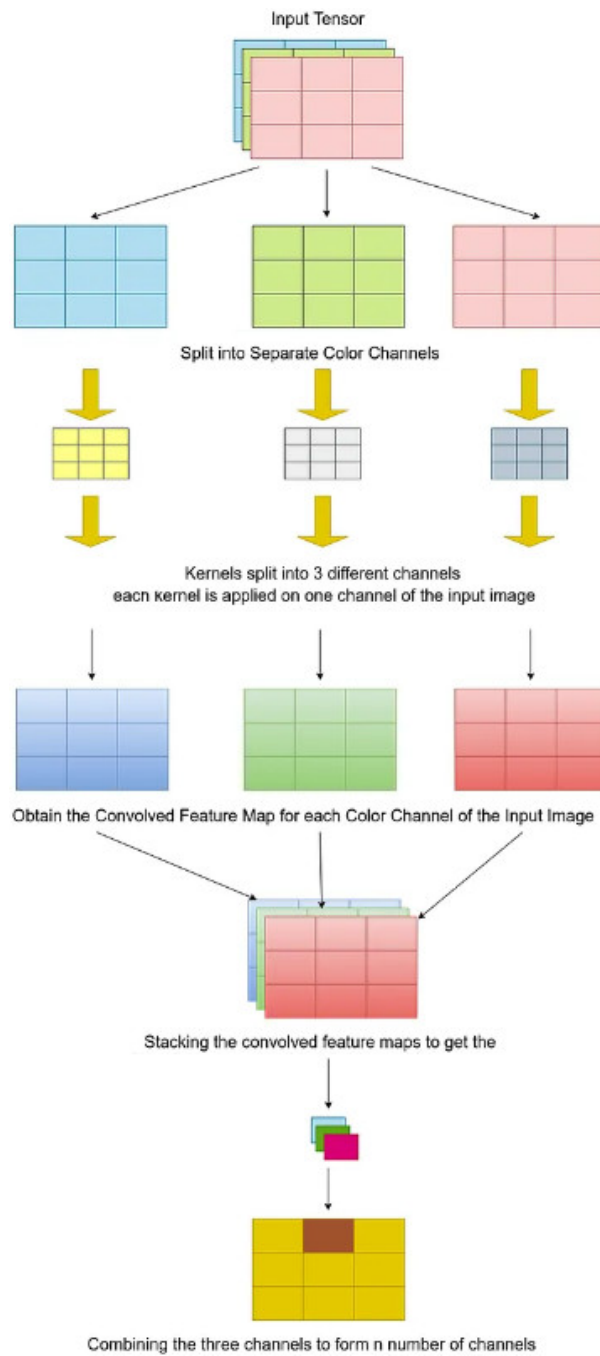


Figure 6.7: Depth-wise convolution workflow. The color scheme is just for representation, depth-wise convolution can be applied to whatever grid-type input having  $N$  channels.

volutions, models can become better at capturing information from a larger area with the same computational budget, making them efficient and powerful at understanding wide-ranging patterns.

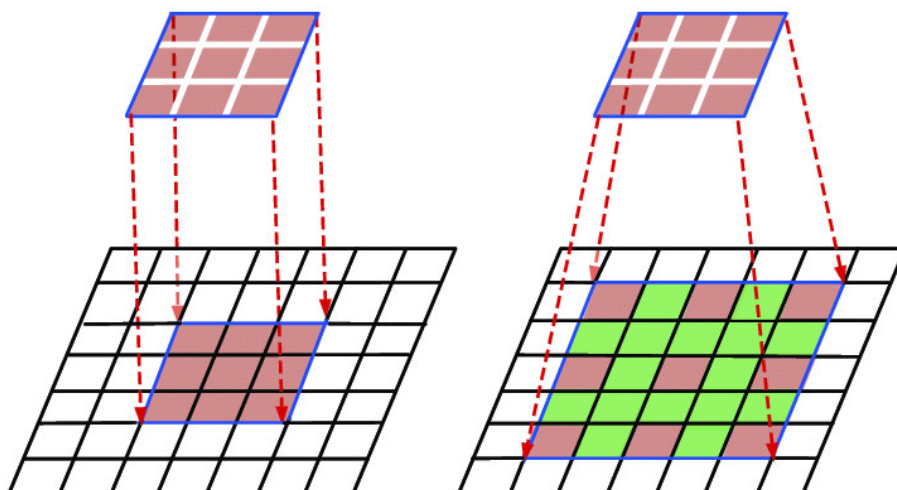


Figure 6.8: Dilated convolution applied to a  $3 \times 3$  kernel, with dilation  $d = 1, 2$ .

## 6.4.2 Pooling

Another stage is usually involved in the layer of a convolutional neural network, a pooling layer 6.9. This serves for reducing the spatial dimension of the input feature maps, decreasing the computational load, memory usage, and also providing an abstracted form of the features, making the network more robust to variations of the input. There are different way of performing the pooling operation withing a CNN, but the most classical and common kind of pooling layers are the **max pooling** and the **average pooling**, which output the maximum value within a rectangular neighborhood, and its average, respectively.

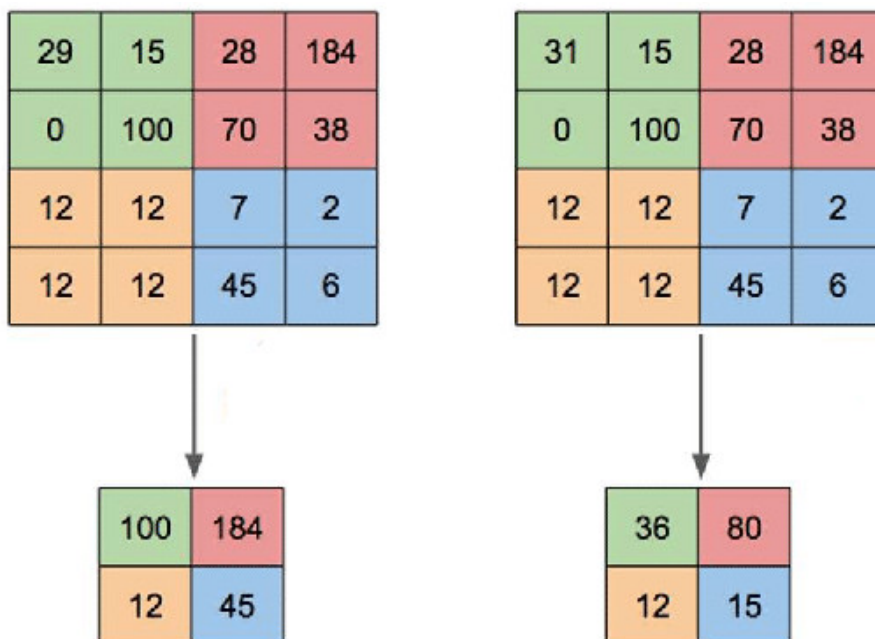


Figure 6.9: Max Pooling and Average Pooling applied with a kernel of size  $2 \times 2$ .

### 6.4.3 Batch normalization

Batch normalization, introduced in 2015 [44], is a pivotal technique in the training of deep neural networks, particularly convolutional neural networks (CNNs). This layer aims to standardize activations, helping the network to learn on a more stable distribution of inputs, significantly improving the training speed and allowing for higher stability and efficiency, especially when dealing with deep learning models. Batch normalization is achieved by adjusting and scaling the output of the activations, so that the input to the next layer will have a mean and variance of 0 and 1, respectively. This way, batch normalization is able to reduce the **internal covariant shift** problem, which refers to the changes in the distribution of network activations due to the update of its weights in previous layers, and that can interfere with the training process, slowing it down and making it more difficult to train at higher learning rate ratio.

## 6.5 Transformers

Although gated Recurrent Neural Networks (RNNs) have long been recognized as the state-of-the-art for processing sequential data, the emergence of **transformers** and their *attention* mechanism has set a new benchmark in this domain.

Like many other architectures, transformers are built on an encoder-decoder framework: the input sequence  $x = (x_0, \dots, x_n)$  is transformed into a continuous representation  $z = (z_0, \dots, z_n)$  by the encoder. Subsequently, the decoder uses  $z$  to generate the output sequence  $y = (y_0, \dots, y_m)$ . Furthermore, the transformer model employs an *auto-regressive* approach, incorporating the previous symbol as additional input to compute the current output, enhancing its predictive capabilities.

### 6.5.1 Architecture

When it was first presented in [45], the encoder-decoder stack that define the transformer architecture was described as follow:

- **Encoder:** composed of  $N$  identical layers stacked together, each one of them exploit two sub-layers: the first is a *multi-head self-attention* mechanism, while the second is a *position-wise fully connected feed-forward network*. Residual connections are used between each of the sub-layers, followed by layer normalization, hence the output of a sub-layer will be

$$\text{LayerNorm}(x + \text{Sublayer}(x)) \tag{6.5}$$

- **Decoder:** Similar to the encoder, it is also composed of  $N$  identical layers. Besides the two sub-layers found in the encoder, it includes *multi-head attention* over the encoder's output. Residual connections are also used here, followed by layer normalization. Additionally, the self-attention sub-layers are modified to prevent attending to subsequent positions.

The attention mechanism used in the Transformer is a function that maps a **query** and a set of **key-value** pairs into an output. The queries, the keys, and the values are all vectors, and their combination is a *weighted sum* of the values, where the weight is computed by a compatibility function between the query and the associated key.

### 6.5.2 The attention mechanism

The attention mechanism described in [45] is called **Scaled Dot-Product Attention**. Here the input consists of query and key vectors of dimension  $d_k$ , and value vectors of dimension  $d_v$ .

After computing the dot product between queries and keys, the resulting vector is divided by  $\sqrt{d_k}$ , and a *softmax* function is applied to obtain the weights associated with each value. Despite this formulation, in practice, each component is packed into a matrix (instead of a vector), to make the computation faster, and the final matrix of outputs is computed as:

$$\text{Attention}(Q, K, V) = \text{Softmax} \left( \frac{QK^T}{\sqrt{d_k}} \right) V \quad (6.6)$$

Then, instead of a single attention function mapping  $d_{model}$ -dimensional queries, keys, and values into an output, Transformers exploit linear projection on these three inputs  $h$  times (i.e., the number of **heads**), to project queries, keys, and values to  $d_q$ ,  $d_k$ , and  $d_v$  dimensions, respectively. Then, the attention function is performed on the projected version of the inputs, and the  $h$  results of these operations are lastly concatenated and once again re-projected to return the final output. This is called **Multi-Head Attention**, and allows the Transformers to jointly retain information from different representation sub-spaces at different positions. Multi-Head Attention is regulated by the following equation

$$\begin{aligned} \text{MultiHead}(Q, K, V) &= \text{Concat}(\text{Head}_1, \dots, \text{Head}_h)W^O, \\ \text{where Head}_i &= \text{Attention}(QW_i^Q, KW_i^K, VW_i^V). \end{aligned} \quad (6.7)$$

### 6.5.3 Foundation Model

A **Foundation Model** (FM) is a type of neural network that is trained on a enormous amount of data and that can then be adapted for a wide range of tasks and operations. FMs exploit the Transformer technology to effectively learn the data, and usually contain billions of parameters.

The type of data on which a FM is trained determines its *mode* (the term *multi-modal learning* indeed refers to the training of a model using data of different nature), for instance a large-language model (LLM) is a FM trained on a vast number of text data, while image-generation models are trained on image data (possibly coupled with text).

Similarly to any other NN, the first step to define a Foundation Model is to design its architecture, by choosing its size and structure among many other parameters. Then, before starting with the actual training, data should be tokenised into a format suitable for training the model. The training of a FM starts with a pre-trained phase in which thousands of gigabytes are used to build the initial knowledge of the model, fine-tuning is then exploited to add specific capabilities or improvements. For fine-tuning a model, two main approaches can used:

- **Alignment**, which is the process of improving the behaviour of the model so that it adhere to the expectations or preferences that the user may have.

- **Domain/Task specific**, which consists of training the model to specialize it to a particular domain or task.

In our framework we used **Depth Anything**, a Foundation Model for monocular depth estimation that, despite being a quite large model, in terms of parameters it contains, it allows for fast inference and high accurate predictions.



# Bibliography

- [1] B. J. Emran and H. Najjaran, “A review of quadrotor: An underactuated mechanical system,” *Annual Reviews in Control*, vol. 46, pp. 165–180, 2018. doi: <https://doi.org/10.1016/j.arcontrol.2018.10.009>.
- [2] K. Ebadi, L. Bernreiter, H. Biggie, *et al.*, “Present and future of slam in extreme underground environments,” *arXiv preprint arXiv:2208.01787*, 2022.
- [3] L. Yang, B. Kang, Z. Huang, X. Xu, J. Feng, and H. Zhao, *Depth anything: Unleashing the power of large-scale unlabeled data*, 2024. arXiv: 2401.10891 [cs.CV].
- [4] Y. Ming, X. Meng, C. Fan, and H. Yu, “Deep learning for monocular depth estimation: A review,” *Neurocomputing*, vol. 438, pp. 14–33, 2021.
- [5] S. Gasperini, N. Morbitzer, H. Jung, N. Navab, and F. Tombari, “Robust monocular depth estimation under challenging conditions,” in *Proceedings of the IEEE/CVF international conference on computer vision*, 2023, pp. 8177–8186.
- [6] P. Geneva, K. Eickenhoff, W. Lee, Y. Yang, and G. Huang, “OpenVINS: A research platform for visual-inertial estimation,” in *Proc. of the IEEE International Conference on Robotics and Automation*, Paris, France, 2020. [Online]. Available: [\url{https://github.com/rpng/open\\_vins}](https://github.com/rpng/open_vins).
- [7] A. Bhoi, “Monocular depth estimation: A survey,” *arXiv preprint arXiv:1901.09402*, 2019.
- [8] X. Yang, Y. Gao, H. Luo, C. Liao, and K.-T. Cheng, “Bayesian denet: Monocular depth prediction and frame-wise fusion with synchronized uncertainty,” *IEEE Transactions on Multimedia*, vol. 21, no. 11, pp. 2701–2713, 2019.
- [9] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” *Advances in neural information processing systems*, vol. 25, 2012.
- [10] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, 2014.

- [11] C. Szegedy, W. Liu, Y. Jia, *et al.*, “Going deeper with convolutions,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.
- [12] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.
- [13] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, “Densely connected convolutional networks,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4700–4708.
- [14] A. G. Howard, M. Zhu, B. Chen, *et al.*, “Mobilenets: Efficient convolutional neural networks for mobile vision applications,” *arXiv preprint arXiv:1704.04861*, 2017.
- [15] X. Zhang, X. Zhou, M. Lin, and J. Sun, “Shufflenet: An extremely efficient convolutional neural network for mobile devices,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 6848–6856.
- [16] K. Han, Y. Wang, Q. Tian, J. Guo, C. Xu, and C. Xu, “Ghostnet: More features from cheap operations,” in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 1580–1589.
- [17] J. Cho, D. Min, Y. Kim, and K. Sohn, “Deep monocular depth estimation leveraging a large-scale outdoor stereo dataset,” *Expert Systems with Applications*, vol. 178, p. 114877, 2021.
- [18] D. Eigen, C. Puhrsch, and R. Fergus, “Depth map prediction from a single image using a multi-scale deep network,” *Advances in neural information processing systems*, vol. 27, 2014.
- [19] D. Eigen and R. Fergus, “Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture,” in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 2650–2658.
- [20] J. M. Facil, B. Ummenhofer, H. Zhou, L. Montesano, T. Brox, and J. Civera, “Cam-convs: Camera-aware multi-scale convolutions for single-view depth,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 11 826–11 835.
- [21] S. Zhao, H. Fu, M. Gong, and D. Tao, “Geometry-aware symmetric domain adaptation for monocular depth estimation,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 9788–9798.

- [22] H. Fu, M. Gong, C. Wang, K. Batmanghelich, and D. Tao, “Deep ordinal regression network for monocular depth estimation,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 2002–2011.
- [23] R. Ranftl, K. Lasinger, D. Hafner, K. Schindler, and V. Koltun, “Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 44, no. 3, pp. 1623–1637, 2020.
- [24] S. F. Bhat, R. Birkl, D. Wofk, P. Wonka, and M. Müller, “Zoedepth: Zero-shot transfer by combining relative and metric depth,” *arXiv preprint arXiv:2302.12288*, 2023.
- [25] S. Farooq Bhat, I. Alhashim, and P. Wonka, “Adabins: Depth estimation using adaptive bins,” Jun. 2021. doi: 10.1109/cvpr46437.2021.00400. [Online]. Available: <http://dx.doi.org/10.1109/CVPR46437.2021.00400>.
- [26] S. F. Bhat, I. Alhashim, and P. Wonka, “Localbins: Improving depth estimation by learning local distributions,” 2022. arXiv: 2203.15132 [cs.CV].
- [27] L. Yang, B. Kang, Z. Huang, X. Xu, J. Feng, and H. Zhao, “Depth anything: Unleashing the power of large-scale unlabeled data,” *arXiv preprint arXiv:2401.10891*, 2024.
- [28] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, “Mobilenetv2: Inverted residuals and linear bottlenecks,” 2019. arXiv: 1801.04381 [cs.CV].
- [29] T. maintainers and contributors, *Torchvision: Pytorch’s computer vision library*, <https://github.com/pytorch/vision>, 2016.
- [30] P. K. Nathan Silberman Derek Hoiem and R. Fergus, “Indoor segmentation and support inference from rgb-d images,” in *ECCV*, 2012.
- [31] N. Simon and A. Majumdar, *Mononav: Mav navigation via monocular depth estimation and reconstruction*, 2023. arXiv: 2311.14100 [cs.RD].
- [32] J. Mao, G. Li, S. Nogar, C. Kroninger, and G. Loianno, “Aggressive visual perching with quadrotors on inclined surfaces,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2021, pp. 5242–5248.
- [33] M. Diehl, H. G. Bock, J. P. Schlöder, R. Findeisen, Z. Nagy, and F. Allgöwer, “Real-time optimization and nonlinear model predictive control of processes governed by differential-algebraic equations,” *Journal of Process Control*, vol. 12, no. 4, pp. 577–585, 2002.
- [34] R. Verschueren, G. Frison, D. Kouzoupis, *et al.*, “Acados – a modular open-source framework for fast embedded optimal control,” *Mathematical Programming Computation*, 2021.

- [35] G. Frison and M. Diehl, “HIIPM: A high-performance quadratic programming framework for model predictive control,” *IFAC*, 2020.
- [36] G. Frison, D. Kouzoupis, T. Sartor, A. Zanelli, and M. Diehl, “BLASFEO: Basic linear algebra subroutines for embedded optimization,” *ACM Transactions on Mathematical Software*, 2018.
- [37] S. Sun, A. Romero, P. Foehn, E. Kaufmann, and D. Scaramuzza, “A comparative study of nonlinear mpc and differential-flatness-based control for quadrotor agile flight,” *IEEE Transactions on Robotics*, pp. 1–17, 2022.
- [38] H. Nguyen, M. Kamel, K. Alexis, and R. Siegwart, “Model predictive control for micro aerial vehicles: A survey,” in *European Control Conference*, 2021, pp. 1556–1563.
- [39] C. Zhang, D. Han, Y. Qiao, *et al.*, *Faster segment anything: Towards lightweight sam for mobile applications*, 2023. arXiv: 2306.14289 [cs.CV].
- [40] A. Kirillov, E. Mintun, N. Ravi, *et al.*, *Segment anything*, 2023. arXiv: 2304.02643 [cs.CV].
- [41] K. He, X. Chen, S. Xie, Y. Li, P. Dollár, and R. Girshick, *Masked autoencoders are scalable vision learners*, 2021. arXiv: 2111.06377 [cs.CV].
- [42] A. Radford, J. W. Kim, C. Hallacy, *et al.*, *Learning transferable visual models from natural language supervision*, 2021. arXiv: 2103.00020 [cs.CV].
- [43] Y. LeCun, B. Boser, J. S. Denker, *et al.*, “Backpropagation applied to handwritten zip code recognition,” *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989. doi: 10.1162/neco.1989.1.4.541.
- [44] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” in *International conference on machine learning*, pmlr, 2015, pp. 448–456.
- [45] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, “Attention is all you need,” *Advances in neural information processing systems*, vol. 30, 2017.