Visual Guidance for Unmanned Aerial Vehicles with Deep Learning

Xingshuai Dong

A thesis submitted in fulfilment
of the requirements for the degree of
Doctor of Philosophy

School of Engineering and Information Technology
University College
University of New South Wales
Australian Defence Force Academy

25 July 2023

Declarations

Thesis Title and Abstract | **Declarations** | Inclusion of Publications Statement | Corrected Thesis and Responses

ORIGINALITY STATEMENT

☑ I hereby declare that this submission is my own work and to the best of my knowledge it contains no materials previously published or written by another person, or substantial proportions of material which have been accepted for the award of any other degree or diploma at UNSW or any other educational institution, except where due acknowledgement is made in the thesis. Any contribution made to the research by others, with whom I have worked at UNSW or elsewhere, is explicitly acknowledged in the thesis. I also declare that the intellectual content of this thesis is the product of my own work, except to the extent that assistance from others in the project's design and conception or in style, presentation and linguistic expression is acknowledged.

COPYRIGHT STATEMENT

☑ I hereby grant the University of New South Wales or its agents a non-exclusive licence to archive and to make available (including to members of the public) my thesis or dissertation in whole or part in the University libraries in all forms of media, now or here after known. I acknowledge that I retain all intellectual property rights which subsist in my thesis or dissertation, such as copyright and patent rights, subject to applicable law. I also retain the right to use all or part of my thesis or dissertation in future works (such as articles or books).

For any substantial portions of copyright material used in this thesis, written permission for use has been obtained, or the copyright material is removed from the final public version of the thesis.

AUTHENTICITY STATEMENT

☑ I certify that the Library deposit digital copy is a direct equivalent of the final officially approved version of my thesis.

Inclusion of Publications Statement

Thesis Title and Abstract | Declarations | **Inclusion of Publications Statement** | Corrected Thesis and Responses

UNSW is supportive of candidates publishing their research results during their candidature as detailed in the UNSW Thesis Examination Procedure.

Publications can be used in the candidate's thesis in lieu of a Chapter provided:

- The candidate contributed **greater than 50%** of the content in the publication and are the "primary author", i.e. they were responsible primarily for the planning, execution and preparation of the work for publication.
- The candidate has obtained approval to include the publication in their thesis in lieu of a Chapter from their Supervisor and Postgraduate Coordinator.
- The publication is not subject to any obligations or contractual agreements with a third party that would constrain its inclusion in the thesis.

☑ The candidate has declared that **some of the work described in their thesis has been published and has been documented in the relevant Chapters with acknowledgement**.

A short statement on where this work appears in the thesis and how this work is acknowledged within chapter/s:

> My literature review is partially comprised of a survey paper that I published in the Journal of "IEEE Transactions on Intelligent Transportation Systems". Acknowledgement of the work of the other authors of this paper has been made at the beginning of Chapter 2.
>
> The results from the Journal paper "MobileXNet: An efficient convolutional neural network for monocular depth estimation" that published in "IEEE Transactions on Intelligent Transportation Systems" are contained in parts in Chapter 4. Acknowledgement of the work of the other authors of this paper has been made at the beginning of Chapter 4.
>
> The results from the Conference paper "Lightweight monocular depth estimation with an edge guided network" which accepted by "2022 17th International Conference on Control, Automation, Robotics and Vision (ICARCV 2022)" are contained in parts in Chapter 5. Acknowledgement of the work of the other authors of this work has been made at the beginning of Chapter 5.

Candidate's Declaration

I declare that I have complied with the Thesis Examination Procedure.

Abstract

Unmanned Aerial Vehicles (UAVs) have been widely applied in the military and civilian domains. In recent years, the operation mode of UAVs is evolving from teleoperation to autonomous flight. In order to fulfill the goal of autonomous flight, a reliable guidance system is essential. Since the combination of Global Positioning System (GPS) and Inertial Navigation System (INS) systems cannot sustain autonomous flight in some situations where GPS can be degraded or unavailable, using computer vision as a primary method for UAV guidance has been widely explored. Moreover, GPS does not provide any information to the robot on the presence of obstacles.

Stereo cameras have complex architecture and need a minimum baseline to generate disparity map. By contrast, monocular cameras are simple and require less hardware resources. Benefiting from state-of-the-art Deep Learning (DL) techniques, especially Convolutional Neural Networks (CNNs), a monocular camera is sufficient to extrapolate mid-level visual representations such as depth maps and optical flow (OF) maps from the environment. Therefore, the objective of this thesis is to develop a real-time visual guidance method for UAVs in cluttered environments using a monocular camera and DL.

The three major tasks performed in this thesis are investigating the development of DL techniques and monocular depth estimation (MDE), developing real-time CNNs for MDE, and developing visual guidance methods on the basis of the developed MDE system. A comprehensive survey is conducted, which covers Structure from Motion (SfM)-based methods, traditional handcrafted feature-based methods, and state-of-the-art DL-based methods. More importantly, it also investigates the application of MDE in robotics. Based on the survey, two CNNs for MDE are developed. In addition to promising accuracy performance, these two CNNs run at high frame rates (126 fps and 90

fps respectively), on a single modest power Graphical Processing Unit (GPU).

As regards the third task, the visual guidance for UAVs is first developed on top of the designed MDE networks. To improve the robustness of UAV guidance, OF maps are integrated into the developed visual guidance method. A cross-attention module is applied to fuse the features learned from the depth maps and OF maps. The fused features are then passed through a deep reinforcement learning (DRL) network to generate the policy for guiding the flight of UAV. Additionally, a simulation framework is developed which integrates AirSim, Unreal Engine and PyTorch. The effectiveness of the developed visual guidance method is validated through extensive experiments in the simulation framework.

Acknowledgments

First and foremost, I would like to express my great gratitude to my supervisor, Prof. Matthew A. Garratt, for giving me the opportunity to pursue my Ph.D. study at UNSW Canberra. I thank him for his guidance and our discussions helped me grow as a researcher and an individual. I also thank him for his suggestions and support both in academic and non-academic matters. I could not have imagined having a better advisor and mentor for my Ph.D study.

I thank my co-supervisors, Dr. Sreenatha G. Anavatti and Prof. Hussein A. Abbass, for their guidance and support throughout my Ph.D. study. My sincere thanks also to my joint supervisor from Ocean University of China, Prof. Junyu Dong. I thank him for his help and providing me the opportunity to join his team as a visiting student. I thank Dr. David Paull for providing the nice photos of using a UAV to monitor the impact of wild horse on creek bank erosion.

I sincerely thank UNSW Canberra for offering me the TFS scholarship and HDR completion scholarship which supported my Ph.D. study.

I thank Dr. Huanneng Qiu for helping me to start my life in Canberra when I first came here. I thank Dr. Min Wang and Dr. Jing Liu for helping me check and reboot my computer during the time of the COVID crisis. I thank my colleagues in the office, Dr. Praveen Kumar Muthusamy, Dr. Junpeng Zhang, Jue Zhang, and James Qin for the wonderful times spent in the office together. I am grateful to all the friends I met here, for all the enjoyable and memorable times. I thank Yuan Rao for the academic discussions, and the happy lunch and dinner times we have shared in Ocean University of China.

Finally, I would like to express my gratitude to my parents, my elder brother, my elder sister and the rest of the family for their consistent support, encouragement and everything that they have been doing for me. Special thanks to my elder brother for his encouragement and support in my life and academics.

List of Publications

Journal Papers

[1] **Xingshuai Dong**, Matthew A Garratt, Sreenatha G Anavatti, and Hussein A Abbass. Towards real-time monocular depth estimation for robotics: A survey. IEEE Transactions on Intelligent Transportation Systems, 23(10):16940–16961, 2022.

[2] **Xingshuai Dong**, Matthew A Garratt, Sreenatha G Anavatti, and Hussein A Abbass. MobileXNet: An efficient convolutional neural network for monocular depth estimation. IEEE Transactions on Intelligent Transportation Systems, 23(11):20134–20147, 2022.

[3] **Xingshuai Dong**, Matthew A Garratt, Sreenatha G Anavatti, and Hussein A Abbass. Frontier guided area coverage for unmanned aerial vehicles with deep reinforcement learning and monocular vision. Under review.

Conference Papers

[1] **Xingshuai Dong**, Matthew A Garratt, Sreenatha G Anavatti, Hussein A Abbass, and Junyu Dong. Lightweight monocular depth estimation with an edge guided network. In 2022 17th International Conference on Control, Automation, Robotics and Vision (ICARCV), pages 204–210. IEEE, 2022.

Contents

List of Figures

List of Tables

List of Abbreviations

1D	One dimensional
2D	Two dimensional
3D	Three dimensional
A3C	Asynchronous Advantage Actor Critic
Abs REL	Absolute Relative Difference
ANN	Artificial Neural Network
API	Application Programming Interface
AV	Autonomous Vehicle
BCE	Binary Cross Entropy
BN	Batch Normalization
BP	Back Propagation
BVLOS	Beyond Visual Line of Sight
CAFF	Channel Attention-based Feature Fusion
CARLA	Car Learning to Act
CC	Competitive Collaboration
cGAN	conditional Generative Adversarial Network
CNN	Convolutional Neural Network
ConvLSTM	Convolutional Long Short-Term Memory
CR	Coverage Rate
CRF	Conditional Random Field
D3QN	Deep Double-Q Network
DDPG	Deep Deterministic Policy Gradient
DDQN	Double Deep Q-Network

DL	Deep Learning
DNN	Deep Neural Network
DoF	Degrees of Freedom
DQN	Deep Q-Network
DRL	Deep Reinforcement Learning
DRQN	Deep Recurrent Q-Network
DTN	Depth-To-Normal
EDDQN	Extended Double Deep Q-Network
EGB	Edge Guidance Branch
FC	Fully Connected
FCN	Fully Convolutional Network
FCRN	Fully Convolutional Residual Network
FCU	Feature Coupling Unit
FLOPs	Floating-Point Operations Per Second
FoV	Field of View
GAN	Generative Adversarial Network
GASDA	Geometry-Aware Symmetric Domain Adaptation Network
GCS	Ground Control Station
GeoNet	Geometric Neural Network
GPS	Global Positioning System
GPU	Graphics Processing Unit
HIL	Hardware-in-the-Loop
ICP	Iterative Closest Point
IMU	Inertial Measurement Unit
IRB	Inverted Residual Block
KLT	Kanade-Lukas-Tomashi
LPG	Local Planar Guidance
LSTM	Long Short-Term Memory
MAV	Micro Aerial Vehicle

MDE	Monocular Depth Estimation
MDP	Markov Decision Process
ML	Machine Learning
MLP	Multilayer Perceptron
MMIDRL	Multi-modal Information-based Deep Reinforcement Learning
MRF	Markov Random Field
MSFF	Multi-Scale Feature Fusion
NLP	Natural Language Processing
NTD	Normal-To-Depth
OF	Optical Flow
PBEP	Projection Batchnorm Expansion Projection
PID	Proportional Integral Derivative
PnP	Plug-and-Play
POMDP	Partially Observable Markov Decision Process
PPO	Proximal Policy Optimization
RANSAC	Random Sample Consensus
RL	Reinforcement Learning
RMSE	Root Mean Square Error
RMSE log	The logarithm Root Mean Square Error
RNN	Recurrent Neural Network
ROS	Robot Operating System
RRT	Rapidly-exploring Random Tree
SAC	Soft Actor-Critic
SARPN	Structure-Aware Residual Pyramid Network
SENet	Squeeze-and-Excitation Network
SfM	Structure from Motion
SfSM	Structure from Small Motion
SID	Spacing Increasing Discretization
SIFT	Scale-invariant Feature Transform

SLAM	Simultaneous Localization and Mapping
SpyNet	Spatical Pyramid Network
Sq REL	Squared Relative Difference
SURF	Speeded Up Robust Features
SVO	Semi-direct Visual Odometry
TD3	Twin-Delayed Deep Deterministic Policy Gradient
TRFA	Transformer-based Feature Aggregation
UAV	Unmanned Aerial Vehicle
UD	Uniform Discretization
UE	Unreal Engine
UGV	Unmanned Ground Vehicle
US	United States
UV	Unmanned Vehicle
V-SLAM	Visual Simultaneous Localization and Mapping
VAE	Variational Autoencoder
VI-SLAM	Visual-Inertial SLAM
ViT	Vision Transformer
VO	Visual Odometry
VR	Virtual Reality

Chapter 1

Introduction

1.1 Aim

The primary aim of this thesis is to investigate and develop a real-time visual perception and guidance method for small unmanned aerial vehicles (UAVs) flying at low altitudes in cluttered outdoor environments. To be specific, the idea is to develop the visual guidance algorithms necessary for a small UAV to cover as much of the accessible areas as possible in an operating environment whilst collecting image data and avoiding obstacles.

Both stereo and monocular vision based methods have been explored in the research community. However, stereo cameras have complex architecture, and need a minimum baseline to generate a disparity map. By contrast, monocular cameras are simple and require less hardware resources. Therefore, monocular vision-based guidance is better suited to small UAVs with limited payload and is the target of this thesis.

Inspired by the success of deep learning (DL) in computer vision, the DL-based method has previously been applied to solve the UAV guidance problem. A

typical example is the work by Loquerico et al. [5], where the authors designed a DL-based network named DroNet to control the flight of a UAV in urban environments. DroNet is a convolutional neural network (CNN) which takes as input monocular RGB images and learns policies including a steering angle and a collision probability. The steering angle is used to keep the UAV flying while avoiding obstacles, and the collision probability is used to modulate the forward speed of the UAV. These policies are trained through imitating a human using a large amount of annotated data that was collected from a real-world environment. It is worth noting that collecting these data requires an expert to demonstrate the desired behaviour. Furthermore, annotating this data requires lots of human resources and is time consuming. Therefore, in this thesis we explore a different method to avoid the abovementioned problems.

Reinforcement learning (RL) is a technique that trains an autonomous agent the optimal actions for achieving its goals. The agent receives information about the current state in the environment and performs actions to change it. At each step, the agent performs an action and gets a positive or a negative reward from the environment. RL works similar to the human decision-making process where decisions are made through interaction with the environment and the policy model is developed through trial and error according to the response of the environment. Compared to DL, RL does not require a human annotated policy as supervisory information. Deep reinforcement learning (DRL) combines artificial neural networks (ANNs) with a framework of RL to map the high dimensional inputs (eg., images from a camera) to the optimal action [6].

Depth estimation aims to estimate the distance of each pixel relative to the camera. A depth map contains information relating to the distance of the surfaces of scene objects from a view point as well as the layout of the scene. As a

kind of mid-level visual representation, depth maps are more generic than raw RGB images, and they have demonstrated faster training and improved policy performance [7]. Optical flow (OF) represents the pattern of apparent motion of image objects between two consecutive frames resulting from the movement of the object or camera. It is a 2D vector field where each vector is a displacement vector representing the movement of points between two consecutive frames [8]. OF relates to both the speed of the observer and distance to objects in the scene and also provides rich information about the environment which can be leveraged for obstacle avoidance. This thesis adopts the combination of those two different modalities of information, depth and OF. In particular, we propose a multi-modal information-based DRL framework for UAV guidance in cluttered environments. Benefiting from the CNNs for monocular depth estimation (MDE) and OF estimation, a monocular camera is sufficient to recover depth and OF. The proposed DRL agent takes as input depth and OF maps, producing high-level commands for guiding the UAV to perform coverage and exploration of an area while collecting imagery and avoiding collisions.

1.2 Background

An unmanned vehicle (UV) refers to a vehicle without a human operator onboard. It can either be remotely controlled or it can be an autonomous vehicle (AV) which is capable of sensing its environment and navigating on its own. The most common types of AVs are UAVs and unmanned ground vehicles (UGVs). Compared to UGVs, UAVs have the advantage of hovering and flying over the working environment which generally makes them less challenged by ground terrain and obstacles. UAVs tend to move faster than UGVs and can provide

observations from altitude that UGVs cannot. Therefore, this thesis is focused on application to UAV platforms. However, many of the methods developed will apply to all kinds of UVs.

UAVs, commonly known as drones, are a kind of aircraft that can fly without an onboard human pilot. UAVs can be classified into two main categories, fixed wing and rotorcraft [9]. The fixed wing UAV has a rigid wing that is designed to work like an aeroplane. This kind of UAV requires forward motion for the wing to generate lift to support its weight. Moreover, it requires a runway to take-off and land whilst not being able to hover, or perform vertical take-off and landings. Rotorcraft UAVs use rotating blades instead of fixed wings to produce the necessary aerodynamic lift force. This provides rotorcraft UAVs with distinct maneuverability advantages over fixed wing UAV, such as hovering, vertical take-off and landing, and low-altitude flights. Micro aerial vehicles (MAVs) are a subset of UAVs that are much smaller in size (< 50 cm) and much lighter in weight (< 2 kg) [10]. MAVs can also be either fixed wing or rotary wing.

The most popular rotorcraft platform is the quadcopter UAV with four rotors. Due to their small size and simplicity, quadcopters are suited for operation in cluttered environments or tasks where static (slow) motion is required. Furthermore, other features of quadcopters, such as relatively low production and technical support costs, contribute to their growing popularity in a range of applications such as forest fire detection and monitoring [11, 12], ecological environment and wildlife monitoring [13,14], search and rescue [15,16], aerial surveillance [17], and critical infrastructure protecting and monitoring [18, 19].

An example of the use of small autonomous drones is the Skeyetech drone developed by Azur Drones for enhanced security and airborne surveillance over sensitive sites [17]. Equipped with the high definition visible camera and high

precision thermal camera, the drone can perform surveillance tasks day and night. However, the Skeyetech drone requires pre-configured flight paths to execute surveillance flights. In scenarios of forest fire detection and monitoring [11, 12], ecological environment and wildlife monitoring [13, 14], search and rescue [15, 16], and critical infrastructure protecting and monitoring [18, 19], the workspace for UAVs has an apparent commonality: a cluttered environment with varying illumination conditions and/or under the tree canopy where global positioning system (GPS) is not always reliable. In these scenarios, it is difficult to define a flight path in advance. Furthermore, the UAV is expected to be able to cover as much of the accessible areas as possible while collecting images and avoiding collision with obstacles within the workspace.

Traditionally, UAVs are controlled by human operators through a ground control station (GCS), where paths and waypoints to be executed by the UAV are planned in advance. Under this circumstance, UAVs can fly Beyond Visual Line of Sight (BVLOS) of the operator. The onboard camera captures images within the environment and sends it to the GCS. Based on the received images, the operator decides and controls the movement of the UAV. However, in environments such as a dense forest or mines, the occurrence of an overhead canopy may influence the wireless communication connection between the UAV and GCS or prevent GPS from working. Without implementation of sophisticated autonomy and sensing, the UAV may quickly become out of control or collide with obstacles such as trees. Therefore, an autonomous guidance method that enables UAVs to explore or sweep the entire investigated area and avoid collision with obstacles is an important enabler for future operations and a focus of the robotics and autonomous systems research community.

Active sensing devices such as laser scanners and LiDAR are bulky, expensive

and energy hungry, which inhibit their deployment on small sized UAVs. By contrast, machine vision, has become relatively low cost, light weight and passive. More importantly, it does not require energy to interrogate the environment, and can gather richer information. Stereo cameras usually have complex architecture, and need a minimum baseline to generate a disparity map, while monocular cameras are simple and require less hardware resources. Hence, monocular vision-based guidance is better suited to UAVs [20].

In recent years, significant progress has been made in the fields of computer vision and robotics. One important factor should be attributed to the application of deep neural networks (DNNs), especially CNNs. These techniques can learn both low level image features and high level abstracted concepts from large-scale datasets. CNN-based features can cope well with different variances, such as illumination, translation and viewpoint. Previous research mainly depends on hand-crafted image features, which are designed beforehand by human experts to model a given set of chosen characteristics. According to the literature review carried out for this thesis, little work has focused on CNN-based methods to solve the monocular sensing problem and guidance of UAVs. Motivated by the above described context, the objective of this thesis is to exploit DL-based perception methods to guide the movement of UAVs in cluttered environments.

1.3 Motivation

This thesis aims to explore novel and efficient visual guidance method whereby the autonomous navigation and perception of the unstructured workspace are performed in real-time. Unlike UGVs that are limited to 2D space, UAVs can hover and fly fast in 3D environments. Images captured by UAVs that are flying a

few metres above the ground have the potential to fill the gap between expensive, weather-dependent and low resolution images provided by satellites.

Recent years have witnessed an increasing use of UAVs in the applications of search and rescue [15,16], infrastructure monitoring [18,19], and wildlife monitoring [13,14]. For example, after hurricane Katrina struck the southeastern United States (US) in 2005, the US government deployed UAVs to survey damage and deliver essential goods to remote locations. It was the first reported application of UAVs for disaster response [21, 22]. On September 5, 2022, a 6.8-magnitude earthquake occurred in Sichuan province, China. Communication infrastructures were damaged by the disaster, so that the trapped people were unable to make an emergency call and lead rescue workers to the location they were trapped. The Chinese government applied UAVs to conduct survey and emergency communication support [23]. The UAVs helped set up an airborne communication network and sent real-time images of the earthquake areas, supporting relief work and ensuring effective rescue operations.

In recent times, circumstances where humans are both the victim and the main transmission route of an infectious virus, have highlighted the potential of UAVs to manage the situation. UAVs are not sensitive to the virus, and can be used to enforce social distancing and monitor the population. During the Covid-19 episode in 2020, UAVs were applied to detect abnormal gatherings of people and to remotely measure temperatures [24]. The mobility and flexible characteristics of these UAVs make the mission of law enforcement agencies simpler, while avoiding close contact with the population.

A UAV can hover in relative close proximity to wild animals or plants to capture images, this makes the UAV a useful tool in the domain of wildlife monitoring. During the summer of 2019-2020, Australia experienced extreme

(a) (b) (c)

Figure 1.1: The application of UAV for ecological environment monitoring. UNSW Canberra Senior Lecturer Dr David Paull is using drones to track the erosion caused by feral horses. (a) Wild horses (copyright free [26]); (b) The drone used by Dr. Paull's team (image was provided by Dr. Paull); and (c) The constructed 3D model (image was provided by Dr. Paull).

bushfires across many areas of the country. After the bushfire, UAVs equipped with infrared and RGB cameras were applied to detect and monitor injured wild animals [25]. Furthermore, UAVs were also applied to monitor creek bank impacts of wild horses in the Australian Alps [26]. Researchers utilized UAVs to capture high resolution images of creek banks. With the collected images, 3D models of creek banks were built to monitor the stream morphology and volumes of soil lost. In scenario shown in Figure 1.1, the majority of the terrain is too dangerous for people to enter on the ground. While UAVs have advantages for such tasks as they can hover and fly over inspected areas, and they are not as influenced by undergrowth or terrain elevation changes as ground robots.

Conventionally, UAVs need to be operated by an experienced pilot or operator. This professional should be aware of what kind of information the inspectors need and adapt to any difficulties such as obstacles, time of day and weather conditions. The recent development towards autonomous inspections using UAVs equipped with image-based sensors is promising for reducing human error and assists in safer, faster and more accurate inspections. Besides, UAVs can conduct repeated inspections in challenging and dynamic environments, making it easier

to monitor infrastructural changes over time. In order to guarantee safety and efficiency, UAVs must demonstrate the ability to sense and avoid obstacles whilst attempting to cover all accessible area in the workspace to complete the task.

In a shared workspace, UAVs may encounter not only static objects such as buildings and trees, but also moving objects like ground vehicles, cyclists and pedestrians. An effective guidance system must guarantee that the UAV can cover as much ground as possible, identify objects within the shared workspace and avoid colliding with them. Autonomous guidance systems require UAVs to sense the environment. The primary method for UAVs to acquire information about the surrounding environment is through a variety of on-board sensors. In recent years, novel sensors such as LiDARs, laser scanners and RGB-D cameras have been applied to robotics. However, these sensors require more payload capacity and higher power consumption than RGB cameras which are light weight, relatively low-cost, and have low power consumption. More importantly, RGB cameras can provide richer information about the environment, which can be processed and applied in real-time. Therefore, vision based guidance methods have attracted great attention from robotics and machine vision researchers.

Conventional visual guidance methods are mainly based on handcrafted features, such as OF [27–29], scale-invariant feature transform (SIFT) [30] and speeded up robust features (SURF) [31]. These features were designed beforehand by human experts to extract a given set of chosen characteristics that overcome specific issues like occlusions and variations in scale and illumination [32]. Therefore, the accuracy of traditional approaches depends on different factors, such as viewing angle, illumination and so on. Furthermore, the design of these hand-crafted features is usually made under some assumption. For example, the Lucas-Kanade algorithm [28] assumes that the displacement of the image

contents between two nearby frames is small and approximately constant within a neighborhood of the point p under consideration. Thus, it may fail when it is applied to environments with illumination changes and/or long-range motions.

By contrast, DL techniques especially CNNs have the ability to create powerful object representations without the need to hand design features. In particular, CNNs can learn both low level image features and high level abstracted concepts as well as abstract input-output relations from large-scale datasets such as ImageNet [33], NYU Depth v2 [3], KITTI [1] and the UnrealDataset [34]. Combined with CNN's translation invariance property, CNNs provide a good substitute for conventional hand-crafted feature extractors. Considering the increasingly challenging environment that UAVs face, this research will focus on the design of novel and efficient CNN-based visual guidance method for UAVs.

1.4 Contributions of Thesis

In this thesis, we focus on the development of novel and real-time vision-based guidance method for UAVs in cluttered environments where only a monocular camera is available to perceive the environment.

The following contributions are made in this thesis:

- **Contribution 1:** A comprehensive survey on MDE, which includes: Structure from Motion (SfM)-based methods, traditional handcrafted feature-based methods, and state-of-the-art DL-based methods. This survey also investigates the application of MDE in robotics (Chapter 2).

- **Contribution 2:** A real-time CNN architecture called MobileXNet for MDE. MobileXNet generates comparable accuracy to the state-of-the-art methods which use either extremely deep and complex architecture or

post-processing but also runs much faster on a single Nvidia GTX 1080 GPU used for benchmarking (Chapter 4).

- **Contribution 3:** A novel lightweight MDE network, named EGD-Net, which employs edge attention features to guide the task of depth estimation. EGD-Net has only 2.21 million parameters and runs at about 96 frames-per-second (fps) on an Nvidia GTX 1080 GPU whilst achieving state-of-

  the-art performance in terms of accuracy (Chapter 5).

- **Contribution 4:** A simulation framework for autonomous UAV monocular vision-based navigation and control which integrates AirSim, Unreal Engine, Python and PyTorch (Chapter 3).

- **Contribution 5:** Application of the proposed MobileXNet and EGD-Net to visual guidance of a UAV and demonstration of its effectiveness through extensive simulation experiments (Chapter 6).

- **Contribution 6:** A frontier guided area coverage algorithm for UAVs that uses a monocular RGB camera and DRL framework. This algorithm enables the UAV to achieve the task of area coverage while avoiding collision with objects in the environment (Chapter 6).

- **Contribution 7:** A multi-modal information-based DRL framework for UAV guidance. Unlike existing methods that use an RGB camera and a laser rangefinder, our designed framework only depends on a single monocular RGB camera. With the developed CNNs for MDE and OF estimation, our method enables the UAV to perceive the environment through depth and OF information (Chapter 6).

1.5 Organization of Thesis

This thesis includes a total of 7 chapters. The review of related literature is shown in Chapter 2. Chapter 3 introduces the simulation framework that utilized in this thesis. Chapter 4 introduces the designed real-time CNN for MDE. Chapter 5 presents a lightweight MDE network. Chapter 6 introduces the design and development of visual guidance system for UAVs. The conclusion of this thesis and a few future directions for the presented work are proposed in Chapter 7.

Chapter 2

Literature Review

The work in this chapter is partially published in the following journal article [35]:

Xingshuai Dong, Matthew A Garratt, Sreenatha G Anavatti, and Hussein A Abbass. Towards real-time monocular depth estimation for robotics: A survey. IEEE Transactions on Intelligent Transportation Systems, 23(10):16940-16961, 2022.

2.1 Introduction

This thesis address the issue of visual guidance of unmanned aerial vehicles (UAVs) with deep learning (DL) based methods. The primary goal is to guide the movement of an UAV in a simulated cluttered outdoor environment using a monocular camera. This chapter gives an overview of related work and is organized as follows. Section 2.2 introduces the development of DL. A comprehensive survey of monocular depth estimation (MDE) is conducted in Section 2.3. Section 2.4 describes literature on optical flow (OF) estimation. Section 2.5 discusses DL-based methods in robotics. Deep reinforcement learning (DRL) and

deep reinforcement learning-based methods in robotics are reviewed in Section 2.6 and 2.7 respectively. Software for running simulations are discussed in Section 2.8.

2.2 The Development of Deep Learning

During the past few years, deep neural networks (DNNs) have set all sorts of records and defeated traditional approaches on many computer vision problems such as image classification [36], depth estimation [2], and OF estimation [37]. The success of these systems comes from their ability to construct powerful object representations without the need to hand design features. As a special branch of DNNs, convolutional neural networks (CNNs) have enjoyed great popularity in recent times. The concept of CNNs came from the late 1970s, and they have been applied to different detection and classification problems proposed in the late 1980s [38] and 1990s [39].

The pioneered CNN architecture, LeNet5, was proposed by LeCun et al. [39] in 1998. It has 7 layers, was first applied to recognise hand-written numbers in 32×32 sized grayscale input images. The AlexNet developed by Krizhevsky et al. [36] was a breakthrough of CNN in computer vision. It achieved the first place in the ILSVRC2012 challenge, wining it with a top-5 accuracy of 84.7% which was by far better that the 73.8% achieved by the second-best contest entry based on traditional methods. AlexNet has a similar architecture as LeNet, but is deeper, and with more filters per layer. It consists of 8 weight layers, the first 5 are convolutional and the following 3 are fully connected.

In 2014, Simonyan and Zisserman [40] increased the network depth by using smaller (3×3) filters. They designed five CNN configurations, and found that

the performance can be improved by pushing the depth to 16-19 weight layers. Due to the problem of vanishing gradients, training becomes more difficult to converge when the network becomes deeper. To tackle this issue, He et al. [41] proposed a residual learning framework. The main idea of residual learning is to create an identity shortcut connection that skips one or more layers. Unlike the previous work learning unreferenced functions, He et al. cast the layers as learning residual functions with reference to the layer input. This technique makes training a CNN with more layers possible. Based on [41], Huang et al. [42] and Hu et al. [43] developed DenseNet and SENet, respectively.

The previous development trend has been to design deeper (stack more layers) and wider (include more filters in each layer) CNNs to increase accuracy. As the architecture of CNNs becomes deeper, they require more memory which sometimes is a big inconvenience when working with graphics processing units (GPUs). These advances neglect the need to make networks more efficient with respect to the size and speed required for some real-time applications such as small sized UAVs. To bridge this gap, Howard et al. [44] designed the first lightweight CNN architecture, MobileNet, for mobile and embedded vision applications. MobileNet is buit on top of depthwise separable convolutions [45], which factorize a standard convolution into a depthwise convolution and a 1×1 pointwise convolution. The application of depthwise separable convolution enables MobileNet to be $32\times$ smaller and $27\times$ less computationally intensive than VGG-16 [40], whilst achieving comparable accuracy on the ImageNet dataset.

Later, Sandler et al. [46] extended [44] through building an inverted residual block (IRB). This block takes as input a low dimensional feature map, which is first widen to high dimension and convolved with a depthwise convolution. The produced feature map is then projected back to low dimension with a linear

convolution and added with the input feature map. In particular, the IRB applies a "narrow → wide → narrow" design with the number of channels, which is the inverted operation of the traditional residual block [41]. ShuffleNet [47] applied channel shuffle operators in the channel dimension of feature maps to make cross-group information flow for group convolution layers. Tan et al. [48] developed an EfficientNet-B0 baseline network by using neural architecture search and scaled it up to get a set of models, named EfficientNets. In [49], Howard et al. introduced the latest version of MobileNet, named MobileNetV3. MobileNetV3 applies the combination of linear bottleneck, inverted residual structure [46] and Squeeze-and-Excite layers [43] as building blocks.

In 2017, Vaswani et al. [50] designed a sequence-to-sequence model, called a Transformer. It relies on attention mechanisms to learn global dependencies between input and output. Transformer models have an encoder and decoder architecture. The encoder has six identical blocks, each block has a multi-head self-attention layer and a position-wise feed-forward layer. To construct a deeper model, residual connections [41] are employed around each block, followed by layer normalization [51]. The decoder also consisting of six blocks, and each block has three layers. The first two layers are similar to the encoder, while an additional cross-attention layer is inserted between the multi-head self-attention layer and the position-wise feed-forward layer. Compared to CNNs, transformers more suit for capturing long-range relationships [50].

Inspired by the success of transformers in natural language processing (NLP), Dosovitskiy et al. [52] proposed a vision-based transformer (ViT) architecture by replacing the CNN backbone with pure transformer which takes the 1D sequence of token embeddings as input. Given 2D images with the size of $224 \times 224 \times 3$, they reshaped it into a sequence of flattened 2D patches with

a fixed size of $16 \times 16 \times 3$. The produced patches are then passed through a sequence of transformer layers to learn global relations and extract features for image classification. Transformer-based models take as input the 1D sequence of patches, this is helpful in capturing the long-range dependencies between patches. However, it ignores the 2D structure and spatial local information within each patch. Besides, it is difficult for ViT models to extract low-resolution and multi-scale feature maps because of the fixed patch size.

To solve the above discussed issues of ViT, Peng et al. [53] designed a dual network architecture that integrates CNN-based local features and transformer-based global features. The dual network architecture combines local convolution blocks, self-attention modules, and Multilayer Perceptron (MLP) units. The produced local and global features are fused with the Feature Coupling Unit (FCU) in an interactive manner. Meanwhile, Guo et al. [54] combined a transformer-based network with convolutional layers to design the CMT (CNNs meet transformers). CMT applies an input block consisting of a 3×3 convolution with a stride of 2, and two 3×3 convolution with stride of 1 to extract local feature maps. The extracted feature maps are passed through several CMT blocks to learn local and global feature representations.

Due to the ability to model long-range dependencies within an image, vision transformers achieve remarkable performance. However, the computational complexity of self-attention based transformers are quadratic to the resolution of input images. Katharopoulos et al. [55] formulated the self-attention as a linear dot-product of kernel feature maps to reduce the computational complexity of the transformer. In addition, the associative property of matrix products are adopted to calculate the self-attention weights. Therefore, the computation complexity is reduced from $\mathcal{O}(N^2)$ to $\mathcal{O}(N)$.

2.3 Monocular Depth Estimation

This section provides a comprehensive survey of monocular depth estimation (MDE), which is based on the author's journal publication [35]. Depth estimation refers to the process of estimating a dense depth map from the corresponding input image(s). Depth information can be utilized to infer the 3D structure, which is an essential part in many robotics and autonomous system tasks, such as ego-motion estimation [56], obstacle avoidance [57] and scene understanding [58]. Active methods depend on RGB-D cameras, LiDAR, Radar or ultrasound devices to directly get the depth information of the scene [59]. However, RGB-D cameras suffer from a limited measurement range. Meanwhile, LiDAR and Radar are limited to sparse coverage, and ultrasound devices are limited by inherently imprecise measurements. In addition, the above devices are large in size and energy-consuming, which is a deficiency when it comes to small sized robots such as MAVs.

On the contrary, RGB cameras are cheaper and light weight. More importantly, they can provide richer information about the environment. Many methods [60–62] depend on stereo matching to estimate depth maps from stereo images. Stereo methods are more accurate, however, collecting stereo images require complex alignment and calibration procedures. Besides, stereo vision-based methods are limited by the baseline distance between the two cameras. To be specific, the estimated depth values tend to be inaccurate when the considered distance are large. With recent advancements in computer vision algorithms, it is more convenient to infer a dense depth map from RGB images.

In this section, we restrict our literature review to MDE for dense depth maps. We extensively review more than 150 relevant articles spanning over 50

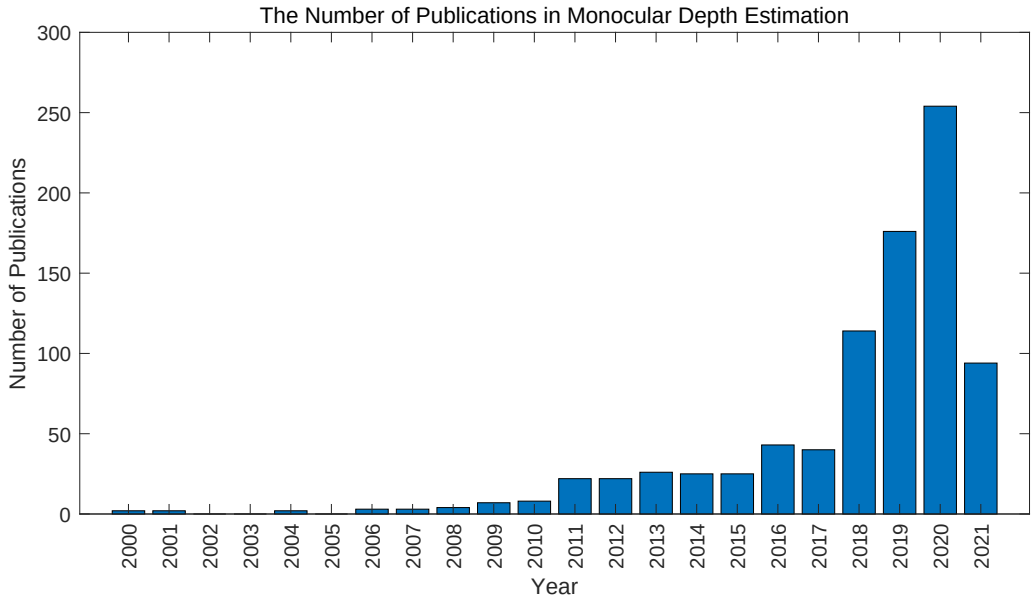


Figure 2.1: The number of published articles on MDE from 2000 to June 2021. (Data from Google Scholar advanced search.)

years (from 1970 to 2021). Our aim is to assist readers to navigate this research field, which has attracted great attention from the computer vision and robotics communities. Figure 2.1 shows the number of published articles on MDE from 2000 to 2021, while Figure 2.2 illustrates the milestones of MDE in recent years.

We classify the reviewed methods into three main categories: Structure from Motion (SfM)-based methods, traditional handcrafted feature-based methods, and state-of-the-art DL-based methods. SfM-based methods [63–66] track a set of corresponding pixels, across a series of images taken in a given scene, and compute depth values at the pixels where features are matched. Therefore, the obtained depth maps are sparse. For traditional handcrafted feature-based methods [67–74], features are first extracted from monocular images, which are then utilized to estimate dense depth maps by optimizing a probabilistic model such as a Markov Random Field (MRF) or a Conditional Random Field (CRF). Over the past few years, the success of DNNs has greatly motivated the development of MDE.

A variety of models [2, 75–81] manifest their effectiveness to recover the depth information from a single image. A possible reason is that the monocular cues can be better modeled with the larger capacity of DNNs.

The remainder of this section is organized as follows: In Section 2.3.1, we introduce the background of MDE. MDE with SfM and traditional handcrafted feature-based methods will be reviewed in Section 2.3.2 and Section 2.3.3 respectively. Section 2.3.4 reviews state-of-the-art DL-based methods. Section 2.3.5 will review other related methods. Section 2.3.6 presents a discussion and comparison on different MDE methods. The applications of MDE will be reviewed in Section 2.3.7. Conclusions are given in Section 2.3.8. We show the overall organization of this section in Figure 2.3.

2.3.1 Background of MDE

Problem Definition

The problem definition of MDE can be viewed as follows. Let I be a single RGB image with size $w \times h$, D is the corresponding depth map with the same size as I. The task of MDE is to formulate a non-linear mapping Ψ: $I \rightarrow D$. Whilst requiring less computational resources and avoiding the baseline issue, MDE is an ill-posed problem as a monocular image may be captured from different distinct 3D scenes. Therefore, MDE algorithms exploit different monocular cues such as texture, occlusion, known object size, lighting, shading, haze and defocus.

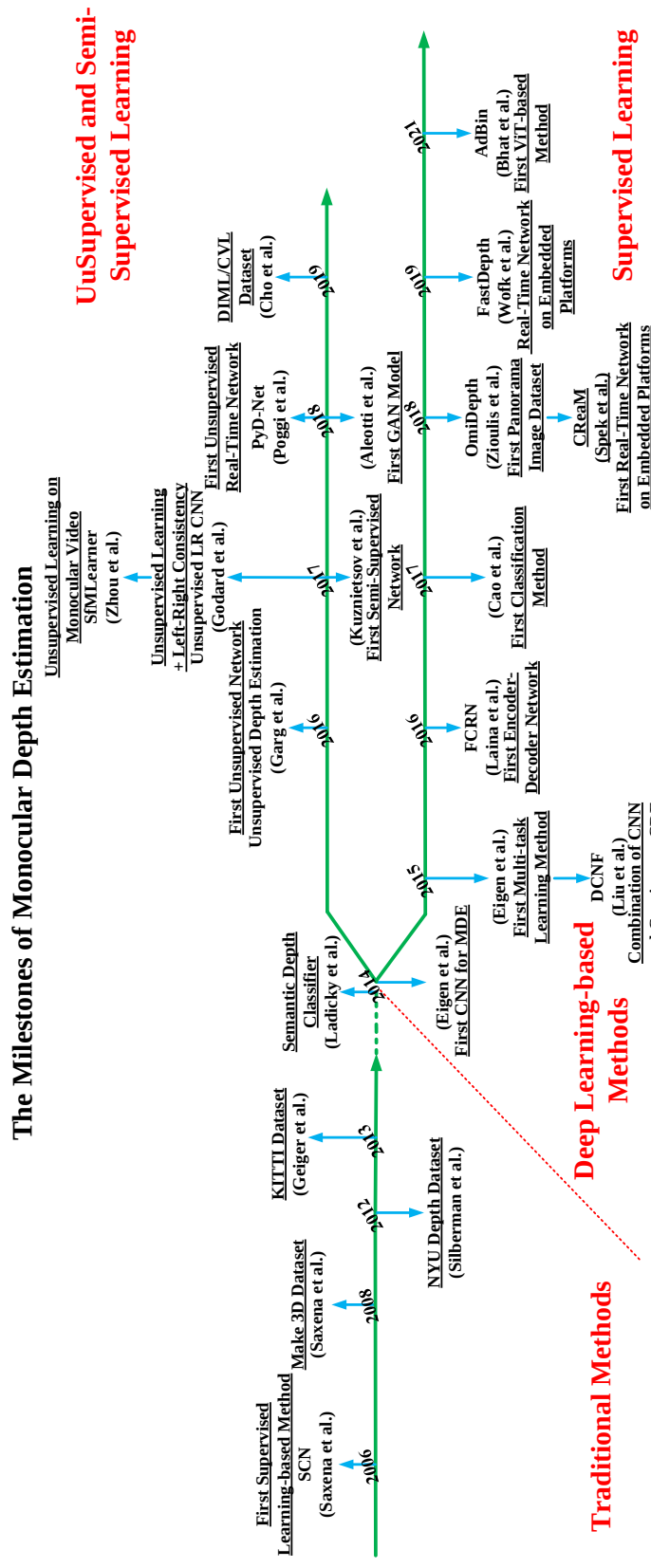


Figure 2.2: Milestones of MDE, including traditional handcrafted feature-based methods [1, 3, 68, 71, 82] and state-of-the-art deep learning-based methods [2, 75, 76, 81, 83–93].

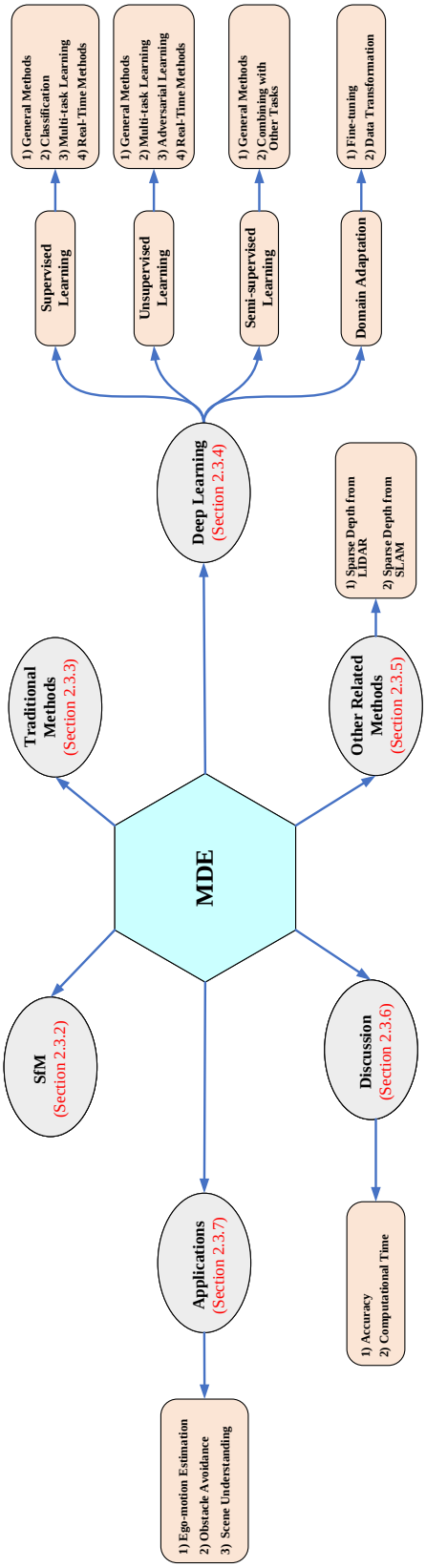


Figure 2.3: An overview of the organization of Section 2.3. ("MDE": Monocular Depth Estimation and "SfM": Structure from Motion.)

MDE Performance Evaluation

Given an estimated depth map D and the corresponding ground-truth D^*, let D_i and D_i^* represent the estimated and ground-truth depth values at the pixel indexed by i, respectively, and N represent the total number of pixels for which there exist both valid ground-truth and estimated depth pixels. As for the quantitative comparison of the estimated depth map and ground-truth, the commonly used evaluation metrics in prior works are listed as follows:

- **Absolute Relative Difference (Abs REL)**: defined as the average value over all the image pixels of the L_1 distance between the ground-truth and the estimated depth, but scaled by the estimated depth:

$$AbsRel = \frac{1}{N}\sum_{N}\frac{|D_i^* - D_i|}{D_i}. \tag{2.1}$$

- **Squared Relative Difference (Sq REL)**: defined as the average value over all the image pixels of the L_2 distance between the ground-truth and the estimated depth, but scaled by the estimated depth:

$$SqRel = \frac{1}{N}\sum_{N}\frac{|D_i^* - D_i|^2}{D_i}. \tag{2.2}$$

- **The linear Root Mean Square Error (RMSE)**: defined as:

$$RMSE = \sqrt{\frac{1}{N}\sum_{N}|D_i^* - D_i|^2}. \tag{2.3}$$

- **The logarithm Root Mean Square Error (RMSE log)**: defined as:

$$RMSE\ log = \sqrt{\frac{1}{N}\sum_{N} |log\ D_i^* - log\ D_i|^2}. \tag{2.4}$$

- **Threshold Accuracy**: is the percentage of predicted pixels where the relative error is within a threshold. The formula is represented as:

$$max(\frac{D_i}{D_i^*}, \frac{D_i^*}{D_i}) < threshold, \tag{2.5}$$

  where the values of threshold usually set to 1.25, 1.25^2, 1.25^3.

In addition, Eigen et al. [2] design a scale-invariant error to measure the relationships between points in the scene, irrespective of the absolute global scale. The scale-invariant mean squared error in log space is defined in Equation (2.6):

$$E(D, D^*) = \frac{1}{2N}\sum_{i=1}^{N}(log\, D_i - log\, D^*{}_i + \alpha(D, D^*))^2, \tag{2.6}$$

where $\alpha(D, D^*) = \frac{1}{N}\sum_i(log\, D^*{}_i - log\, D_i)$ is the value of α which minimizes the difference for a given (D, D^*). For any estimation D, err^α is the scale that best aligns it to the ground-truth, and err is the difference between D and D^* in log space. Since all scalar multiples of D^* have the same error, the scale is invariant.

MDE Datasets

Datasets play a critical role in developing and evaluating depth estimation algorithms. In depth estimation, a number of well-known datasets have been released. Beginning with Make3D [82], representative datasets include NYU depth v2 [3], KITTI [1], Cityscapes [94] and Virtual KITTI [95, 96]. The features of the

datasets for depth estimation are summarized in Table 2.1.

2.3.2 Structure from Motion Based Methods

Structure from motion (SfM) refers to the process of predicting camera motion and/or 3D structure of the environment from a sequence of images taken from different viewpoints [115]. Given a sequence of input images taken from different viewpoints, features such as Harris, SIFT, or SURF are first extracted from all the images. Then the extracted features will be matched. Because some features maybe incorrectly matched, RANSAC (random sample consensus) is typically applied to remove the outliers. These matched features are tracked from image to image to estimate the 3D coordinates of the features. This produces a point cloud which can be transformed to a depth map.

SfM Methodologies

Wedel et al. [63] estimate the scene depth from the scaling of supervised image regions using SfM. Define a point $\boldsymbol{X}(t) = (X(t), Y(t), Z(t))^T$ in 3D space at time t, and its corresponding projected image point $\boldsymbol{x}(t)$. The camera translation in depth between time t and $t + \tau$ is $\boldsymbol{T}(t, \tau)$, while the point at time $t + \tau$ is $\boldsymbol{X}(t + \tau) = \boldsymbol{X}(t) + \boldsymbol{T}(t, \tau)$. The motion of image regions are divided into two parts, with the correctly computed vehicle translation and displacement of image points, the scene depth can be computed through Equation (2.7):

$$d \equiv Z(t) = \frac{s(t, \tau)}{1 - s(t, \tau)} T_Z(t, \tau), \tag{2.7}$$

where $s(t, \tau)$ is the scale, and $T_Z(t, \tau)$ is the camera translation in depth.

Prakash et al. [64] present a SfM-based sparse depth estimation method. The

Table 2.1: A summary of the datasets for depth estimation.

Year	Dataset	Scenario	Sensors	Resolution	Type	Images	Anno.
2008	Make3D [82]	Outdoor	Laser	2272 × 1704	R	534	D
2012	NYU-v2 [3]	Indoor	Kinect v1	640 × 480	R	1449	D
2012	RGB-D SLAM [97]	Indoor	Kinect v1	640 × 480	R	48K	D
2013	KITTI [1]	Driving	LiDAR	1238 × 374	R	44K	S
2015	SUN RGB-D [98]	Indoor	-	-	R	10335	D
2016	DIW [99]	Outdoor	-	-	R	495K	SP
2016	Cityscapes [94]	Driving	Stereo	2048 × 1024	R	5000	Dis.
2016	CoRBS [100]	Indoor	Kinect v2	1920 × 1080, 512 × 424	R	-	D
2016	Virtual KITTI [95]	Outdoor	-	1242 × 375	S	21260	D
2017	2D-3D-S [101]	Indoor	Matterport	1080 × 1080	R	71909	D
2017	ETH3D [102]	Indoor, Outdoor	Laser	940 × 490	R	-	D
2017	Matterport3D [103]	Indoor	Matterport	1280 × 1024	R	194400	D
2017	ScanNet [104]	Indoor	Structure	1296 × 968, 640 × 480	R	2.5M	D
2017	SceneNet RGB-D [105]	Indoor	-	320 × 240	S	5M	D
2017	SUNCG [106]	Indoor	-	640 × 480	S	45000	D
2018	MegaDepth [107]	Indoor, Outdoor	-	-	R	130K	D, O
2018	Unreal [34]	Outdoor	-	256 × 160	S	107K	D
2018	SafeUAV [108]	Outdoor	-	640 × 480	S	8137	D
2018	3D60 [90]	Indoor	-	-	S	35995	D
2018	NUSTMS [109]	Outdoor	Radar	576 × 160, 144 × 40	R	3600	D
2019	DIML/CVL [92]	Indoor, Outdoor	Kinect v2, Stereo	1920 × 1080, 1280 × 720	R	1M	D
2019	DrivingStereo [110]	Driving	LiDAR	1762 × 800	R	182K	S
2019	DIODE [111]	Indoor, Outdoor	Laser	1024 × 768	R	25458	D
2019	Mid-Air [112]	Outdoor	-	1024 × 1024	S	119K	D
2020	Forest Environment [113]	Forest	Depth	640 × 480	R	134K	D
2020	Shanghaitech-Kujiale [114]	Indoor	-	1024 × 512	S	3500	D
2020	Virtual KITTI 2 [96]	Outdoor	-	1242 × 375	S	21260	D

"K": thousand, "M": million, "-": not available, "R": real, "S": Synthetic, "D": dense, "S": sparse, "SP": single pair, and "O": ordinal .

proposed approach takes a sequence of 5 to 8 images captured by a monocular camera to estimate a depth map. With the captured images, features are detected by a multi-scale Fast detector. After matching the detected features from a reference frame and any other frame in the input subset, the two-view geometry is computed between the considered frames. The sparse depth values at the matched feature locations are calculated and reconstructed through a metric transformation.

Ha et al. [65] propose a Structure from Small Motion (SfSM) method which utilizes a plane sweep technique to estimate a depth map. The Harris corner detector is applied to extract features in the reference image and the corresponding features in other images are found by the Kanade-Lukas-Tomashi (KLT) algorithm. Then, the plane sweeping technique is applied to get dense depth maps. Although [65] generates dense depth maps, it takes about 10 minutes to process just one image. To solve the problem of computing efficiency, Javidnia and Corcoran [66] utilize the ORB algorithm as the feature extractor. It reduces the run time to minutes but still cannot run in real-time. In addition, as a corner detector, the ORB algorithm is highly sensitive to the texture present in the scene. Therefore, the estimated depth maps are erroneous in low-textured environments.

Summary of SfM Methods

SfM relies on feature detection and matching to get the correspondence between the detected features and the accuracy of produced depth values depends on the quality of feature matching. The number of detected features relies on the environment, for example, less features are detected in textureless or low-contrast surroundings. Therefore, most of the existing SfM methods produce the sparse

depth maps. These depth maps are adequate for the task of localization, but are not sufficient for applications such as autonomous flight which requires a dense depth map to enable UAVs to avoid frontal obstacles. Although using more features and images produces better estimations, it requires more time to generate a depth map.

2.3.3 Traditional Handcrafted Feature Based Methods

Due to the loss of 3D information in the process of capturing images with a monocular camera, it is not straightforward to infer a depth map from a single-view image. Unlike stereo vision-based methods that can perform stereo matching between the left and right images to estimate depth, earlier MDE algorithms mainly use texture variations, occlusion boundaries, defocus, color/haze, surface layout and size of known objects as cues for predicting depth maps. Although MRF and its variants are a branch of machine learning (ML), they are often combined with handcrafted features to incorporate more contextual information. Therefore, we review methods with MRF in this subsection to distinguish from the DNN-based methods.

The handcrafted feature based-methods roughly work as follows. First, the input images are over-segmented into a set of small regions, called superpixels. Each such superpixel is assumed as a coherent region in the scene that all the pixels have similar properties. Then a number of color, location, texture, motion and geometric context-based features are computed from the obtained superpixels. With the computed features, depth cues will be computed to estimate the depth for each superpixel. Finally, a MRF model is applied to combine superpixel-based depth estimation with information between different superpixels to construct the final depth map.

Traditional Methods Methodologies

According to Google Scholar, the pioneering work in depth estimation from monocular images is [116]. In this work, the intensity or color gradients of a monocular image are exploited to estimate the depth information of objects. The intrinsic images correspond to physical properties of the scene such as depth, reflectance, shadows and surface shape, provide complementary information [117]. Inspired by this point, Kong and Black [118] formulate dense depth estimation as an intrinsic image estimation problem. They combine [73] with a method that extracts consistent albedo and shading from monocular video. A contour detector is trained to predict surface boundaries from albedo, shading and pixel values and the predicted contour is applied to replace image boundaries to enhance the qualities of depth maps.

Torralba and Oliva [67] propose the first learning-based approach, which infers absolute depth from monocular images by incorporating the size of known objects in the image. As the recognition of objects under unconstrained conditions is difficult and unreliable, the absolute scene depth of the images is derived from the global image structure represented as a set of features from Fourier and wavelet transforms. Real-world images contain various objects, while the work in [67] handles different objects with the same method. Hence, it is unsuitable because it disregards the object's own properties. Jung and Ho [69] design an MDE algorithm using a Bayesian learning-based object classification method. With the property of linear perspective, objects in a monocular image are categorized into four types: sky, ground, cubic and plane. According to the type, a relative depth value to each object and 3D model is generated.

Saxena et al. [68] introduce a supervised learning-based method to estimate depth from monocular images. They divide the input image into small patches

and estimate a single depth value for each patch. Two kinds of features, absolute and relative depth features are applied. The former is used to estimate the absolute depth at a particular patch and the latter is for distinguishing the depth magnitude between two patches. Considering the depth of a particular patch relies on the features of the patch and the depths of other parts of the image, a MRF is utilized to model the relation between the depth of a patch and the depths of its neighbouring patches. Raza et al. [74] combine the texture features, geometric context, motion boundary-based monocular cues with co-planarity, connectivity and spatio-temporal consistency constraints to infer depth from monocular videos. Given a monocular video, they first decompose it into spatio-temporal regions. For each region, depth cues that model the relationship of depth to visual appearance, motion and geometric classes are computed and utilized to estimate depth with random forest regression. Subsequently, the estimated depth is refined by incorporating 3D scene properties in MRF with occlusion boundaries.

Besides image features, semantic labels are also used as a cue for inferring depth. The semantic classes of a pixel or region usually have geometry constraints, for example, sky is far away and ground is horizontal. Therefore, depth can be estimated by measuring the difference in appearance with respect to a given semantic class. Liu et al. [70] propose a method that uses semantic information as context to estimate depth from a single image. The proposed method consists of two steps. In the first step, a learned multi-class image labeling MRF is applied to infer the semantic class for each pixel in the image. The obtained semantic information is incorporated in the depth reconstruction model in the second step. Two different MRF models, a pixel-based and a superpixel-based, are designed. Both MRF models define convex objectives that are solved by using the L-BFGS algorithm to compute a depth value for each

pixel in the image.

Ladicky et al. [71] demonstrate how semantic labeling and depth estimation can benefit each other under a unified framework. They propose a pixel-wise classifier by using the property of perspective geometry. Conditioning the semantic label on the depth promotes the learning of a more discriminative classifier. Conditioning depth on semantic classes enables the classifier to overcome some ambiguities of depth estimation. The relationship between different parts of the image is another cue for estimating depth. Liu et al. [72] model MDE as a discrete-continuous optimizing problem. The continuous variables encode the depth of the superpixels in the input image, and the additional discrete variables encode the relationship of two neighboring superpixels. With these variables, the depth estimation can be solved by an inference problem in a discrete-continuous CRF.

Karsch et al. [73] design a non-parametric, data-driven method for estimating depth maps from 2D videos or single images. Given a new image, the designed algorithm first searches similar images from a dataset by applying GIST matching. Subsequently, the label transfer between the given image and the matched image are applied to construct a set of possible depth values for the scene. Finally, the spatio-temporal regularization in an MRF formulation is conducted to make the generated depths spatially smooth.

Summary of Handcrafted Methods

In the abovementioned methods, handcrafted features are extracted from the monocular images to estimate depth maps by optimizing a probabilistic model. These features are designed beforehand by human experts to extract a given set of chosen characteristics, while some corner cases may be missed. Therefore, it

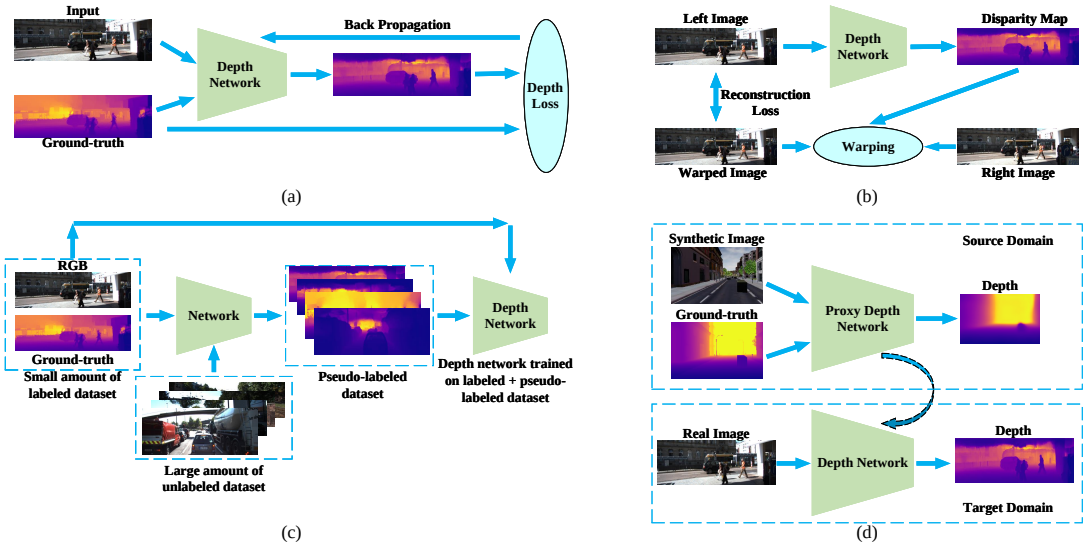


Figure 2.4: The general architecture of DL-based MDE. (a) Supervised learning network, which takes an RGB image and ground-truth depth as input and outputs the estimated depth map, (b) Unsupervised learning network takes as input the stereo image, (c) Semi-supervised learning network, which uses a small amount of image-depth pairs and a large amount of unlabeled images, and (d) Domain adaptation method, where the network in the target domain is trained on synthetic data, the arrow with dotted line represents adaptation. (Best viewed in color).

may result in unsatisfactory performance when applied in new environments. In addition, these methods need pre-processing or post-processing, which imposes a computational burden and makes them unsuitable for the real-time control of robots.

2.3.4 Deep Learning Based Methods

The success of DL in image classification also boosts the development of MDE. In this section, we review DL-based MDE methods. According to the dependency on ground-truth, there are three types of learning approaches: supervised, unsupervised and semi-supervised. These three types of methods are trained on the real data, but we also review methods trained on the synthetic data and then

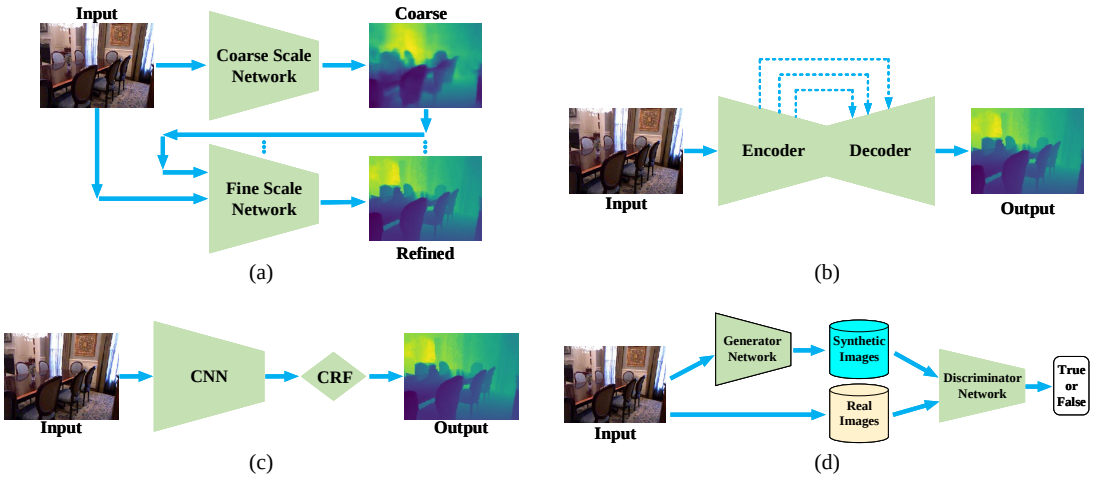


Figure 2.5: Taxonomy of different network architectures. (a) Multi-scale Network [2, 75], (b) Encoder-Decoder Network (dotted lines represent skip connections) [76, 80, 119–121], (c) CNN Combines with CRF [77, 122, 123], and (d) GANs [88, 124]. (Best viewed in color).

transferred to the real data. The implementations and download links of the source code of some algorithms are summarized in Tables 2.2, 2.3 and 2.4.

Depth Estimation with Supervised Learning

The pipeline of supervised learning-based MDE methods can be described as follows (see Figure 2.4(a)). The MDE network incorporates a single image I and the corresponding ground-truth depth map D^* to learn the scene structure information for estimating a dense depth map (D). Then the parameters of the network is updated by minimizing a loss function $L(D^*, D)$, which measures the difference between D and D^*. The network converges when D is as close as possible to D^*.

Table 2.2: A summary of supervised learning-based MDE algorithms with open-source implementations.

Year	Algorithm	Type	Implementation	Source code
2014	Eigen et al. [2]	Supervised	Python	https://cs.nyu.edu/ deigen/depth/
2015	Eigen et al. [75]	Supervised	Python	https://cs.nyu.edu/ deigen/dnl/
2016	Laina et al. [76]	Supervised	MatConvNet, TensorFlow	https://github.com/iro-cp/FCRN-DepthPrediction
2017	Xu et al. [78]	Supervised	Caffe	https://github.com/danxuhk/ContinuousCRF-CNN.git
2018	Alhashim and Wonka [125]	Supervised	PyTorch, TensorFlow	https://github.com/ialhashim/DenseDepth
2018	Fu et al. [126]	Supervised	Caffe	https://github.com/hufu6371/DORN
2018	Guo et al. [127]	Supervised	PyTorch	https://github.com/xy-guo/Learning-Monocular-Depth-by-Stereo
2018	Li and Snavely [107]	Supervised	PyTorch	https://github.com/zhengqili/MegaDepth
2018	Ma and Karaman [119]	Supervised	PyTorch	https://github.com/fangchangma/sparse-to-dense.pytorch
2018	Zioulis et al. [90]	Supervised	PyTorch	https://github.com/VCL3D/SphericalViewSynthesis
2019	Bian et al. [128]	Supervised	PyTorch	https://github.com/JiawangBian/SC-SfMLearner-Release
2019	Chen et al. [121]	Supervised	PyTorch	https://github.com/Xt-Chen/SARPN
2019	Hu et al. [80]	Supervised	PyTorch	https://github.com/JunjH/Revisiting-Single-Depth-Estimation
2019	Lee et al. [129]	Supervised	PyTorch, TensorFlow	https://github.com/cogaplex-bts/bts
2019	Liebel and Korner [130]	Supervised	PyTorch	https://github.com/lukasliebel/MultiDepth
2019	Nekrasov et al. [131]	Supervised	PyTorch	https://github.com/DrSleep/multi-task-refinenet
2019	Qiu et al. [132]	Supervised	PyTorch	https://github.com/JiaxiongQ/DeepLiDAR
2019	Wofk et al. [81]	Supervised	PyTorch	https://github.com/dwofk/fast-depth
2019	Yin et al. [133]	Supervised	PyTorch	https://tinyurl.com/virtualnormal
2020	Fang et al. [134]	Supervised	PyTorch	https://github.com/zenithfang/supervised-dispnet
2020	Sartipi et al. [135]	Supervised	PyTorch	https://github.com/MARSLab-UMN/vi-depth-completion
2020	Xian et al. [136]	Supervised	PyTorch	https://github.com/KexianHust/Structure-Guided-Ranking-Loss
2021	Bhat et al. [93]	Supervised	PyTorch	https://github.com/shariqfarooq123/AdaBins

Table 2.3: A summary of unsupervised learning-based MDE algorithms with open-source implementations.

Year	Algorithm	Type	Implementation	Source code
2016	Garg et al. [84]	Unsupervised	Caffe	https://github.com/Ravi-Garg/Unsupervised-Depth-Estimation
2017	Godard et al. [85]	Unsupervised	TensorFlow	https://github.com/mrharicot/monodepth
2017	Zhou et al. [86]	Unsupervised	TensorFlow	https://github.com/tinghuiz/SfMLearner
2018	Pilzer et al. [137]	Unsupervised	TensorFlow	https://github.com/andrea-pilzer/unsup-stereo-depthGAN
2018	Poggi et al. [89]	Unsupervised	TensorFlow	https://github.com/mattpoggi/pydnet
2018	Qi et al. [138]	Unsupervised	TensorFlow	https://github.com/xjqi/GeoNet
2018	Zhan et al. [139]	Unsupervised	Caffe2	https://github.com/Huangying-Zhan/Depth-VO-Feat
2019	Casser et al. [140]	Unsupervised	TensorFlow	https://github.com/tensorflow/models/tree/archive/research/struct2depth
2019	Elkerdawy et al. [141]	Unsupervised	TensorFlow	https://github.com/selkerdawy/joint-pruning-monodepth
2019	Fei et al. [142]	Unsupervised	TensorFlow	https://github.com/feixh/GeoSup
2019	Godard et al. [143]	Unsupervised	PyTorch	https://github.com/nianticlabs/monodepth2
2019	Ranjan [144]	Unsupervised	PyTorch	https://github.com/anuragranj/cc
2019	Tosi et al. [145]	Unsupervised	TensorFlow	https://github.com/fabiotosi92/monoResMatch-Tensorflow
2019	Watson et al. [146]	Unsupervised	PyTorch	https://github.com/nianticlabs/depth-hints
2019	Zioulis et al. [147]	Unsupervised	PyTorch	https://github.com/VCL3D/SphericalViewSynthesis
2020	Guizilini [148]	Unsupervised	PyTorch	https://github.com/TRI-ML/packnet-sfm
2020	Klingner [149]	Unsupervised	PyTorch	https://github.com/ifnspaml/SGDepth
2020	Peng et al. [150]	Unsupervised	TensorFlow	https://github.com/kspeng/lw-eg-monodepth
2020	Shu et al. [151]	Unsupervised	PyTorch	https://github.com/sconlyshootery/FeatDepth
2020	Xue et al. [152]	Unsupervised	PyTorch	https://github.com/TJ-IPLab/DNet

Table 2.4: A summary of semi-supervised learning and domain adaption-based MDE algorithms with open-source implementations.

Year	Algorithm	Type	Implementation	Source code
2017	Kuznietsov [87]	Semi-supervised	TensorFlow	https://github.com/Yevkuzn/semodepth
2018	Ramirez [153]	Semi-supervised	TensorFlow	https://github.com/CVLAB-Unibo/Semantic-Mono-Depth
2019	Amiri [154]	Semi-supervised	TensorFlow	https://github.com/jahaniam/semiDepth
2018	Atapour et al. [155]	Domain adaptation	PyTorch	https://github.com/atapour/monocularDepth-Inference
2018	Guo et al. [127]	Domain adaptation	PyTorch	https://github.com/xy-guo/Learning-Monocular-Depth-by-Stereo
2018	Zheng et al. [156]	Domain adaptation	PyTorch	https://github.com/lyndonzheng/Synthetic2Realistic
2019	Zhao et al. [157]	Domain adaptation	PyTorch	https://github.com/sshan-zhao/GASDA

General Supervised Methods

The general supervised methods treat MDE as a regression problem. To our knowledge, Eigen et al. [2] introduced the first DL-based MDE algorithm. To exploit global and local information, two CNNs are employed in this work (see Figure 2.5(a)). In addition to the common scale-dependent errors, a scale-invariant error is used as the loss function to optimize the training. The real scale of depth information is recovered without any post-processing. This work remarkably improved the accuracy of MDE on the NYU depth v2 [3] and KITTI [1] datasets. Considering the continuous nature of depth values, Liu et al. [83] cast depth estimation as a deep continuous conditional random fields (CRF) learning problem. They design a network which includes three modules: unary part, pairwise part and continuous CRF loss layer. The input images are first segmented into superpixels. Only image patches centered around each super-pixel are passed to the designed network to predict depth values. The work in [2, 83] depends on fully connected (FC) layers to predict depth values. While the FC layer yields a full receptive field, it has a huge number of trainable parameters resulting in [83] needing over a second to estimate a single depth map from a test image.

Laina et al. [76] introduce a fully convolutional residual network (FCRN) for depth estimation. FCRN consists of two parts, encoder and decoder (see Figure 2.5(b)). The encoder is modified from ResNet-50 [41] by removing the FC layers and the last pooling layer. The decoder guides the network into learning its upscaling via a series of upsample and convolutional layers. The removal of FC layers significantly reduces the number of learnable parameters. Their experiments demonstrate that with the increase of network depth, the accuracy apparently increases, because a deeper network leads to a larger receptive field and captures more context information. Inspired by this finding, CNNs with more

than 100 layers e.g., ResNet-101/152 [41], DenseNet-169 [42] or SENet-154 [43] have been applied to MDE.

Based on DenseNet-169 [42], Alhashim and Wonka [125] design a densely connected encoder-decoder architecture. Unlike [76], they use a simple decoder method which consists of a bilinear upsampling and two convolution layers. With the deeper network architecture, elaborated augmentation and training strategies, the designed network generates more accurate results on the NYU depth v2 [3] and KITTI [1] datasets. To effectively guide the mapping from the densely extracted features to the desired depth estimation, Lee et al. [129] design a local planar guidance (LPG) layer and apply it to each decoding stage. The output from the LPG layers has the same size as the desired depth map. Then, the outputs are combined to get the final estimation. Yin et al. [133] construct a geometric constraint in the 3D space for depth estimation through designing a novel loss function. The designed loss function combines geometric with pixel-wise depth supervisions, which enables the depth estimation network produces accurate depth map and high quality 3D point cloud.

Hu et al. [80] combine an encoder-decoder module with a multi-scale feature fusion (MSFF) module and a refinement module. The MSFF module upscales feature maps from different encoder layers to the same size and then concatenate it channel by channel. Features from the MSFF module are combined with features from the decoder and then fed to the refinement module to generate the final prediction. The other contribution of [80] is a hybrid loss function, which measures errors in depth, gradients and surface normals. Inspired by [80], Chen et al. [121] design a Structure-Aware Residual Pyramid Network (SARPN) to exploit scene structures in multiple scales for MDE. SARPN includes three parts, an encoder which extracts multi-scale features, an adaptive dense feature

fusion module for dense feature fusion and a residual pyramid decoder. The residual pyramid decoder estimates depth maps at multiple scales to restore the scene structure in a coarse-to-fine manner.

Fu et. al. [126] discretize the continuous depth into a series of intervals and transfer depth estimation to an ordinal regression problem. As the uncertainty of the estimated depth increases along with the ground-truth depth values, the common uniform discretization (UD) strategy may result in an over-strengthened loss for the large depth values. To solve this problem, a spacing increasing discretization (SID) strategy is designed to discretize the depth values. With the obtained discrete depth values, an ordinal regression loss which involves the ordered information between discrete labels is applied to train the network.

Bhat et al. [93] divide the estimated depth range into bins where the bin widths change per image. This enables the network to learn to adaptively focus on the regions of different depths. The main contribution of [93] is a Mini-ViT module consisting of four transformer layers [50]. Being designed as a variant of ViT [52], Mini-ViT takes as input the multi-channel feature map of the input image to compute global information at a high resolution and outputs bin-widths and range-attention-maps showing the likelihood of each bin. Unlike Fu et al. [126] estimate depth as the bin center of the most likely bin. The final depth of [93] is the linear combination of bin centers weighted by the probabilities. Therefore, this approach generates smoother depth maps.

In recent years, omnidirectional cameras have become more and more popular. Depth estimation from single 360° images [90, 158] also being explored by researchers. Compared with the regular cameras, omnidirectional cameras have a larger field of view (FoV) which enables them to record the entire surroundings. According to Google Scholar, the first work to estimate a depth map from an

omnidirectional image is OmniDepth [90]. The main contribution of [90] is a dataset consisting of 360° RGB-depth pairs. Since acquiring 360° datasets with ground-truth is difficult, the authors resort to re-use recently released 3D datasets to produce diverse 360° view images. Wang et al. [158] propose a two-branch framework which combines equirectangular and cubemap projection to infer depth from monocular 360° images. The two branches take equirectangular image and cubemap as input, respectively. The produced features are combined by a bi-projection fusion block to exploit the shared feature representations.

In addition to CNNs, Recurrent Neural Networks (RNNs) are also applied to MDE. RNNs are a class of neural networks that model the temporal behavior of sequential data through hidden states with cyclic connections. Unlike CNNs that back-propagate gradient through the network, RNNs additionally back-propagate the gradient through time. Therefore, RNNs can learn dependencies across time. As an extension of the regular RNN, the long short-term memory (LSTM) is able to learn long-term dependencies within the input sequence.

Kumar et al. [159] design a convolutional LSTM (ConvLSTM) based encoder and decoder architecture to learn depth from the spatio-temporal dependencies between video frames. The encoder consists of a set of ConvLSTM layers and the decoder includes a sequence of deconvolutional and convolutional layers. Each ConvLSTM layer has N states that correspond to the number of timestamps, thus, the network learns depth maps from N consecutive video frames. Zhang et al. [160] exploit spatial and temporal information for depth estimation by combining ConvLSTM and Generative Adversarial Network (GAN). In addition, a temporal consistency loss is designed to further maintain the temporal consistency among video frames. The designed temporal loss is combined with the

spatial loss to update the model in an end-to-end manner.

A RNN-based multi-view method for learning depth and camera pose from monocular video sequences is introduced by [161]. The ConvLSTM units are interleaved with convolutional layers to exploit multiple previous frames in each estimated depth map. With the model of multi-views, the image reprojection constraint between multi-view images can be incorporated into the loss function. Additionally, a forward-backward flow-consistency constraint is applied to solve the ambiguity of image reprojection by providing additional supervision.

Monocular Depth Estimation by Classification

For different pixels in a single image, the possible depth values may have different distributions. Therefore, depth estimation can be formulated as a pixel-wise classification task by discretizing the continuous depth values into segments [77, 79, 162–164]. Cao et al. [77] design a fully convolutional deep residual network to estimate the depth range. Li et al. [79] propose a hierarchical fusion dilated CNN to learn the mapping between the input RGB image and corresponding depth map. A soft-weighted-sum inference is proposed to transfer the discretized depth scores to continuous depth values.

Following [77, 79, 162], Zou et al. [164] cast depth estimation as a classification problem but take probability distribution into account in the training step. The main contribution of [164] is a novel mean-variance loss which consists of a mean loss and a variance loss. The mean loss is used to penalize the error between the mean of estimated depth distribution and the ground-truth. Meanwhile, the variance loss is complementary to the mean loss and makes the distribution sharper. The mean-variance loss is combined with the softmax loss to supervise the training of the depth estimation network.

In addition, depth estimation can also be solved by combining depth regression and depth interval classification together [130, 165]. Song et al. [165] exploit the shared features from the semantic labels, contextual relations and depth information in a unified network. They use a FCN-based network to encode the input RGB images into high-level semantic feature maps. The generated feature maps are passed to a two step decoder. In the first step, the feature maps are sampled and fed into a "semantic decoder" to up-sample semantic class labels, and a "depth decoder" to estimate the depth map. Subsequently, the semantic map and depth map are refined at the class and pixel levels by a CRF layer. At last, the classification and regression tasks are integrated to model the depth estimation process by a joint-loss layer and produce the final depth map.

Multi-Task Learning Based Methods

Depth estimation and other applications such as semantic segmentation and surface normal estimation are correlated and mutually beneficial. For example, semantic maps and depth maps reveal the layout and object boundaries/shapes [166]. In order to take advantage of the complementary properties of these tasks, multi-task learning in a unified framework has been explored [75, 131, 138, 167–172].

Eigen and Fergus [75] design a unified three-scale network for three different tasks, depth estimation, surface normal estimation and semantic segmentation. The first scale block estimates a coarse global feature from the entire image. Then the feature map is passed to the second and third scale blocks. The second scale block produces mid-level resolution estimations, and the third scale outputs higher resolution estimations at half size of the input image. The designed network can be trained for three different tasks by changing the output layer

and the loss function.

Inspired by the global network of [2], Wang et al. [167] propose a CNN that jointly estimates pixel-wise depth values and semantic labels. To obtain fine-level details, the authors decompose input images into local segments and use the global layouts to guide the estimation of region-level depth and semantic labels. With the global and local estimations, the inference problem is formulated to a two-layer hierarchical CRF to produce the refined depth and semantic map. Jafari et al. [168] design a modular CNN to jointly solve MDE and semantic segmentation problems. The designed network consists of an estimation module and a refine module. The estimation module utilizes [75] and [173] as sub-networks for different tasks respectively. During training, the two sub-networks positively enforce each other and mutually improve each other. The refine module incorporates the output of the estimation module and produces refined estimations.

The abovementioned methods [75,167,168] require training images have pixel level labels with depth and semantic class ground-truth. It is difficult to collect such datasets, especially for outdoor scenarios. Gurram et al. [169] solve this problem by leveraging depth and semantic information from two heterogeneous datasets to train a depth estimation CNN. The training process is divided into two steps. In the first step, a multi-task learning scheme is applied for pixel-level depth and semantic classification. In the second step, the regression layers take as input the classified depth maps in order to generate the final depth maps.

Qi et al. [138] design a Geometric Neural Network (GeoNet) that jointly learns depth and surface normals from monocular images. Being designed as a two-stream CNN, GeoNet consists of a depth to normal (DTN) network and a normal to depth (NTD) network. The DTN network infers surface normal

from depth map via the least square solution and a residual module, while the NTD network refines the depth estimation from the estimated surface normal and initial depth map. Hesieh et al. [171] propose a multi-task learning network through adding a depth estimation branch to YOLOv3 [174]. During the process of training, a L_1 distance depth estimation loss ($\sum_i^N |depth_i - depth_i^*|$, where N is the amount of objects in a batch, $depth_i$ and $depth_i^*$ denote the estimated and ground-truth object depth) is added to the object detection loss function to update the parameters of the network.

Abdulwahab et al. [172] introduce a framework for predicting the depth and 3D pose of the main objects shown in the input image. The proposed framework stacks a GAN block and a regression CNN block in series connection. The GAN block is trained with a loss function for feature matching which enables the network to generate a dense depth map from an input image. The regression block incorporates the generated depth map to predict the 3D pose. The supervised multi-task learning methods estimate depth maps with other tasks such as semantic estimation and surface normal estimation can improve the accuracy of the depth map. However, it is difficult to collect datasets with depth labels and other labels.

Real-Time Supervised Monocular Depth Estimation

The aforementioned algorithms are based on complex DNNs which are challenging for real-time requirements. In order to enable MDE network to run at the real-time speed on embedded platforms, Spek et al. [91] build a lightweight depth estimation network on top of the "non-bottleneck-1D" block [175]. The designed network runs about 30fps on the Nvidia-TX2 GPU, but its accuracy is inferior.

Later, Wofk et al. [81] develop a lightweight encoder-decoder network for

MDE. Moreover, a network pruning algorithm is applied to further reduce the amount of parameters. Experimental results on the NYU depth v2 dataset show that the obtained depth estimation network runs at 178 fps on an Nvidia-TX2 GPU, while the RMSE and δ_1 values are 0.604 and 0.771 respectively. Inspired by the densely-connected encoder-decoder architecture [125], Wang et al. [176] design a highly compact network named DepthNet Nano. DepthNet Nano applies densely connected projection batchnorm expansion projection (PBEP) modules to reduce network architecture and computation complexity while maintaining the representative ability.

Supervised learning-based methods require vast amounts of depth images as ground-truth for training, allowing these methods to achieve high accuracy for MDE. However, collecting this ground-truth data from the real world requires depth sensing devices such as LiDAR or RGB-D cameras, which increases the expense. In addition, these sensors require accurate extrinsic and intrinsic calibration, any error in calibration results in an inaccurate ground-truth. Therefore, unsupervised learning-based methods [84–86] which do not require ground-truth are attracting attention.

Depth Estimation with Unsupervised Learning

Unsupervised learning methods take as input stereo images or video sequences with small changes in camera positions between frames as two continuous frames can be treated as stereo images. These methods formulate depth estimation as an image reconstruction problem, where depth maps are an intermediate product that integrates into the image reconstruction loss. The pipeline of unsupervised learning-based methods (see Figure 2.4(b)) can be described as follows: the network incorporates two images (name it I_L and I_R) of the same scene but

with slightly different perspectives. Subsequently, the depth map is estimated for I_L, and the obtained depth map is represented by D_L. With D_L and the camera motion between images, I_R can be warped to an image which is similar to I_L through Equation (2.8):

$$I_R(D_L) \rightarrow \tilde{I}_L, \tag{2.8}$$

where $\tilde{I}_L$ is the warped image. The network can be trained with the reconstruction loss formulated in Equation (2.9):

$$loss = L(\tilde{I}_L, I_L). \tag{2.9}$$

General Unsupervised Methods

According to our literature review, Garg et al. [84] developed the first unsupervised learning method for MDE. In this work, image pairs with known camera motion are fed to the network to learn the non-linear transformation between the source image and depth map. The color constancy error between the input image and the inverse-warped target image is used as the loss to optimize the update of network weights. In addition to camera motion, the correspondence between the left and right images is another cue for unsupervised MDE. Godard et al. [85] train an encoder-decoder network in an unsupervised manner by designing a left-right consistency loss. With the calibrated stereo image pairs and epipolar geometry constraints, the designed method does not need ground-truth depth as supervisory signal. Both [84] and [85] require calibrated stereo pairs for training. Therefore, datasets without stereo images [3,82] cannot be applied to train these methods.

Mahjourian et al. [177] alleviate the dependence on stereo images by exploiting the consistency between depth and ego-motion from continuous frames as supervisory signal for training. Tosi et al. [145] propose an unsupervised framework that infers depth from a single input image by synthesizing features from a different point of view. The designed network includes three parts: multi-scale feature extractor, initial disparity estimator and disparity refinement module. Given an input image, high-level features at different scales are extracted by the multi-scale feature extractor. The extracted features are passed to the initial disparity estimator to predict multi-scale disparity maps that aligned with the input and synthesized right view image. The refinement module refines the initial disparity by performing stereo matching between the real and synthesized feature representations.

Ma et al. [178] extend [119] to an unsupervised approach. The input sparse depth maps and the RGB images are pre-processed by initial convolutions separately. The output features are concatenated into a single tensor, which are passed to the encoder-decoder framework. The network is trained in an unsupervised scheme. Besides, the authors use the Perspective-n-Pose method to estimate pose, which assumes that the input sparse depth is noiseless and susceptible to failure in low image texture situations. Based on [178], Zhang et al. [179] design a framework that jointly learns depth and pose from monocular images. The temporal constraint is applied to measure the reprojection error and provides a training signal to depth and pose CNNs simultaneously. The reprojection error signal works on the noisy sparse depth input, while the ground-truth depth provides scale information and supervises the training of depth estimation.

Fei et al. [142] design an unsupervised network that uses the global orientation and the semantics of the scene as the supervisory signal. Unlike previous work

that computes the surface normals from the depth values first and then impose regularity, they directly regularize the depth values via the scale-invariant constraint. Guizilini et al. [180] utilize semantic information to guide the geometric representation learning of MDE. The designed architecture is built within an unsupervised scheme [148]. It consists of two networks, one responsible for depth estimation whilst the other performs semantic segmentation. During training, only the depth estimation network is optimized, while the weights of the semantic segmentation network are fixed to guide the depth estimation network to learn features via pixel-adaptive convolutions. Recently, Johnston and Carneiro [181] introduce a discrete disparity volume to regularize the training of an unsupervised network. The designed method enables the network to predict sharper depth map and pixel-wise depth uncertainties.

Multi-Task Learning Based Methods

Zhou et al. [86] present a method that jointly learning depth maps and camera motion from monocular videos. The proposed framework stacks a depth network [182] and a pose network. The produced depth maps and relative camera pose are applied to inverse warp the source views to reconstruct the target view. By using view synthesis as the supervisory signal, the entire framework can be trained in an unsupervised manner. Due to the dependence on two frame visual odometry estimation method, this network suffers from the per frame scale ambiguity problem.

Inspired by [86], Prasad and Bhowmich [183] use epipolar constrains to optimize the joint learning of depth and ego-motion. The main idea behind the training is similar to [182]. Instead of using epipolar constrains as labels for training, the authors apply it to weight the pixels to guide the training. Klodt

and Vedaldi [184] modify [86] in the following aspects. Firstly, a structural similar loss is imported to strengthen the brightness constancy loss. Besides, an explicit model of confidence is incorporated to the network by predicting each pixel a distribution over possible brightnesses. Finally, a SfM algorithm [185] is applied to the network to provide supervisory signal for the training of depth estimation network.

Vijayanarasimhan et al. [186] design "SfM-Net," a geometry-aware network capable of estimating depth, camera motion and dynamic object segmentation. The designed network includes two sub-networks, the structure network learns to estimate depth while the motion network predicts camera and object motion. The outputs from both sub-networks are then transformed into optical flow by projecting the point cloud from depth estimation to the image space. Thus, the network can be trained in an unsupervised manner through minimizing the photometric error. Dai et al. [187] propose a self-supervised learning framework for jointly estimating individual object motion and depth from monocular video. Instead of modeling the motion by 2D optical flow or 3D scene flow, the object motion is modeled and predicted in the form of full 6 degrees of freedom (DoF).

Joint learning of depth estimation and pose estimation is usually done under the assumption that a consistent scale of CNN-based MDE and relative pose estimation can be learned across all input samples. This hypothesis degrades the performance in environments where the changes of relative pose across sequences are significantly remarkable. In order to tackle the problem of scale inconsistency, Bian et al. [128] design a geometry consistency loss as shown in Equation (2.10). With the proposed loss function, the depth and ego-motion networks are trained in monocular videos to predict scale-consistent results. Given any two continuous images (I_a, I_b) from an unlabeled video, they first use a depth network to compute

the corresponding depth maps (D_a, D_b), and then compute the relative 6 DoF pose P_{ab} between them using a pose network. With the obtained depth and relative camera pose, the warped D_b^a is computed by transforming D_a to 3D space and projecting it to I_b using P_{ab}. The inconsistency between D_b^a and $D_b^{'}$ are used as geometric consistency loss L_{GC} to supervise the training of the network. L_{GC} is defined in Equation (2.10).

$$L_{GC} = \frac{1}{|V|}\sum_{p \in V} D_{diff}(p), \tag{2.10}$$

where V represents valid points that are successfully projected from I_a to the image plane of I_b, $|V|$ means the number of points in V, and D_{diff} stands for the depth inconsistency map. For each point p in V, D_{diff} is defined in Equation (2.11):

$$D_{diff}(p) = \frac{|D_b^a(p) - D_b^{'}(p)|}{D_b^a(p) + D_b^{'}(p)}. \tag{2.11}$$

In order to mitigate the influence of moving objects and occlusions on network training, Bian et al. design a self-discovered mask M ($M = 1 - D_{diff}$) which assigns low/high weights for inconsistent/consistent pixels.

Zhao et al. [188] present a joint learning method for depth and pose. Unlike [86] and [183] utilize PoseNet [189] to recover relative pose, the work in [188] directly predicts relative pose by solving the fundamental matrix from dense optical flow correspondence and apply a differentiable two-view triangulation module to recover an up-to-scale 3D structure. The depth error is measured after a scale adaptation from the estimated depth to the triangulated structure and the reprojection error between depth and optical flow is computed to further enforce the end-to-end joint training.

Zou et al. [190] present an unsupervised framework to jointly learn depth

and optical flow from monocular video sequences. In addition to the regular photometric and spatial smoothness loss, a cross-task consistency loss is designed to provide additional supervisory signals for both tasks. Yin and Shi [191] jointly learn depth, optical flow and camera pose in a unified network. They use a rigid structure reasoning module to infer scene architecture, and a non-rigid motion refinement module to cope with the effect of dynamic objects. These two modules work in different stages, and the view synthetics are used as a basic supervision for the unsupervised learning paradigm. Furthermore, an adaptive geometric consistency loss is designed to tackle the occlusions and texture ambiguities that is not included in pure view synthesis objectives.

Ranjan et al. [144] learn depth along with camera motion estimation, optic flow estimation and motion segmentation. To achieve the goal of joint learning, they design a Competitive Collaboration (CC) learning method. It consists of two modules, the static scene reconstructor infers the static scene pixels using depth and camera motion, and the moving region reconstructor reasons about pixels in the independently moving regions. Those two modules compete for a resource whilst being regulated by a moderator, the motion segmentation network. The CC method coordinates the training of multiple tasks and achieves performance gains in both tasks.

Adversarial Learning Based Methods

In addition to learning depth from view-synthesis or minimizing photometric reconstruction error, unsupervised MDE [88, 124, 137, 192, 193] has also been solved by generative adversarial networks (GANs). GANs consist of a generator network and a discriminator network (see Figure 2.5(d)). Those two networks are trained by the back-propagation algorithm, thus they can work together to

construct unsupervised learning models. Since there is no ground-truth depth in unsupervised learning, the discriminator distinguishes between the synthesized and the real images.

Aleotti et al. [88] present the first GAN for unsupervised MDE. The generator network is trained to infer a depth map from the input image to generate a warped synthesized image. The discriminator network is trained to distinguish the warped image and the input real image. Since the quality of the estimated depth maps has an effect on the warped synthesized images, the generator is forced to generate more accurate depth maps. Mehta et al. [124] introduce a structural adversarial training method which predicts dense depth maps using stereo-view synthesis. Given a monocular image, the generator network outputs a dense disparity maps. With the produced disparity map, multi-view stereo pairs corresponding to the input image view are generated. The discriminator network distinguishes these reconstructed views from the real views in the training data.

Wang et al. [192] integrate adversarial learning with spatial-temporal geometric constraints for the joint learning of depth and ego-motion. The generator combines depth-pose net with direct visual odometry DVO to produce a synthesized image. The combination of PoseNet and DVO generates a fine-grained pose estimation and provides an effective back-propagation gradient to the depth network. Meanwhile, the discriminator takes the synthesized and original images to distinguish the reconstructed and real images. Almalioglu et al. [193] design an adversarial and recurrent unsupervised learning framework. The designed network consists of a depth generator and a pose regressor. With the produced depth map, 6 DoF camera pose and color values from the source images, the view reconstruction module synthesizes a target image. The discriminator network distinguishes the synthesized target image from the real target image.

Real-Time Unsupervised Monocular Depth Estimation

Although these works achieve promising performance, however, they all have fairly deep and complex architectures. Therefore, real-time speed can only be achieved on high performance GPUs, which inhibits their application in autonomous driving or robotics. To tackle the problem of running speed, Poggi et al. [89] stack a simple encoder and multiple small decoders working in a pyramidal structure. The designed network only has 1.9M parameters and requires 0.12s to produce a depth map on a i7-6700K CPU, which is close to a real-time speed. Liu et al. [194] introduce a lightweight model (named MiniNet) trained on monocular video sequences for unsupervised MDE. The core part of MiniNet is DepthNet, which iteratively utilizes the recurrent module-based encoder to extract multi-scale feature maps. The obtained feature maps are passed to the decoder to generate multi-scale disparity maps. MiniNet achieves real-time speed about 54fps with 640×192 sized images on a single Nvidia 1080Ti GPU.

Unsupervised learning methods formulate MDE as an image reconstruction problem and use geometric constraints as supervisory signal. Those methods take stereo images or monocular image sequences as input to learn geometry constraints between the left and right images or continuous frames. Unsupervised learning methods do not require ground-truth in the training process, which avoids the expense of collecting ground-truth depth maps. However, due to the absence of ground-truth the accuracy rate is inferior to supervised learning methods (see Tables 2.5 and 2.6).

Depth Estimation with Semi-supervised Learning

Unsupervised learning methods eliminate the dependence on ground-truth, which is time-consuming and expensive to obtain. However, their accuracy is limited by

stereo construction. With this motivation, semi-supervised methods [87, 92, 153, 154, 195–197] use a small amount of labeled data and a large amount of unlabeled data to improve the accuracy of depth estimation.

The general semi-supervised MDE network works as follows (see Figure 2.4(c)). First, the model is trained with a small amount of labeled training data until it achieves good performance. Then the trained network is used with unlabeled training data to produce outputs known as pseudo labels which may not be quite accurate. The labels and input images from the labeled training data are linked with the generated pseudo labels and input images in the unlabeled training data. Finally, the model is trained in the same way as the first step.

General Semi-supervised Methods

Kuznietsov et al. [87] design the first semi-supervised learning MDE method by combining supervised and unsupervised loss terms together. The MDE network is trained with the image-sparse depth pairs and unlabeled stereo images. The unsupervised learning-based on direct image alignment between the stereo images is utilized to complement supervised training. Experiments show that the semi-supervised results outperform the supervised and unsupervised results (see Tables 2.5 and 2.6 for numerical indicators). Amiri et al. [154] extend [85] to a semi-supervised network through using sparse ground-truth data as additional labels for supervised learning. In the training stage, LiDAR data is used as the supervisory signal, and rectified stereo images are used for unsupervised training.

Ji et al. [196] introduce a semi-supervised adversarial learning network that is trained on a small number of image-depth pairs and a large number of unlabeled monocular images. The proposed framework consists of a generator network for depth estimation and two discriminator networks to measure the quality

of the estimated depth map. During training, unlabeled images are passed to the generator to output depth maps. The two discriminator networks provide feedback to the generator as a unified loss to enable the generator output depth map that accords with the natural depth value distribution. Meanwhile, Guizilini et al. [197] propose a novel supervised loss which optimizes the re-projected depth in the image space. The designed loss term operates under the same conditions as the photometric loss, by re-projecting depth errors back onto the image space. Hence, the depth labels are incorporated into an appearance-based unsupervised learning method and generates a semi-supervised approach.

Semi-supervised Methods with Other Tasks

Ramirez et al. [153] propose a semi-supervised network for joint learning of semantic segmentation and MDE. The network has a shared encoder, a depth decoder and a semantic decoder. The MDE task is trained with unsupervised image re-projection loss, while semantic segmentation is trained in the supervised manner. During training time, the semantic segmentation branch provides feedback to the encoder which enables a shared feature representation of both tasks. In addition, a cross-domain discontinuity is proposed to improve the accuracy of depth estimation. Yue et al. [198] present a semi-supervised MDE framework which consists of a symmetric depth estimation network and a pose estimation network. The RGB image and its semantic map are passed to each sub-network of the framework to produce an initial depth map and a semantic weight map separately. The two are integrated to generate the final depth map. The pose estimation network outputs a 6 DoF pose for view synthesis. The depth estimation network is trained by minimizing the difference between the synthesized view and target view.

Tian and Li [195] introduce a confidence learning-based semi-supervised algorithm by stacking a depth network and a confidence network. The depth network can be any MDE network, e.g., [2, 76], which takes an RGB image as input and produces a depth map, while the confidence network incorporates an RGB image image and the produced depth map to generate a spatial confidence map. The produced confidence map is then utilized as the supervisory signal to guide the training of depth network on unlabeled data. Inspired by the student-teacher strategy, Cho et al. [92] design a semi-supervised learning framework that stacks a stereo matching network and an MDE network. The stereo matching network [199] which trained with ground-truth is used as teacher to produce depth maps from the stereo image pairs. Then stereo confidence maps are predicted to cope with the estimation error from the stereo matching network. The generated depth maps and stereo confidence maps are used as "pseudo ground-truth" to supervise the training of a shallow MDE network. With this method, the MDE network performs as accurately as the deeper teacher network, and yields better performance than directly learning with ground-truth data.

Semi-supervised learning methods learn depth from a small amount of labeled data and a large amount of unlabeled data. The labeled data can be some auxiliary information, e.g., sparse depth or semantic maps, which enables the estimated depth maps more accuracy than unsupervised learning methods. Semi-supervised learning methods alleviate the dependence on ground-truth to some extent, however, it still requires a large amount of unlabeled data in training.

Monocular Depth Estimation with Domain Adaptation

The above subsections review DNN-based MDE methods trained on data collected in real world. Recent advances in computer graphics and modern high-level

generic graphic platforms such as game engines make it possible to generate a large set of synthetic 3D scenes. With the constructed scenes, researchers can capture a large amount of synthetic images and their corresponding depth maps to train the MDE model. While training MDE models on synthetic data mitigates the cost of collecting real datasets consist of a large set of image-depth pairs, the produced models normally do not generalize well to the real scenes because of the inherent domain gap[1]. To tackle this problem, domain adaptation-based methods first train MDE networks on synthetic data to mitigate the effect of domain gap, making the synthetic data representative of real data (see Figure 2.4(d)) have been proposed.

Domain Adaptation via Fine-tuning

Approaches reviewed in this subsection first train a network on images from a certain domain such as synthetic data, and then fine-tune it on images from the target domain. According to our investigation, DispNet [182] is the first work that apply fine-tuning to overcome domain gap for depth estimation. DispNet is first trained on a large synthetic dataset, and then fine-tuned on a smaller dataset with ground-truth. Guo et al. [127] first apply synthetic data to train a stereo matching network. Subsequently, the stereo matching network is fine-tuned on real data. Finally, the produced disparity maps from the stereo network are used as ground-truth to train the MDE network. Experimental results show that [127] outperforms Eigen et al. Fine [2], [85, 86], and Kuznietsov et al. supervised [87].

The fine-tuning based methods require a certain amount of ground-truth depth from the target domain. However, suitable ground-truth depth is only available for a few benchmark datasets, e.g., KITTI. Furthermore, in practical

[1]Due to the distinctions in the intrinsic nature of different domains, the model trained on data from one domain is often incapable of performing well on data from another domain.

settings collecting RGB images with corresponding ground-truth depth maps requires expensive sensors (e.g., LiDAR) and accurate calibration. Since this procedure is complicated and costly, collecting enough real data to perform fine-tuning in the target domain is seldom feasible [200].

Domain Adaptation via Data Transformation

Methods reviewed in this part transform data in one domain to look similar in style to the data from another domain. Atapour-Abarghouei et al. [155] introduce a GAN-based style transfer approach to adapt the real data to fit into the distribution approximated by the generator in the depth estimation model. In order to infer depth maps, a stereo matching network is applied to compute disparity from pixel-wise matching. Compared with methods which directly learn from synthetic data, [155] generalizes better from synthetic domain to real domain. Zheng et al. [156] develop an end-to-end trainable framework that consists of an image translation network ($G_{S\to R}$, where S means synthetic and R means real) and a MDE network (f_T). The image translation network takes as input the synthetic and real training images. For the real images, $G_{S\to R}$ behaves as an autoencoder and uses a reconstruction loss to apply minimal change to the images. For the synthetic data, $G_{S\to R}$ uses a GAN loss to translates synthetic images into the real domain. The translated images are fed to f_T to estimate depth maps which are compared to the synthetic ground-truth depth maps.

However, [155, 156] does not consider the geometric structure of the natural images from the target domain. Zhao et al. [157] exploit the epipolar geometry between the stereo images and design a geometry-aware symmetric domain adaptation network (GASDA) for MDE. The designed framework consists of a style transfer network and a depth estimation network. Since the style transfer

network considers both real-to-synthetic and synthetic-to-real translations, two depth estimators can be trained on the original synthetic data and the generated realistic data in supervised manners respectively.

The data transformation-based methods achieve domain invariance in terms of visual appearance by mitigating the cross-domain discrepancy in image layout and structure. It suffers a drop in accuracy when dealing with environments that are different in appearance and/or context from the source domain [200]. Moreover, sudden change of the illumination or the saturation in images may influence the quality of the transformed images, which will impair the performance of depth estimation [201].

Domain adaptation enables MDE networks trained on the synthetic data are adapted to real data, which reduces the cost of collecting ground-truth depth in real-world environments. It is a promising technique for addressing the unavailability of large amounts of labeled real data.

2.3.5 Other Related Methods of MDE

In this section, we review methods for constructing a dense depth map on top of a sparse depth map from LiDAR or Simultaneous Localization and Mapping (SLAM). These methods take as input the sparse depth maps and the aligned RGB images to fill-in missing data in the sparse depth maps. The reviewed methods are referred to in the literature as "depth completion."

Sparse Depth Maps from LiDAR

To the best of our knowledge, Liao et al. [202] is the first to perform depth completion. Given a partially observed depth map, the first step is to generate a dense reference depth map via projecting the 2D planar depth values along the gravity

direction. The reference depth map is concatenated with the corresponding image and passed to the network which combines both classification and regression losses for estimating the continuous depth value. Ma and Karaman [119] concatenate a set of sparse depth points from LiDAR with an RGB image in channel dimension to train a depth estimation network. Unlike [202], the sparse depth data is randomly sampled from the ground-truth depth image in order to complement the RGB data.

Since the depth data and RGB intensities represent different information, Jaritz et al. [203] fuse sparse depth data and RGB images in a late fusion method. Specifically, the RGB image and sparse depth are processed by two encoders separately. The generated feature maps are concatenated along the channel axis and then fed to the decoder network to generate a dense depth map. [119,202,203] use depth data to update model weights in the process of training. Wang et al. [204] design a Plug-and-Play (PnP) module to improve the accuracy of existing MDE networks by using sparse depth data in the process of inference. For the general training of the MDE network, the aim is to minimize the error between the estimation $f(I)$ and ground-truth D^*, with respect to the network f parametrized by θ through Equation (2.12):

$$\theta^* = argminL(f(I;\theta), D^*), \tag{2.12}$$

where $L(\cdot,\cdot)$ is the loss function. Both the model parameters θ and the input I can affect the estimated depth $f(I;\theta)$, but only the parameters are updated in the process of training. The designed PnP module utilizes the gradient computed from the sparse depth map to update the intermediate feature representation, which is a function of I.

Chen et al. [205] design a 2D-3D fusion block for the joint learning of 2D and 3D feature representations. The designed block consists of a multi-scale 2D convolution branch and a 3D continuous convolution branch. These two branches extract features from the RGB image and the sparse depth data separately, and the generated feature maps are fused via element-wise summation. With this design, various sized networks can be created by stacking 2D-3D fusion block sequentially. Qiu et al. [132] infer dense depth maps from the sparse depth maps and the RGB images while using surface normals as the intermediate representation. The designed network consists of a color branch and a surface normal branch. These two branches take as input the RGB image and sparse depth respectively. The color branch directly outputs a dense depth map. The surface normal branch first produces a surface normal image which is fused with the sparse input and a confidence mask from the color branch to produce a dense depth map. Depth maps from different branches are then fused by an attention mechanism to compute the final depth.

Sparse Depth Maps from SLAM

Yang et al. [57] utilize sparse depth maps from ORB-SLAM [206] to guide the learning of a dense depth map and a confidence map. The RGB image and sparse depth map are separately processed by a convolutional layer and a max pooling layer. The generated feature maps from the RGB image and the sparse depth map are then concatenated together and processed by another convolutional layer. The fused feature maps are passed to an encoder-decoder network to generate a dense depth map. Sartipi et al. [135] use RGB images, learned surface normals and sparse depth from visual-inertial SLAM (VI-SLAM) to infer dense depth maps. Since the depth map from VI-SLAM is more sparse, a sparse-depth enrichment

step is performed to increase its density. The enriched sparse depth maps along with the RGB images and surface normals are passed to the depth completion network to produce dense depth maps.

Depth completion methods integrate the RGB images and sparse depth information to generate dense depth maps. The RGB images provide color, texture, contextual and scene structure information, while the sparse depth maps provide a rough geometric structure of the scene. Since the two input data are complementary, depth completion methods produce higher accuracy rate than MDE methods [119, 202].

2.3.6 Discussion and Comparison

In order to evaluate and compare the MDE methods, we summarize the quantitative results of 42 representative methods on the KITTI dataset [1]. The performance comparison of the summarized methods is listed in Tables 2.5 and 2.6, including error metric (Abs Rel, Sq Rel, RMSE and RMSE log, *lower is better*), accuracy metrics (δ_1, δ_2 and δ_3, *higher is better*) and running time (t_{GPU}). The results listed in Tables 2.5 and 2.6 are from their respective papers.

Accuracy

According to Tables 2.5 and 2.6, all DL-based methods show much better results than the traditional Make3D method [82]. Thus, Make3D is not applicable in any recent application. We observe that the overall development trend of MDE is to push the increase of accuracy. Among the four categories of methods, the supervised learning method generates the best error and accuracy metric results, followed by the semi-supervised, domain adaptation and unsupervised methods. It demonstrates that supervised learning method can learn more representative

features from the ground-truth depth. Note that Bhat et al. [93] generate the best performance among supervised learning methods, suggesting that explicitly utilizing global information at a high resolution decisively improves the performance of MDE.

Regarding the domain adaptation methods [127, 155, 157], Guo et al. [127] yields the best performance. Unlike [155] and [157], [127] first pre-trained with synthetic data and then fine-tuned on real data. It demonstrates that when the fine-tuning dataset is similar to the test dataset, the fine-tuning method performs better than the data transformation method. The best domain adaptation method [127] has superior performance to the best unsupervised method [180]. Regarding the best semi-supervised method [180] and the best domain adaptation method [127], [180] outperforms [127]. In particular, their accuracy metrics are almost equal, while the error metrics especially Sq Rel, RMSE and RMSE log are highly variable. This suggests that even small amounts of labeled data can make a great contribution to the performance of depth networks.

Computational Time

In Tables 2.5 and 2.6, we do not show the running time of all summarized methods because many publications do not report it. Since some authors did not provide enough information to replicate their results, it is impractical to test the running time on our computer. However, as the number of network parameters affects memory footprint and running time required to infer depth, we use this information as an additional information to compare the running time. For example, Poggi et al. [89] has 1.9M parameters and requires 20ms to infer a depth map on a popular Nvidia Titan-X GPU.

Table 2.5: Comparison of many supervised learning-based MDE methods on the KITTI dataset [1] using the data split in [2].

Year	Method	Type	Abs Rel	Sq Rel	RMSE	RMSE log	δ_1	δ_2	δ_3	t_{GPU}	Device
2008	Saxena et al. [82]	Traditional	0.412	5.712	9.635	0.444	0.556	0.752	0.870	-	-
2014	Eigen et al. [2]	Supervised	0.190	1.515	7.156	0.270	0.692	0.899	0.967	13	NVidia Titan Black
2017	Cao et al. [77]	Supervised	0.115	-	4.712	0.198	0.887	0.963	0.982	-	-
2017	Kuznietsov et al. [87]	Supervised	0.122	0.763	4.815	0.194	0.845	0.957	0.987	48	Nvidia GTX 980Ti
2018	Alhashim and Wonka [125]	Supervised	0.093	0.589	4.170	0.171	0.886	0.965	0.986	333.3	Jetson AGX Xavier
2018	Fu et al. [126]	Supervised	0.072	0.307	2.727	0.120	0.932	0.984	0.994	500	-
2018	Guo et al. [127]	Supervised	0.105	0.717	4.422	0.183	0.874	0.959	0.983	-	-
2018	Gurram et al. [169]	Supervised	0.100	0.601	4.298	0.174	0.874	0.966	0.989	-	-
2018	Kumar et al. [159]	Supervised	0.137	1.019	5.187	0.218	0.809	0.928	0.971	-	-
2018	Li et al. [79]	Supervised	0.104	0.697	4.513	0.164	0.868	0.967	0.990	-	-
2019	Lee et al. [129]	Supervised	0.059	0.241	2.756	0.096	0.956	0.993	0.998	-	-
2019	Wang et al. [161]	Supervised	0.088	0.245	**1.949**	0.127	0.915	0.984	0.996	-	-
2019	Yin et al. [133]	Supervised	0.072	-	3.258	0.117	0.938	0.990	0.998	-	-
2020	Patil et al. [207]	Supervised	0.102	0.655	4.148	0.172	0.884	0.966	0.987	**10**	-
2020	Wang et al. [176]	Supervised	0.103	0.511	3.916	-	0.894	0.978	0.994	71.84	Jetson AGX Xavier
2021	Bhat et al. [93]	Supervised	**0.058**	**0.190**	2.360	**0.088**	**0.964**	**0.995**	**0.999**	-	-

Depth range from 0m to 80m. The results of Saxena et al. [82] are reproduced from Eigen et al. [2]; the running time of Fu et al. [126] is reported in Patil et al. [207]; the running time of Alhashim et al. [125] is reported in Wang et al. [176].t_{GPU}: running time (**ms**) tested on for a single forward pass and "-": not available. The best results are shown in red and **bold** values.

Table 2.6: Comparison of many unsupervised learning, semi-supervised learning, and domain adaption-based MDE methods on the KITTI dataset [1] using the data split in [2].

Year	Method	Type	Abs Rel	Sq Rel	RMSE	RMSE log	δ_1	δ_2	δ_3	t_{GPU}	Device
2017	Godard et al. [85]	Unsupervised	0.148	1.344	5.927	0.247	0.862	0.960	0.964	35	Nvidia Titan-X
2017	Kuznietsov et al. [87]	Unsupervised	0.308	9.367	8.700	0.367	0.752	0.904	0.952	48	Nvidia GTX 980Ti
2017	Zhou et al. [86]	Unsupervised	0.208	1.768	6.865	0.283	0.678	0.885	0.957	30	Nvidia Titan-X
2018	Aleotti et al. [88]	Unsupervised	0.118	0.908	4.978	**0.150**	0.855	0.948	0.976	-	-
2018	Mahjourian et al. [177]	Unsupervised	0.163	1.240	6.220	0.250	0.762	0.916	0.968	**10.5**	Nvidia GTX 1080
2018	Pilzer et al. [137]	Unsupervised	0.152	1.388	6.016	0.247	0.789	0.918	0.965	140	Nvidia k80
2018	Poggi et al. [89]	Unsupervised	0.153	1.363	6.030	0.252	0.789	0.918	0.963	20	Nvidia TiTan-X
2018	Qi et al. [138]	Unsupervised	0.155	1.296	5.857	0.233	0.793	0.931	0.973	870	Nvidia TiTan-X
2018	Zou et al. [190]	Unsupervised	0.150	1.124	5.507	0.223	0.806	0.933	0.973	-	-
2019	Almalioglu et al. [193]	Unsupervised	0.150	1.141	5.448	0.216	0.808	0.939	0.975	-	-
2019	Bian et al. [128]	Unsupervised	0.137	1.089	5.439	0.217	0.830	0.942	0.975	-	-
2019	Godard et al. [143]	Unsupervised	0.115	0.882	4.701	0.190	0.879	0.961	0.982	-	-
2019	Ranjan et al. [144]	Unsupervised	0.140	1.070	5.326	0.217	0.826	0.941	0.975	-	-
2019	Tosi et al. [145]	Unsupervised	0.111	0.867	4.714	0.199	0.864	0.954	0.979	160	Nvidia TiTan-Xp
2020	Guizilini et al. [180]	Unsupervised	**0.102**	**0.698**	**4.381**	0.178	**0.896**	**0.964**	**0.984**	-	-
2020	Liu et al. [194]	Unsupervised	0.141	1.080	5.264	0.216	0.825	0.941	0.976	18.57	Nvidia GTX 1080Ti
2020	Zhao et al. [188]	Unsupervised	0.113	0.704	4.581	0.184	0.871	0.961	**0.984**	-	-
2017	Cho et al. [92]	Semi-supervised	0.099	0.748	4.599	0.183	0.880	0.959	0.983	-	-
2017	Kuznietsov et al. [87]	Semi-supervised	0.113	0.741	4.621	0.189	0.862	0.960	0.986	48	Nvidia GTX 980Ti
2019	Amiri et al. [154]	Semi-supervised	0.096	0.552	3.995	0.152	0.892	**0.972**	**0.992**	-	-
2019	Dos et al. [208]	Semi-supervised	0.123	0.641	4.524	0.199	0.881	0.966	0.986	-	-
2020	Guizilini et al. [197]	Semi-supervised	**0.072**	**0.340**	**3.265**	**0.116**	**0.934**	-	-	-	-
2020	Zhao et al. [209]	Semi-supervised	0.143	0.927	4.679	0.246	0.798	0.922	0.968	-	-
2018	Atapouret al. [155]	Domain adaptation	0.110	0.929	4.726	0.194	**0.923**	**0.967**	0.984	**22.7**	Nvidia GTX 1080Ti
2018	Guo et al. [127]	Domain adaptation	**0.096**	**0.641**	**4.095**	**0.168**	0.892	**0.967**	**0.986**	-	-
2019	Zhao et al. [157]	Domain adaptation	0.149	1.003	4.995	0.227	0.824	0.941	0.973	-	-

Depth range from 0m to 80m. t_{GPU}: running time (**ms**) tested on for a single forward pass and "-": not available. The best results of unsupervised learning-based method are shown in blue and **bold** values, the best results of semi-supervised learning-based method are shown in green and **bold** values, and the best results of domain adaptation method are shown in cyan and **bold** values..

According to [93], the proposed network has 78M parameters, which is 40× more than Poggi et al. [89]. Therefore, [93] requires much more time to infer a depth map. This suggests that [93] can only be applied to accuracy-first tasks. The supervised method by Wang et al. [176] runs at about 14fps on an Nvidia Jetson AGX Xavier embedded device, which is close to a real-time speed. Moreover, [176] has less parameters than [89] (1.75M vs 1.9M) and yields much better results. This suggests that [176] is suitable for real-time tasks. Among unsupervised methods, [89, 177, 194] show the three fastest speeds. Although the running time is tested on different GPUs, the order of GPU computation capacity is "Titan-X > 1080Ti > 1080". Regarding the error and accuracy metrics, [194] is inferior to [177] and [89]. In addition, [194] runs faster than [89] on a less powerful GPU. Therefore, Liu et al. [194] can be applied to real-time tasks where large amount of labeled data is not available.

2.3.7 Applications in Robotics

Autonomous vehicles need to detect obstacles, other cars and pedestrians and depth estimation is a fundamental component required to do this in a 3D environment. Depth estimation is a basic component in perceiving the 3D environment. Although autonomous vehicles and robots can perceive depth information through LiDAR, they only produce sparse depth maps. The sparsity of these depth measurements makes it hard to meet the perception requirements needed for safe self-driving car applications. MDE predicts dense depth maps from single images. The resulting dense depth maps have the potential to provide the absolute distances to surfaces of objects in real-time with a single sensor, whilst meeting the requirements of autonomous navigation and obstacle avoidance systems [210].

Autonomous vehicles operate in real-world environments where real-time performance is crucial. Moreover, small autonomous vehicle platforms (e.g., micro aerial vehicles or mini ground vehicles) normally have limited memory and computational resource. The onboard sensor on such platforms may be limited to a monocular RGB camera, and no additional information (e.g., sparse depth point clouds) may be present. Motivated by this fact, lightweight CNNs [44, 46, 175] and MDE networks [81, 89, 91, 194, 211] that run on real-time embedded devices have been developed on top of depthwise separable convolutions, factorized convolutions or network architecture search techniques. It is worth noting that these lightweight implementations [81, 89, 91, 194, 211] achieve a real-time speed on mobile platforms (e.g., Nvidia-TX2 or Jetson AGX Xavier GPU) and produce more accurate depth maps than their traditional counterparts (e.g., [73,82]) which cannot run in real-time. Therefore, it enables the autonomous vehicles or robots to perceive more accurate depth information than [73, 82], whilst not limiting the reactive speed of these vehicles.

Due to the relatively low cost, size and energy consumption, MDE has been applied to the task of ego-motion estimation [91, 212–215], obstacle avoidance [57, 216–220] and scene understanding [58, 221, 222].

Ego-motion Estimation

Systems for calculating ego-motion from vision generally need an absolute range sensor to provide scale to the visual motions. MDE can provide the range measurements needed to provide this information. However, in some cases this can be done using inertial sensing with an appropriate sensor fusion filter. Li et al. [223] use feature depth and onboard Inertial Measurement Unit (IMU) data to compute optical flow to estimate the motion of UAV. Such techniques need the

vehicle to be constantly moving and will fail if there is not motion, and are prone to noise. Wang et al. [224] design an ego-motion estimation method for UAV by fusing data from an RGB-D camera and an IMU. The utilized depth camera suffers from a limited measurement range (0.5m-4m), which can be replaced by a real-time MDE algorithm, such as [81, 89, 91, 194, 211].

DNNs can predict the absolute scale information in the process of MDE, it is helpful in tackling the scale ambiguity and drift problem in monocular ego-motion estimation and improving the mapping process. [212, 214, 215] incorporate CNN-based depth estimations into monocular visual odometry (VO). The obtained VO algorithms show robustness to scale drift and achieve comparable performance to stereo VO methods. Tateno et al. [56] fuse CNN predicted dense depth maps with semi-dense depth measurements from SLAM [225] to solve the scale ambiguity and drift problem of monocular SLAM. To improve the computing speed, dense depth maps only computed from every key-frame. In addition, LOO et al. [226] combine the semi-direct visual odometry (SVO) with a depth estimation network [85]. The depth estimation network provides depth priors in the map points initialization process when a key-frame is selected. With the prior knowledge of the scene geometry, the proposed CNN-SVO is able to obtain a much better prediction of the mean and a smaller initial variance of the depth-filter than the original SVO.

The fusion of depth maps and VO or V-SLAM algorithms improves the performance of ego-motion estimation [56, 226], while the applied depth estimation CNNs [76, 85] require a high-end GPU (e.g., Titan-X) to achieve a real-time speed. This limits the application of [56, 226] in small sized platforms with limited memory and computational resource. Spek et al. [91] integrate the estimated depth maps from a lightweight CNN with ORB-SLAM2 system [185]. The fused

system runs tracking and mapping on mobile platforms at a real-time speed while effectively reducing scale-drift and improving the accuracy of a standard monocular SLAM system.

Obstacle Avoidance

Depth maps contain information about the distance between the surface of objects to the camera [227, 228]. With the estimated depth maps, it is possible for autonomous vehicles or robots to perceive the environment and achieve the goal of avoiding obstacles in stationary scenes [57, 216–219]. Michels et al. [216] use a supervised learning algorithm to learn depth cues that can accurately represent the distance of the nearest obstacles in the scene. The estimated depth information is then converted to steering commands for controlling a ground vehicle in the static outdoor environments. Chakravarty et al. [218] pass depth maps to a behaviour arbitration-based control algorithm to guide a UAV which flies at a particular height in indoor environment to avoid obstacles. Given a depth map, a vertical and horizontal strip through the middle of the depth map is selected. Then the averaged depth values within each vertical and horizontal bin $vert_i$ and $horz_i$ are used as depth values d_i from the angle. These depth values d_i are used to compute an angular velocity for steering the UAV away from obstacles. Zhang et al. [219] use the estimated depth maps to compute the rotation angle to guide a UAV to avoid obstacles and fly towards a destination.

Depth maps can also be used to select collision-free waypoints in order to steer robots in a safe path. Alvarez et al. [217] first compute a dense depth map from a small set of consecutive images. Then the depth map is applied to generate the next obstacle-free waypoints to proceed in a forward direction. To this end, the most distant point in 3D space reachable by the UAV without collisions will be

computed. Yang et al. [57] design an Ego Dynamic Space (EDS)-based obstacle avoidance method by embedding the dynamic motion constrains of the UAV and the confidence values into the spatial depth map. With the estimated depth and confidence maps, the distances D_{eff} to the obstacles can be written as in Equation (2.13):

$$\begin{aligned} D_{eff} &= D - D_{brake} - D_{error} \\ &= D - (vT - \frac{aT^2}{2}) - D_{error}, \end{aligned} \tag{2.13}$$

where D is the perceived distance to the obstacles (i.e., the estimated depth map), D_{brake} is the deceleration distance to stop a UAV which moves at velocity v using deceleration a in a sampling interval T, and D_{error} is the depth measurement error which is computed by $D_{error} = -ln(C)$, C is the estimated confidence. Subsequently, D_{eff} is converted into a binary depth map, which includes obstacle free regions that a UAV can move safely in.

It should be noted that the CNNs used in [57, 218, 219] were trained on data recorded through cameras on moving ground vehicles, which is limited in viewing angles in recorded images. Moreover, the ground vehicles normally move in constrained environments such as roads or corridors. The onboard cameras only encounter a limited subset of motion types, and do not completely explore the 3D environment. This results in the perspective and angle of view of the captured images being different from the UAV view. Therefore, this may raise concerns about the generalization potential of the trained CNNs to the application of obstacle avoidance for UAVs [112].

Depth information offers opportunities and complexities for the determination of the collision-point and time-to-collision when a robot is navigating in a dynamic environment. In particular, depth information could offer the robot a space to navigate in an otherwise more constrained environment. However, error in

depth estimation could have a profound impact on a robots ability to estimate a collision-point with a moving object and subsequently, the time-to-collision with that object. While there exists an extensive literature on collision avoidance research in dynamic 3D environments [229–231], including research on point-clouds from sensors such as LiDAR, literature tackling collision avoidance in dynamic 3D environments with depth estimation is almost non-existent.

Scene Understanding

Autonomous vehicles and robots require a full understanding of the geometric structure of environment to interact with it. The task of scene understanding is to obtain 3D geometric information from 2D image. Depth maps encode the 3D structure of the scene, which helps to resolve ambiguities and to avoid a physical implausible labeling [221]. Scharwachter and Franke [221] propose an approach to infer the coarse layout of street scenes from color, texture and depth information.

When an agent moves through the world, the apparent motion of scene elements is usually inversely proportional to their depth. For example, as the agent moves, faraway mountains do not move much, while nearby trees move a lot. Inspired by this point, Jiang et al. [58] use the MDE network as the base for city scene understanding. They first train a deep network to infer relative scene depth from single images, and then fine-tune it for tasks such as semantic segmentation, joint semantic reasoning of road segmentation and car detection.

Rojas-Perez et al. [222] propose a MDE-based landing zone detection method for UAVs. The detection is divided into two stages: depth estimation from single aerial images and classification of possible landing zones. Based on the extracted patches from RGB images, a multi-layer CNN architecture is designed and trained to infer depth from aerial images. The obtained depth map is then fed to another

CNN to detect possible landing zones for UAVs.

2.3.8 Conclusions and Recommendations regarding MDE

In this section, we presented the first comprehensive survey of MDE. We reviewed the literature from the SfM-based methods, traditional handcrafted feature-based methods to the state-of-the-art DL-based methods. We also summarized the publically available datasets, commonly used performance evaluation metrics, and open-source implementations of some representative methods. In addition, we compared and analyzed the performance of 42 representative methods from different perspectives, including error, accuracy and running time metrics. Since MDE plays an important role in robotics, we provided a review of the application of MDE in ego-motion estimation, obstacle avoidance and scene understanding. Based on the reviewed literature, we concluded that the promising future research directions of MDE may focus on but are not limited to the following aspects:

Collecting Rich Scene Datasets: DL-based models show great performance in MDE. However, training robust models requires a dataset consists of various scenes, in order that the models can learn various scene features. Compared with real-world environments which include complex scenarios such as moving objects, cluttered scenes, occlusions, illumination changes and weather changes, the existing public datasets are not rich enough. Specifically, these datasets focus on certain scenes such as indoor [3], driving [1], campus [82] and forest [113]. Thus, it is essential to collect datasets that encompass richness and high-diversity of environmental scenarios.

Real-Time MDE with Accuracy And Efficiency Balance: The overall development trend of MDE is to push the increase of accuracy using extremely deep CNNs or by designing a complex network architecture, which are computa-

tionally expensive for current mobile computational devices which have limited memory and computational capability. Therefore, it is difficult for these networks to be deployed in small sized robots which depend on mobile computational devices. Under this context, researchers have begun to develop real-time MDE methods [81, 89, 176, 194]. However, the accuracy of these methods is inferior to state-of-the-art methods. Therefore, developing real-time MDE network is assumed to achieve the trade-off between accuracy and efficiency.

MDE with Domain Adaptation: The training of a supervised MDE network requires a large amount of ground-truth data. However, collecting these ground-truth data requires LiDAR or RGB-D cameras, which increase the cost. With computer graphic techniques, it is easier to obtain a large set of synthetic images and its corresponding depth maps. Applying domain adaptation techniques to training MDE models on synthetic and transferring it to real data seems to be a popular direction in MDE.

Semi-supervised MDE: The training of supervised MDE network relies on a large amount of labeled data. The process of collecting depth maps is time-consuming, expensive and inefficient. Unsupervised methods do not need the ground-truth data, but suffer from lower accuracy. Developing semi-supervised methods that are trained on a small number of labeled images and a large number of unlabeled images is of great importance for reducing labor costs and improving prediction accuracy.

Depth Estimation with Information Fusion: Depth estimation with multiple sources/modalities of data, such as sparse depth maps, optical flow and surface normal, show the best performance. Some open questions include: how to apply well-designed depth estimation to different modalities of data, how to efficiently fuse different information to improve the accuracy of depth estimation

and how to use optical flow to handle independently moving objects in a dynamic scene?

Interpretability of MDE Networks: While DL-based MDE has achieved remarkable improvement in accuracy, there remain questions about these networks. For example, what exactly are MDE networks learning? What is a minimal network architecture that can achieve a certain accuracy? Although studies in [232, 233] explored the mechanism of MDE networks, a specific study of the underlying behavior/dynamics of these networks was not available. Thus, research on the interpretability of MDE networks is an important topic.

2.4 Optical Flow Estimation

Optical flow (OF) refers to the apparent motion of objects, brightness patterns or feature points, observed from the eye or the camera. Following this definition, OF can be computed from the pixel-wise motions between consecutive images. Traditional methods such as [27] and [28] have been widely used by robotic communities. The success of CNNs also promoted their application in OF estimation. Early developed CNN-based OF estimation methods normally utilize CNNs as feature extractors. The core idea is to replace the hand-crafted feature extracting with a CNN model.

Weinzaepfel et al. [234] applied CNN techniques to OF estimation in their milestone work, DeepFlow. DeepFlow first extracts features in non-rigid local frames by means of sparse convolutions and max-pooling, then performs dense matching in all image regions. This method is able to efficiently handle large displacements occurring in realistic image sequences, and shows competitive performance on OF benchmarks. Later, Simo-Serra et al. [235] applied CNN

to compute a 128 dimensional descriptor, which was utilized as a substitute for SIFT [30]. It has been shown that this descriptor is efficient and can be generalized well against scaling, rotation, perspective transformation, non-rigid deformation and illumination changes.

Meanwhile, the end-to-end regression-based CNN architectures that can directly estimate OF from a pair of input images also being developed. Unlike methods that combine CNN feature extractors with traditional regularizers, the regression-based methods approximate CNN as a function, which learns the relationship between the input image pairs and the desired OF output given the labeled training dataset. Dosovitskiy et al. [37] developed the first end-to-end CNN for estimating OF based on an encoder-decoder architecture. Since obtaining dense ground-truth OF in real-world images is difficult, Dosovitskiy et al. collected a synthetic dataset from CAD models of chairs, which move in front of a static background. Pairs of RGB images with ground-truth OF map are used to train the network. FlowNet demonstrated that a CNN-based regression architecture is able to predict OF directly.

Later, Ilg et al. [236] presented FlowNet2 that boosts the OF estimation accuracy over the FlowNet method. The main problem behind the FlowNet is blurry flow maps from the decoder network. The authors stacked several FlowNet-style networks in a unified framework to refine the output from the previous network modules step by step. FlowNet2 demonstrated that end-to-end regression networks can outperform traditional methods.

Ranjan and Black designed the spatial pyramid network (SpyNet) [237] which combines the advantages of the traditional "coarse-to-fine" concept and a CNN architecture. SpyNet is composed of 5 pyramid levels. Each pyramid level has a shallow CNN for predicting OF map between a source image and a target image,

which is warped by the current flow prediction. Compared with Dosovitskiy et al. [37], SpyNet reduces the number of network parameters by 96%, while achieving comparable accuracy to [37]. In 2018, Sun et al. [238] introduced the PWC-Net, which depends on pyramidal processing, warping, and the cost volume principles. Compared with FlowNet2 [236], PWC-Net is 17× smaller in model size and 2× faster in inference while achieving the higher accuracy.

2.5 Deep Learning in Robotics

Inspired by the breakthrough achieved by using CNN techniques in the field of computer vision, much interest has been given to apply these techniques to robotics. Considering its relatively low sample complexity (i.e., not much sample data is required to generalize) and its implementation simplicity, supervised learning has become the predominant tool used to learn control, guidance and obstacle avoidance policies from images.

Chakravarty et al. [218] passed depth maps to a behaviour arbitration-based control algorithm to guide the UAV to avoid obstacles. Given a depth map, a vertical and horizontal strip through the middle of the depth map is selected, and the averaged depths within each vertical and horizontal bin $vert_i$ and $horz_i$ are used as depth values d_i from the angle. These depth values d_i are then fed to the avoid behaviour for computing an angular velocity that guide the UAV away from obstacles. It should be noted that [218] adopts the coarse network of [2] to produce depth maps, while the coarse global prediction limited the accuracy of obstacle avoidance. One year later, Jung et al. [239] introduced another CNN-based guidance system for UAV. They design a CNN to estimate the center of the gates used for drone racing competitions. Based on obtained

center points, a guidance system was applied to control a UAV fly through the gates. The designed method outperforms its counterparts with traditional methods in autonomous drone racing.

Inspired by Giusti et al. [240], Smolyanskiy et al. [241] proposed an approach for navigating a UAV through forest trails and avoid obstacles. They combine two different networks, one for trail detection and the other for obstacle detection. The output of these two networks were fed to the control module to generate desired movement commands. Although [241] demonstrated promising results, the designed method suffers low speed due to the complex network architecture. Mancini et al. [34] introduced an end-to-end architecture that jointly learns how to detect obstacles and estimate their depth for UAV flight applications. The designed network has a feature extractor and two task dependent branches, a MDE branch and an obstacle detection branch. The MDE branch has 4 deconvolutional layers and a final convolution layer which outputs the estimated depth map at original input resolution. The obstacle detection branch consists of 9 convolutional layers. Given an input image, the obstacle detection branch divides it into an 8×5 grid of square-shaped cells. For each cell, the detector estimates the coordinates of the bounding box center, the bounding box width and height, a confidence score, the average distance of the detected obstacle from the camera and the variance of its depth distribution. Since [34] applies the fully convolutional part of VGG-19 CNN as feature extractor, its complex architecture increases the running time.

Giusti et al. [240] introduced a DL-based monocular vision method for recognizing forest trail directions and navigates a quadrotor UAV in the forest environment. The authors treat trail perception problem as an image classification task: they estimate the approximate direction of the trail with respect

to the direction of view through a CNN. Since they treat trail perception as a classification problem, the CNN architecture relies on FC layers which induce loads of trainable parameters. In addition, the images were captured when hiker walks a long trail and assuming that person always looking straight along its direction of motion. However, it is difficult to keep looking to the same direction in the whole process. Therefore, this man-made deviation may leads to incorrect image labels.

Loquercio et al. [5] proposed DroNet, a CNN that guides a UAV to fly through the streets of a city. Designed as an 8-layer residual network, DroNet generates two outputs for each single input image: a heading angle to keep the drone flying while avoiding obstacles, and a collision probability is used to modulate the forward speed of the UAV. To avoid the challenge of collecting data in an unstructured outdoor environment, the authors trained DroNet from datasets collected by cars and bicycles, which were already integrated into the city streets. It should be noted that DroNet navigates the UAV to move in a horizontal 2D plane, whilst the process of vertical maneuvering and landing are not considered.

One year later, Maciel-Pearson et al. [242] designed an end-to-end multi-task learning based approach to predict the orientation quaternions and positional waypoints in NED coordinates while exploring an unknown environment. The predicted position and rotational quaternions are passed to the rate and attitude control loops to generate the motion commands. In [5, 240, 242], data for training the DL models was captured manually. The data collection requires an experienced operator and the predicted behaviors tend to be influenced by the operator [243].

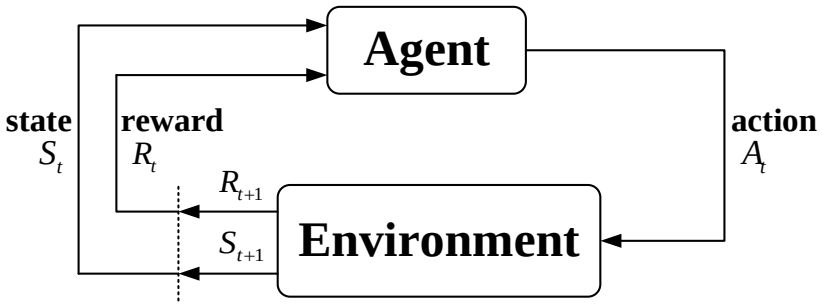


Figure 2.6: The interaction between agent and environment in reinforcement learning.

2.6 Deep Reinforcement Learning

Reinforcement learning (RL) is a subset of machine learning (ML) which concerns how an agent ought to take actions in an environment whilst maximizing the cumulative reward. Policy, reward, value and environment are essential components of RL. In particular, the agent learns about the optimal policy, which is needed in current state. The agent interacts with the environment to produce an optimal policy. As shown in Figure 2.6, the agent environment interaction produces a sequence of state-action-reward-next state. The whole process is modeled by the Markov Decision Process (MDP) which is represented by $< S, A, P, R, \lambda >$. At each time step t, the agent selects an action from the action space A and then performs the action to reach its next state $S_{t+1} \in \{S\}$. The transition from current state S_t to next state S_{t+1} is governed by transition probabilities. After state transition, the agent receives a reward from the environment. The reward is defined as: $R : S \times A \times S \rightarrow \mathbb{R}$. The goal of the agent is to maximize the total accumulated reward received at time step t.

Deep reinforcement learning (DRL) combines DL with RL to produce an optimal solution based on experience. The key difference between RL and DRL is the fact that RL is a computational agent learning problem for making decisions through trail-and-error experience, while DRL allows the agent to make decisions

from unstructured input data free from manual design of the state space.

Mnih et al. [244] developed a deep Q-network (DQN) which demonstrate how a CNN can learn control policies from sensory input. The input to the CNN consists of a $84 \times 84 \times 4$ tensor generated from 4 consecutive stacked frames to capture the temporal information. The first layer consists of 32 filters of 8×8 with stride 4 and a rectifier nonlinearity. The second layer consists of 64 filters of 4×4 with stride 2 and a rectifier nonlinearity. The second layer is followed by a convolutional layer that has 64 filters of 3×3 with stride 1. The final intermediate layer is a fully connected layer with 512 rectifier units. The output layer is a fully connected layer with a single output corresponding to each valid action. Through consecutive layers, the network learns how to combine features for identifying the action most likely to generate the best result. DQN has shown outstanding performance and has been applied to UAV path planning, navigation and attitude control.

Hausknecht and Stone [245] designed a deep recurrent Q-network (DRQN) by combining the Long Short Term Memory (LSTM) with the DQN. Specifically, they replaced the first post-convolutional FC layer with a recurrent LSTM layer. Therefore, the DRQN is capable of integrating information across frames to capture information such as velocity of objects. To solve the over-estimate problem in Q-learning, Van Hasselt et al. [246] proposed a double DQN (DDQN) algorithm. Unlike DQN, DDQN employs a max operator in select action selection. Meanwhile, Wang et al. [247] introduced a dueling network architecture which includes two separate estimators, one for the state value function and another for the advantage function. Then, the two estimators are combined to estimate the action value function.

In 2016, Lillicrap et al. [248] designed a deep deterministic policy gradient

(DDPG) algorithm through extending DQN and deterministic policy gradient. With an actor-critic model, DDPG avoids the optimization of action at every time step to get a greedy policy. The policy of DDPG is deterministic, which is infeasible in complex environments with noise as the policy is required to perform with a certain randomness. The proximal policy optimization (PPO) algorithm [249] is based on the Actor-Critic model, and it requires low computation time. The actor-critic models uses two separate networks. The actor network estimates the optimal action, while the critic network estimates the reward of the action and uses rewards to train the actor. After each action selection, the critic network evaluates the new state to tell whether the result of the selected action is better or worse than expected.

2.7 Deep Reinforcement Learning in Robotics

In this section, we review literature on the application of DRL in robotics. Mobile robots are robotic platforms that are able to move around in an environment through remote control or using an autonomous guidance system. Mobile robots operating in unstructured and dynamic environments have been applied to applications such as search and rescue operations and surveillance. These applications require autonomous robots having the ability of choosing appropriate actions from the perception and interaction with the environment. DRL is an end-to-end learning method which takes as input raw sensor data and produces robot actions. There is a variety of work in the literature applying DRL to robotic applications such as navigation, obstacle avoidance and autonomous exploration.

Wang et al. [250] proposed an autonomous navigation method that enables a UAV to fly from arbitrary start points to destinations. Without using local

or global maps and path planning, the proposed method passes sensory data captured from the local environment and the GPS signal to the DRL module to navigate the flight of UAV. Xiang et al. [251] applied a DRL algorithm, i.e., Soft Actor-Critic (SAC) to navigate a simulated UAV. The SAC network takes as input the laser scanning data and information of the destination and outputs continuous linear and angular velocities for controlling the UAV. Barros et al. [252] trained the SAC algorithm for controlling a simulated UAV in the go-to-destination task. The state space consists of the relative position and orientation of the UAV to the destination, the relative linear and angular velocities, the rotation matrix, and the actions taken in the previous step for all motors.

While DRL-based methods achieve promising performance, they are prone to local minima and lack of long term memory [253]. To solve this problem, Kastner et al. [254] integrated a DRL-based local planner with traditional global planners. The Asynchronous Advantage Actor Critic (A3C) algorithm takes as input the 360 degree LiDAR data and produces continuous action states which are utilized in the local planner. The traditional global planners can be rapidly-exploring random tree (RRT) [255], A^* [256] or Dikstra [257] algorithms. The global and local planners are connected by an interconnection waypoint generator. In [258], Huang et al. combined DRL with multi-modal fusion to navigate an unmanned aerial vehicle (UGV) in real-world scenes. Features learned from raw images and LiDAR data, and the measurement of UGV velocities are passed through the PPO-based policy module to output linear and angular velocities for steering the UGV.

In terms of obstacle avoidance, Xie et al. [259] developed a dueling architecture based deep double-Q network (D3QN). Based on the combination of dueling and double Q mechanisms, D3QN learns policies from depth maps estimated from

RGB image. Singla et al. [260] designed a deep recurrent Q-network (DRQN) with temporal attention for UAV obstacle avoidance in indoor environments. The designed approach first applies the conditional generative adversarial network (cGAN) to estimate depth maps from monocular RGB images. Then, the produced depth maps are fed to the DRQN to learn an optimal action. In 2021, Thomas et al. [261] proposed an obstacle avoidance algorithm for a UAV by combining a self-attention model with duelling deep Q-network. Xue et al. [262] combined the Variational Autoencoder (VAE) and the Twin-Delayed Deep Deterministic Policy Gradient (TD3) algorithm to solve the problem of obstacle avoidance for a UAV. The VAE converts the RGB image captured by the front camera into 32 variables. The TD3 algorithm consists of actor networks (actor network and actor target network) and critic networks (critic network and critic target network). The actor networks incorporate 32 variables from VAE and output velocities in the y direction (left and right directions) and z direction (up and down directions). The critic networks estimate the Q value when a certain action value is selected in a certain state.

LiDAR perceives accurate depth information, however, it captures less redundant information than image data which is important to train a DRL model. Gao et al. [263] applied semantic segmentation to encode the depth maps for generating the one-dimensional pseudo-laser data. Given an input image, the depth map and semantic segmentation mask are extracted through the corresponding CNN models. The semantic segmentation mask is subsequently used to cull out the traversable region from the depth map. The processed depth map is passed through a dynamic local minimum pooling operation to generate the pseudo-laser data. Finally, the pseudo-laser data along with the relative goal position and robot's current velocity are fed to the DRL module to produce actions. Com-

pared with traditional LiDAR data which only includes one-dimensional distance information, the pseudo-laser data encodes context information of objects in the scene. Therefore, it provides more reliable information for DRL network than the pure laser-based measurement.

Robot exploration is the operation of controlling a robot moving in an unknown environment while constructing a map which can be applied to the subsequent navigation [264]. The main objective of robot exploration is to construct the map of the working environment without depending on prior knowledge. A classical method for robot exploration is frontier search that origins from the work introduced by Yamauchi [264]. Yamauchi defined the frontier as regions located at the border between explored and unexplored space in the occupancy grid map. The robot keeps track of the frontiers and selects the best frontier as the next desired position. DRL algorithms have also been applied to enable robots to learn how to explore the environment through the observations/perception of their surroundings. Zhu et al. [265] introduced a method that adopts DRL to learn exploration knowledge over office floor maps. The Asynchronous Advantage Actor-critic (A3C) network incorporates the current map, the agents location and orientation to predict the next visiting direction. Niroui et al. [266] combined DRL and frontier-based exploration to solve the robot exploration problem. The 2D occupancy grid, coordinates of possible frontiers, and the robot location are all passed to the A3C network which then predicts coordinates of the next goal frontier. Li et al. [267] proposed an autonomous exploration method where the DRL-based decision module is applied to select the next goal location in the grid map. The decision module is formulated as:

$$g_T = F_{decision}(l_{0:T}, m_T), \tag{2.14}$$

where g_T denotes the goal point, $l_{0:T}$ represents robot positions from step 0 to T, and m_T is the environment built at step T.

Area coverage or coverage path planning is a representative exploration task which aims to enable a robot to visit as many free areas as possible in an environment without colliding with obstacles. Tran et al. [268] introduced a frontier search driven swarming algorithm that controls robot swarm to perform area coverage. It is worth noting that robot swarming is beyond the scope of this thesis, we will review literature on area coverage for a single robot. According to our literature review, many traditional methods to solve the area coverage problem depend on knowing the model of the environment [269]. Since an environmental model is usually not available, DRL techniques have been applied to solve this problem. In [270], Saha et al. combined DQN with prioritized experience replay to solve the problem of area coverage in room sized environments.

Maciel-Pearson et al. [271] developed an Extended Double Deep Q-Network (EDDQN) based method to solve the exploration and obstacle avoidance problems in a partially observable environment for a UAV. The main innovation of the developed method is a double input state that combines the acquired knowledge from the raw image and a local map containing positional information. The positional information aids the DRL network to understand where the UAV has been and how far it is from the target position, the feature map from the image of current scene highlights cluttered areas that are to be avoided. Piciarelli et al. [272] designed a DDQN-based method to find optimal patrolling strategies for UAV visual coverage task. The network input consists of the current agent state and the relevance map whose values represent the relevance of an observed area, i.e., the importance of its observation or the cost for the system if that area is not observed. The network outputs the Q values for all the possible agent actions.

In summary, the articles described in this section give a brief overview of DRL-based navigation, obstacle avoidance and autonomous exploration methods for robotics. Based on the literature review it is observed that a DRL-based area coverage method that applies pure vision perception method is missing. Motivated by this context, this thesis is mainly concerned with a method that only uses a monocular camera to perceive the environment and guide the movement of UAVs. Considering the trade-off between computational efficiency and accuracy, real-time CNN models for MDE and OF estimation will be applied to this research.

2.8 Simulators

Robot simulator is an essential tool for researchers in the field of robotics. The comprehensive review of currently used simulators is beyond the scope of this section, instead, we describe some notable simulators here. Gazebo [273] is one of the most commonly used simulators in the field of autonomous systems. Benefiting from the modular design, Gazebo allows to utilize different physic engines, sensor models and build 3D worlds. However, it is difficult for Gazebo to construct large scale visually realistic environments [274]. Meyer et al. [275] developed a simulator for quadrotor UAVs through integrating Robot Operating System (ROS) and the Gazebo simulator. Since the developed simulator tightly relies on ROS and Gazebo softwares, it is limited by richness of simulated environments.

In [276], Furrer et al. presented a modular framework to build MAVs, and develop control and state estimation algorithms. However, the presented framework also applies Gazebo as its platform, which limits its perception related capabilities. jMavSim [277] is a simple multirotor simulator which is designed

for the purpose of testing PX4 firmware and devices. Therefore, it is tightly integrated with PX4 Application Programming Interfaces (APIs). Additionally, jMavSim applies simpler sensor models and depends on simple rendering engine without any objects in the environment.

In 2017, Dosovitskiy et al. [278] developed a simulator for autonomous driving in urban environments, named Car Learning to Act (CARLA). CARLA is built over Unreal Engine 4 (UE4) [279] and supports training, prototyping, and validation of perception and control models for autonomous driving. Being designed as a simulator for UGVs, CARLA does not support the simulation of UAVs. One year later, Microsoft introduced AirSim [274], a simulator providing physically and visually realistic scenarios for the simulation of UAVs and UGVs. Similar to CARLA, AirSim is also built on top of Unreal Engine (UE). Due to the powerful graphical capabilities, UE enables construction of photorealistic environments. Thus, it benefits the development of perception algorithms and sim-to-real transfer techniques. Since this thesis aims to develop the real-time visual guidance method for UAVs in photorealistic environments, we choose *AirSim* as the simulator.

2.9 Chapter Summary

In this chapter, the key concepts relate to visual guidance for UAVs have been reviewed. The main contribution of this chapter is a comprehensive survey on MDE, which covers classical and state-of-the-art DL-based methods as well as the application of MDE in robotics. Inspired by this survey, we propose two novel MDE networks in Chapter 4 and Chapter 5 respectively. Furthermore, we discussed the development trend of DL techniques, state-of-the-art methods for

navigation, obstacle avoidance and autonomous exploration that harnesses the power of various DL methods and DRL methods. Based on the literature review, we obtained the research gaps to be filled in this thesis.

Chapter 3

Simulation Framework

3.1 Introduction

In this thesis, several critical components are adopted to achieve the task of monocular vision-based guidance for Unmanned Aerial Vehicles (UAVs). Currently, there are two main test methods for UAVs. The direct way is to use a real UAV. However, actual flights are inconvenient due to the following problems. First, the UAV in testing is easy to crash which will bring about economic losses. Furthermore, real flight tests need to consider factors such as safety, aviation regulation compliance, maintenance, weather, battery life and so on. Those factors will influence flight efficiency and increase the testing costs. Therefore, in this thesis experiments relate to UAV guidance are performed in a simulation environment. This chapter introduces the deep learning (DL) framework for training convolutional neural networks (CNNs) in Chapters 4 and 5, and the deep reinforcement learning (DRL) network in Chapter 6, as well as the simulator for running simulation experiments in Chapter 6.

Figure 3.1: Image produced by Unreal Engine [279], a screenshot from the open-source "Rural Australia" environment [280].

3.2 Unreal Engine

The Unreal Engine (UE) [279] is an open-source and cross-platform 3D computer graphics game engine developed by Epic Games. UE supports Windows, macOS, and Linux, and offers solutions for constructing large-scale simulated environments. Benefiting from the open-source community, UE has a high availability of free plugins and extensions. UE has full open-source code in C++, which can be modified to meet user's specific requirements. It also provides a node-based editor called Blueprints. Blueprints allows users to implement logic without writing C++ code, as many functions are already implemented. This can accelerate implementation and allows for quickly executed experiments.

The UE Marketplace has dozens of pre-built extra-ordinarily detailed simulated environments. Many of these environments are free and can be used after a few build steps. Apart from the available assets in the UE Marketplace, assets from most of the modeling software can be easily imported to UE. To sum up, UE has the following highlights: (1) It is open-source and can be easily modified to meet the user's requirement; (2) It has the ability to generate photorealistic

images (see Figure 3.1); (3) It offers tools and assets for building the simulated environment. Assets from other modeling software integrate well with UE; (4) In addition to game industry, UE is also a popular choice for virtual reality (VR) and architectural visualization. Therefore, high-quality 3D contents are easily accessible.

3.3 AirSim

AirSim [274] is a simulator for UAVs, Unmanned Ground Vehicles (UGVs) and other objects, developed on top of the UE. It is an open-source, cross-platform software designed by Microsoft. AirSim supplies physically and visually realistic simulations through supporting hardware-in-the-loop (HIL) simulations with different controllers. In particular, AirSim has an integral controller named "simple flight", which is used by default. It also supports PX4 [281], an open-source flight control solution for advanced users. The "simple flight" controller controls the vehicle through incorporating the desired input information such as angle rate, angle level, velocity or position. In essence, simple flight applies a sequence of proportional integral derivative (PID) controllers to produce actuator signals. The PID position loop output drives the PID velocity loop setpoint, which then drives the PID attitude controller setpoint which then finally drives the PID angular rate loop setpoint.

AirSim is designed as a plugin in UE that can be imported in any UE project. This enables AirSim to take advantage of features of UE regarding its rendering and physics simulation, user interface, and computational efficiency. AirSim was developed for solving two main problems in the research community of autonomous systems: the requirement of large-scale datasets for training and

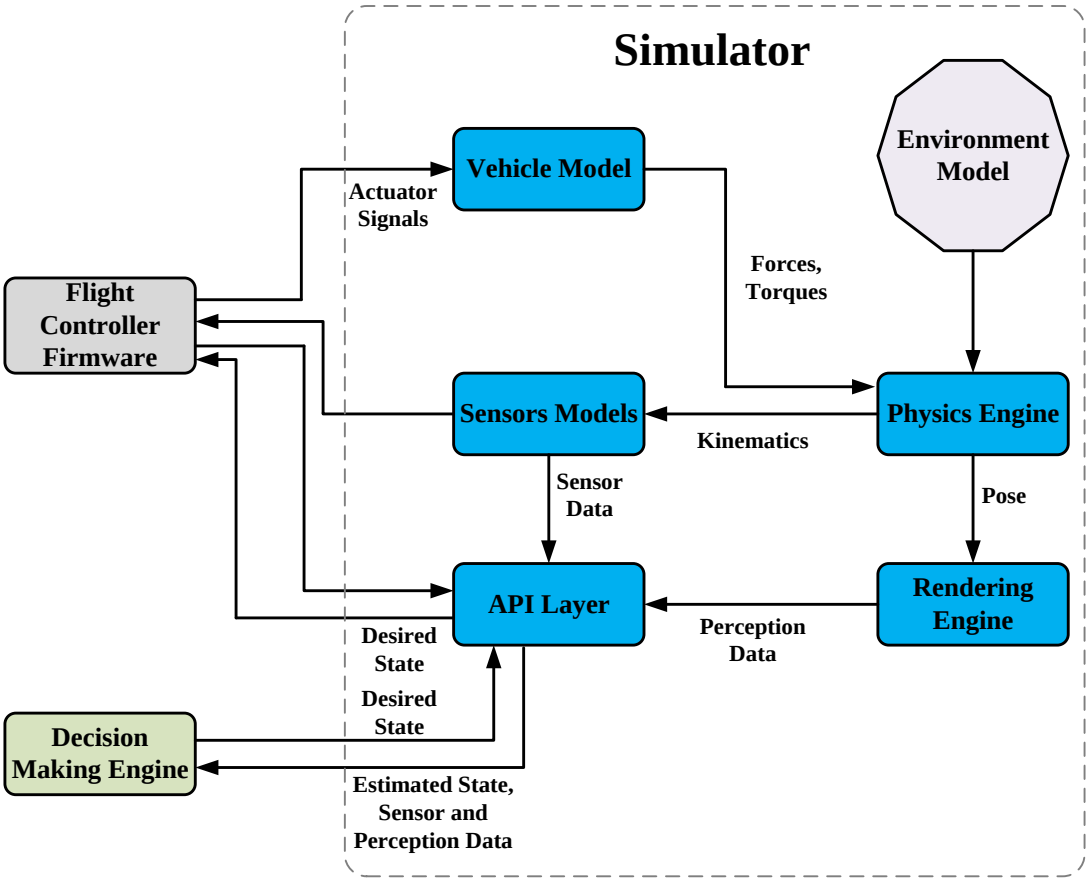


Figure 3.2: The architecture of AirSim.

testing the developed systems, and the ability to debug in a simulator. For this reason, AirSim provides APIs for independent platform recovery of data and control vehicles.

3.3.1 Architecture

AirSim adopts the modular design and highlights extensible ability. The main modules of AirSim are vehicle model, environment model, sensor models, rendering engine, physics engine, public API layer, and an interface layer for vehicle firmware. The architecture of AirSim is illustrated in Figure 3.2. The details of each individual module in AirSim are described as follows.

Vehicle Model

AirSim allows users to define the vehicle as a rigid body that has an arbitrary number of actuators producing forces and torques through the provided interface [274]. The vehicle model consisting of a series of parameters, including mass, inertia, coefficients for linear and angular drag, and coefficients of friction and restitution. These parameters are passed through the physics engine to produce rigid body dynamics.

The defined vehicle is normally represented by a stack of N vertices placed at positions $\{\mathbf{r}_1, \mathbf{r}_2, ..., \mathbf{r}_n\}$ and normals $\{\mathbf{n}_1, \mathbf{n}_2, ..., \mathbf{n}_n\}$. Each vertex takes as input a unitless vehicle specific scaler control input $\{u_1, u_2, ..., u_n\}$. The produced forces and torques from these vertices are assumed to have the same directions as their normals. During the process of simulation, the positions and normals are allowed to change.

Figure 3.3 illustrates how a defined UAV can be represented by four vertices. The control input u_i drives the rotational speed of the propellers, which located at the four vertices. The produced forces ($\mathbf{F}_i$) and torques (τ_i) from the four propellers are calculated by Equation (3.1) and Equation (3.2) respectively.

$$\mathbf{F}_i = C_T \rho \omega_{max}^2 D^4 u_i, \tag{3.1}$$

$$\tau_i = \frac{1}{2\pi} C_{pow} \rho \omega_{max}^2 D^5 u_i, \tag{3.2}$$

where C_T and C_{pow} represent the thrust and the power coefficients respectively, ρ means the air density, D stands for the propeller's diameter, and ω_{max}^2 indicates the max angular velocity in revolutions per minute.

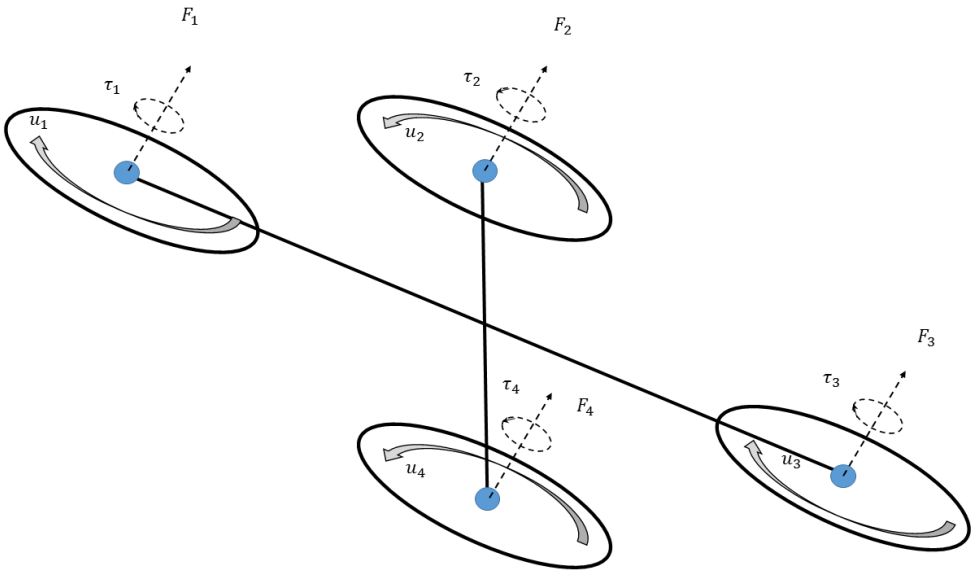


Figure 3.3: The vehicle model for the quadrotor UAV in AirSim [274]. The four vertices (represented as blue dots) take as input the controls $\{u_1, u_2, u_3, u_4\}$ and produce four forces $\{\mathbf{F}_1, \mathbf{F}_2, \mathbf{F}_3, \mathbf{F}_4\}$ and four torques $\{\tau_1, \tau_2, \tau_3, \tau_4\}$. Reproduced with permission from Springer Nature, Field and Service Robotics, Shah et al. [274].

Environment Model

In the real-world environment, a vehicle is exposed to diverse physical phenomena. While AirSim is able to generate computationally expensive models of those phenomena, it is focused on modeling accurate gravity, air density, air pressure and magnetic field in order to enable a real-time operation with HIL.

Sensors Models

AirSim provides a variety of simulated sensors including accelerometer, gyroscope, barometer, magnetometer, GPS, LiDAR, IMU, and cameras. The models of these sensors are written in C++ header-only libraries, which can be independently applied outside of AirSim. Additionally, sensor models are represented as abstract interfaces. Therefore, it is easy to replace or add new sensors in AirSim.

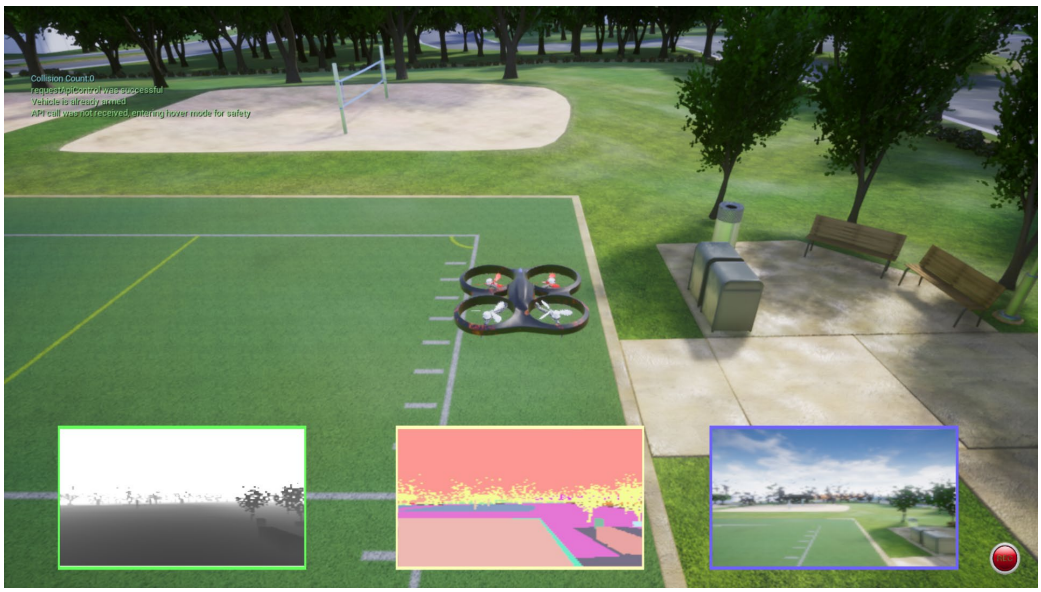


Figure 3.4: A snapshot from AirSim shows a UAV flying in a simulated soccer field environment where it is surrounded by different objects. The sub-windows illustrate depth, semantic segmentation and RGB images from the viewpoint of the front view camera.

In this thesis, we only use the front center camera to perceive the environment. AirSim provides image APIs for retrieving synchronized images from multiple cameras along with ground-truth including depth, disparity, surface normals, semantic segmentation and optical flow. Furthermore, image parameters (e.g., resolution, FoV, motion blur, etc) can be set in the *settings.json* file.

Rendering Engine

Due to advanced rendering and detailed environments are core requirements for a simulator, AirSim [274] applies UE or Unity as the rendering platforms. UE is open-source and cross-platform, which makes it an attractive choice for the research community. Besides, UE has cutting edge graphics features such as physically-based materials, photometric lights, planar reflections, ray traced distance field shadows, lit translucency, etc. Therefore, it can generate a series of

highly photorealistic environments including urban, mountain and forest scenarios. In this thesis, we use UE as the rending engine because it supports faster rendering and has superior graphics quality. Figure 3.4 illustrates a screenshot from a simulated soccer field environment[1] that demonstrate the near photo-realistic rendering capabilities.

Physics Engine

AirSim applies six parameters to represent its kinematic state: position, orientation, linear and angular velocities, and linear and angular accelerations. The function of the physics engine is to calculate the next kinematic state for each body given the forces and torques acting on it. In AirSim, the physics engine can run its update loop at a high frequency up to 1000Hz which is expected to enable real-time simulation scenarios.

3.3.2 Environments and Models

AirSim provides several simulated environments which can be directly used in simulation experiments. Those environments are only available in release binaries. Due to copyright, they cannot be edited through the UE editor. The UE marketplace provides dozens of assets and users can setup their own environments by creating an Unreal project [282]. In addition, environments and 3D models in file formats like *Filmbox (.fbx)*, *Wavefront OBJ (.obj)*, and *STL (.stl)* formats, can also be imported to the UE editor.

[1] https://github.com/Microsoft/AirSim/releases

3.4 PyTorch

PyTorch [283] is an open-source machine learning (ML) framework developed on the basis of Torch library. Many ML frameworks such as Caffe [284], CNTK [285], TensorFlow [286] and Theano [287] have been developed before PyTorch. Those frameworks use a static dataflow graph to model the computation and these are applied repeatedly to batches of data. This has the advantages of visualizing the whole computation ahead of time and being leveraged to improve performance and scalability. However, it suffers from difficulty in use, difficulty in debugging, and it cannot cope well with different types of computation that can be represented. By contrast, PyTorch adopts dynamic computation graphs where the graph structure is defined on-the-fly via the actual forward computation. Specifically, the graph is rebuilt from scratch on every iteration. This enables PyTorch to scale well for different dimensional inputs and makes it easy to debug.

3.4.1 Highlights of PyTorch

The highlights of PyTorch are summarized as follows:

- **Deep Learning Models are Just Python Programs:** In PyTorch, layers are typically represented as Python classes. For models, they are normally expressed as classes which consisting of individual layers. Since PyTorch is tightly integrated with Python, many Python debugging tools can be easily used with it.

- **Interoperability and Extensibility:** Easy and efficient interoperability is one of the most apparent features of PyTorch. It allows users leveraging Python libraries as part of their programs. Thus, PyTorch supports bidirectional exchange of data with external libraries. For instance,

the NumPy arrays and PyTorch tensors can be converted through the **torch.from_numpy()** function and **.numpy()** tensor method. Furthermore, many of the core systems in PyTorch are designed to be extensible.

- **Automatic Differentiation:** Back propagation (BP) is the most frequently used algorithm for training neural networks. It adjusts network parameters (model weights) according to the gradient of the loss function with respect to the given parameter. To compute these gradients, PyTorch provides a built-in automatic differentiation module called **torch.autograd**. During backpropagation, **torch.autograd** automatically computes the gradient of functions defined in **torch.nn**. It is worth noting that **torch.autograd** supports the automatic computation of gradient for any computational graph.

3.4.2 PyTorch Basic Components

PyTorch has five main components:

- **Tensors:** Tensors are the core data type in PyTorch. They are N-dimensional arrays similar to Numpy arrays, but can run on GPUs to accelerate their numeric computations.

- **Variable:** A variable is a wrapper around a tensor, and represents a node in a computational graph. For example, if x represents a variable then $x.data$ is a tensor giving its value, while $x.grad$ is another variable containing the gradient of x with respect to some scalar value. In PyTorch, variables have the same API as tensors. Any operation that users can do on a tensor can also do on a variable, the difference is that autograd allows users to automatically compute gradients.

- **Parameter:** Parameter is a kind of tensor that is to be considered as a module parameter. Parameters are sub-classes of **Tensor**, and it usually used to create some tensors in PyTorch model.

- **Functions:** In PyTorch, functions perform transform operations, and they does not require memory to store any state.

- **Module:** Module is the basic class for neural networks. A module can contain other modules. For example, a neural network is a module itself consisting of other modules (layers). In PyTorch, the **torch.nn** namespace provides all blocks that users need to build their own neural network. Every module is the subclass of **nn.Module**.

3.5 Chapter Summary

In this chapter, the game engine for rendering, the simulation platform, and the deep learning framework are elaborated. The Unreal Engine is used as the rendering engine of AirSim, which simulates a UAV flying in a complex environment. PyTorch, a popular deep learning framework is adopted to train CNNs for depth perception and the deep reinforcement learning network for controlling the movement of UAV. With these software packages, simulation experiments for UAV guidance will be performed in Chapter 6.

Chapter 4

MobileXNet for Real-Time Monocular Depth Estimation

The work, reported in this chapter, has partially been published in the following journal article [288]:

Xingshuai Dong, Matthew A Garratt, Sreenatha G Anavatti, and Hussein A Abbass. MobileXNet: An efficient convolutional neural network for monocular depth estimation. IEEE Transactions on Intelligent Transportation Systems, 23(11):20134-20147, 2022.

4.1 Introduction

This chapter aims to develop a real-time convolutional neural network (CNN) for monocular depth estimation (MDE). State-of-the-art methods [76, 77, 79, 80, 119, 121, 289, 290] normally utilize CNNs to learn features in order to predict a depth value for each pixel, and significantly outperform their classical counterparts [68, 73, 82]. However, these methods were developed based on extremely deep and

complex network architectures. Therefore, they suffer a heavy computational cost and cannot be run in real-time without using high-end GPUs. It is impractical to deploy those methods on platforms where reaction time is critical, such as small drones.

On the other hand, Wofk et al. [81] introduced an encoder-decoder network using the lightweight CNN [44]. Network pruning techniques [291] were used to further reduce the size of the model. Despite their method achieving a significant improvement in speed, the accuracy was not comparable to the state-of-the-art results.

In general, the above-mentioned methods employ the fully convolutional part of the CNNs [41–44] designed for image classification as the feature extractor/encoder. However, these networks downsample the resolution of the final feature maps to 1/32 scale of the input image. Although successive downsampling increases the receptive field and enables the generated feature maps to include more semantic information, it results in loss of spatial information [175]. It is difficult to accurately recover this type of information from these feature maps. Hence, the accuracy of depth estimation will be impaired, in particular, when small sized images are processed. Moreover, the loss functions (i.e., the L_1 loss, L_2 loss and berHu loss) used by those methods are insensitive to the errors which occur at step edges. This leads to distorted and blurry edges in the estimated depth maps.

To address the above problems, we propose a novel CNN and a hybrid loss function for MDE in this chapter. In particular, we aim to develop a network which has a proper trade-off between accuracy and speed. To be specific, the proposed network assembles two relatively shallow encoder-decoder style sub-networks back-to-back in a unified framework. Compared with the previous

studies [76, 77, 79, 80, 119, 121, 289, 290], our network has a much shallower and simpler architecture as it is built on top of a series of simple convolutional layers rather than Inception modules [292] or Residual blocks [41]. In addition, each subnetwork involves less downsampling operations. This enables the intermediate feature maps to have a larger resolution and contain more spatial information, which is beneficial for depth estimation on small sized images. To the best of our knowledge, this work is the first attempt that stacks shallow encoder-decoder style CNNs in a unified framework to avoid the loss of spatial information and enhance the network's representative ability for depth estimation. Different from existing works [79, 290], our method avoids the loss of spatial details but does not incorporate a separate spatial branch or fuse the hierarchical features of the encoder network. To preserve more edge details of objects, we design a hybrid loss function which adds constraints on the gradient data of depth maps.

Since the proposed network is designed for autonomous driving and mobile robots as well as is inspired by the XNet [293], we refer to it as "MobileXNet" in this chapter. Compared with the XNet network, MobileXNet has three different characteristics. First, it is developed for the regression application rather than the classification task. In addition, the encoder of the first subnetwork consists of a series of depthwise separable convolutional layers [45], which enables a faster computational speed. Finally, a bridge module is inserted between the encoder and decoder in each subnetwork, including a set of dilated convolutional layers with different dilation rates, to capture the context information in multiple scales. This configuration generates better results than those derived using a single dilation rate. As a result, the proposed MobileXNet can be used for depth estimation in a faster and more effective manner in contrast to the XNet method.

The contributions of this chapter are summarized as follows: (1) We introduce

a novel CNN architecture for achieving running efficient and accurate depth estimation from a single image; (2) We design a hybrid loss function; (3) We evaluate the proposed method on a dataset consisting of small sized synthetic images. In particular, we show that the size of the filters used in the input layer of CNNs has an influence on the performance of MDE when small sized images are processed; (4) We use Pareto Optimality to compare the error and running time over different methods, which has not been exploited in depth estimation; and (5) We demonstrate that the proposed method not only generates comparable accuracy to the state-of-the-art methods which use either extremely deep and complex architecture or post-processing but also runs much faster on a single less powerful GPU.

The rest of this chapter is organized as follows. In Section 4.2, our methodology is introduced. The experimental setup and results are reported in Sections 4.3 and 4.4 respectively. Finally, conclusions are presented in Section 4.5.

4.2 Methodology

In this section, we first introduce the details of the proposed CNN architecture for MDE and then describe the loss functions used for training.

4.2.1 CNN Architecture

Targeting at addressing the problems with state-of-the-art approaches and achieving the trade-off between accuracy and speed, we design a novel network which learns the end-to-end mapping from an RGB image to the corresponding depth map. We fulfil the task by assembling two simple and shallow encoder-decoder style subnetworks in a unified framework. As shown in Figure 4.1(b), each

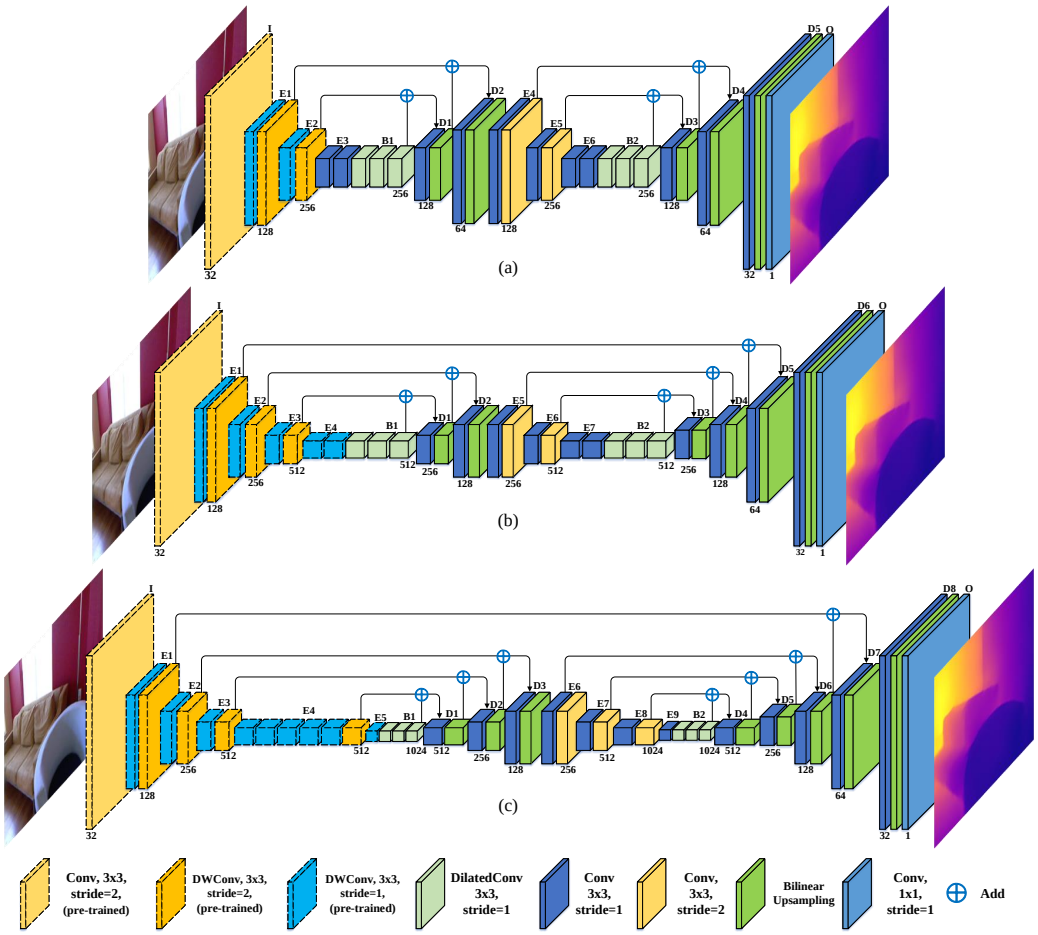


Figure 4.1: Architecture of the proposed MDE networks (best viewed in color). (a) MobileXNet-Small, (b) MobileXNet, and (c) MobileXNet-Large. I represents the input layer, E_i means the i_{th} encoder block, B_i stands for the i_{th} bridge module, D_i indicates the i_{th} decoder block and O denotes the output layer.

subnetwork only consist of simple convolutional layers.

The encoder of the first subnetwork takes a single RGB image as input. After four times downsampling operation, it produces feature maps (referred to as F_1) at 1/16 scale of the input image. This avoids a greater number of successive downsamplings which impairs the accuracy of recovering boundary level detail and the depth estimation on the small sized images. Subsequently, F_1 are passed through two upsampling steps and the generated feature maps F_2 are fed to the second subnetwork, which consists of two downsampling and four upsampling

steps and outputs a dense depth map. In order to capture fine-grained image details, skip connections are applied to different levels of the designed network. Instead of concatenating, feature maps are fused via addition to avoid the increase of feature map channels processed in the decoder. This results in a further improvement in running speed.

To reduce network latency, we use depthwise separable convolution to design the encoder of the first subnetwork (see Figure 4.1(b)). The depthwise separable convolution factorizes a regular convolution into a depthwise convolution and a pointwise convolution. Specifically, the depthwise convolution applies a single filter at each input channel, and the pointwise convolution is used to create a linear combination of the output of the depthwise layer.

The depthwise convolution with one filter per input channel is defined as:

$$\hat{G}_{k,l,m} = \sum_{i,j} \hat{K}_{i,j,m} \cdot F_{k+i-1,l+j+1,m}, \tag{4.1}$$

where $\hat{K}$ is the depthwise convolutional kernel with size of $D_K \times D_K \times M$, and the m_{th} filter is operated on the m_{th} channel in input feature map F to produce the m_{th} channel of the output feature map $\hat{G}$. Hence, it has a computational complexity of $D_K \cdot D_K \cdot M \cdot D_F \cdot D_F$, where D_K is the spatial dimension of the kernel, M is the number of input channels and D_F is the feature map size. Since the depthwise convolution only operates on input channels, and is followed by a 1×1 convolution which combines the output to generate the final feature map, the computational cost of depthwise separable convolution is:

$$D_K \cdot D_K \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F, \tag{4.2}$$

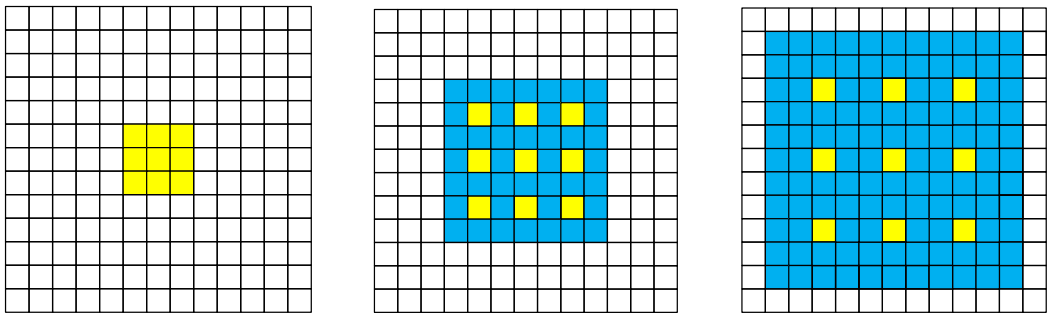
where N indicates the number of output channels. While for the regular convolution which has the same filter kernel size, the computation cost is:

$$D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F. \tag{4.3}$$

By using depthwise separable convolution we can reduce the number of parameters in the convolutional layer to:

$$\frac{D_K \cdot D_K \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F}{D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F} = \frac{1}{N} + \frac{1}{D_k^2}. \tag{4.4}$$

For depthwise separable convolutions with kernel size of 3, the amount of computation is 8 to 9 times less than the regular convolution. Thus, it is helpful to improve the computation speed of MobileXNet. In order to take advantage of the pre-trained weights on the ImageNet dataset [294], we design the encoder (boxes with dotted line in Figure 4.1(b)) of the first subnetwork with the same configuration as the first nine layers of MobileNet [44].



(a) (b) (c)

Figure 4.2: Illustration of the dilated convolution, which expands the convolution kernel through inserting holes between the kernel elements. (a), (b), (c) are 1-dilated, 2-dilated, 3-dilated convolution respectively. The yellow squares represent kernel elements, and blue squares indicate inserted holes.

A larger receptive field enables the network to capture more context informa-

tion, which can be leveraged to improve depth estimation performance [76]. The dilated convolution expands the convolution kernel by inserting holes between the kernel elements. This increases the receptive field of the kernel elements without increasing the number of model parameters [295]. Therefore, we insert a bridge module between the encoder and decoder in each subnetwork. Specifically, the bridge module consists of three dilated convolutional layers. Given a discrete function defined by $\mathcal{F} : \mathcal{Z}^2 \rightarrow \mathcal{R}$, let $\Omega_r = [-r, r]^2 \cap \mathcal{Z}^2$ where r represents the kernel radius, and $k : \Omega_r \rightarrow \mathcal{R}$ be a discrete filter of size $(2r + 1)^2$. The discrete convolution operator $*$ can be expressed as:

$$(F * k)(p) = \sum_{s+t=p} F(s)k(t). \tag{4.5}$$

Based on this filter, the dilated convolution with a dilation rate l can be formulated as:

$$(F *_l k)(p) = \sum_{s+lt=p} F(s)k(t), \tag{4.6}$$

where $*_l$ is an l-dilated convolution. The regular convolution $*$ is simply the 1-dilated convolution. The illustration of dilated convolution is shown in Figure 4.2. In this study, the dilation rates are set as 1, 2 and 3. By doing this, the decoders can get the context information captured in multiple scales.

The task of the decoder is to refine and upsample the output of the encoder. The core part of the decoder is the upsampling layer to upsample the spatial resolution. The commonly applied methods are unpooling, up-projection, transpose convolution and interpolation combined with convolution. According to [76], up-projection produces better performance than unpooling and transpose convolution. However due to the relatively complex architecture, it significantly increases the number of network parameters. Inspired by [296], we choose the

combination of a bilinear interpolation and a 3×3 convolutional layer as the upsample layer in this study. The 3×3 convolutional layer reduces the number of output channels to half of the number of input channels. The bilinear interpolation increases the spatial resolution of intermediate feature maps to the same size as the output of the corresponding encoder layer. The main advantage of this design is the simple network architecture and reduced computational time.

We also design two variants that perform three and five times of downsampling in the first subnetwork, and we name them as MobileXNet-Small and MobileXNet-Large respectively. We train and test MobileXNet and the variants on the NYU depth v2 dataset [3] to compare the performance. The network architectures are shown in Figure 4.1.

4.2.2 Loss Functions

A loss function is applied to measure how far the predicted depth map is from the ground-truth, and this is used to supervise the update of network weights. Thus, it plays an important role in training CNNs. A commonly used loss function for depth estimation is the L_2 loss as it is more effective for pixels with larger error [297]. The L_2 loss is defined as:

$$L_2(d, d^*) = \frac{1}{N} \sum_{i}^{N} |d_i - d_i^*|^2, \tag{4.7}$$

where N is the number of pixels being considered, d and d^* are the predicted and ground-truth depth respectively. In [76], Laina et al. introduce a reverse Huber loss:

$$L_{berHu}(d, d^*) = \begin{cases} |d - d^*| & if |d - d^*| \leq c, \\ \frac{(d-d^*)^2 + c^2}{2c} & otherwise, \end{cases} \tag{4.8}$$

where c is the threshold and set as $c = \frac{1}{5}max_i(|d - d^*|)$. The berHu loss is equal to L_2 loss when the error exceeds c, whilst when the error falls in $[-c, c]$ the berHu loss is equal to the L_1 loss:

$$L_1(d, d^*) = \frac{1}{N}\sum_{i}^{N}|d_i - d_i^*|, \tag{4.9}$$

which is more robust to outliers in the dataset [298]. However, the above described loss function are insensitive to errors emerging at blur edges [170]. To penalize the errors around edges, we define a hybrid loss which combines the regular L_1 loss with the image gradient-based L_1 loss:

$$L_{grad}(d, d^*) = \frac{1}{N}\sum_{i}^{N}|\bigtriangledown_x(d_i, d_i^*)| + |\bigtriangledown_y(d_i, d_i^*)|, \tag{4.10}$$

where $\bigtriangledown_x$ and $\bigtriangledown_y$ are spatial derivative in x and y direction respectively. Finally, the defined hybrid loss is formulated as:

$$L_{hybrid} = L_{depth} + L_{grad}, \tag{4.11}$$

where L_{depth} is the regular L_1 loss, and L_{grad} is the image gradient-based L_1 loss.

4.3 Experimental Setup

We introduce our experimental setup in this section. It includes three parts: implementation details, data augmentation methods and performance metrics.

4.3.1 Implementation Details

We implement the proposed networks using PyTorch [283]. A desktop with an i7-7700 CPU, 16GB RAM and a single Nvidia GTX 1080 GPU is used for training and testing. In order to improve accuracy and avoid over-fitting, we initialized the weights of the encoder of the first subnetwork of MobileXNet and its variants with the pre-trained model on the ImageNet dataset [294]. The other layers are initialized from He initialization [299] with a standard deviation of $\sqrt{2/N}$, where N represents the number of incoming nodes of one neuron [299]. Following [81, 119], we set the batch size as 8 and use the SGD optimizer with an initial learning rate of 0.01 to train our models. For the NYU depth v2 [3], KITTI [1] and Unreal [34] datasets, we train the network for 20 epochs, and reduce the learning rate to 20% every 5 epochs. For the Make3D dataset [82], we train MobileXNet for 100 epochs, and reduce the learning rate to 20% every 40 epochs.

4.3.2 Data Augmentation

To increase the diversity of training samples, we utilize the following data augmentation methods, which are applied to each RGB and depth image pair in an online fashion:

(1) Random Rotation: RGB and depth image pairs are rotated by a random angle $r \in [-5, 5]$ degrees.

(2) Random Scale: RGB images are scaled by a random factor $s \in [1, 1.5]$, and the corresponding depth maps are divided by s.

(3) Color Jitter: the brightness, contrast, and saturation of the RGB images are scaled by a random factor $c \in [0.6, 1.4]$.

(4) Random Flips: RGB and depth image pairs are horizontally flipped with

the probability of 0.5.

4.3.3 Performance Metrics

In this section, we evaluate each method with four metrics. In addition to the linear Root Mean Square Error (RMSE), Absolute Relative Difference (Abs REL) and Threshold Accuracy (δ_i, $i = 1, 2, 3$) described in Section 2.3.1, we also use the running time (t_{GPU}) to measure the speed of MoboleXNet. The running time is defined as the average execution time of testing each frame on a single GPU. In this section, we report the running time of our method on the Nvidia GTX 1080 GPU. Due to the code and GPU model of some methods not being available to us, we cannot do a direct comparison on the Nvidia GPU. Therefore, we instead compare against their running time and GPU model as reported in the literature.

4.4 Experimental Results

In this section, we present the experimental results on MDE. We evaluate the proposed method on both indoor and outdoor scenes. Four datasets, the NYU depth v2 [3], KITTI [1], Make3D [82] and Unreal [34] are selected as benchmarks. Specifically, the NYU depth v2 dataset was captured in indoor environments, the KITTI [1] and Make3D dataset [82] were collected in real-world outdoor scenes, while the UnrealDataset [34] was gathered from simulated outdoor surroundings. We first evaluate MobileXNet with different loss functions and network configurations on the NYU depth v2 dataset [3]. Then, we compare it with state-of-the-art methods on four benchmarks.

4.4.1 NYU Depth Dataset

The NYU depth v2 dataset [3] consists of about 240k RGB and depth image pairs captured from 464 different indoor scenes through a Microsoft Kinect camera. In this section, we train the designed method on about 48k synchronized RGB and depth images pairs, and test it on 654 images. Both training and testing data are released by [81]. Following [2,76,81], we first downsample the original images from 640×480 pixels to half size, and then center crop a 304×228 pixel region as input to the network.

Table 4.1: Evaluation of loss functions (Rows 1-4) and dilation rates (Rows 4-8) on the NYU depth v2 dataset [3]. $[1, 2, 3]^*$ means depthwise separable convolutions with dilations. The red and **bold** values indicate the best results.

Row	Method	Loss	Rate	RMSE	Abs REL	δ_1	δ_2	δ_3	t_{GPU}
1	MobileXNet	$\mathcal{L}_1$	[1,2,3]	0.558	0.158	0.784	0.945	0.984	8.1 ms
2	MobileXNet	$\mathcal{L}_2$	[1,2,3]	0.553	0.158	0.777	0.948	0.986	7.8 ms
3	MobileXNet	berHu	[1,2,3]	0.550	0.160	0.779	0.947	0.987	8.9 ms
4	MobileXNet	Hybrid	[1,2,3]	0.537	**0.146**	**0.799**	**0.951**	**0.988**	7.9 ms
5	MobileXNet	Hybrid	[1,1,1]	0.552	0.156	0.785	0.946	0.985	6.8 ms
6	MobileXNet	Hybrid	[2,2,2]	0.545	0.156	0.794	0.947	0.987	9.0 ms
7	MobileXNet	Hybrid	[3,3,3]	**0.533**	0.148	0.797	**0.951**	0.987	8.9 ms
8	MobileXNet	Hybrid	$[1, 2, 3]^*$	0.552	0.162	0.786	0.942	0.984	**6.2 ms**

Lower is better | Higher is better | $\delta_1 : \delta < 1.25$, $\delta_2 : \delta < 1.25^2$, $\delta_3 : \delta < 1.25^3$

Evaluation of Loss Functions

In this subsection, we perform a set of experiments over different loss functions based on the designed MobileXNet. Four loss functions described in Section 4.2.2 are tested, and the results are listed in the first 4 rows of Table 4.1. It can be observed that both the berHu loss and the L_2 loss outperform the L_1 loss. In addition, the berHu loss performs slightly better than the L_2 loss. It should be noted that the designed hybrid loss which combines the regular L_1 loss with the image gradient-based L_1 loss generates the best performance. Therefore, we use

the *hybrid* loss function as the default loss in this chapter.

Evaluation of Dilation Rate

In this subsection, we evaluate the performance of different dilation rate configurations in the bridge modules. The dilation rates in the bridge modules are set as [1, 1, 1], [2, 2, 2], [3, 3, 3] and [1, 2, 3], while [1, 1, 1] means the bridge modules have three regular convolutional layers. The results produced from these 4 configurations are listed in Rows 4 to 7 of Table 4.1.

As can be seen, (1) the dilated convolutions improve the performance in both accuracy and error metrics. We attribute this to the fact that it has a larger receptive field and captures more context information. It should be noted that we do not further increase the dilation rate, as we did not obtain empirical improvement when dilation rate was larger than 3; (2) the configuration of [3, 3, 3] produces the best RMSE value, while the values of Abs REL, δ_1, δ_3 and running time are inferior to [1, 2, 3]. It demonstrates that using multiple dilated convolutional layers with different dilation rates to capture the multiple-scale context information is helpful in improving accuracy. Besides, the MobileXNet using the bridge modules with dilation rates of [1, 2, 3] runs faster than [3, 3, 3] by 1ms. This suggests that [1, 2, 3] achieves better accuracy and speed balance; (3) we further apply depthwise separable convolutions with dilation rates of 1, 2 and 3 to the bridge modules (row 8, Table 4.1). According to the 4th and 8th rows of Table 4.1, the depthwise separable convolution lowers the running time by 1.7ms but the accuracy also decreased. However, the MobileXNet using the regular convolutions with dilation rates of [1, 2, 3] generates the best performance and adequate speed for real-time application. Therefore, we choose it as the default configuration.

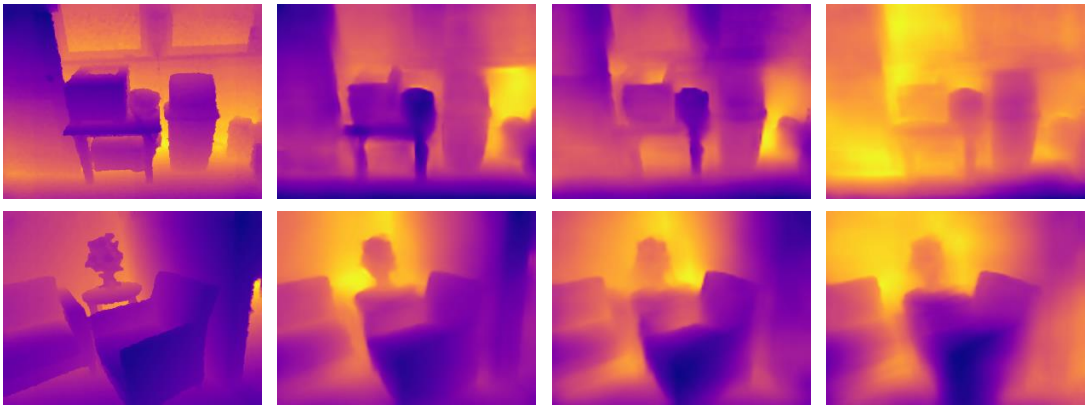


Figure 4.3: Qualitative results of MobileXNet with different weight initialization and convolution type in the encoder of the second sub-network. (a) Ground-truth, (b) Conv-IG, (c) DWConv-IG, and (d) DWConv. Color represents depth (yellow is far, blue is close).

Effect of Weight Initialization

To evaluate the effect of weight initialization in the backbone (the **encoder** of the **first** subnetwork) of MobileXNet, we initialized it with the pre-trained MobileNet model (row 1, Table 4.2) and Gaussian distribution (row 2, Table 4.2) respectively. Furthermore, we design a variant of MobileXNet by replacing the depthwise separable convolutional layers in the backbone with regular convolutional layers (row 3, Table 4.2). Since the ImageNet dataset pre-trained weights of the regular convolutional layers are not available, we initialize them from the Gaussian distribution.

As can be observed from the *2nd* and *3rd* rows of Table 4.2, when initialized from the Gaussian distribution, the variant with regular convolutional layers (row 3, Table 4.2) outperforms the Gaussian distribution initialized MobileXNet (row 2, Table 4.2) in RMSE, Abs REL, δ_1, δ_2 and δ_3, while the running speed is inferior. We attribute this to the fact that the regular convolution has more parameters than the depthwise separable convolution. However, when initializing

Table 4.2: Evaluation of the weight initialization and convolution types in the encoder of the first sub-network (Rows 1-3) and the second sub-network (Rows 4-6) on the NYU depth v2 dataset [3]. DwConv indicates the depthwise separable convolutional layers initialized from the pre-trained weights, DwConv-IG represents the depthwise separable convolutional layers initialized from the Gaussian distribution, and Conv-IG means the regular convolutional layers initialized from the Gaussian distribution. The red and **bold** values indicate the best results.

Row	Method	Convolution	RMSE	Abs REL	δ_1	δ_2	δ_3	t_{GPU}
1	MobileXNet	DwConv	**0.537**	**0.146**	**0.799**	**0.951**	**0.988**	7.9 ms
2	MobileXNet	DwConv-IG	0.669	0.198	0.692	0.909	0.973	8.5 ms
3	MobileXNet	Conv-IG	0.632	0.186	0.724	0.923	0.976	9.3 ms
4	MobileXNet	DwConv	0.822	0.209	0.584	0.862	0.961	**6.8 ms**
5	MobileXNet	DwConv-IG	0.550	0.156	0.788	0.947	0.986	7.3 ms
6	MobileXNet	Conv-IG	**0.537**	**0.146**	**0.799**	**0.951**	**0.988**	7.9 ms

Lower is better | Higher is better | $\delta_1 : \delta < 1.25$, $\delta_2 : \delta < 1.25^2$, $\delta_3 : \delta < 1.25^3$

the backbone from the pre-trained weights (row 1, Table 4.2), the performance of MobileXNet improved significantly. Thus, we choose the depthwise separable convolution in the backbone and initialize it from the pre-trained weights in this study.

With the backbone initialized from the pre-trained MobileNet model, we further design the **encoder** of the **second** subnetwork (*E5-E7* blocks in Figure 4.1(b)) with depthwise separable convolutional layers and initialize it with the pre-trained weights of the corresponding layers of MobileNet [44]. Experimental results are shown in rows 4 to 6 of Table 4.2. Figure 4.3 also displays the qualitative results.

It can be seen from Table 4.2 that when we initialized the encoder of the second subnetwork with the pre-trained weights (row 4, Table 4.2), MobileXNet does not provide good performance. Furthermore, the produced depth maps are blurry and suffer from sharp discontinuities of shapes of objects (see Figure 4.3(d)). We attribute the bad performance to the following facts. In essence, the MobileXNet stacks two encoder-decoder style sub-networks. The encoder and decoder of the

first sub-network are trained to describe input images and generate feature maps with 1/4 size of the input image, respectively. The produced feature maps are passed to the second subnetwork in order to produce depth maps. The second subnetwork works as a learned refining module which operates with downsampling and upsampling steps. This subnetwork tends to be more dependent on the training dataset than the backbone which is utilized to learn generic features [300]. The ImageNet dataset [294] used for pre-training the MobileNet is different from the NYU depth v2 dataset [3]. Hence, initializing the encoder of the second subnetwork impairs the performance of depth estimation.

To further augment our experiments, we then initialize the parameters of the depthwise separable convolutional layers in the encoder of the second subnetwork with a Gaussian distribution (row 5, Table 4.2). As shown in Table 4.2, initializing the encoder of the second subnetwork from the Gaussian distribution outperforms initialization from pre-trained weights. Compared with the original MobileXNet (row 6, Table 4.2), the MobileXNet with the depthwise separable convolution in the encoder of the second subnetwork has inferior performance. In addition, the boundary of small or thin objects in the produced depth maps (see Figure 4.3(c)) are unclear. Therefore, we use regular convolutional layers in the encoder of the second subnetwork and initialize it from the Gaussian distribution.

Table 4.3: Comparison of the proposed MobileXNet against different variants and U-Net [4] on the NYU depth v2 dataset [3]. △ represents the first 9 layers of MobileNet, □ denotes the first 7 layers of MobileNet, ♢ means the network does not use a CNN designed for image classification. M and G indicate $\times 10^6$ and $\times 10^9$ respectively. The red and **bold** values indicate the best results.

Row	Method	Backbone	Parameters	Flops	Memory	RMSE	Abs REL	δ_1	δ_2	δ_3	t_{GPU}
1	MobileXNet	△	24.95 M	9.78 G	111.12 MB	**0.537**	**0.146**	**0.799**	0.951	**0.988**	7.9 ms
2	MobileXNet-Small	□	6.51 M	8.34 G	104.61 MB	0.606	0.180	0.733	0.930	0.981	**6.1 ms**
3	MobileXNet-Large	MobileNet	91.07 M	11.85 G	121.17 MB	0.538	0.148	**0.799**	**0.952**	0.987	15.2 ms
4	ShuffleXNet	ShuffleNetV2	5.20 M	1.73 G	**41.93 MB**	0.580	0.163	0.766	0.943	0.985	6.9 ms
5	EfficientXNet	EfficientNet-B0	**1.96 M**	**0.39 G**	74.19 MB	0.575	0.162	0.766	0.946	0.985	8.2 ms
6	U-Net [4] (Bilinear)	♢	17.27 M	42.22 G	401.05 MB	0.705	0.206	0.661	0.900	0.972	14.1 ms
7	U-Net [4] (DeConv)	♢	31.04 M	48.55 G	426.32 MB	0.726	0.212	0.644	0.892	0.971	19.8 ms

Lower is better | Higher is better | $\delta_1 : \delta < 1.25$, $\delta_2 : \delta < 1.25^2$, $\delta_3 : \delta < 1.25^3$

Evaluation of Network Architectures

To validate the effectiveness of the MobileXNet, we compare it with different variants. In addition to the MobileXNet-Small and MobileXNet-Large, we build two variants based on the ShuffleNetV2 [47] and EfficientNet [48] backbones. The corresponding variants are named as ShuffleXNet[1] and EfficientXNet[2] respectively. The standard encoder-decoder network, U-Net, is used as the baseline. Unlike [4], the U-Net includes batch normalization in this work. All methods were trained with the hybrid loss. The experimental results are shown in Table 4.3.

It can be observed that: (1) among all methods, MobileXNet generates the best error and accuracy metric results and its running time is only inferior to the fastest method by 1.8ms; (2) regarding the RMSE, Abs REL, δ_1, δ_2 and δ_3 metrics, the performance of MobileXNet-Large is very close to MobileXNet. However, due to its deep network architecture, the running time is almost doubled; (3) MobileXNet-Small yields the fastest speed, while its error and accuracy metric results are inferior to its counterparts except the U-Net [4]; (4) the network parameters, flops and memory footprint of ShuffleXNet are about 4×, 5× and 2× fewer than MobileXNet, but it is only faster than MobileXNet by 1ms. Moreover, the performance of ShuffleXNet with respect to the error and accuracy metrics is not comparable to MobileXNet; (5) EfficientXNet is optimal in terms of network parameters and flops, while its running time is longer than MobileXNet. This should be attributed to the fact that EfficientXNet is built on top of EfficientNet [48], which is optimized for parameter and flops efficiency. Furthermore, EfficientNet scales the feature map resolution and depth at the same time, this

[1]The conv1 to stage 3 layers of ShuffleNetV2-1x [47] are used as backbone
[2]The stage 1 to stage 6 layers of EfficientNet-B0 [48] are used as backbone

leads to the slow GPU inference time [301]; and (6) the performance of U-Net [4] is inferior, no matter which upsampling method is used. Besides, it is the slowest especially when using transpose convolution (DeConv) for upsampling. It should be noted that the weights of U-Net are initialized from a Gaussian distribution.

MobileXNet generates the best performance in terms of error and accuracy metrics and achieves real-time speed about 126 fps on an Nvidia GTX 1080 GPU, which is adequate for autonomous driving and robotic applications. In this context, we choose MobileXNet as the network in this chapter.

Effect of Data Augmentation

We employ data augmentation to increase the diversity of the training data to enable the trained network to have a better depth estimation performance. We train our MobileXNet on the original training data and the training data with online data augmentation to analyze the benefit of data augmentation. The standard 654 testing images are used as testing data. Experimental results are listed in Table 4.4. It can be observed that data augmentation improves the MDE performance, especially the RMSE, Abs REL and δ_1 metrics. Therefore, we use data augmentation in this chapter.

Table 4.4: Evaluation of the benefit of data augmentation.

Row	MobileXNet	RMSE	Abs REL	δ_1	δ_2	δ_3
1	Original	0.555	0.158	0.782	0.946	0.985
2	Augmented	**0.537**	**0.146**	**0.799**	**0.951**	**0.988**
		Lower is better		Higher is better		

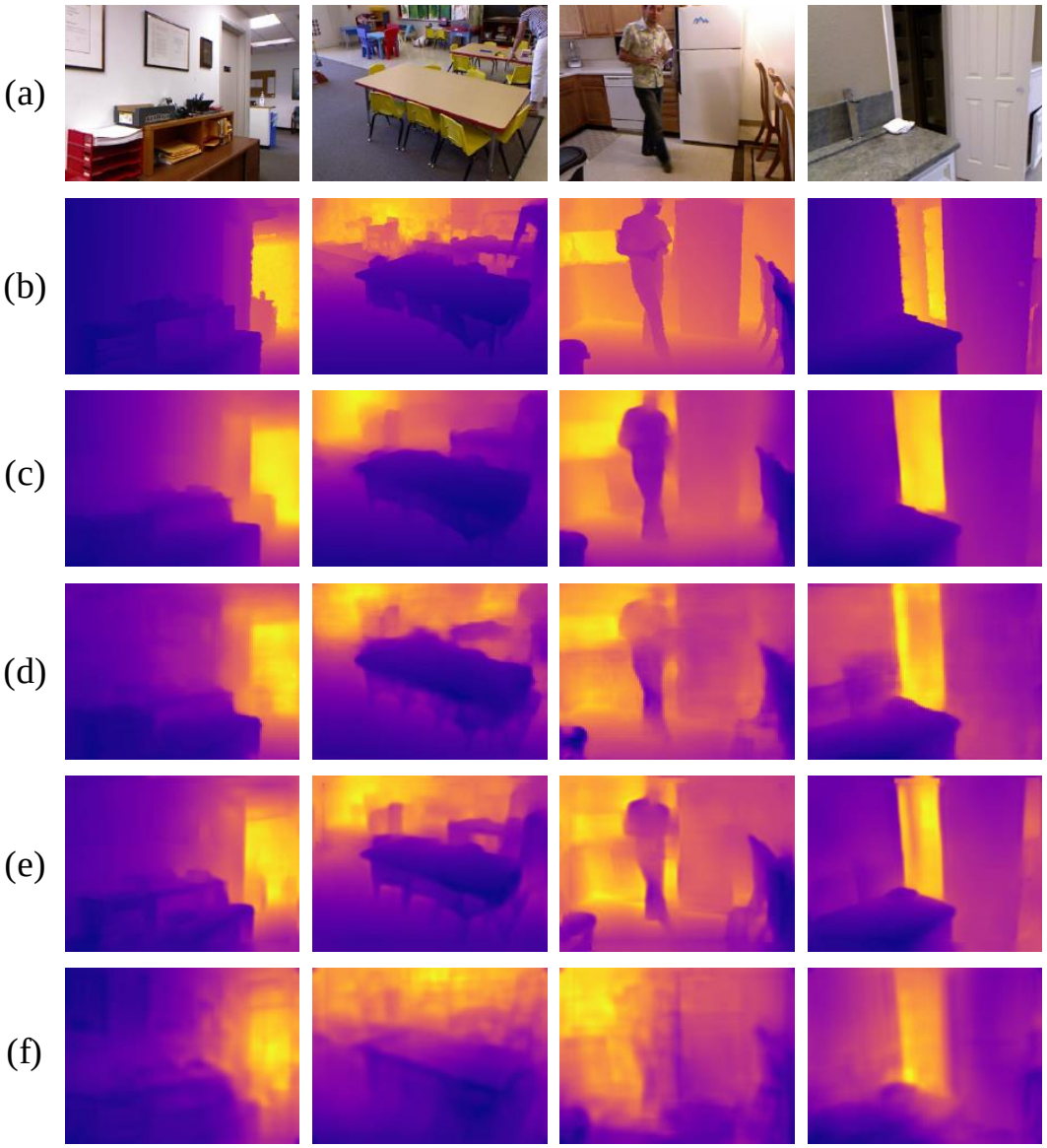


Figure 4.4: Qualitative comparison on the NYU depth v2 dataset. (a) RGB, (b) Ground-truth, (c) Our results, (d) Wofk et al. [81] (Final), (e) Eigen and Fergus [75], and (f) Eigen et al. [2] (Fine). Color represents depth (yellow is far, blue is close).

Table 4.5: Comparison of performances on the NYU depth v2 dataset [3]. ♢ means the network does not use CNN designed for image classification, △ represents the first 9 layers of MobileNet. The red and **bold** values indicate the best results.

Row	Method	Backbone	RMSE	Abs REL	δ_1	δ_2	δ_3	t_{GPU}	Device
1	Eigen et al. [2]	♢	0.907	0.215	0.611	0.887	0.971	N/A	N/A
2	Eigen and Fergus [75]	VGG-16	0.641	0.158	0.769	0.950	0.988	N/A	N/A
3	Liu et al. [83]	VGG-16	0.759	0.213	0.650	0.906	0.976	N/A	N/A
4	Laina et al. [76] (UpConv)	ResNet-50	0.604	0.132	0.789	0.946	0.986	78 ms	Nvidia GTX Titan
5	Laina et al. [76] (UpProj)	ResNet-50	0.573	0.127	0.811	0.953	0.988	55 ms	Nvidia GTX Titan
6	Cao et al. [77]	ResNet-152	0.540	0.141	**0.819**	**0.965**	**0.992**	N/A	N/A
7	Li et al. [79]	ResNet-50	0.601	0.147	0.808	0.957	0.985	N/A	N/A
8	He et al. [302]	VGG-16	0.572	0.151	0.789	0.948	0.986	N/A	N/A
9	Xu et al. [289]	ResNet-50	0.593	**0.125**	0.806	0.952	0.986	150	Nvidia Titan-X
10	Wofk et al. [81] (Original)	MobileNet	0.579	0.164	0.772	0.938	0.982	5.0 ms	Nvidia GTX 1080
11	Wofk et al. [81] (Final)	MobileNet	0.604	0.165	0.771	0.937	0.980	4.0 ms	Nvidia GTX 1080
12	Yang et al. [57]	ResNet-18	0.628	0.199	0.708	0.916	0.975	10 ms	Nvidia Titan-X
13	Hambarde and Murala [303]	♢	0.543	0.160	0.773	0.959	0.989	N/A	N/A
14	Spek et al. [91]	ERFNet	0.687	0.190	0.704	0.917	0.977	**3.2 ms**	Nvidia GTX 1080Ti
15	MobileXNet (Ours)	△	**0.537**	0.146	0.799	0.951	0.988	7.9 ms	Nvidia GTX 1080

Lower is better	Higher is better	$\delta_1 : \delta < 1.25$, $\delta_2 : \delta < 1.25^2$, $\delta_3 : \delta < 1.25^3$

Comparison with the State-of-the-Art

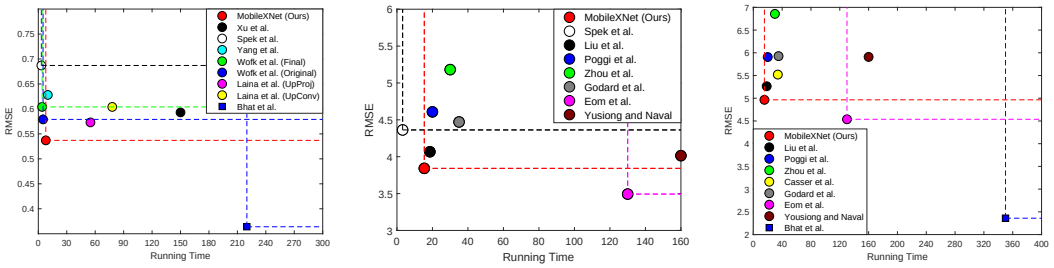
In this subsection, we compare MobileXNet with state-of-the-art methods [2, 57, 75–77, 79, 81, 83, 91, 289, 302, 303]. Since we focus on depth estimation from single RGB images, methods that fuse sparse depth data are not considered in this study. The results of [2, 57, 75–77, 79, 83, 91, 289, 302, 303] are reported in respective literatures. While Wofk et al. [81] only report the values of RMSE and δ_1, we run their released code and models on our desktop to get the values of Abs REL, δ_2, δ_3 and the running time on the Nvidia GTX 1080 GPU. Table 4.5 reports all experimental results.

As can be seen, MobileXNet performs much better than Spek et al. [91] in both error and accuracy metrics, and it generates the best RMSE result. Regarding Abs REL, δ_1, δ_2 and δ_3 values, MobileXNet also outperforms [57, 75, 81, 83, 302, 303]. Moreover, the Abs REL, δ_1, δ_2 and δ_3 values of MobileXNet are on par with [76, 77, 79, 289]. Compared with these four methods, MobileXNet has a shallow and simple architecture, which consists of less layers. Specifically, MobileXNet uses the first 9 layers of MobileXNet as backbone, which is much shallower than the ResNet-50 and ResNet-152 utilized by [76, 77, 79, 289]. Regarding the decoder network, our upsample layer consists of a 3×3 convolutional layer and bilinear interpolation, while Laina et al. [76] (UpProj) utilizes a relatively complex Up-projection method. Additionly, [79] and [289] integrate multi-scale side output feature maps and [77] uses CRF as post-processing operation.

It can be observed from Table 4.5 that Spek et al. [91] yields the fastest running speed, while it performs worse than most of the counterparts in terms of the accuracy and error metrics. Our MobileXNet runs slightly slower than Wofk et al. [81], which is tested at 224×224 sized images. It should be noted that the MobileXNet runs on images having 228×304 pixels. The running speed of

MobileXNet is apparently faster (7.9ms vs 150ms) than Xu et al. [289], which was tested on a Titan-X GPU. While the running time is obtained with different GPUs, the Titan-X GPU has 3584 CUDA cores and 12GB memory, which is much stronger than our 1080 GPU (2580 CUDA cores and 8GB memory). With the same sized images, the speed of MobileXNet is about 7× to 10× faster than [76]. It should be noted that [76] was evaluated on an Nvidia GTX Titan GPU which has 3072 CUDA cores and 12GB memory.

The non-dominated algorithms over running time and RMSE are shown in Figure 4.5(a). As can be seen, MobileXNet dominates Laina et al. [76] (UpConv), Laina et al. [76] (UpProj), Xu et al. [289] and Yang et al. [57]. Besides, MobileXNet, Wofk et al. [81] (Original), Wofk et al. [81] (Final), Spek et al. [91] and Bhat et al. [93] lie on the Pareto Front because none of them is dominated by another. Bhat et al. [93] generates better RMSE performance (0.364), but it runs about 27× slower than MobileXNet (7.9ms vs 220ms) on a single Nvidia GTX 1080 GPU. Moreover, whilst MobileXNet is slower than [81] (Original), [81] (Final) and Spek et al. [91], its accuracy is better than them. More importantly, MobileXNet runs about 126 fps on a less powerful GPU, which is adequate for the real-time application of autonomous driving and robotics. Considering the trade-off between accuracy and speed, MobileXNet is the best compromise solution. In Figure 4.4, we provide qualitative comparison of the proposed method with [2, 75, 81]. It is clearly observed that the results of [2, 75, 81] are blurry. By contrast, our method recovers more details and the predicted depth maps are much clearer than its counterparts.



(a) (b) (c)

Figure 4.5: Pareto Optimality on the NYU depth v2 dataset and the KITTI dataset. (a) The NYU depth v2 dataset, (b) The KITTI dataset (range caps of 50m), and (c) The KITTI dataset (range caps of 80m).

4.4.2 KITTI Dataset

The KITTI dataset [1] consists of outdoor scene images with a resolution of 375×1241 pixels. This dataset has sparse depth images captured by a Velodyne LiDAR. We utilize the same split as Eigen et al. [2], where only left images, which includes 22600 training images and 697 testing images are used. To generate the ground-truth depth maps, we projected corresponding Velodyne data points to the left image plane. The missed depth values in the ground-truth depth maps are ignored both in training and testing. As the LiDAR provides no measurements in the upper part of the images, only the bottom 228×912 pixels region is used in this chapter.

In order to compare with state-of-the-art methods [2,77,79,83,85,86,89,91,140, 194,290,304,305], we evaluate our method with the depth ranging from 0m to 80m and 0m to 50m. Table 4.6 shows the results of our method together with baselines. For the depth ranging from 0m to 80m, MobileXNet achieves the best Abs REL and δ_3 results, while the value of RMSE is comparable to [77,304]. Moreover, the δ_1 and δ_2 of MobileXNet are only slightly inferior to Cao et al. [77]. It should be noted that [77] is built on top of an extremely deep CNN, ResNet-152, and uses fully connected CRF for post-processing. In addition, Eom et al. [304] adopts

a two-stream encoder in order to learn features from RGB images and optical flow. However, MobileXNet does not exploit any multi-stream architecture or post-possessing operation, and has much less layers than [77]. Furthermore, our method outperforms [2, 79, 85, 86, 89, 140, 194, 290, 305]. For the depth range of 0 to 50 meters, MobileXNet produced the best performance on Abs REL, δ_2, and δ_3, and the second best performance of δ_1. It demonstrates that MobileXNet works better for close-range depth.

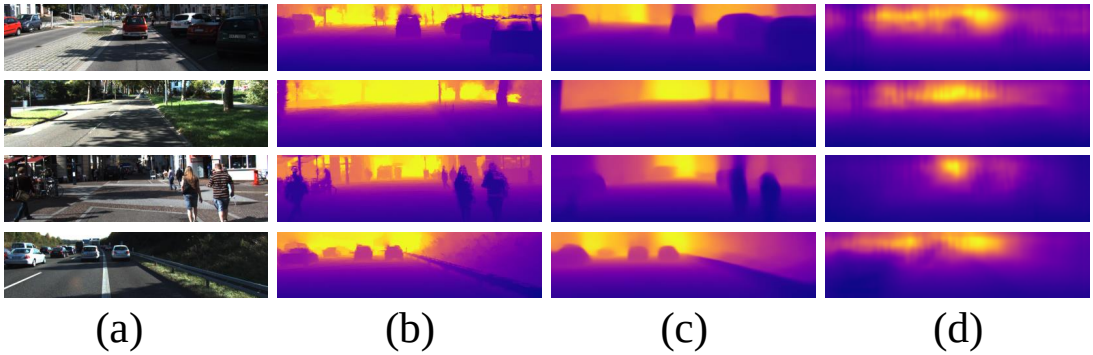


Figure 4.6: Qualitative comparison on the KITTI dataset. (a) RGB, (b) Ground-truth, (c) Our results, and (d) Eigen et al. [2] (Fine). The ground-truth has been interpolated for visualization. Color represents depth (yellow is far, blue is close).

In terms of running speed, our MobileXNet only needs 15.4ms to process a 228×912 sized image on an Nvidia GTX 1080 GPU, which is faster than its counterparts such as [85, 86, 89, 140, 194, 304, 305]. It is worth noting that [85, 86, 89, 140, 194, 304, 305] were tested on GPUs which are much stronger than ours. Specifically, the Titan-X GPU has 3584 CUDA cores and 12GB memory, the GTX 1080Ti GPU has 3584 CUDA cores and 11 GB memory, while our GTX 1080 GPU only has 2560 CUDA cores and 8GB memory.

Table 4.6: Comparison of performances on the KITTI Eigen-split [2]. ♢ means the network does not use CNN designed for image classification, △ represents the first 9 layers of MobileNet. The red and **bold** values indicate the best results.

Row	Method	Cap	Backbone	RMSE	Abs REL	δ_1	δ_2	δ_3	t_{GPU}	Device
1	Eigen et al. [2]	80 m	♢	7.156	0.190	0.692	0.899	0.967	N/A	N/A
2	Liu et al. [83]	80 m	VGG-16	6.986	0.217	0.647	0.882	0.961	N/A	N/A
3	Cao et al. [77]	80 m	ResNet-152	4.712	0.115	**0.887**	**0.963**	0.982	N/A	N/A
4	Godard et al. [85]	80 m	VGG-16	5.927	0.148	0.803	0.922	0.964	35 ms	Nvidia Titan-X
5	Zhou et al. [86]	80 m	♢	6.856	0.208	0.678	0.885	0.957	30 ms	Nvidia Titan-X
6	Li et al. [79]	80 m	ResNet-50	5.325	0.128	0.833	0.956	**0.985**	N/A	N/A
7	Poggi et al. [89]	80 m	♢	5.907	0.146	0.801	0.926	0.967	20 ms	Nvidia Titan-X
8	Casser et al. [140]	80 m	♢	5.521	0.142	0.820	0.942	0.976	34 ms	Nvidia GTX 1080Ti
9	Yusiong and Naval [305]	80 m	♢	5.909	0.145	0.824	0.936	0.970	160 ms	Nvidia GTX 1080Ti
10	Eom et al. [304]	80 m	♢	**4.537**	0.117	0.865	0.958	0.983	130 ms	Nvidia Titan-X
11	Hambarde and Murala [303]	80 m	♢	5.285	0.142	0.797	0.932	0.975	N/A	N/A
12	Liu et al. [194]	80 m	♢	5.264	0.141	0.825	0.941	0.976	18.57 ms	Nvidia GTX 1080Ti
13	Ye et al. [290]	80 m	ResNet-101	4.978	0.112	0.842	0.947	0.973	N/A	N/A
14	MobileXNet (Ours)	80 m	△	4.965	**0.103**	0.873	0.959	**0.985**	**15.4 ms**	Nvidia GTX 1080
15	Cao et al. [77]	50 m	ResNet-152	3.605	0.107	**0.898**	**0.966**	0.984	N/A	N/A
16	Godard et al. [85]	50 m	VGG-16	4.471	0.140	0.818	0.931	0.969	35 ms	Nvidia Titan-X
17	Zhou et al. [86]	50 m	♢	5.181	0.201	0.696	0.900	0.966	30 ms	Nvidia Titan-X
18	Poggi et al. [89]	50 m	♢	4.608	0.145	0.813	0.934	0.972	20 ms	Nvidia Titan-X
19	Yusiong and Naval [305]	50 m	♢	4.014	0.122	0.864	0.953	0.978	160 ms	Nvidia GTX 1080Ti
20	Eom et al. [304]	50 m	♢	**3.493**	0.113	0.877	0.963	0.985	130 ms	Nvidia Titan-X
21	Liu et al. [194]	50 m	♢	4.067	0.135	0.838	0.947	0.978	18.57 ms	Nvidia GTX 1080Ti
22	Spek et al. [91]	50 m	ERFNet	4.363	0.156	0.818	0.940	0.977	**3.2 ms**	Nvidia GTX 1080Ti
23	MobileXNet (Ours)	50 m	△	3.842	**0.098**	0.886	**0.966**	**0.987**	15.4 ms	Nvidia GTX 1080

Lower is better	Higher is better	$\delta_1 : \delta < 1.25$, $\delta_2 : \delta < 1.25^2$, $\delta_3 : \delta < 1.25^3$

We use non-dominance to compare the running time and RMSE of the proposed MobileXNet, [85, 86, 89, 91, 93, 140, 194, 304, 305]. Bhat et al. [56] requires 350ms to produce a depth map on a single Nvidia GTX 1080 GPU, and its RMSE value is 2.360. It can be observed from Figure 4.5(c) that MobileXNet, Eom et al. [304] and Bhat et al. [93] lie on the Pareto Front in the depth range between 0m and 80m. However, the MobileXNet is more than 8× faster than Eom et al. [304], even though [304] was tested on a more powerful Titan-X GPU. While Bhat et al. [93] produces better RMSE performance, its running speed (2 fps) is not adequate for real-time autonomous driving and robotic applications. As can be observed from Table 4.6, MobileXNet also outperforms Eom et al. [304] in Abs REL, δ_1, δ_2, and δ_3 for depth range from 0m to 80m as well as 0m to 50m. According to the non-dominated set and Pareto front distribution, MobileXNet is the best solution among these methods.

For the depth range of 0 to 50m, MobileXNet and Spek et al. [91] are none dominated by each other. According to Table 4.6, the MobileXNet performs much better than Spek et al. [91] with respect to the error and accuracy metrics. Moreover, the MobileXNet achieves a real-time speed of 65 fps at a larger resolution (228 × 912 pixels). Therefore, MobileXNet is the best solution among these methods. Qualitative results are shown in Figure 4.6. As can be observed, our method generates much clearer depth maps than [2].

According to [85, 306], the depth measurements from the LiDAR may be influenced by many factors such as rotation of the LiDAR and incorrect depth readings resulting from the occlusion at object boundaries. To better validate the performance of our proposed MobileXNet, we applied the same split of images with more accurate ground-truth labels provided by the KITTI official [306]. The annotated depth maps are generated by filtering LiDAR points with a computed

Table 4.7: Comparison of performances on 93.5% of the KITTI Eigen-split with accurate ground-truth labels released by the KITTI evaluation benchmark. K denotes the KITTI dataset, CS refers to the Cityscapes dataset, CS + K represents training on the Cityscapes dataset then fine-tuning on the KITTI dataset. △ represents the first 9 layers of MobileNet. The red and **bold** values indicate the best results.

Row	Method	Cap	Backbone	Dataset	RMSE	Abs REL	δ_1	δ_2	δ_3
1	Amiri et al. [154]	80 m	ResNet-50	K	**3.995**	0.096	0.892	0.972	0.992
2	Godard et al. [85]	80 m	ResNet-50	CS+K	4.279	0.097	0.898	0.973	0.991
3	Aleotti et al. [88]	80 m	VGG-16	CS+K	4.236	0.096	0.899	0.974	0.992
4	MobileXNet (Ours)	80 m	△	K	4.128	**0.087**	**0.905**	**0.976**	**0.993**
5	Godard et al. [85]	50 m	ResNet-50	CS+K	4.100	0.095	0.896	0.975	0.992
6	Aleotti et al. [88]	50 m	VGG-16	CS+K	4.110	0.094	0.897	0.976	0.993
7	MobileXNet (Ours)	50 m	△	K	**3.081**	**0.083**	**0.916**	**0.981**	**0.994**

Lower is better | Higher is better | $\delta_1 : \delta < 1.25$, $\delta_2 : \delta < 1.25^2$, $\delta_3 : \delta < 1.25^3$

disparity map from the Semi-Global Matching algorithm [307] to remove outliers from the raw measurements. It is worth noting that the annotated depth maps are not available for 315 training and 45 testing images of the original Eigen split, thus, the number of training and testing images are 22 285 and 652 respectively. The missed values in the annotated depth maps are ignored during both training and evaluation.

We compare our method with [85, 88, 154]. The results of [85] and [88] are reported in [88]. Quantitative results are listed in Table 4.7. As shown in rows 1 to 4 of Table 4.7, for the cap of 0-80m, MobileXNet outperforms all baselines in terms of Abs REL, δ_1, δ_2 and δ_3, while the RMSE is slightly inferior to [154]. It should be noted that [154] applies ResNet-50 as encoder, which is much deeper than the network of MobileXNet. Since [154] only reports results in the range of 0-80m, thus, we compare MobileXNet with [85] and [88] for the cap of 0-50m. It can be seen from rows 5 to 7 of Table 4.7, with a simple and shallow network architecture the proposed MobileXNet outperforms [85] and [88] in all metrics.

4.4.3 Make 3D Dataset

In this section, we evaluate MobileXNet on the Make3D dataset [82], consisting of 400 training and 134 testing images. The RGB images have a resolution of 2272×1704 pixels, and the corresponding ground-truth depth map is 55×305 sized. Following [126] all images are downsampled to 568×426 pixels. In this section, we train our network on the 460×345 pixels center cropped region. To compare with the state-of-the-art, we evaluate the trained model with *C1* (depth range from 0m to 70m) and *C2* (depth range from 0m to 80m) criterions as introduced in [72].

Table 4.8: Qualitative results on the Make3D dataset. The red and **bold** values indicate the best results.

Method	C1 error			C2 error		
	Abs REL	log10	RMSE	Abs REL	log10	RMSE
Liu et al. [72]	0.335	0.137	9.49	0.338	0.134	12.60
Karsch et al. [73]	0.355	0.127	9.20	0.361	0.148	15.10
Li et al. [122]	0.278	0.092	7.12	0.279	0.102	10.27
Liu et al. [83]	0.287	0.109	7.36	0.287	0.122	14.09
Roy and Todorovic [308]	N/A	N/A	N/A	0.260	0.119	12.40
Godard et al. [85]	0.443	0.143	8.860	N/A	N/A	N/A
Kuznietsov et al. [87]	0.421	0.190	8.24	N/A	N/A	N/A
Fu et al. [126]	0.236	**0.082**	7.02	0.238	**0.087**	10.01
Zhao et al. [157]	0.403	N/A	10.424	N/A	N/A	N/A
MobileXNet (Ours)	**0.229**	0.086	**6.807**	**0.233**	0.089	**8.020**
			Lower is better			

The previous methods [72,73,83,85,87,122,126,157,308] are used as baselines, all results are shown in Table 4.8. As can be observed, MobileXNet outperforms [72, 73, 83, 85, 87, 122, 157, 308] in all metrics. As for [126], our method generates better Abs REL and RMSE results, while the log10 value is slightly inferior. It is worth noting that [126] applies a deep CNN as backbone and combines a scene understanding module. Furthermore, our MobileXNet only needs 0.012s to compute a depth map on a single Nvidia GTX 1080 GPU. For Liu et al. [83],

1.1s is required to infer depth map from a test image. However, the time for computing superpixels is not included. Qualitative results from this dataset are shown in Figure 4.7.

4.4.4 UnrealDataset

To demonstrate the ability of predicting depth maps from small sized images, we evaluate the proposed MobileXNet on the UnrealDataset [34]. This dataset contains over 100K images collected in a series of simulated urban and forest scenarios, and the ground-truth depth up to 40 meters. Compared with the aforementioned datasets [1, 3, 82], the RGB and depth images in the UnrealDataset have a much smaller size, 160×256 pixels. The Unreal Engine and AirSim softwares simulate an MAV flying in the simulated environments. The RGB and depth image pairs are collected from the MAV's frontal camera using a plugin. We first evaluate the proposed method on original images, and compare it with [4, 34, 81, 119]. Considering the limitation of GPU memory, we choose the ResNet50-DeConv3[3] combination of [119]. We train [81,119] with the official released source code and default parameters. As in section 4.4.1, we train the PyTorch implemented U-Net (Bilinear) with the hybrid loss function.

From rows 1 to 5 of Table 4.9, we show the quantitative comparison between MobileXNet and baselines on the original images. Following [34], we only report the RMSE and running time of J-MOD2. As can be observed, MobileXNet outperforms [81,119] in all metrics. It performs best in Abs REL, δ_3 and speed, while the RMSE value of MobileXNet is slightly inferior to J-MOD2. J-MOD2 is a multi-task learning method, which jointly learns object detection

[3]The fully convolutional part of ResNet-50 is used as the encoder and the deconvolution with 3×3 kernel is applied for upsampling.

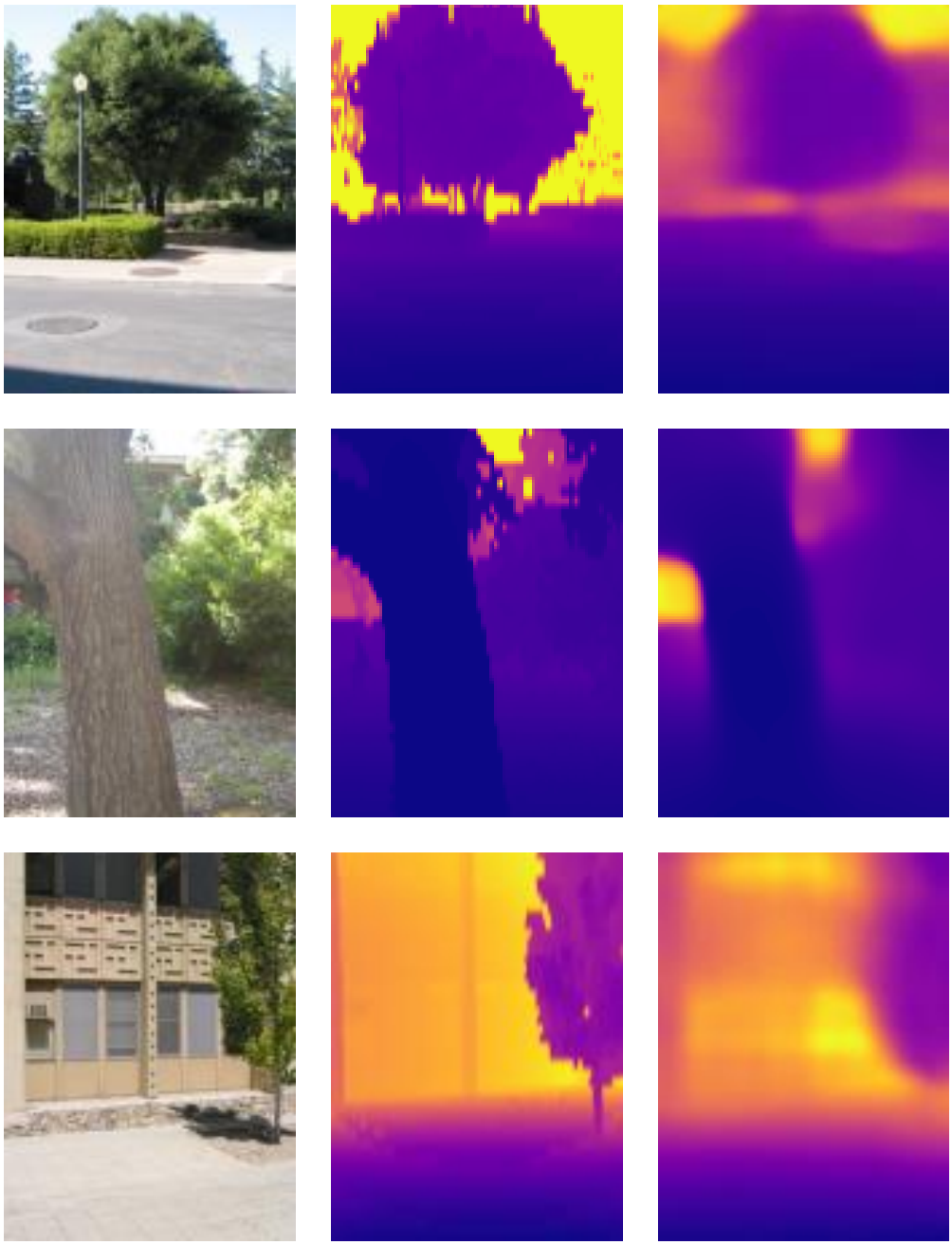


Figure 4.7: Quantitative results on the Make3D dataset. (a) RGB, (b) Ground-truth, and (c) Our results. Color represents depth (yellow is far, blue is close).

and depth estimation. The object detection branch informs the depth estimation branch with object structures, which improves the accuracy of depth estimation. However, our method only works out MDE. According to Figure 4.9(a), both MobileXNet and J-MOD2 are non-dominated solutions. The running time of J-MOD2 is longer than MobileXNet, even though J-MOD2 was evaluated on a more powerful GPU. The threshold metric of MobileXNet is on par with U-Net [4]. It is worth noting that images from the UnrealDataset are much smaller than the NYU depth v2 dataset [3]. As both U-Net and MobileXNet downsample the feature maps to 1/16 scale of the input images, the produced feature maps include more spatial information than [81,119]. In addition, U-Net [4] is twice as wide as our MobileXNet, which enables U-Net to learn more information. Some qualitative results of our method and baselines are shown in Figure 4.8, which further demonstrates the superior performance of MobileXNet.

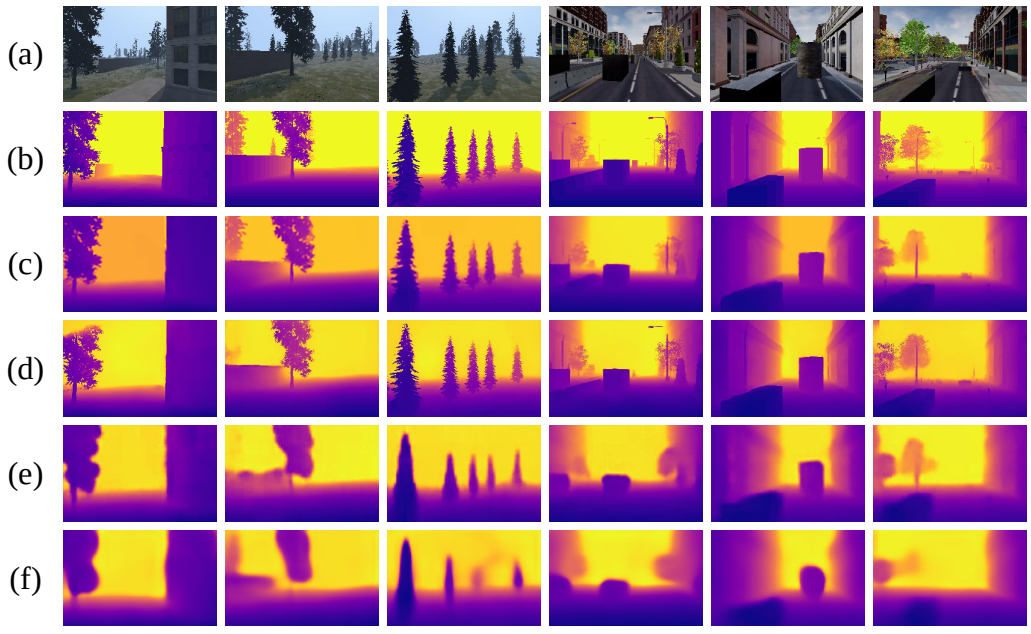
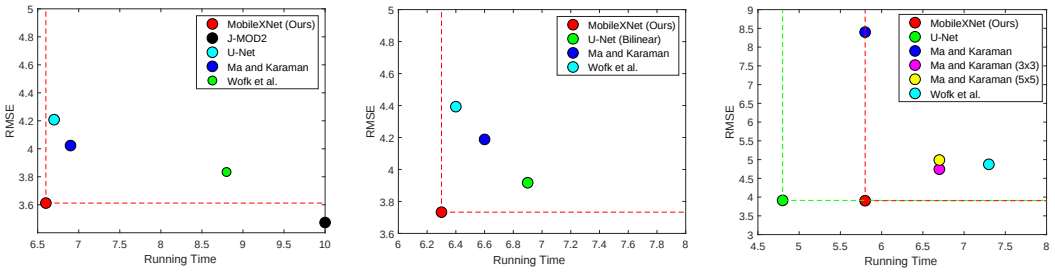


Figure 4.8: Qualitative comparison on the UnrealDataset (original). (a) RGB, (b) Ground-truth, (c) Our results, (d) U-Net [4], (e) Ma and Karaman [119], and (f) Wofk et al. [81] (Original). Color represents depth (yellow is far, blue is close).

Table 4.9: Comparison of performances on the UnrealDataset. ♢ means the network does not use CNN designed for image classification, △ represents the first 9 layers of MobileNet. Ma and Karaman (3×3) and Ma and Karaman (5×5) denote the use of 3×3 and 5×5 sized filters in the input layer respectively. The red and **bold** values indicate the best results.

Row	Method	Backbone	Input Size	RMSE	Abs REL	δ_1	δ_2	$\delta 3$	t_{GPU}	Device
1	J-MOD2 [34]	VGG-19	160×256	**3.473**	N/A	N/A	N/A	N/A	10 ms	Nvidia Titan-X
2	U-Net [4]	♢	160×256	3.834	0.130	**0.884**	**0.957**	**0.977**	8.8 ms	Nvidia GTX 1080
3	Ma and Karaman [119]	ResNet-50	160×256	4.023	0.126	0.870	0.951	0.974	6.9 ms	Nvidia GTX 1080
4	Wofk et al. [81]	MobileNet	160×256	4.208	0.155	0.860	0.945	0.972	6.7 ms	Nvidia GTX 1080
5	MobileXNet (Ours)	△	160×256	3.612	**0.124**	0.878	0.954	**0.977**	**6.6 ms**	Nvidia GTX 1080
6	U-Net [4]	♢	128×204	3.917	0.134	**0.879**	**0.956**	0.969	6.9 ms	Nvidia GTX 1080
7	Ma and Karaman [119]	ResNet-50	128×204	4.188	0.138	0.856	0.946	0.973	6.6 ms	Nvidia GTX 1080
8	Wofk et al. [81]	MobileNet	128×204	4.393	0.157	0.852	0.940	0.976	6.4 ms	Nvidia GTX 1080
9	MobileXNet (Ours)	△	128×204	**3.733**	**0.133**	0.871	0.954	**0.977**	**6.3 ms**	Nvidia GTX 1080
10	U-Net [4]	♢	80×128	3.912	**0.135**	**0.873**	**0.956**	**0.977**	**4.8 ms**	Nvidia GTX 1080
11	Ma and Karaman [119]	ResNet-50	80×128	8.399	0.444	0.511	0.681	0.868	5.8 ms	Nvidia GTX 1080
12	Ma and Karaman (5×5) [119]	ResNet-50	80×128	4.988	0.178	0.798	0.923	0.962	6.7 ms	Nvidia GTX 1080
13	Ma and Karaman (3×3) [119]	ResNet-50	80×128	4.744	0.174	0.804	0.930	0.967	6.7 ms	Nvidia GTX 1080
14	Wofk et al. [81]	MobileNet	80×128	4.874	0.166	0.814	0.928	0.967	7.3 ms	Nvidia GTX 1080
15	MobileXNet (Ours)	△	80×128	**3.904**	0.140	0.853	0.948	0.975	5.8 ms	Nvidia GTX 1080

Lower is better | Higher is better | $\delta_1 : \delta < 1.25$, $\delta_2 : \delta < 1.25^2$, $\delta_3 : \delta < 1.25^3$



(a) (b) (c)

Figure 4.9: Pareto Optimality on the UnrealDataset. (a) The original images (160×256), (b) The resized images (128×204), and (c) The resized images (80×128).

To further validate the ability of MobileXNet to process small sized images, we downsample the original images to 128×204 and 80×128 pixels. We compare our method with [4,81,119]. The rows 6 to 9 and rows 10 to 15 of Table 4.9 show results on 128×204 and 80×128 sized images respectively. It can be observed that reducing the image size decreases the accuracy of depth estimation. It should be noted that when images are downsampled to 80×128 pixels, the performance of Ma and Karaman [119] dropped significantly. Ma and Karaman [119] use the fully convolutional part of ResNet-50 as encoder. Besides the inherent characteristic of the encoder network, the 7×7 sized filters in the input layer may have influence on the performance of depth estimation. For small sized images, large filters could capture global information, while the detailed information is missed. Unlike image classification, depth estimation is a pixel-wise application, both global and detailed information are important. Our hypothesis is that using smaller sized filters in the input layer will help to improve the performance of Ma and Karaman [119]. As a result, we modified [119] by reducing the filter kernel size in the input layer from 7 to 5 and 3 respectively.

It can be observed from rows 11 to 13 of Table 4.9, both 5×5 and 3×3 sized filter-based input layers improve the performance of Ma and Karaman

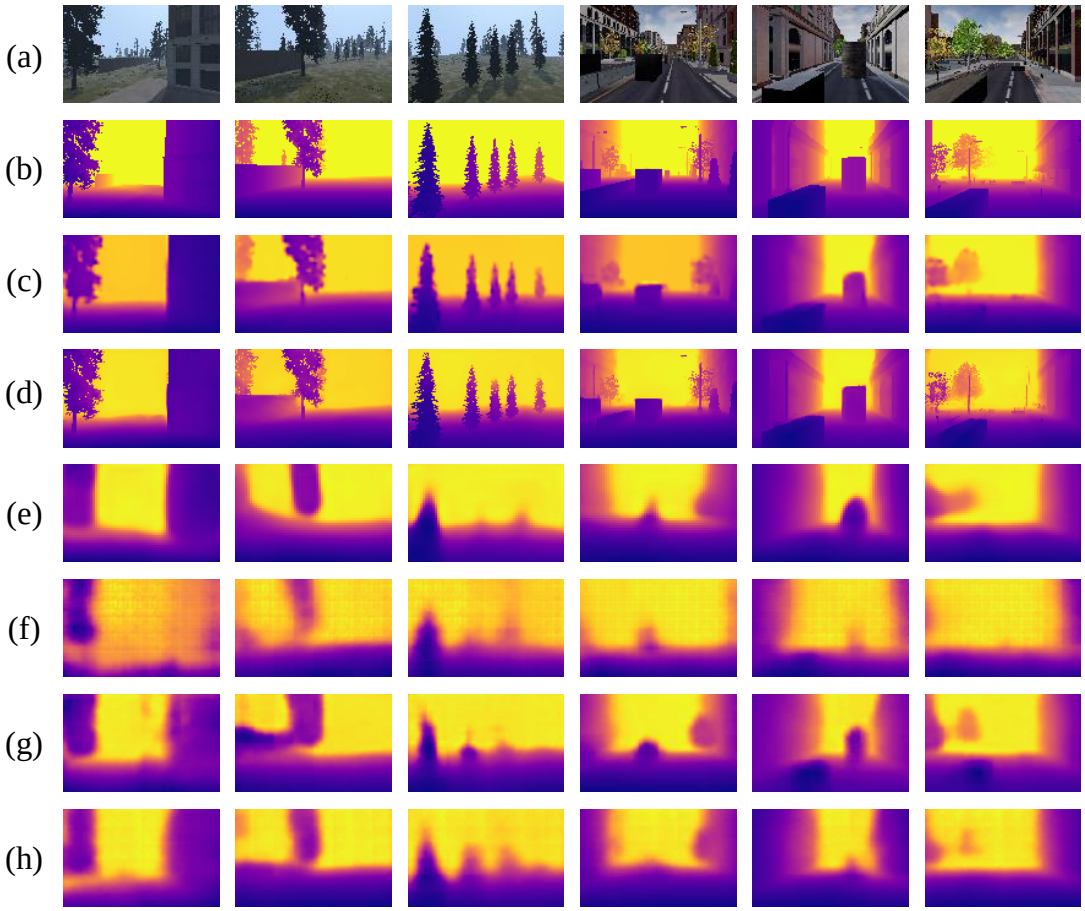


Figure 4.10: Qualitative comparison on the UnrealDataset (80×128). (a) RGB, (b) Ground-truth, (c) Our results, (d) U-Net [4], (e) Wofk et al. [81] (Original), (f) Ma and Karaman [119], (g) Ma and Karaman [119] with 5×5 sized filters in the input layer, and (h) Ma and Karaman [119] with 3×3 sized filters in the input layer. Color represents depth (yellow is far, blue is close).

[119]. We present a qualitative comparison between our method and baselines in Figure 4.10. It is clear that the modified Ma and Karaman [119] with small sized filter-based input layers performs better than the original. Thus, we can conclude that the input layer with small sized filters is helpful in predicting depth from small sized images. It is worth noting that MobileXNet outperforms the modified Ma and Karaman [119] in all metrics and generates much clearer depth maps, although the latter has a very deep encoder network. Moreover, according to

Figure 4.9(b) and Figure 4.9(c), the proposed MobileXNet is the non-dominated solution in 128×204 and 80×128 sized images. This demonstrates the superior performance of MobileXNet on small sized images.

4.5 Chapter Summary

In this chapter, we paid attention to the trade-off between the accuracy and speed of MDE. To this end, we introduced a novel and real-time CNN architecture, namely, MobileXNet. Specifically, we assembled two shallow and simple encoder-decoder subnetworks back-to-back in a unified framework, which avoids a greater number of successive downsamplings. We also designed a hybrid loss function by employing an additional spatial derivative-based L_1 loss function together with the regular L_1 function.

Extensive experiments on four datasets demonstrated that the proposed method achieved promising performance and run much faster than state-of-the-art methods, which is critical for real-time autonomous driving and robotic applications.

Chapter 5

Lightweight Monocular Depth Estimation with an Edge Guided Network

The work, reported in this chapter, has partially been published in the following conference paper [309]:

Xingshuai Dong, Matthew A Garratt, Sreenatha G Anavatti, Hussein A Abbass, and Junyu Dong. Lightweight monocular depth estimation with an edge guided network. In 2022 17th International Conference on Control, Automation, Robotics and Vision (ICARCV), pages 204-210, 2022.

5.1 Introduction

This chapter introduces a novel lightweight monocular depth estimation (MDE) network. Recently developed methods [76, 80, 121, 125, 310] learn deeper (stack more layers) and wider (include more filters in each layer) models that produce

better depth estimation performance. However, these methods focused more on improving depth estimation accuracy than achieving real-time running speed. This makes them difficult to run in edge platforms where reaction time is crucial for operating safety such as for obstacle avoidance in small sized autonomous robots.

In order to solve the problem of achieving real-time speed on embedded platforms, lightweight convolutional neural networks (CNNs) such as ERFNet [175] and MobileNets [44, 46] have been employed to design lightweight MDE networks [81, 91]. These methods achieved real-time speed on embedded platforms, but their accuracies were inferior to the state-of-the-art methods. Additionally, both [91] and [81] applied encoder-decoder style network architectures. The successive downsampling operations in the encoder network distort fine details in lower resolution layers, which can result in blurry boundaries or neglect small objects in the produced depth maps. To avoid the loss of spatial information, MobileXNet [288] stacked two relatively shallow encoder-decoder style subnetworks back-to-back in a unified framework.

It should be noted that all above discussed approaches did not utilize the low-level features in input images, such as edge features which define the boundaries of object or regions within an image. Recent interpretability studies on MDE [232, 233] demonstrated that the edges in input images are an important cue for CNNs to predict depth. However, to the best of our knowledge, existing MDE methods normally do not exploit edge features in their network.

This chapter aims to address the above described problems. To be specific, this chapter introduces a novel lightweight MDE network which adopts edge attention features to guide the task of depth estimation. We integrate the depth estimation branch with an edge guidance branch in a unified network, named

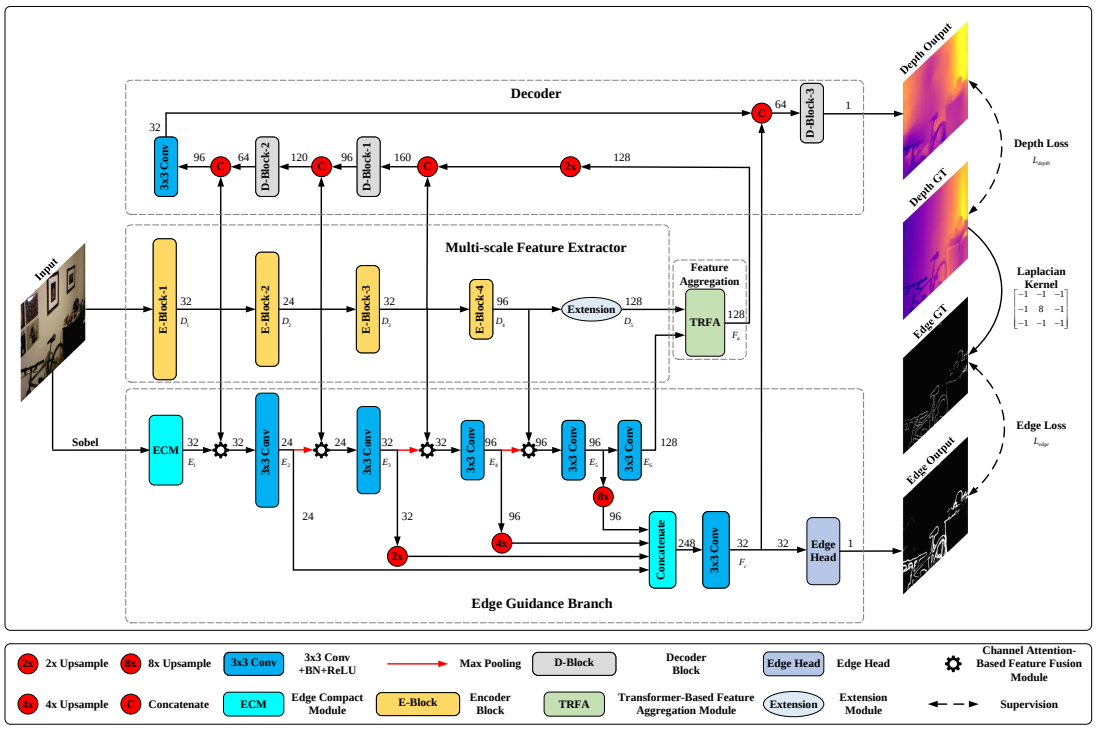


Figure 5.1: Illustration of our proposed EGD-Net.

Edge Guided Depth Estimation Network (EGD-Net). EGD-Net is built on top of a shallow (stacks less layers) and narrow (includes less filters in each layer) encoder-decoder network and applies edge attention features to guide the depth estimation task. The contributions of this chapter are summarized as follows: (1) We propose a novel lightweight MDE network, named EGD-Net, which employs edge attention features to guide the depth estimation task; (2) We design a channel attention-based feature fusion module; (3) We design a transformer-based feature aggregation module, which captures the long-range dependencies between the edge attention features and the context information; (4) We demonstrate the effectiveness of our designed method through extensive experiments.

5.2 Methodology

This section introduces our proposed lightweight MDE network, EGD-Net. Figure 5.1 illustrates the architecture of EGD-Net, which is composed of four parts: multi-scale feature extractor, edge guidance branch, feature aggregation module and decoder.

5.2.1 Multi-scale Feature Extractor

The multi-scale feature extractor (MSFE) consists of a backbone (E-Block-1, 2, 3, 4) and an extension module. We adopt a lightweight CNN, MobileNetV2 [46] as the backbone. To adapt MobileNetV2 to the MDE task, we remove the layers after the fourth convolutional stage. Thus, the final features from the backbone are 1/16 size of the input image. To further enhance the representation ability of produced features, we add an extension module after the backbone. The extension module is comprised of six dilated Inverted Residual Blocks (IRBs) and each IRB has different dilation rate to capture multiple scales context information. In particular, the dilation rates are set as 1, 2, 3, 1, 2 and 3. Furthermore, the expansion factor and stride values in IRB are set as 4 and 1 respectively. For convenience, we define the output feature maps from the MSFE as D_1, D_2, D_3, D_4, D_5, with strides of $2^1, 2^2, 2^3, 2^4, 2^4$, respectively.

5.2.2 Edge Guidance Branch

Edge information is an important cue for CNNs to predict depth [232, 233]. To model the edge attention features for guiding the depth estimation task, we design an edge guidance branch (EGB) which is composed of a stack of convolutional layers interleaved with channel attention-based feature fusion (CAFF) modules.

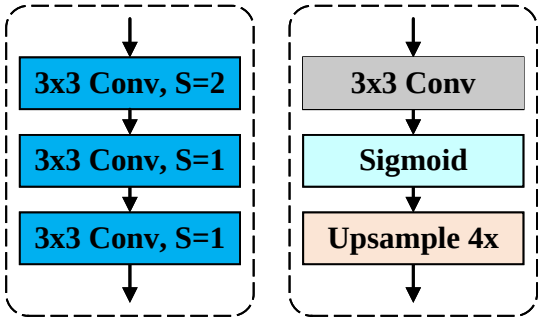


Figure 5.2: Illustration of the edge compact module (a) and the edge head (b). S denotes stride, and "3×3 Conv" represents 3×3 convolutional layer has no batch normalization and ReLU.

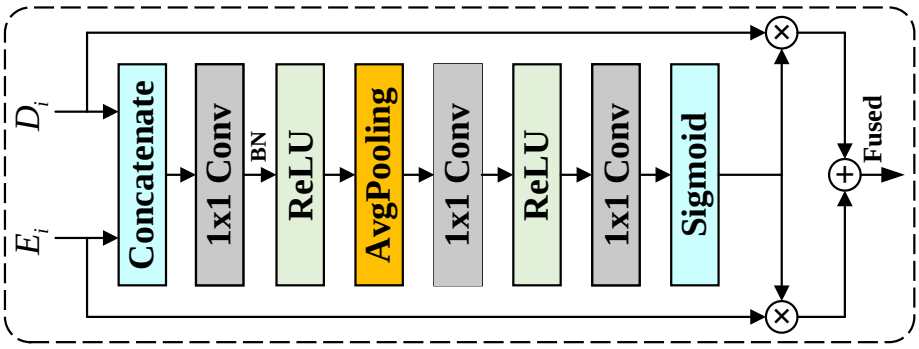


Figure 5.3: Illustration of the CAFF module, where "1×1 Conv" has no batch normalization (BN) and ReLU, "$\times$" and "$+$" represent multiply and add operations respectively.

EGB takes image gradients as well as the intermediate features from the MSFE as input and outputs edge images and high resolution feature maps. We first extract image gradients in x and y directions from the input image with the Sobel operator. Image gradients are processed by an edge compact module (see Figure 5.2(a)) to get features (E_1) at 1/2 size of the input image. The produced features (E_1) are fused with D_1 through the CAFF module.

As shown in Figure 5.3, intermediate feature maps generated by the MSFE (D_i) and EGB (E_i) are concatenated along the channel dimension. The concatenated features are passed through a "1×1 Conv+BN+ReLU" layer to reduce

the channel dimension to half size. Next, we pool the reduced features to a feature vector, which is then processed by 1×1 convolutions, ReLU and Sigmoid operators to get an attention map. The attention map is multiplied with D_i and E_i respectively. These multiplied features are added element-wise to build a fused feature. The fused feature is then convolved with a "3×3 Conv-BN-ReLU" layer. We denote the feature maps generated by the EGB as $E_1, E_2, E_3, E_4, E_5, E_6$, with strides of $2^1, 2^1, 2^2, 2^3, 2^4, 2^4$, respectively.

Feature maps from EGB (E_3, E_4, E_5) are resized to $1/2$ size of the input image and concatenated with E_2 and then pass through a "3×3 Conv-BN-ReLU" layer. As shown in Figure 5.1, the output feature (F_c) from the "3×3 Conv-BN-ReLU" layer is used as the input of two directions. The first direction is the edge head, which produces edge maps for supervision. The second direction is passed to the decoder to act as the edge guidance feature. In addition, E_5 is convolved with a "3×3 Conv-BN-ReLU" layer and generates E_6, which is fed to the TRFA module to aggregate with the context rich feature from the MSFE. In this chapter, the edge detection is modeled as a binary segmentation task. To obtain the ground-truth binary edge images, we adopt the Laplacian operator to extract edge maps from the ground-truth depth maps. The edge guidance branch is directly supervised by the binary edge labels. Therefore, it learns edge attention features.

5.2.3 Transformer-Based Feature Aggregation Module

In order to combine the edge attention features from the EGB and context rich features from the MSFE to produce high resolution depth maps, we design a transformer-based feature aggregation (TRFA) module. TRFA consists of two linear transformer encoder layers [55] and a "1×1 Conv-BN-ReLU" layer. The

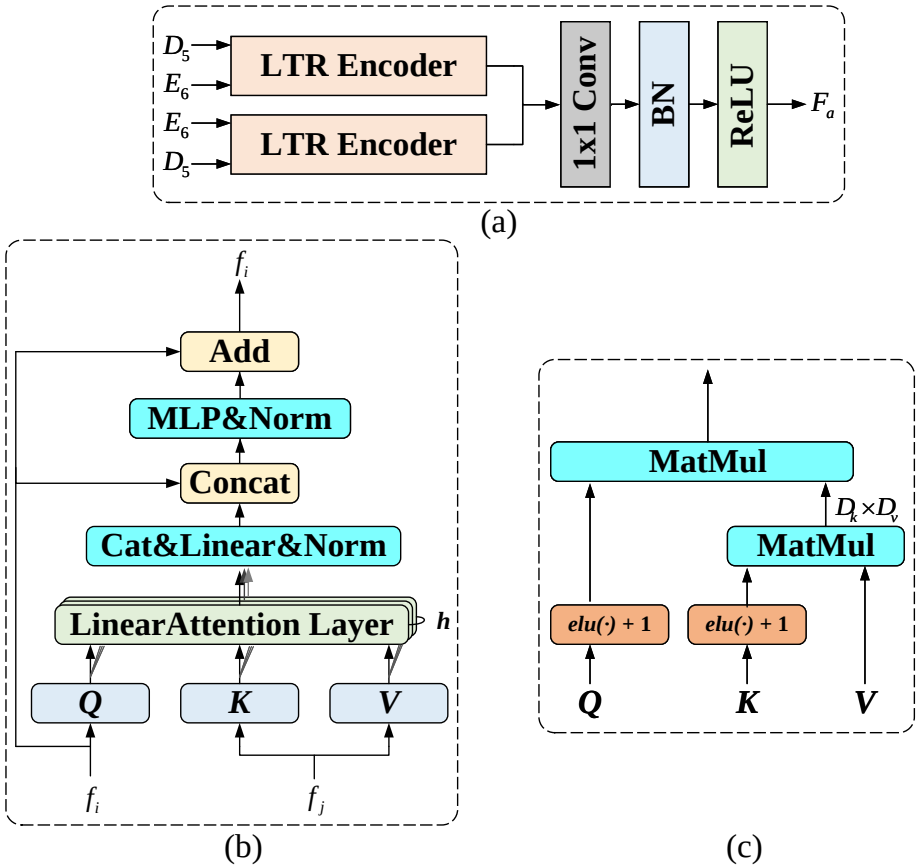


Figure 5.4: Illustration of the TRFA module. (a) The transformer-based feature aggregation (TRFA) module, where LTR represents linear transformer. (b) Transformer encoder layer, h means the multiple heads of attention, which is set as 4 in this study. (c) Linear attention layer.

core element of the linear transformer encoder layer is that the linear attention layer computes the attention between a set of query vectors (Q) and key vectors (K) using dot-product similarity, which is then used to weigh a set of value vectors (V). Thus, the attention computation selects the relevant information through measuring the similarity between the query vector and each key vector.

Inspired by [311], we adopt the linear transformer encoder layer to capture the long-range dependencies (or global context) between the edge and context features through cross-attention in two directions. As shown in Figure 5.4(a), the input features to linear transformer encoders are (D_5, E_6) and (E_6, D_5) respec-

tively. Then, features from the linear transformer encoder layers are concatenated and passed through the "1×1 Conv-BN-ReLU" layer to aggregate them together.

The output features from the TRFA module are first upsampled to two times in size and then fed to the decoder. The decoder is composed of a "3×3 Conv-BN-ReLU" layer and three decoder blocks (D-Block-1, 2, 3), each decoder block includes a "3×3 Conv-BN-ReLU" layer and a bilinear interpolation with a scale factor of 2. The decoder incorporates features (F_a) from the TRFA module, low-level features (D_1, D_2, D_3) from the MSFE, and the high resolution features (F_c) from the EGB to produce depth maps with the same size as input images.

5.2.4 Loss Function

The designed network includes two branches that output depth maps and edge maps respectively. For the task of MDE, we adopt the loss function proposed in [288]. This loss stacks the regular L_1 loss:

$$L_1(d, d^*) = \frac{1}{N} \sum_{i}^{N} |d_i - d_i^*|, \tag{5.1}$$

and the image gradient based L_1 loss:

$$L_{grad}(d, d^*) = \frac{1}{N} \sum_{i}^{N} |\bigtriangledown_x(d_i, d_i^*)| + |\bigtriangledown_y(d_i, d_i^*)|, \tag{5.2}$$

where N is the total number of pixels being considered, d and d^* are the predicted and ground-truth depth, $\bigtriangledown_x$ and $\bigtriangledown_y$ are the spatial derivatives in x and y directions. The depth estimation loss can be written as:

$$L_{depth} = L_1 + L_{grad}. \tag{5.3}$$

For edge detection, we employ the standard Binary Cross Entropy (BCE) loss L_{edge}, which is defined as:

$$L_{edge} = -\sum_{i}(e_i^* log e_i + (1 - e_i^*) log(1 - e_i)), \tag{5.4}$$

where e_i and e_i^* are the detected and ground-truth edges respectively. Finally, the whole loss function is formulated as:

$$L = \lambda_1 L_{depth} + \lambda_2 L_{edge}, \tag{5.5}$$

where λ_1 and λ_2 are the hyper-parameters, we empirically set $\lambda_1 = 1$ and $\lambda_2 = 20$.

5.3 Experiments

To demonstrate the effectiveness of our proposed EGD-Net, we evaluate it on the NYU depth v2 dataset [3].

5.3.1 Implementation Details

The designed network is implemented in PyTorch [283]. A workstation with a single Nvidia RTX 3090 GPU is used for training and testing. The weights of the backbone of the MSFE are initialized with the weights pre-trained on ImageNet. The other layers are randomly initialized. The training is optimized by using the SGD optimizer, and the batch size is set as 8. We train the network for 25 epochs. The poly learning rate policy is adopted, the learning rate for the n^{th} epoch is $init\_lr \times (1 - \frac{n}{max\_epoch})^{power}$, where the $init\_lr$ and power are set as 0.01 and 0.9 respectively.

During training, we employ data augmentation approaches to increase the diversity of training samples. Data augmentations are applied to each RGB and ground-truth depth image pair in an online fashion:

- Random Flips: RGB and ground-truth depth image pairs are horizontally flipped at a probability of 0.5.
- Random Rotation: RGB and ground-truth depth image pairs are randomly rotated by a degree of $r \in [-5, 5]$.
- Color Jitter: the brightness, contrast and saturation values of the RGB images are randomly scaled by a factor of $c \in [0.6, 1.4]$.

5.3.2 Dataset and Evaluation Metric

We evaluate the proposed method on the commonly used NYU depth v2 dataset [3], which was collected in real-world indoor surroundings with a Microsoft Kinect camera. The original images have a resolution of 640×480 pixels. In this chapter, we train our method on the training set proposed by Hu et al. [80] and evaluate it on the offical testing set including 654 RGB and depth image pairs. Each image pair is downsampled to 342×256 and then center cropped to 320×240. We first compare our proposed method with state-of-the-art methods and then perform ablation experiments to validate the contribution of each component of the proposed network.

We adopt three widely used metrics, linear Root Mean Square Error (RMSE), Absolute Relative Difference (Abs REL) and Threshold Accuracy (δ_i, $i = 1, 2, 3$) to evaluate the proposed method. These metrics are defined in Section 2.3.1. Since our aim is to predict depth maps from RGB images, only the error and accuracy metrics of depth estimation are compared.

5.3.3 Comparison with State-of-the-art

In this section, we compare the performance of our EGD-Net with state-of-the-art methods [80, 81, 121, 288, 312, 313] in terms of the number of network parameters (Parameters, million), error (RMSE and Abs REL) and accuracy (δ_1, δ_2 and δ_3) metrics. Quantitative results of our EGD-Net and state-of-the-art methods are listed in Table 5.1. For [80, 121] we report the corresponding results from their papers. The results of [312, 313] are reported in [313]. We retrained [81, 288] with the same training and testing procedure as described in Section 5.3.1. To compare fairly, [81, 288] are supervised by the depth loss described in Section 5.2.4.

It can be observed that: (1) among all methods, EGD-Net has the lowest number of parameters. In particular, EGD-Net has >2.5× fewer parameters than [312, 313], >11× fewer parameters than MobileXNet [288], >71× fewer parameters than Hu et al. [80], and >95× fewer than Chen et al. [121]; (2) with much fewer network parameters, EGD-Net generates the best RMSE performance while its δ_2 and δ_3 metrics are very close to Hu et al. [80] and Chen et al. [121]; (3) EGD-Net outperforms [81, 288, 312, 313] in terms of all error and accuracy metrics.

As regards the running speed, EGD-Net runs at about 96 fps (GPU reference time is 10.4 ms) on an Nvidia GTX 1080 GPU (2580 CUDA cores and 8GB memory), which is adequate for real-time robotic and autonomous driving applications. We present the qualitative comparison of our results with Wofk et al. [81] and MobileXNet [288] in Figure 5.5. It can be observed that our proposed method can detect small objects (e.g., lamp) in the image and predict finely detailed object boundaries while [81, 288] cannot predict clearly.

Table 5.1: Comparison of performances on the NYU depth v2 dataset [3]. ↑ means higher is better, ↓ means lower is better. The red and **bold** values indicate the best results.

Method	Backbone	Parameters ↓	RMSE ↓	Abs REL ↓	δ_1 ↑	δ_2 ↑	δ_3 ↑
Hu et al. [80]	SENet-154	157.0 M	0.530	0.115	0.866	0.975	0.993
Chen et al. [121]	SENet-154	210.3 M	0.514	**0.111**	**0.878**	**0.977**	**0.994**
Wofk et al. [81]	MobileNet	20.67 M	0.529	0.155	0.789	0.950	0.987
Tu et al. [312]	MobileNetV2	5.7 M	0.531	0.147	0.801	0.956	0.989
Rudolph et al. [313]	DDRNet-23-slim	5.8 M	0.501	0.138	0.823	0.961	0.990
MobileXNet [288]	MobileNet	24.95 M	0.507	0.149	0.807	0.953	0.989
Ours	MobileNetV2	**2.21 M**	**0.486**	0.136	0.825	0.960	0.990

Figure 5.5: Qualitative results from the NYU depth v2 dataset. (a) RGB image, (b) Ground-truth depth, (c) Wofk et al. [81], (d) MobileXNet [288] and (e) Our results. Color represents depth (yellow is far, blue is close).

5.3.4 Ablation Studies

To analyze the contribution of each component of the designed network, we perform experiments with different deployments on the NYU Depth v2 dataset [3]. The training and testing strategies are kept the same as Section 5.3.1.

Contribution of Different Components

We setup a baseline network that consists of the multi-scale feature extractor and the decoder shown in Figure 5.1. The baseline method is trained with the depth loss described in Section 5.2.4. The EGB and TRFA module are added to the baseline step by step.

Table 5.2: Ablation study on contribution of different components. "w CAFF" means the EGB includes the CAFF module. "w/o CAFF" indicates the EGB does not include the CAFF module. ↑ means higher is better, ↓ means lower is better. The red and **bold** values indicate the best results.

Method	Parameters ↓	RMSE ↓	Abs REL ↓	δ_1 ↑	δ_2 ↑	δ_3 ↑
Baseline	1.56 M	0.520	0.149	0.796	0.955	0.988
Baseline + EGB	2.01 M	0.514	0.149	0.808	0.959	0.990
Baseline + TRFA	2.19 M	0.506	0.144	0.810	0.955	0.986
Baseline + EGB (w CAFF) + TRFA	2.21 M	**0.486**	**0.136**	**0.825**	**0.960**	**0.990**
Baseline + EGB (w/o CAFF) + TRFA	2.16 M	0.489	0.144	0.817	0.960	0.989

As shown in Table 5.2, the baseline has the lowest number of parameters, while it yields the worst results. When we combine the proposed EGB with the baseline, it outperforms the baseline in terms of RMSE, δ_1, δ_2 and δ_3. To explore the influence of the proposed TRFA module, we append it to the baseline network and train it with the depth loss. With the same training procedure, it outperforms the baseline and the combination of "Baseline + EGB". Finally, our whole EGD-Net (line 4, Table 5.2), with both EGB and TRFA module and trained with the whole loss function, yields the best performance in terms of all metrics. To evaluate the contribution of the proposed CAFF module, we design a variant by replacing CAFF with pixel-wise addition (Baseline + EGB (w/o CAFF) + TRFA). According to the last two lines, the CAFF improves the performance of EGD-Net in term of all error and accuracy metrics.

Comparison of Different Backbones

In this subsection, we investigate the influence of adopting different backbones in the MSFE. We compare MobileNetV2 [46] with three CNNs, ResNet-18 [41], EfficientNet-B0 [48] and ShuffleNetV2 [47]. Specifically, ResNet-18 [41] is a general CNN, ShuffleNetV2 [47] and EfficientNet-B0 [48] are lightweight CNNs. The weights of all backbones are initialized from the pre-trained models on ImageNet. To make the backbones compatible with the fixed feature aggregation module, we fixed the channel dimension of the final feature maps from the MSFE (D_5) and EGB (E_6) to 128. We report the results of four backbones in Table 5.3. As can be observed, when using MobileNetV2 [46] as the backbone the proposed method achieves the best trade-off between accuracy and computation complexity. In particular, it has the lowest number of parameters and yields the best performance in terms of both error and accuracy metrics.

Table 5.3: Comparison of different backbones. ↑ means higher is better, ↓ means lower is better. The red and **bold** values indicate the best results.

Encoder	Parameters ↓	RMSE ↓	Abs REL ↓	δ_1 ↑	δ_2 ↑	δ_3 ↑
ResNet-18 [41]	6.43 M	0.489	0.144	0.818	0.960	0.990
EffcientNet-B0 [48]	2.63 M	0.555	0.161	0.771	0947	0.988
ShuffleNetV2 [47]	3.37 M	0.514	0.155	0.798	0.949	0.986
MobileNetV2 [46]	2.21 M	**0.486**	**0.136**	**0.825**	**0.960**	**0.990**

5.4 Chapter Summary

In this chapter, we introduced a novel lightweight MDE network, named EGD-Net. Specifically, we designed an Edge Guidance Branch to detect edges and produce edge attention features that contain edge information. Moreover, a transformer-based feature aggregation module has been designed to learn the long-range dependencies between the edge and context features and aggregate them together. Extensive experiments on the NYU depth v2 dataset demonstrated the effectiveness of our proposed network.

Chapter 6

Frontier Guided Area Coverage for Unmanned Aerial Vehicles with Deep Reinforcement Learning

The work, reported in this chapter, has partially been included in the following journal manuscript:

Xingshuai Dong, Matthew A Garratt, Sreenatha G Anavatti, and Hussein A Abbass. Frontier guided area coverage for unmanned aerial vehicles with deep reinforcement learning and monocular vision. Under review.

6.1 Introduction

The objective of this chapter is to address the area coverage problem for unmanned aerial vehicles (UAVs). The area coverage problem has been widely

explored in the literature. It is observed that a deep reinforcement learning (DRL) based area coverage method based on pure visual perception for obstacle avoidance is missing.

Motivated by this context, we propose a frontier guided area coverage method that uses a monocular RGB camera and DRL framework. In particular, we design a multi-modal information-based DRL (MMIDRL) framework for the UAV area coverage problem. The proposed framework depends on a minimal vision sensor setup consisting of a monocular RGB camera and applies two convolutional neural networks (CNNs) to predict depth maps and optical flow (OF) maps from monocular RGB images. Then, the predicted depth maps and OF maps are passed to the DRL network to learn policies for guiding the flight of a UAV. Due to the fact that RL requires repeated trial and error learning through interaction with the environment, training the agent in real-world environment may result in unexpected behaviours such as collision. Therefore, we train the DRL agent in a simulated environment.

The core contribution of this chapter is the proposed MMIDRL framework for the UAV area coverage problem. Unlike prior methods that use an RGB-D camera or a LiDAR, our proposed MMIDRL framework only depends on a monocular RGB camera for obstacle avoidance. Furthermore, the proposed MMIDRL framework uses CNNs for MDE and OF estimation to extract depth maps and OF maps from RGB images, and then takes advantage of the features learned from depth maps and OF maps to learn policies for guiding the flight of the UAV. We design a frontier guided area coverage method for the UAV, apply two real-time MDE networks to the designed area coverage method and perform a series of simulation experiments, and demonstrate the effectiveness of the proposed area coverage method through simulation experiments.

The rest of this chapter is organized as follows. Section 6.2 describes our methodology. Section 6.3 depicts the experimental setup. The experimental results are introduced in Section 6.4. Finally, conclusions and future work are presented in Section 6.5.

6.2 Methodology

Our primary objective is to enable a UAV to cover as much of the accessible area in an environment as possible while avoiding obstacles. We formulate this objective as an RL problem, and propose a MMIDRL framework to map the input data to command actions. In this section, we first introduce the proposed frontier guided area coverage method and then we present the proposed MMIDR framework.

6.2.1 Frontier Guided Area Coverage

In this chapter, we assume that the UAV has access to accurate position and velocity information and we obtain these from AirSim [274] [1]. We define a working area we wish to cover within the 3D simulated environment in AirSim [274]. For the purpose of simplicity, we let the UAV fly across a 2D plane, i.e., the UAV flies across the X and Y axes of the 3D space while maintaining nearly constant altitude. This enables us to approximate the working area into a 2D grid map, where each grid cell will be set as an occupancy value of 1 if the cell is explored (covered), or 0 if the cell is unexplored (uncovered). Additionally, we define an extended area (see Figure 6.1) which is still legal for the UAV to fly within but not part of the area needing to be covered. This avoids needlessly terminating

[1] In AirSim, the default flight controller "simple flight" does not simulate a state estimator, so we only have access to the ground-truth state values.

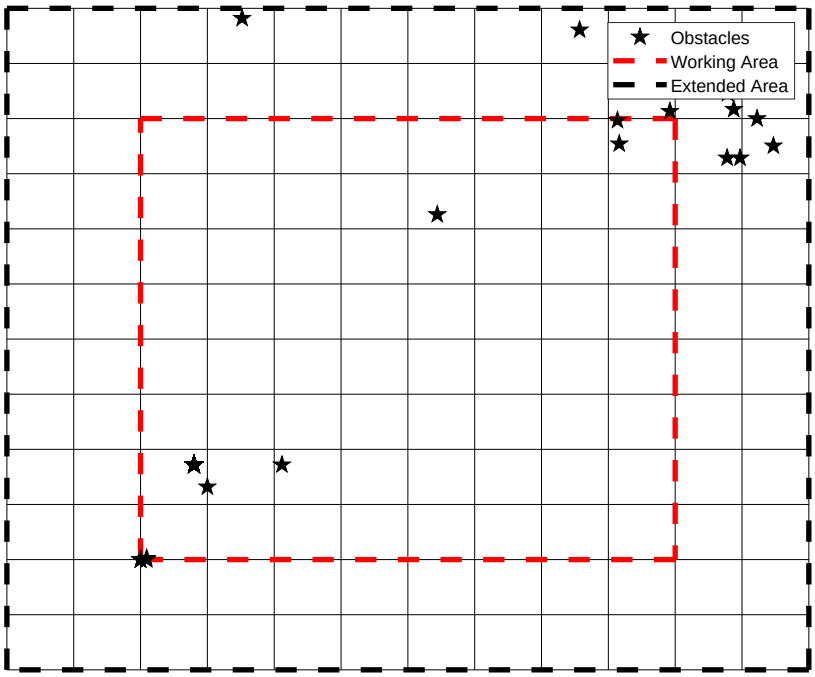


Figure 6.1: Illustration of the 2D grid map with working area and extended area.

the simulation when the UAV moves outside the desired coverage area but is still in a safe space.

Frontiers are regions located at the border between the explored and unexplored space in the environment. Any grid cell between an explored and unexplored cell is considered as a frontier cell. Clusters of connected frontier cells are named as frontier regions. The main idea behind the frontier search-based area coverage method is to discover frontiers and use them as the desired destinations for the UAV. By moving to successive frontiers, a UAV can incrementally visit all accessible areas in an environment.

The frontier search method [264] is described as follows. Given a timestep t, the ith frontier cell is represented as f_t^i. First of all, all valid frontier cells are grouped together to form a list of frontier cells $\Pi_t = [f_t^0, f_t^1, ..., f_t^I]$. Subsequently, the breadth-first search algorithm is applied to each frontier cell in Π_t to convert

the frontier cells and corresponding unexplored cells to a frontier region. The obtained frontier regions are added to a set $\mathcal{F}_t = \{\eta_t^0, \eta_t^1, ..., \eta_t^J\}$ $(J \leq I)$ and then ranked to determine the desired frontier region by minimizing the following utility function:

$$O(\eta_t^j) = \lambda_D * D_t^{\eta_t^j} + \lambda_S * S_t^{\eta_t^j}, \tag{6.1}$$

$$\eta_t^* = \arg\min_{\eta_t^j \in \mathcal{F}_t} \big(O(\eta_t^j)\big), \tag{6.2}$$

where $D_t^{\eta_t^j}$ is the Euclidean distance between the UAV and the closest frontier cell of that frontier region, $S_t^{\eta_t^j}$ is the grid size of that frontier region, and λ_D and λ_S are weighting parameters associated with $D_t^{\eta_t^j}$ and $S_t^{\eta_t^j}$ respectively. It can be observed that Equation (6.1) is minimized when the Euclidean distance is small and the frontier size is large. The determined frontier region is selected as the desired destination. After the desired destination is selected, the relative position between the UAV's current position and the centroid of the desired destination is used as a direction vector for guiding the flight of the UAV.

6.2.2 Deep Reinforcement Learning

Problem Definition

The objective of an area coverage algorithm is to control the robot's trajectory to visit as much of the accessible areas in the environment as possible. This enables a thorough map of the environment to be generated or an exhaustive search to be carried out such that no area is missed out. This task can be formulated as an RL problem, where the agent repeatedly interacts with an environment by performing actions, to collect observations and rewards. The main objective of the trained agent is to explore and cover an unknown environment depending

only on a monocular RGB camera to perceive the environment. Benefiting from our previously developed CNN for MDE [288, 309] and using the MobileFlow adapted from FlowNet [37] CNN for OF estimation, depth maps and OF maps can be predicted directly from the monocular RGB images. The predicted depth maps and OF maps are then passed to the RL agent to learn policies for guiding the flight of the UAV.

Reinforcement Learning Setup

We define a Partially Observable Markov Decision Process (POMDP) for the UAV area coverage problem. The POMDP is represented by a tuple $< \mathcal{O}, \mathcal{A}, \mathcal{R}, \mathcal{S}, \mathcal{P}, \gamma >$, where $\mathcal{O} = \{o\}$ represents the set of observations and referred to as "observation space", $\mathcal{A} = \{a\}$ means the set of actions and referred to as "action space", $\mathcal{R} : s \times a \rightarrow r$ is the reward function that provides feedback to the agent for action selection, $\mathcal{S} = \{s\}$ is the set of states and referred to as "state space", $\mathcal{P} : s \times a \rightarrow s$ is the transition function that models the transitions of states based on selected actions, and $\gamma \in (0, 1)$ indicates the discount factor. At each timestep t, the agent receives the observation (o_t) of the current state (s_t) and selects an action a_t from $\mathcal{A}$. Then, the agent receives a reward $r(s_t, a_t)$ and transitions to a new state s_{t+1} and gets a new observation o_{t+1}. The objective of the agent is to find an optimal policy $\pi(a_t|o_t; \theta_\pi)$ which maximizes the expected sum of discounted rewards:

$$\pi^* = \arg\min_{\theta_\pi} \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t)\right], \tag{6.3}$$

where π^* represents the optimal action, θ_π means the model parameters, t denotes the timestep, and γ represents the discount factor.

The agent is defined as a quadcopter UAV that is free to move across the X and Y axes of a 3D space. In particular, the learned policy (π) consists of eight discrete actions. At the beginning of each episode, a start point is randomly selected from the working environment. The agent starts to fly and visits the reachable areas in the working environment. The episode ends when the agent fulfills the area coverage task, collides with any objects in the environment, flies outside of the defined extended area, or the simulation reaches the maximum number of steps.

A commonly applied model-free policy gradient method is the deep deterministic policy gradient (DDPG) [248], which has been shown to improve the accuracy of position control and tracking. Additionally, the proximal policy optimisation (PPO) [249] algorithm has demonstrated promising performance with control tasks in continuous state-action domains. However, we apply the double deep Q-network (DDQN) [246] to train the agent, as a stochastic policy-based model offers the flexibility of adaptive learning in partially or totally unknown environments [271].

Observation Space and Action Space In this chapter, the observation o_t consists of three parts: RGB images from the onboard front view camera, the relative position, and the agent's current linear velocity. The RGB images are passed through the perception module to predict depth maps and OF maps for training the DRL model. The relative position between the desired destination and the agent's current position serves as a desired direction vector to let the agent learn where it should go to reach the next frontier.

We consider a 2D action space and define eight discrete actions. As illustrated in Figure 6.2, the defined action space consists of linear velocities in eight direc-

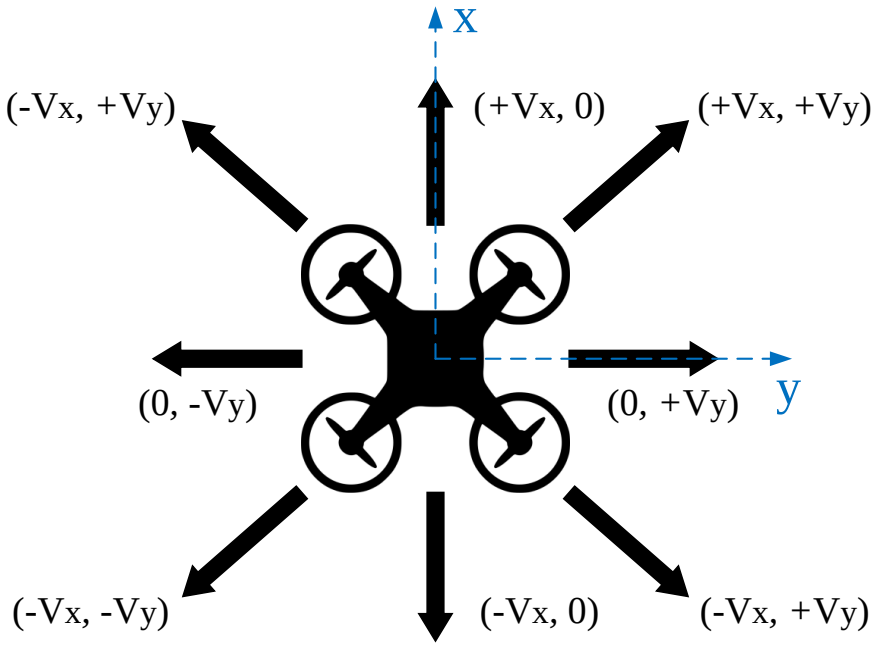


Figure 6.2: Illustration of the AUV reference system and action space.

tions. The values of velocities are set as $1m/s$. At each timestep, the agent selects an action based on observations from the environment. To maintain guidance using the front view camera, we condition the UAV's nose to always align with the direction of travel.

Reward Function A reward function enables the RL agent to learn the preferable actions within the defined working environment. A straightforward formulation of the reward is to give the agent a reward value only if it completes the area coverage task. However, this reward makes initial training difficult as it is hard to get any positive reward at the beginning of training. Therefore, optimization algorithms would take an extremely long time or even fail to converge [314]. To solve this problem, we design a non-sparse reward function for the UAV area coverage problem. The designed reward function is described as follows.

First, we assign a time reward at each timestep until the episode ends:

$$r_{time} = \begin{cases} 0 & \text{on episode termination,} \\ -0.01 & \text{otherwise,} \end{cases} \tag{6.4}$$

where the smaller negative contribution to the total reward encourages the agent to complete the task faster.

Second, if the agent flies outside of the working area but within the extended area, we apply a smaller negative reward r_{valid}:

$$r_{valid} = \begin{cases} -0.05 & \text{if agent flies outside of working area,} \\ 0 & \text{otherwise.} \end{cases} \tag{6.5}$$

Then, when the agent arrives at an uncovered cell, we apply a smaller positive reward r_{visit}:

$$r_{visit} = \begin{cases} 0.1 & \text{if agent arrives an uncovered cell,} \\ -0.2 & \text{if agent arrives an occupied cell,} \\ 0 & \text{otherwise.} \end{cases} \tag{6.6}$$

Besides, in order to encourage the agent to fly to the desired destination, we apply a reward ($r_{destination}$) which relates to the distance between the agent and the desired destination:

$$r_{destination} = \begin{cases} R & \text{if agent flies close to the desired destination,} \\ -R & \text{if agent flies away from the desired destination,} \end{cases} \tag{6.7}$$

where R is based on the distance between the agent and the desired destination

in two consecutive timesteps. $R = C \times D_i$, C is a positive constant value (set as 6) and D_i is the difference of the distance between the agent and the desired destination at two consecutive timesteps.

Finally, we assign the terminal reward at the end of each episode. For a successful episode, for example when the agent completes the area coverage task, we assign a positive terminal reward, while in case of collisions or flight outside of the extended area, the agent receives a penalty as follows:

$$r_{terminal} = \begin{cases} 10 & \text{if agent completes the task,} \\ -10 & \text{if agent collides with an object,} \\ -10 & \text{if agent flies outside of the extended area,} \\ 0 & \text{otherwise.} \end{cases} \tag{6.8}$$

Therefore, the overall reward function is defined as the sum of the above five terms:

$$r = r_{time} + r_{valid} + r_{visit} + r_{destination} + r_{terminal}. \tag{6.9}$$

In addition to the case of completing the defined task, we define three termination cases: 1) the agent collides with any object in the environment; 2) the agent flies outside of the extended area; 3) the current episode timestep T_{step} reaches 1000.

Network Structure

The overview of the proposed MMIDRL framework is shown in Figure 6.3. It consists of three components: simulator module, perception module and policy module. In the simulator module, we use AirSim [274] as the simulator and the Unreal Engine as the render engine. AirSim provides a Python API that can be used to obtain sensor readings such as RGB images, position and velocity of

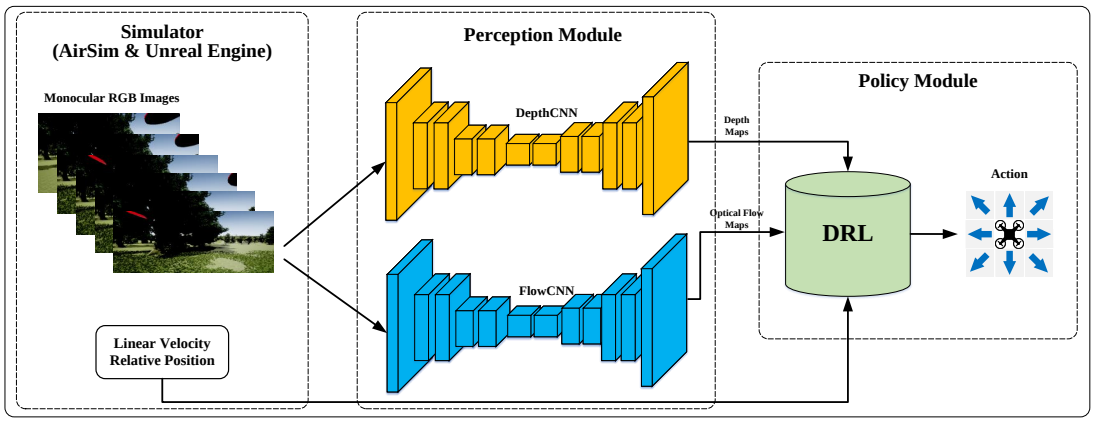


Figure 6.3: Illustration of the proposed MMIDRL framework.

the agent (UAV). The MMIDRL framework takes as input the RGB image from AirSim, the relative position of the UAV with respect to the desired destination[2], and the UAV's current velocity to learn the policy for guiding the flight of the UAV. Details of Unreal Engine and AirSim have been introduced in Sections 3.2 and 3.3 respectively.

The designed MMIDRL framework depends on a minimal vision sensor setup composed of a monocular RGB camera (the onboard front view camera) to capture RGB images from the environment. The captured images are passed to the perception module to predict depth maps and OF maps through two CNNs. The depth map contains the distance between the camera to the surface of objects and the structure of the environment. OF carries rich information about range in monocular images and this is believed to have a lot of additional information as inspired from insects, which use OF for ranging given their visual system is effectively monocular. CNNs use texture and color cues to find depth from monocular images, this can sometimes fail in cases where images are blurry [233].

[2]In this work, the desired destination has two definitions. For the flying to destinations task, the desired destination is the destination point defined at the beginning of each episode. For the area coverage task, the desired destination is the centroid of the selected frontier cell at each timestep.

Meanwhile, OF estimation exploits temporal information (e.g., change between two consecutive frames) and may work in those cases [315]. Therefore, depth maps and OF maps are two complementary visual representations of the environment. In essence, the perception module perceives the environment through MDE and OF estimation. The combination of depth and OF modalities enables a complete and detailed interpretation of the environment [316].

To predict depth maps from monocular RGB images, we apply MobileXNet [288] or EGD-Net [309] as the DepthCNN in the perception module. The details of MobileXNet [288] and EGD-Net [288] have been described in Sections 4 and 5, respectively. In FlowCNN, a deep CNN such as FlowNet [37] is expected to be used. However, FlowNet produces OF maps at 1/4 size of the input images. Since the simulator generates RGB images with the resolution of 128×128, FlowNet produces OF maps with the resolution of 32×32, which are not compatible with the DRL network architecture. Therefore, we design a variant of FlowNet, which has a lightweight encoder-decoder architecture. The designed variant is built on top of MobileNet [44], and we name it MobileFlow.

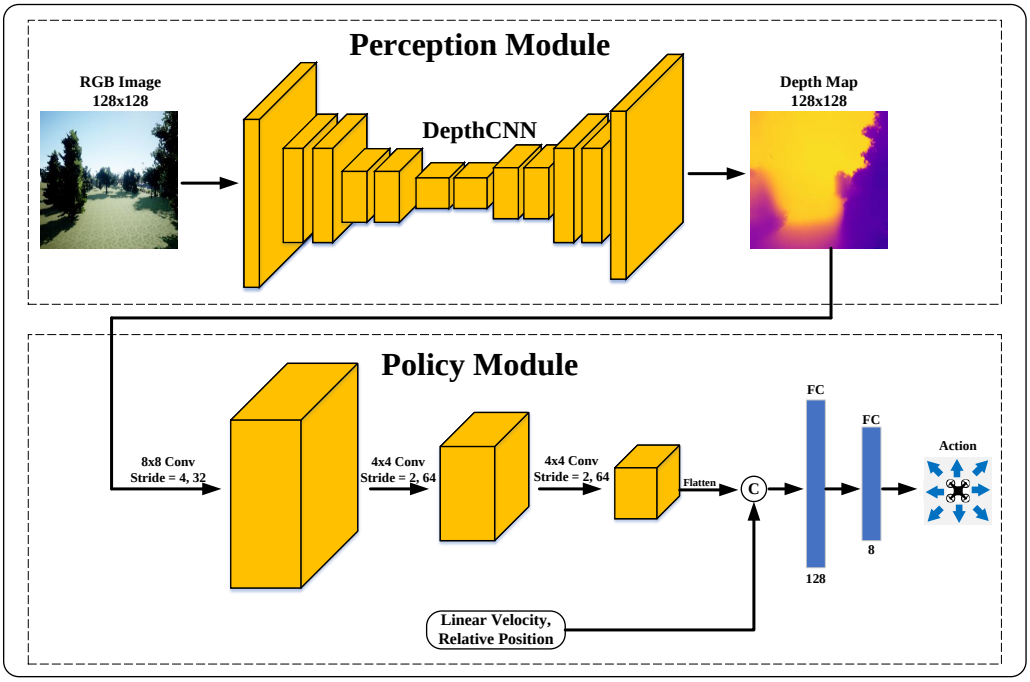
The configuration of the encoder of MobileFlow is illustrated in Figure 6.4. It consists of four convolutional stages. Considering the size of the input images ($128\times$ 128), in the first stage we use 3×3 sized filters to capture more detailed information [288]. To increase the receptive field of the first stage, we stack three 3×3 convolutional layers. This design has the same receptive field as a 7×7 convolutional layer but the number of parameters is less. Stages 2, 3 and 4 are based on depthwise separable convolutions [45] for the purpose of reducing network latency. In particular, these three stages have the same configuration as the corresponding layers in MobileNet [44]. Therefore, we can use the pre-trained weights on the ImageNet dataset [294] to initialize these layers in training. The

	Stage	Layer	Kernel	Padding	Stride	Output Channels
Encoder	Stage 1	Conv1_1	3×3	1	2	32
		Conv1_2	3×3	1	1	32
		Conv1_3	3×3	1	1	32
	Satge2	DWConv2_1	3×3	1	1	32
		Conv2_1	1×1	0	1	64
		DWConv2_2	3×3	1	2	64
		Conv2_2	1×1	0	1	128
	Stage 3	DWConv3_1	3×3	1	1	128
		Conv3_1	1×1	0	1	128
		DWConv3_2	3×3	1	2	128
		Conv3_2	1×1	0	1	256
	Stage 4	DWConv4_1	3×3	1	1	256
		Conv4_1	1×1	0	1	256
		DWConv4_2	3×3	1	2	256
		Conv4_2	1×1	0	1	512

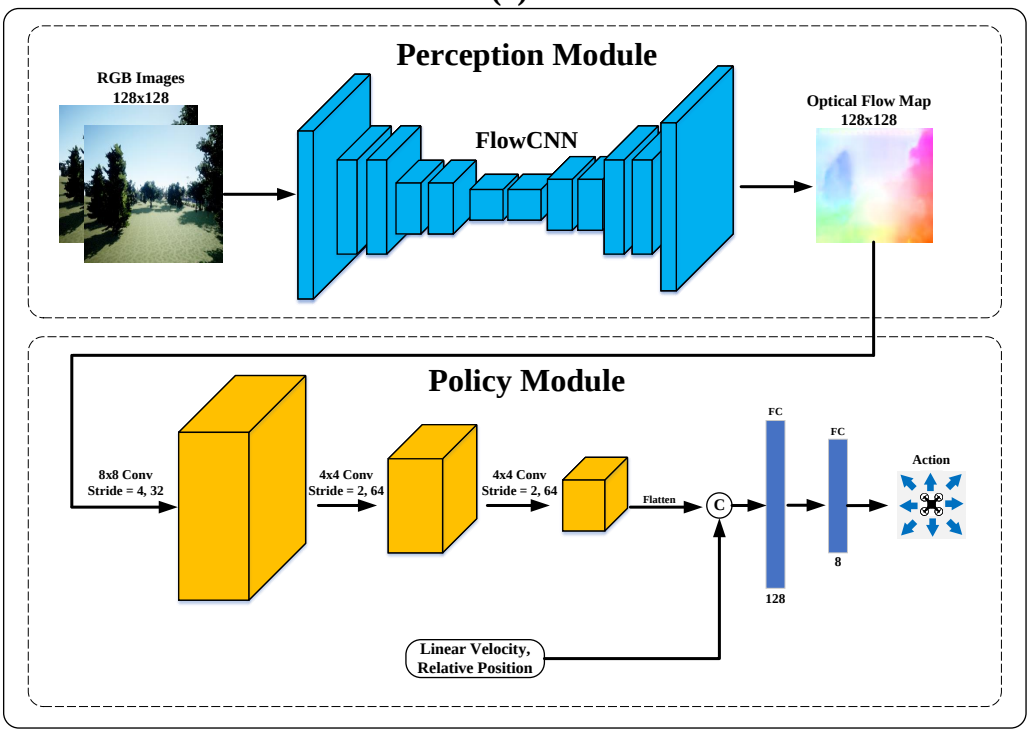
Figure 6.4: Detailed structure of the encoder of the MobileFlow network. "Conv" represents the regular convolutional layer and "DWConv" means depthwise convolution layer.

encoder incorporates a tensor generated through concatenating two consecutive RGB images and produces intermediate feature maps with 1/16 size of the input images. The decoder of MobileFlow has the same architecture as the refine module of FlowNet [37], which has four 2D transposed convolution layers. Each transposed convolution layer has an up-sampling factor of 2. The decoder takes as input feature maps from the encoder and produces OF maps with the same resolution as the input images. The designed MobileFlow network has 3.73 million parameters and runs about 480 fps on an Nvidia RTX 3090 GPU.

The objective of the policy module is to learn the guidance policy through RL. As shown in Figure 6.5, the policy module has three convolutional layers and two FC layers. Each convolutional layer uses ReLU as the activation function. The convolutional layers downsample the depth map or OF map to 2D feature maps, which are then flattened to a 1D feature vector. Subsequently, the flattened feature vector is concatenated with the relative position and agent's current linear velocity. The FC layers incorporate the concatenated feature vector and output the Q values of eight discrete actions.



(a)



(b)

Figure 6.5: Illustration of the baseline policy architectures using the MDE network (a) and the OF estimation network (b).

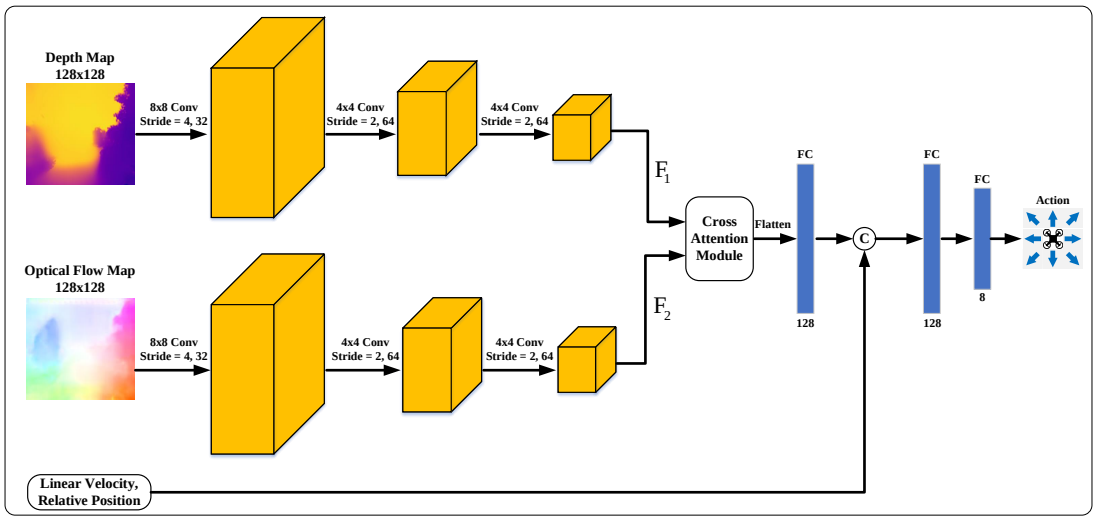


Figure 6.6: Illustration of the proposed multi-modal information-based policy architecture.

In the multi-modal information-based policy module (see Figure 6.6), both the depth and OF maps are passed to convolutional layers to extract feature maps (referred to as F_1 and F_2), respectively. Instead of concatenating F_1 and F_2 in channel dimension, we apply a cross-attention module to fuse them. As shown in Figure 5.4, the cross-attention module consists of two linear transformer encoder layers, and a "1×1 Conv-BN-ReLU" layer. It fuses F_1 and F_2 through the cross-attention mechanism in two directions ((F_1, F_2) and (F_2, F_1)). The fused feature is flattened and concatenated with the relative position and agent's linear velocity and passed to two FC layers to learn policy π. This process is formulated as:

$$\pi = f_{policy}(concatenate(fusion(F_1, F_2), Vel, RelPos), \theta_{policy})) \tag{6.10}$$

where F_1 and F_2 represent features extracted from depth maps and OF maps respectively, Vel is the agent's linear velocity, $RelPos$ denotes the relative position, and f_{policy} stands for the policy module with learnable parameters θ_{policy}.

Figure 6.7: Example images captured from front view camera of the UAV during flight in simulated environments. From left to right: Outdoor Forest Environment, Rural Australia Environment, and the Outdoor Soccer Field Environment.

6.3 Experimental Setup

6.3.1 Simulated Environments

We adopt three 3D highly realistic environments, two environments are forest surroundings and the other one is an outdoor soccer field. For the forest environments, one is called "Outdoor Forest" and the other is called "Rural Australia". The "Outdoor Forest" environment was taken from [317]. We downloaded the source file of the "Rural Australia" environment [280] from the Unreal Engine Marketplace and set it up according to the tutorial of AirSim[3]. For the outdoor soccer field environment, we used the binary model released by AirSim[4]. The binary model was built through the Unreal Engine [279] and can be directly applied to simulation experiments. These simulated environments contain a large variety of lighting conditions and objects (e.g., trees and benches). Figure 6.7 shows the example images captured from the front view camera of the UAV.

[3]https://microsoft.github.io/AirSim/unreal_custenv/
[4]https://github.com/Microsoft/AirSim/releases

6.3.2 Implementation Details

We adopt AirSim [274] as the simulator and implement the designed MMIDRL framework in Pytorch [283]. The simulation agent is a quadrotor UAV equipped with onboard sensors including an RGB camera and IMU. It is noteworthy that only the front view camera is utilized to capture images from the environment. For the weighting parameters in frontier search, we set $\lambda_D = 0.001$ and $\lambda_S = 1$. We train the DDQN-based agent with hyper-parameters illustrated in Table 6.1. Both training and testing are conducted on a workstation with Windows 10 OS and a single Nvidia GeForce RTX 3090 GPU. As regards CNN models in the perception module, we use the MobileXNet [288] or EGD-Net [309] for MDE, and MobileFlow for OF estimation. Both MobileXNet [288] and EGD-Net [309] are trained on the UnrealDataset [34], which was collected from a series of simulated urban and forest scenarios in AirSim [274]. All data for training DepthCNNs are resized to $128 \times 128 \times 3$ pixels, and the hyper-parameters for training are kept same as [288, 309]. We train MobileFlow on the Sintel dataset [318] and the hyper-parameters for training are the same as [37].

Table 6.1: Hyper parameters in our simulation experiments.

Item	Value
Discount factor (γ)	0.99
Mini-batch size	32
Learning rate	0.001
Optimization algorithm	Adam
Loss function	Smooth L1 Loss
Update frequency	50

6.3.3 Baselines

In this chapter, we compare our proposed MMIDRL framework with three baseline methods:

- **Random:** The random policy uniformly, with equal probability at each timestep, selects an action from the action space. For consistency of comparisons, the action space and maximum timesteps of this method are the same as the DRL-based methods.

- **Monocular Depth Perception-Based Method (MDP):** This method only uses an MDE network ((i.e., MobileXNet [288] or EGD-Net [309])) in the perception module, and maps the predicted depth maps to policies for UAV guidance.

- **Optical Flow Perception-Based Method (OFP):** This method only uses an OF estimation network, MobileFlow, in the perception module. The predicted OF maps are used to learn policies for UAV guidance.

6.3.4 Performance Metrics

The adopted performance metrics are described as follows:

- **Success Rate:** The percentage of times the agent completes the defined task. Ideally, this number will be close to 100% as it reflects the algorithms' functionality.

- **Average Time:** The average time that the agent spends completing a task within the simulated environment.

- **Average Distance:** The average travel distance while completing the area coverage task or flying to destination point.

In the stage of evaluation, 50 episodes are run for each method. At the start of each evaluation episode, the agent starts at a random position.

6.4 Experimental Results

In this section, we evaluate the performance of our proposed MMIDRL framework in visual guidance of the UAV. First, we apply the MMIDRL framework to the scenario of controlling the UAV flying to a fixed destination. In this scenario, the objective of the UAV is to reach the pre-defined destination point from a random start point and avoiding collision with objects in the environment. Then, we apply the MMIDRL framework to the scenario of UAV area coverage.

6.4.1 Flying to Destinations

We first evaluate the performance of the MMIDRL framework to control a simulated UAV autonomously flying from start points to destinations in cluttered forest environments. We compare our MMIDRL framework against four baselines. The MMIDRL framework and the DRL-based baselines are trained in the "Outdoor Forest" environment, and evaluated in an unseen forest environment, "Rural Australia". We train each DRL agent for 1000 episodes. At the beginning of each episode, the start point and destination point are randomly selected from an area of size 100m $\times$ 100m. The reward function in this experiment includes three terms: $r = r_{time} + r_{destination} + r_{terminal}$. In terms of the $r_{terminal}$, the termination cases are: (1) the UAV collides with any object in the environment; (2) the UAV flies outside of the pre-defined area; and (3) the current episode timesteps T_{step} reach 500.

We consider an episode to be completed if the UAV reaches the destination point (i.e., the distance between the UAV and destination point less a threshold value, $T_{destination}$) without colliding with objects in the environment and the simulation does not exceed the allowed maximum time steps. The value of

Table 6.2: Comparative performance of the proposed MMIDRL framework in flying to destinations task. Random means the random policy, MDP-E and MDP-M represent monocular depth perception-based method using EGD-Net and MobileXNet respectively, OFP denotes optical flow perception-based method, MMIDRL (E) and MMIDRL (M) indicate MMIDRL framework using EGD-Net and MobileXNet respectively. The red and **bold** values indicate the best results.

Method	Completed Episodes	Failed Episodes	Success Rate
Random	4	46	8%
MDP-E	45	5	90%
MDP-M	42	8	84%
OFP	38	12	76%
MMIDRL (E)	49	1	98%
MMIDRL (M)	50	0	**100%**

$T_{destination}$ is set as $3m$. Since the start point and destination point are randomly generated, the distance between these two points are different in each episode. Therefore, we only use the success rate as the performance metric. The summary of the results are listed in Table 6.2.

As can be seen, among all methods, the random policy yields the worst performance, and only completes four episodes. The monocular depth perception-based methods (MDP-E and MDP-M) outperform the optical flow perception-based method (OFP). Among the two monocular depth perception-based methods, MDP-E (EGD-Net) performs better than MDP-M (MobileXNet). Combining the two complementary inputs of MDE and OF estimation improves the performance of the DRL agent. The multi-modal information based methods (MMIDRL (E) and MMIDRL (M)) outperforms both monocular depth perception-based methods (MDP-E and MDP-M) and optical flow perception-based method (OFP). In the next section, we will evaluate the performance of our proposed MMIDRL framework in the scenario of UAV area coverage.

6.4.2 Area Coverage

The main objective of the experiments in this section is to evaluate the performance of the MMIDRL framework in the scenario of UAV area coverage. For this purpose, we train the MMIDRL framework and the DRL-based baselines in the "Rural Australia" environment and evaluate the trained models in the "Soccer Field" environment. Both the MMIDRL framework and the DRL-based baselines are trained for 500 episodes. We define a working area and an extended area within the simulated environment, and then approximate it into a grid map through discretization. In particular, the working area and extended area are constrained to $40m \times 40m$ and $60m \times 60m$ respectively. The size of each grid cell is set as $5m \times 5m$, as we found that a smaller size leads to an apparent increase of training and testing time for each episode. This results in a working area with 8×8 grid cells.

At the beginning of each episode, a start point is randomly selected from the boundary of the working area. Since the UAV is expected to simply visit as many free grid cells as possible, no destination point is defined. In order to measure the performance of area coverage in a separate episode, we define a coverage rate (CR) metric:

$$CR = \frac{N_{covered}}{N_{free}} \times 100\% \tag{6.11}$$

where $N_{covered}$ and N_{free} represent the number of covered free cells and the total number of free cells in the grid map respectively. If the value of CR in each episode achieves a threshold value ($T_{coverage}$), this episode will be considered as complete. In general, a larger CR value results in a longer path and requires longer time in training and evaluation [267]. In this section, the value of $T_{coverage}$ is set as 95%. The proposed MMIDRL framework is compared with four baselines, and

Table 6.3: Comparative performance of the proposed MMIDRL framework in area coverage task. Random means the random policy, MDP-E and MDP-M represent the monocular depth perception-based method using EGD-Net and MobileXNet respectively, OFP denotes optical flow perception-based method, MMIDRL (E) and MMIDRL (M) indicate MMIDRL framework using EGD-Net and MobileXNet respectively. The red and **bold** values indicate the best results.

Method	Completed Episodes	Failed Episodes	Success Rate	Average Distance (m)	Average Time (s)
Random	0	50	0	*fail*	*fail*
MDP-E	50	0	**100%**	385.7	516.9
MDP-M	48	2	96%	342.7	432.3
OFP	44	6	88%	376.3	499.4
MMIDRL (E)	50	0	**100%**	366.9	484.4
MMIDRL (M)	50	0	**100%**	**305.3**	**384.0**

the comparison is conducted using the three performance metrics described in Section 6.3.2. Table 6.3 illustrates the quantitative results of all methods.

As shown in Table 6.3, the random policy fails in all episodes. Specifically, 44 episodes fly out of the expanded area, four episodes fail to complete the coverage mission and reach the maximum timesteps, and two episodes result in collisions with objects in the environment. These results offer objective evidence that a sophisticated designed algorithm is essential. The monocular depth perception-based methods (MDP-E and MDP-M) outperform the optical flow perception-based method (OFP) in terms of success rate. According to the 2nd and 3rd rows in the Table, the EGD-Net based method (MDP-E) achieves higher success rate than MobileXNet-based method (MDP-E), but its average distance and average time metrics are inferior. When combining MobileXNet with OF estimation, the produced multi-modal information-based method (MMIDRL(M)) generates the best performance in all the three metrics. These results suggest that combining OF estimation and MDE improves the performance of area coverage. The promising results should be attributed to the fact that the DRL model was trained in small sized images (128×128 pixels) and the design of MobileXNet

considered the situation of MDE from small sized images.

6.4.3 Ablation Studies

In order to further analyze the performance of our proposed method, we perform ablation studies to compare its performance when using different action spaces and feature fusion methods. We use the multi-modal information-based method with MobileXNet (MMIDRL (M)) as the benchmark in this section due to its superior performance in Section 6.4.2. The training and testing protocols are kept the same as those adopted in Section 6.4.2.

Comparison of Different Action Spaces

In this section, we compare the effect of using different action spaces in the performance of the proposed area coverage method. To this end, we train the proposed MMIDRL framework with the action space consisting of discrete positions. Instead of controlling the simulated UAV through linear velocities, we control it with positions. Specifically, at each time step, the agent selects an action and changes the position of the UAV in the simulation environment. The quantitative results are shown in Table 6.4.

It can be seen from Table 6.4 that the average distance and average time results produced by position-based action space (MMIDRL (M-P)) are apparently inferior to the velocity-based action space (MMIDRL (M-V)). We visualize example trajectories generated by these two methods and illustrate it in Figure 6.8 and Figure 6.9, respectively. Compared with the velocity-based action space (MMIDRL (M-V)), the position-based action space (MMIDRL (M-P)) suffers from a jerky motion.

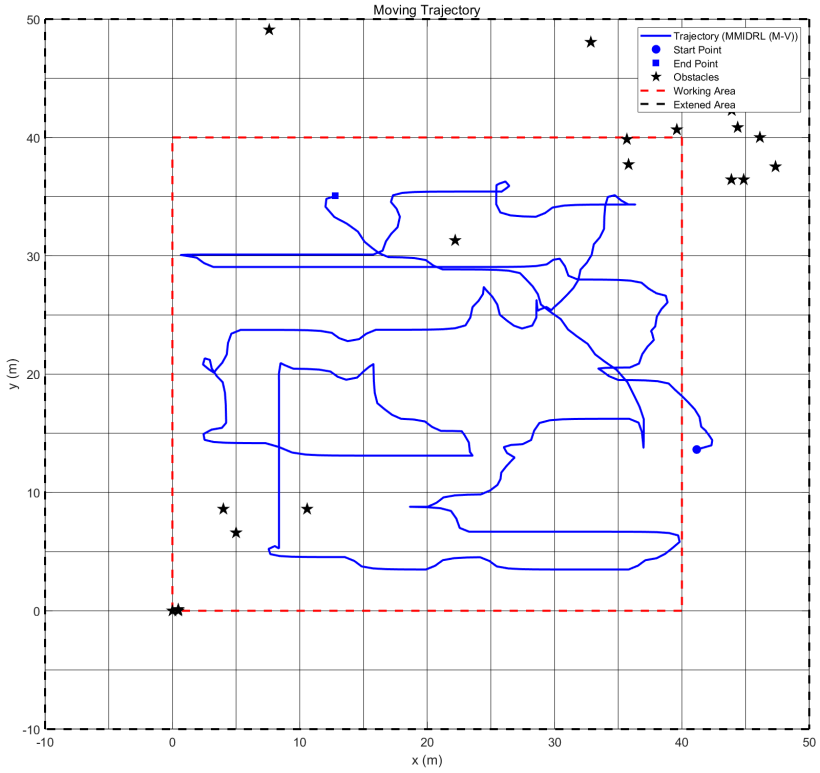


Figure 6.8: An example trajectory generated by the MMIDRL framework with velocity-based action space (MMIDRL (M-V)).

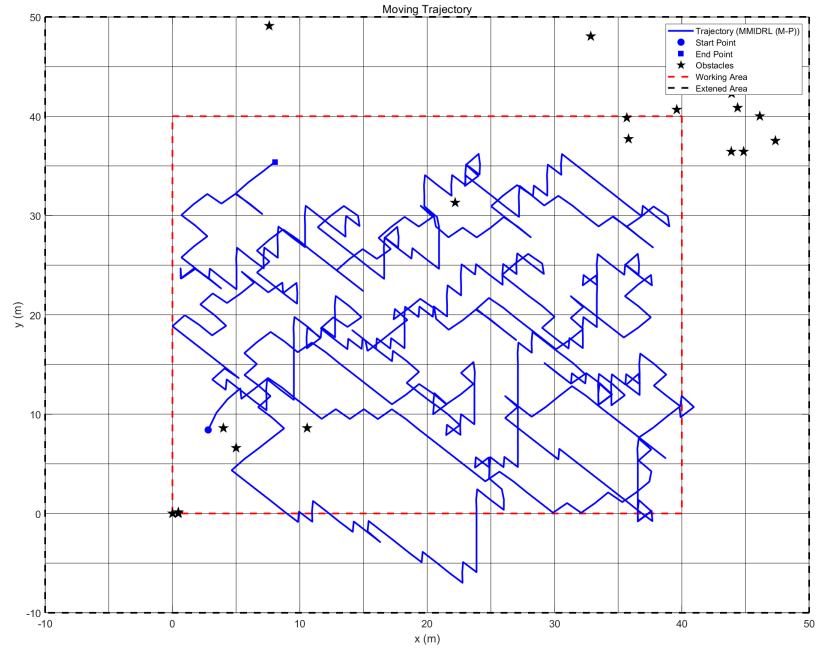


Figure 6.9: An example trajectory generated by the MMIDRL framework with position-based action space (MMIDRL (M-P)).

Table 6.4: Comparative performance of the proposed MMIDRL framework with different action spaces. MMIDRL (M-P) means multi-modal information-based method using MobileXNet and discrete position actions, MMIDRL (M-V) means multi-modal information-based method using MobileXNet and discrete velocity actions. The red and **bold** values indicate the best results.

Method	Completed Episodes	Failed Episodes	Success Rate	Average Distance (m)	Average Time (s)
MMIDRL (M-V)	50	0	**100%**	**305.3**	**384.0**
MMIDRL (M-P)	50	0	**100%**	663.7	1336.6

Comparison of Different Feature Fusion Methods

We evaluate the contribution of the cross-attention module by comparing its performance against a baseline constructed by replacing the cross-attention module with a concatenation operation. The baseline method is named as "MMMIDRL (Concat)". We train and evaluate the baseline method and proposed method (referred to as "MMIDRL (Att)") with the same parameters. The quantitative results are reported in Table 6.5.

Table 6.5: Comparative performance of the proposed MMIDRL framework with different feature fusion methods. MMIDRL (Concat) means MMIDRL method fuses features through concatenation. MMIDRL (Att) represents MMIDRL fuses features through cross-attention module. The red and **bold** values indicate the best results.

Method	Completed Episodes	Failed Episodes	Success Rate	Average Distance (m)	Average Time (s)
MMIDRL (Concat)	50	0	**100%**	307.6	439.7
MMIDRL (Att)	50	0	**100%**	**305.3**	**384.0**

The cross-attention module-based MMIDRL framework (MMIDRL (Att)) outperforms the baseline method (MMIDRL (Concat)) in terms of the average distance and average time metrics. We attribute the promising results to the fact that the cross-attention module captures the long-range dependencies between the features learned from depth maps and OF maps. The fused feature maps further facilitates the performance of UAV area coverage.

6.5 Chapter Summary

In this chapter, we proposed a multi-modal information-based deep reinforcement learning (MMIDRL) framework to solve the area coverage problem for UAVs. The designed MMIDRL framework applies a monocular RGB camera as perception sensor and takes advantages of two CNNs for MDE and OF estimation to learn policies for UAV area coverage. Furthermore, we designed a frontier guided area coverage method which searches the next desired destinations for the UAV based on frontiers. Extensive experiments have been conducted in simulated environments to demonstrate the effectiveness of the proposed method. However, our proposed method did not record the positions of obstacles during the coverage task. In future work, a more rigorous implementation would be to construct an obstacle map in real-time from the predicted depth maps. This map could then be used to mark cells as unavailable, which are blocked by obstacles to prevent the frontier vector from directing the UAV back to areas which could not be visited.

Chapter 7

Conclusions and Future Work

7.1 Summary of Results

In this thesis, a vision guidance method for unmanned aerial vehicles (UAVs) in cluttered outdoor environment using monocular camera and deep learning (DL) has been designed. In particular, the designed method enables a UAV depending on a monocular camera to perceive the environment through depth and optical flow (OF) information to perform area coverage and avoid collisions with obstacles in the simulation environment. Motivated by the literature review in Chapter 2, a campaign of research work has been conducted step by step throughout the thesis.

In Chapter 2, we conducted the literature review. Especially, we performed a comprehensive survey on monocular depth estimation (MDE). We reviewed Structure from Motion (SfM)-based methods, traditional handcrafted feature-based methods, state-of-the-art DL-based methods, and the application of MDE in robotics. It was concluded that developing real-time MDE network with accuracy and efficiency balance is a promising direction.

In Chapter 4, we proposed a real-time MDE network, named MobileXNet. MobileXNet stacks two simple and shallow encoder-decoder style subnetworks in a unified framework, which enables MobileXNet to have a shallow and simple architecture. In particular, the first subnetwork of MobileXNet produces feature maps having a larger resolution (1/4 size of input images) and include more spatial information. These feature maps are passed to the second subnetwork to produce depth maps, which is beneficial for depth estimation on small sized images. Extensive experiments on four datasets demonstrated that MobileXNet achieves accuracy and efficiency balance.

The proposed MobileXNet achieved promising results, however, it did not utilize edge features in input images. In chapter 5, we proposed a lightweight MDE network, named Edge Guided Depth Estimation Network (EGD-Net). EGD-Net integrates a depth estimation branch and an edge guidance branch in a unified network. Moreover, it builds on top of a shallow (stacks less layers) and narrow (includes less filters in each layer) encoder-decoder network and applies edge attention features to guide the depth estimation task. Experimental results showed that EGD-Net runs at about 96 fps on an Nvidia GTX 1080 GPU whilst achieving state-of-the-art performance in terms of accuracy.

To achieve the aim of UAV guidance, we designed a frontier guided area coverage method in Chapter 6. The designed method uses a frontier search algorithm to detect frontiers from the environment. The detected frontiers are then used as desired destinations for the UAV. In addition, a multi-modal information-based deep reinforcement learning (MMIDRL) framework was designed. The designed MMIDRL framework only uses a single monocular RGB camera for perception. Benefiting from CNNs for MDE and OF estimation, the proposed method enables the UAV to perceive the environment through depth information and OF

information. Extensive experiments in simulation environments demonstrated the effectiveness of the proposed method.

7.2 Future Work

7.2.1 MDE Networks Run on Embedded Platform

In Chapters 4 and 5, we proposed two CNNs for MDE. Those two CNNs were named MobileXNet and EGD-Net respectively. In addition to the promising accuracy performance, MobileXNet and EGD-Net run at high frame rates (126 fps and 90 fps respectively), on a single modest power GPU. We applied these two CNNs to visual guidance of UAVs in simulation environments. As the simulation experiments were performed in a workstation with a GPU, future work, should investigate the running speed on an embedded platform such as an Nvidia Jetson TX2 or Jetson AGX Xavier. This could validate the potential of deploying these networks on a small UAV equipped with an embedded computation module.

7.2.2 Design of MDE Networks

In Chapter 5, we proposed the EGD-Net, which integrates a depth estimation branch and an edge guidance branch (EGB) in a unified network. To fuse features from these two branches, we designed a transformer-based feature aggregation (TRFA) module. The TRFA module applies two linear transformer encoder layers to model long-range dependencies between those features. It is important to note that EGD-Net has 2.21 millions of parameters but its running speed is lower than MobileXNet (has 24.95 millions of parameters). This should be attributed to the characteristic of the linear transformer encoder layers, which

depend on dot product computation between feature maps. Since these features have a larger resolution, the huge number of FLOPs (floating-point operations per second) slows the running speed. One possible approach will be to apply an efficient algorithm (e.g., separable vision transformer [319]) to reduce the computation complexity of the linear transformer layer. Moreover, the EGB could be discarded in the inference stage. Therefore, the edge attention features can improve the accuracy of depth estimation without any cost in inference.

7.2.3 Replacing GPS with SLAM to Obtain the Position of UAV

In Chapter 6, the position of the UAV was obtained from AirSim [274]. In AirSim, the default controller designated the "simple flight" controller does not include a sensor model or a state estimator. Instead, it uses the ground-truth from the simulator to provide the state information. Therefore, the proposed area coverage method proceeds under the assumption that the position of the UAV is known from a sensor such as GPS. The frontier search algorithm requires the position of the UAV. To make the proposed method work in a GPS-denied environment, a Simultaneous Localization and Mapping (SLAM) algorithm (e.g., ORB-SLAM [185, 320]) could be applied to predict the position of the UAV.

7.2.4 Map Construction from Predicted Depth Maps

In Chapter 6, we presented a frontier guided area coverage method for UAVs. The proposed method applied a frontier search algorithm to detect frontiers from the environment. The detected frontiers are used as the desired destinations for UAV. Experiments in simulation environments demonstrated that the proposed

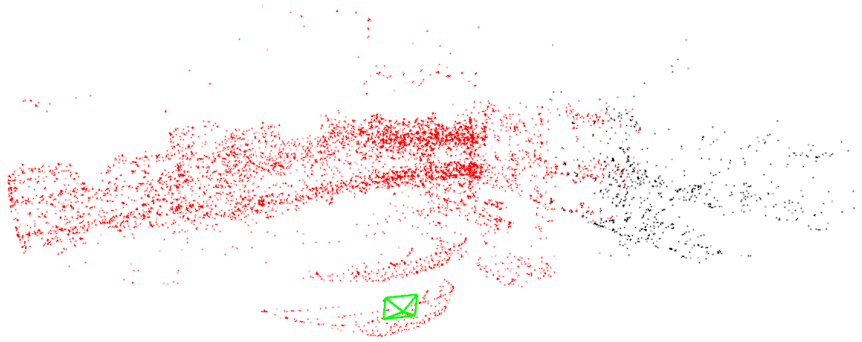
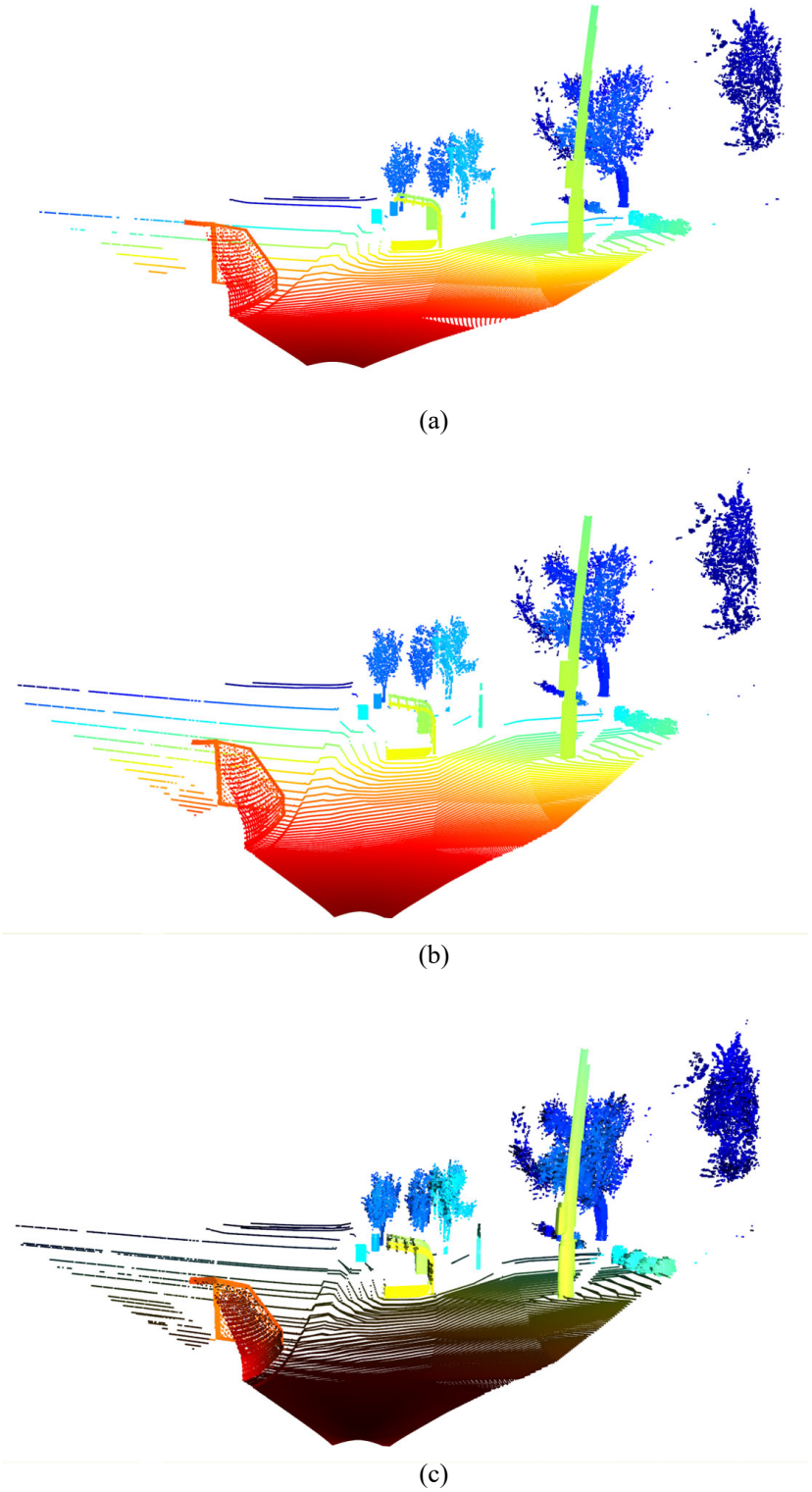


Figure 7.1: Screen shot of 3D point cloud produced by ORB-SLAM2. The green block represents the position of the UAV in the environment.

method can enable a UAV to cover as much of the accessible areas as possible in the environment and avoid obstacles. However, the proposed methods have limitations because the positions of obstacles are not recorded during the coverage task. In this case, the UAV may suffer the problem of trying to revisit the same cells which are blocked by obstacles and cannot be reached. To solve this problem, a real-time grid map of occupied cells should be constructed from the predicted depth maps. The constructed map could be used to mark grid cells as unreachable which are occupied by obstacles to prevent the frontier vector from directing the UAV back to areas that could not be visited.

To build the grid map of occupied cells, real-time SLAM methods such as ORB-SLAM2 [185] could be applied in our future work. As a state-of-the-art visual SLAM algorithm, ORB-SLAM2 is capable of producing a point cloud based 3D map. Figure 7.1 shows the point cloud based 3D map produced by the ORB-SLAM2 algorithm [185]. It is worth noting that the produced 3D map is sparse, it may be difficult to construct an occupancy map that contains most of the obstacles. Therefore, a more denser map should be considered.

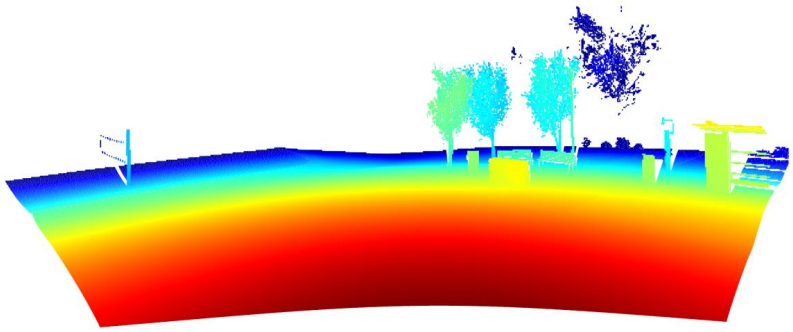


(a)

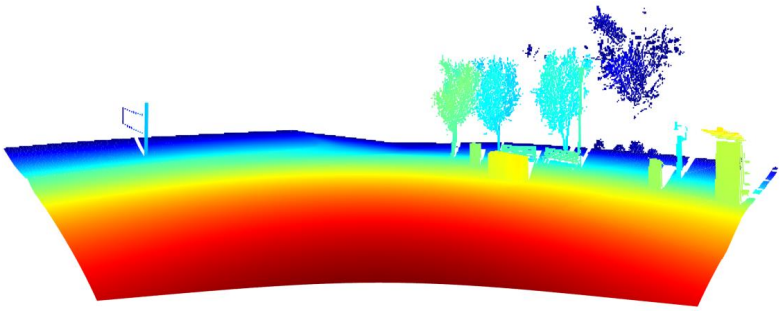
(b)

(c)

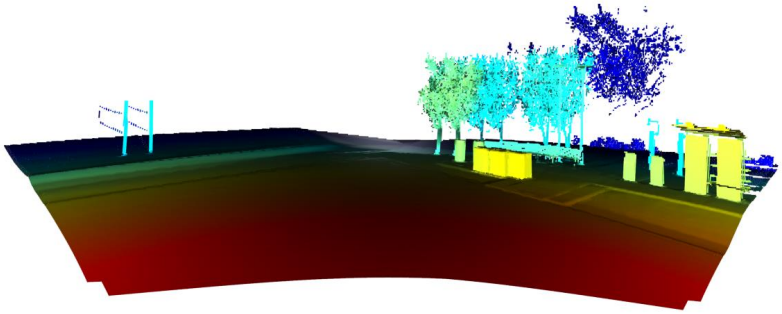
Figure 7.2: Illustration of point cloud registration with small movement between two consecutive depth images. (a) Point cloud converted from the first frame ground-truth depth image, (b) Point cloud converted from the second frame ground-truth depth image, and (c) The registered point cloud.



(a)



(b)



(c)

Figure 7.3: Illustration of point cloud registration with large movement between two consecutive depth images. (a) Point cloud converted from the first frame ground-truth depth image, (b) Point cloud converted from the second frame ground-truth depth image, and (c) The registered point cloud.

In this thesis, we assumed that the UAV only depends on a monocular camera to perceive the environment. Under this situation, any 3D sensors for capturing point cloud from the environment are not available. One possible method will be to convert the depth pixel from the 2D coordinate system of depth map to 3D space. With the obtained point cloud, a 3D map can be constructed through cloud registration algorithm [321]. The accuracy of the cloud registration algorithm determines the quality of the constructed map [322]. Therefore, high registration accuracy is a core requirement for the cloud registration algorithm. Figure 7.2 and Figure 7.3 show examples of point cloud registration through iterative closest point (ICP) algorithm [323,324]. It can be observed that the ICP algorithm does not produce good performance when there is large movement between consecutive frames.

Moreover, point cloud registration encounters two main challenges: noise and outliers, and partial overlap [322]. The sensor noise may result in point clouds including noise and outliers around the same position. Besides, the transformation between depth map and point cloud may bring about noise and outliers because of the error of camera calibration or noise in the depth map. Since the source data of point clouds are collected from different views, especially under the case of abrupt turn, the generated point clouds are partial overlapped. Both SLAM-based and point cloud registration-based methods should be considered in future work.

7.3 Concluding Remarks

Compared with active sensors such as LiDAR, vision sensors, especially monocular cameras are low cost, light weight and require less hardware resources.

Therefore, monocular vision-based visual guidance is promising for small UAVs. Taking advantage of CNNs developed for MDE and OF estimation, a UAV equipped with a monocular camera can perceive the environment through depth and OF information and achieve the aim of visual guidance. There are still issues that need to be tackled to improve the performance of monocular vision-based guidance method in cluttered outdoor environment. Due to limitations of time, we recommend the work of replacing GPS with SLAM and map construction from predicted depth maps as future work. This body of work will be a solution to prevent the frontier vector from directing the UAV back to areas that could not be visited, and a chance to make UAV area coverage in GPS-denied environment a reality.

References

[1] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The KITTI dataset. The International Journal of Robotics Research, 32(11):1231–1237, 2013.

[2] David Eigen, Christian Puhrsch, and Rob Fergus. Depth map prediction from a single image using a multi-scale deep network. In Advances in Neural Information Processing Systems, pages 2366–2374, 2014.

[3] Nathan Silberman, Derek Hoiem, Pushmeet Kohli, and Rob Fergus. Indoor segmentation and support inference from RGBD images. In European Conference on Computer Vision, pages 746–760. Springer, 2012.

[4] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional networks for biomedical image segmentation. In International Conference on Medical Image Computing and Computer-Assisted Intervention, pages 234–241. Springer, 2015.

[5] Antonio Loquercio, Ana I Maqueda, Carlos R Del-Blanco, and Davide Scaramuzza. DroNet: Learning to fly by driving. IEEE Robotics and Automation Letters, 3(2):1088–1095, 2018.

[6] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. Deep reinforcement learning: A brief survey. IEEE Signal Processing Magazine, 34(6):26–38, 2017.

[7] Bryan Chen, Alexander Sax, Gene Lewis, Iro Armeni, Silvio Savarese, Amir Zamir, Jitendra Malik, and Lerrel Pinto. Robust policies via mid-level visual representations: An experimental study in manipulation and navigation. arXiv preprint arXiv:2011.06698, 2020.

[8] Simon Zingg, Davide Scaramuzza, Stephan Weiss, and Roland Siegwart. MAV navigation through indoor corridors using optical flow. In 2010 IEEE International Conference on Robotics and Automation, pages 3361–3368. IEEE, 2010.

[9] Jessica Alvarenga, Nikolaos I Vitzilaios, Kimon P Valavanis, and Matthew J Rutherford. Survey of unmanned helicopter model-based navigation and control techniques. Journal of Intelligent & Robotic Systems, 80(1):87–138, 2015.

[10] Behzad Boroujerdian, Hasan Genc, Srivatsan Krishnan, Wenzhi Cui, Aleksandra Faust, and Vijay Reddi. MAVBench: Micro aerial vehicle benchmarking. In 2018 51st annual IEEE/ACM international symposium on microarchitecture (MICRO), pages 894–907. IEEE, 2018.

[11] E Wardihani, Magfur Ramdhani, Amin Suharjono, Thomas Agung Setyawan, Sidiq Syamsul Hidayat, SARONO WIDODO Helmy, EDDY Triyono, and FIRDANIS Saifullah. Real-time forest fire monitoring system using unmanned

aerial vehicle. Journal of Engineering Science and Technology, 13(6):1587–1594, 2018.

[12] Diyana Kinaneva, Georgi Hristov, Jordan Raychev, and Plamen Zahariev. Early forest fire detection using drones and artificial intelligence. In 2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), pages 1060–1065. IEEE, 2019.

[13] Salvatore Manfreda, Matthew F McCabe, Pauline E Miller, Richard Lucas, Victor Pajuelo Madrigal, Giorgos Mallinis, Eyal Ben Dor, David Helman, Lyndon Estes, Giuseppe Ciraolo, et al. On the use of unmanned aerial systems for environmental monitoring. Remote Sensing, 10(4):641, 2018.

[14] Jarrod C Hodgson, Shane M Baylis, Rowan Mott, Ashley Herrod, and Rohan H Clarke. Precision wildlife monitoring using unmanned aerial vehicles. Scientific Reports, 6(1):1–7, 2016.

[15] Dario Floreano and Robert J Wood. Science, technology and the future of small autonomous drones. Nature, 521(7553):460, 2015.

[16] Yulun Tian, Katherine Liu, Kyel Ok, Loc Tran, Danette Allen, Nicholas Roy, and Jonathan P How. Search and rescue under the forest canopy using multiple UAVs. The International Journal of Robotics Research, 39(10-11):1201–1221, 2020.

[17] Azur Drones. Skeyetech: fully autonomous drone for safety and security. `https://www.azurdrones.com/product/skeyetech/`.

[18] Evangelos Maltezos, Michael Skitsas, Elisavet Charalambous, Nikolaos Koutras, Dimitris Bliziotis, and Kyriacos Themistocleous. Critical infrastructure monitoring using UAV imagery. In Fourth International Conference on Remote Sensing and Geoinformation of the Environment (RSCy2016), volume 9688, page 96880P. International Society for Optics and Photonics, 2016.

[19] Ezedin Barka, Chaker Abdelaziz Kerrache, Hadjer Benkraouda, Khaled Shuaib, Farhan Ahmad, and Fatih Kurugollu. Towards a trusted unmanned aerial system using blockchain for the protection of critical infrastructure. Transactions on Emerging Telecommunications Technologies, page e3706, 2019.

[20] Christoforos Kanellakis and George Nikolakopoulos. Survey on computer vision for UAVs: Current developments and trends. Journal of Intelligent & Robotic Systems, 87(1):141–168, 2017.

[21] Robin Murphy. Drones save lives in disasters, when they're allowed to fly (Op-Ed). `https://https://www.space.com/30555-beginning-with-katrina-drones-save-lives-in-disasters.html`, 2015.

[22] University Of South Florida. USF deploys unmanned aerial vehicles to katrina rescue operation. https://www.sciencedaily.com/releases/2005/09/050908081119.htm, 2005.

[23] Yukai Peng and Jun Liang. China deploys large UAV to support emergency communications in quake-hit Sichuan. http://en.people.cn/n3/2022/0907/c90000-10144282.html.

[24] Stphane Morelli. Robots and UAVs: how machines help humans against coronavirus. https://www.azurdrones.com/robots-uavs-against-coronavirus/.

[25] Douglas Gimesy. Drones and thermal imaging: saving koalas injured in the bushfires. https://www.theguardian.com/australia-news/gallery/2020/feb/11/drones-thermal-imaging-australia-koalas-bushfire-crisis.

[26] Christine Mendoza. Wile horses in snowy mountains, kosciuszko national park, australia. https://unsplash.com/photos/HDDZOfX8pLA, 2018.

[27] Berthold KP Horn and Brian G Schunck. Determining optical flow. Artificial Intelligence, 17(1-3):185–203, 1981.

[28] Bruce D Lucas and Takeo Kanade. An iterative image registration technique with an application in stereo vision. In Proceedings of the International Joint Conference on Artificial Intelligence, pages 674–679. International Joint Conferences on Artificial Intelligence Organization, 1981.

[29] Gunnar Farnebäck. Two-frame motion estimation based on polynomial expansion. In Scandinavian Conference on Image Analysis, pages 363–370. Springer, 2003.

[30] David G Lowe. Distinctive image features from scale-invariant keypoints. International Journal of Computer Vision, 60(2):91–110, 2004.

[31] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (SURF). Computer Vision and Image Understanding, 110(3):346–359, 2008.

[32] Loris Nanni, Stefano Ghidoni, and Sheryl Brahnam. Handcrafted vs. non-handcrafted features for computer vision classification. Pattern Recognition, 71:158–172, 2017.

[33] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 248–255. IEEE, 2009.

[34] Michele Mancini, Gabriele Costante, Paolo Valigi, and Thomas A Ciarfuglia. J-MOD 2: joint monocular obstacle detection and depth estimation. IEEE Robotics and Automation Letters, 3(3):1490–1497, 2018.

[35] Xingshuai Dong, Matthew A Garratt, Sreenatha G Anavatti, and Hussein A Abbass. Towards real-time monocular depth estimation for robotics: A survey. IEEE Transactions on Intelligent Transportation Systems, 23(10):16940–16961, 2022.

[36] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet classification with deep convolutional neural networks. In Advances in Neural Information Processing Systems, pages 1097–1105, 2012.

[37] Alexey Dosovitskiy, Philipp Fischer, Eddy Ilg, Philip Hausser, Caner Hazirbas, Vladimir Golkov, Patrick Van Der Smagt, Daniel Cremers, and Thomas Brox. FlowNet: Learning optical flow with convolutional networks. In Proceedings of the IEEE/CVF International Conference on Computer Vision, pages 2758–2766, 2015.

[38] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. Neural Computation, 1(4):541–551, 1989.

[39] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11):2278–2324, 1998.

[40] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556, 2014.

[41] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 770–778, 2016.

[42] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 4700–4708, 2017.

[43] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-Excitation networks. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 7132–7141, 2018.

[44] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. MobileNets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861, 2017.

[45] François Chollet. Xception: Deep learning with depthwise separable convolutions. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 1251–1258, 2017.

[46] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. MobileNetV2: Inverted residuals and linear bottlenecks.

In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 4510–4520, 2018.

[47] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. ShuffleNet V2: Practical guidelines for efficient CNN architecture design. In Proceedings of the European conference on computer vision (ECCV), pages 116–131, 2018.

[48] Mingxing Tan and Quoc Le. EfficientNet: Rethinking model scaling for convolutional neural networks. In International Conference on Machine Learning, pages 6105–6114. PMLR, 2019.

[49] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. Searching for MobileNetV3. arXiv preprint arXiv:1905.02244, 2019.

[50] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. arXiv preprint arXiv:1706.03762, 2017.

[51] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. arXiv preprint arXiv:1607.06450, 2016.

[52] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. arXiv preprint arXiv:2010.11929, 2020.

[53] Zhiliang Peng, Wei Huang, Shanzhi Gu, Lingxi Xie, Yaowei Wang, Jianbin Jiao, and Qixiang Ye. ConFormer: Local features coupling global representations for visual recognition. In Proceedings of the IEEE/CVF International Conference on Computer Vision, pages 367–376, 2021.

[54] Jianyuan Guo, Kai Han, Han Wu, Chang Xu, Yehui Tang, Chunjing Xu, and Yunhe Wang. CMT: Convolutional neural networks meet vision transformers. arXiv preprint arXiv:2107.06263, 2021.

[55] Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are RNNs: Fast autoregressive transformers with linear attention. In International Conference on Machine Learning, pages 5156–5165, 2020.

[56] Keisuke Tateno, Federico Tombari, Iro Laina, and Nassir Navab. CNN-SLAM: Real-time dense monocular SLAM with learned depth prediction. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 6243–6252, 2017.

[57] Xin Yang, Jingyu Chen, Yuanjie Dang, Hongcheng Luo, Yuesheng Tang, Chunyuan Liao, Peng Chen, and Kwang-Ting Cheng. Fast depth prediction and obstacle avoidance on a monocular drone using probabilistic convolutional neural network. IEEE Transactions on Intelligent Transportation Systems, 2019.

[58] Huaizu Jiang, Gustav Larsson, Michael Maire Greg Shakhnarovich, and Erik Learned-Miller. Self-supervised relative depth learning for urban scene understanding. In Proceedings of the European Conference on Computer Vision (ECCV), pages 19–35, 2018.

[59] Dariush Forouher, Marvin Große Besselmann, and Erik Maehle. Sensor fusion of depth camera and ultrasound data for obstacle detection and robot navigation. In 2016 14th International Conference on Control, Automation, Robotics and Vision (ICARCV), pages 1–6, 2016.

[60] Liang Wang and Ruigang Yang. Global stereo matching leveraged by sparse ground control points. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 3033–3040, 2011.

[61] Mircea Paul Muresan, Mihai Negru, and Sergiu Nedevschi. Improving local stereo algorithms using binary shifted windows, fusion and smoothness constraint. In 2015 IEEE International Conference on Intelligent Computer Communication and Processing (ICCP), pages 179–185. IEEE, 2015.

[62] Robert Spangenberg, Tobias Langner, Sven Adfeldt, and Raúl Rojas. Large scale semi-global matching on the CPU. In 2014 IEEE Intelligent Vehicles Symposium Proceedings, pages 195–201. IEEE, 2014.

[63] Andreas Wedel, Uwe Franke, Jens Klappstein, Thomas Brox, and Daniel Cremers. Realtime depth estimation and obstacle detection from monocular video. In Joint Pattern Recognition Symposium, pages 475–484. Springer, 2006.

[64] Charan D Prakash, Jinjin Li, Farshad Akhbari, and Lina J Karam. Sparse depth calculation using real-time key-point detection and structure from motion for advanced driver assist systems. In International Symposium on Visual Computing, pages 740–751. Springer, 2014.

[65] Hyowon Ha, Sunghoon Im, Jaesik Park, Hae-Gon Jeon, and In So Kweon. High-quality depth from uncalibrated small motion clip. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 5413–5421, 2016.

[66] Hossein Javidnia and Peter Corcoran. Accurate depth map estimation from small motions. In Proceedings of the IEEE International Conference on Computer Vision Workshops, pages 2453–2461, 2017.

[67] Antonio Torralba and Aude Oliva. Depth estimation from image structure. IEEE Transactions on Pattern Analysis and Machine Intelligence, 24(9):1226–1238, 2002.

[68] Ashutosh Saxena, Sung H Chung, and Andrew Y Ng. Learning depth from single monocular images. In Advances in Neural Information Processing Systems, pages 1161–1168, 2006.

[69] Jae-Il Jung and Yo-Sung Ho. Depth map estimation from single-view image using object classification based on bayesian learning. In 2010 3DTV-Conference: The True Vision-Capture, Transmission and Display of 3D Video, pages 1–4, 2010.

[70] Beyang Liu, Stephen Gould, and Daphne Koller. Single image depth estimation from predicted semantic labels. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 1253–1260, 2010.

[71] Lubor Ladicky, Jianbo Shi, and Marc Pollefeys. Pulling things out of perspective. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 89–96, 2014.

[72] Miaomiao Liu, Mathieu Salzmann, and Xuming He. Discrete-continuous depth estimation from a single image. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 716–723, 2014.

[73] Kevin Karsch, Ce Liu, and Sing Bing Kang. Depth transfer: Depth extraction from video using non-parametric sampling. IEEE Transactions on Pattern Analysis and Machine Intelligence, 36(11):2144–2158, 2014.

[74] S Hussain Raza, Omar Javed, Aveek Das, Harpreet Sawhney, Hui Cheng, and Irfan Essa. Depth extraction from videos using geometric context and occlusion boundaries. arXiv preprint arXiv:1510.07317, 2015.

[75] David Eigen and Rob Fergus. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In Proceedings of the IEEE/CVF International Conference on Computer Vision, pages 2650–2658, 2015.

[76] Iro Laina, Christian Rupprecht, Vasileios Belagiannis, Federico Tombari, and Nassir Navab. Deeper depth prediction with fully convolutional residual networks. In 2016 Fourth international conference on 3D vision (3DV), pages 239–248. IEEE, 2016.

[77] Yuanzhouhan Cao, Zifeng Wu, and Chunhua Shen. Estimating depth from monocular images as classification using deep fully convolutional residual networks. IEEE Transactions on Circuits and Systems for Video Technology, 28(11):3174–3182, 2017.

[78] Dan Xu, Elisa Ricci, Wanli Ouyang, Xiaogang Wang, and Nicu Sebe. Multi-scale continuous CRFs as sequential deep networks for monocular depth estimation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 5354–5362, 2017.

[79] Bo Li, Yuchao Dai, and Mingyi He. Monocular depth estimation with hierarchical fusion of dilated CNNs and soft-weighted-sum inference. Pattern Recognition, 83:328–339, 2018.

[80] Junjie Hu, Mete Ozay, Yan Zhang, and Takayuki Okatani. Revisiting single image depth estimation: Toward higher resolution maps with accurate object boundaries. In 2019 IEEE Winter Conference on Applications of Computer Vision (WACV), pages 1043–1051. IEEE, 2019.

[81] Diana Wofk, Fangchang Ma, Tien-Ju Yang, Sertac Karaman, and Vivienne Sze. FastDepth: Fast monocular depth estimation on embedded systems. arXiv preprint arXiv:1903.03273, 2019.

[82] Ashutosh Saxena, Min Sun, and Andrew Y Ng. Make3D: Learning 3D scene structure from a single still image. IEEE Transactions on Pattern Analysis and Machine Intelligence, 31(5):824–840, 2008.

[83] Fayao Liu, Chunhua Shen, Guosheng Lin, and Ian Reid. Learning depth from single monocular images using deep convolutional neural fields. IEEE Transactions on Pattern Analysis and Machine Intelligence, 38(10):2024–2039, 2015.

[84] Ravi Garg, Vijay Kumar BG, Gustavo Carneiro, and Ian Reid. Unsupervised CNN for single view depth estimation: Geometry to the rescue. In European Conference on Computer Vision, pages 740–756. Springer, 2016.

[85] Clément Godard, Oisin Mac Aodha, and Gabriel J Brostow. Unsupervised monocular depth estimation with left-right consistency. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 270–279, 2017.

[86] Tinghui Zhou, Matthew Brown, Noah Snavely, and David G Lowe. Unsupervised learning of depth and ego-motion from video. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 1851–1858, 2017.

[87] Yevhen Kuznietsov, Jorg Stuckler, and Bastian Leibe. Semi-supervised deep learning for monocular depth map prediction. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 6647–6655, 2017.

[88] Filippo Aleotti, Fabio Tosi, Matteo Poggi, and Stefano Mattoccia. Generative adversarial networks for unsupervised monocular depth prediction. In Proceedings of the European Conference on Computer Vision Workshops, pages 337–354, 2018.

[89] Matteo Poggi, Filippo Aleotti, Fabio Tosi, and Stefano Mattoccia. Towards real-time unsupervised monocular depth estimation on CPU. In 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 5848–5854, 2018.

[90] Nikolaos Zioulis, Antonis Karakottas, Dimitrios Zarpalas, and Petros Daras. OmniDepth: Dense depth estimation for indoors spherical panoramas. In Proceedings of the European Conference on Computer Vision (ECCV), pages 448–465, 2018.

[91] Andrew Spek, Thanuja Dharmasiri, and Tom Drummond. CReaM: Condensed real-time models for depth prediction using convolutional neural networks. In 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 540–547, 2018.

[92] Jaehoon Cho, Dongbo Min, Youngjung Kim, and Kwanghoon Sohn. A large RGB-D dataset for semi-supervised monocular depth estimation. arXiv preprint arXiv:1904.10230, 2019.

[93] Shariq Farooq Bhat, Ibraheem Alhashim, and Peter Wonka. AdaBins: Depth estimation using adaptive bins. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 4009–4018, 2021.

[94] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The Cityscapes dataset for semantic urban scene understanding. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 3213–3223, 2016.

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

[96] Yohann Cabon, Naila Murray, and Martin Humenberger. Virtual KITTI 2. arXiv preprint arXiv:2001.10773, 2020.

[97] Jürgen Sturm, Nikolas Engelhard, Felix Endres, Wolfram Burgard, and Daniel Cremers. A benchmark for the evaluation of RGB-D SLAM systems. In 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 573–580. IEEE, 2012.

[98] Shuran Song, Samuel P Lichtenberg, and Jianxiong Xiao. SUN RGB-D: A RGB-D scene understanding benchmark suite. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 567–576, 2015.

[99] Weifeng Chen, Zhao Fu, Dawei Yang, and Jia Deng. Single-image depth perception in the wild. In Advances in Neural Information Processing Systems, pages 730–738, 2016.

[100] Oliver Wasenmüller, Marcel Meyer, and Didier Stricker. CoRBS: Comprehensive RGB-D benchmark for SLAM using Kinect v2. In 2016 IEEE Winter Conference on Applications of Computer Vision (WACV), pages 1–7. IEEE, 2016.

[101] Iro Armeni, Sasha Sax, Amir R Zamir, and Silvio Savarese. Joint 2D-3D-semantic data for indoor scene understanding. arXiv preprint arXiv:1702.01105, 2017.

[102] Thomas Schops, Johannes L Schonberger, Silvano Galliani, Torsten Sattler, Konrad Schindler, Marc Pollefeys, and Andreas Geiger. A multi-view stereo benchmark with high-resolution images and multi-camera videos. In Proceedings

of the IEEE Conference on Computer Vision and Pattern Recognition, pages 3260–3269, 2017.

[103] Angel Chang, Angela Dai, Thomas Funkhouser, Maciej Halber, Matthias Niessner, Manolis Savva, Shuran Song, Andy Zeng, and Yinda Zhang. Matterport3D: Learning from RGB-D data in indoor environments. arXiv preprint arXiv:1709.06158, 2017.

[104] Angela Dai, Angel X Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. ScanNet: Richly-annotated 3D reconstructions of indoor scenes. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 5828–5839, 2017.

[105] John McCormac, Ankur Handa, Stefan Leutenegger, and Andrew J Davison. SceneNet RGB-D: Can 5M synthetic images beat generic ImageNet pre-training on indoor segmentation? In Proceedings of the IEEE International Conference on Computer Vision, pages 2678–2687, 2017.

[106] Shuran Song, Fisher Yu, Andy Zeng, Angel X Chang, Manolis Savva, and Thomas Funkhouser. Semantic scene completion from a single depth image. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 1746–1754, 2017.

[107] Zhengqi Li and Noah Snavely. MegaDepth: Learning single-view depth prediction from internet photos. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 2041–2050, 2018.

[108] Alina Marcu, Dragos Costea, Vlad Licaret, Mihai Pîrvu, Emil Slusanschi, and Marius Leordeanu. SafeUAV: Learning to estimate depth and safe landing areas for UAVs from synthetic data. In Proceedings of the European Conference on Computer Vision (ECCV) Workshops, pages 43–58, 2018.

[109] Shouchuan Wu, Haitao Zhao, and Shaoyuan Sun. Depth estimation from infrared video using local-feature-flow neural network. International Journal of Machine Learning and Cybernetics, 10(9):2563–2572, 2019.

[110] Guorun Yang, Xiao Song, Chaoqin Huang, Zhidong Deng, Jianping Shi, and Bolei Zhou. DrivingStereo: A large-scale dataset for stereo matching in autonomous driving scenarios. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 899–908, 2019.

[111] Igor Vasiljevic, Nick Kolkin, Shanyi Zhang, Ruotian Luo, Haochen Wang, Falcon Z Dai, Andrea F Daniele, Mohammadreza Mostajabi, Steven Basart, Matthew R Walter, et al. DIODE: A dense indoor and outdoor depth dataset. arXiv preprint arXiv:1908.00463, 2019.

[112] Michael Fonder and Marc Van Droogenbroeck. Mid-Air: A multi-modal dataset for extremely low altitude drone flights. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops, pages 0–0, 2019.

[113] Chaoyue Niu, Danesh Tarapore, and Klaus-Peter Zauner. Low-viewpoint forest depth dataset for sparse rover swarms. arXiv preprint arXiv:2003.04359, 2020.

[114] Lei Jin, Yanyu Xu, Jia Zheng, Junfei Zhang, Rui Tang, Shugong Xu, Jingyi Yu, and Shenghua Gao. Geometric structure based and regularized depth estimation from 360 indoor imagery. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 889–898, 2020.

[115] Shimon Ullman. The interpretation of structure from motion. Proceedings of the Royal Society of London. Series B. Biological Sciences, 203(1153):405–426, 1979.

[116] Berthold KP Horn. Shape from shading: A method for obtaining the shape of a smooth opaque object from one view. Technical Report, 1970.

[117] Harry Barrow, J Tenenbaum, A Hanson, and E Riseman. Recovering intrinsic scene characteristics. Computer Vision Systems, 2(3-26):2, 1978.

[118] Naejin Kong and Michael J Black. Intrinsic depth: Improving depth transfer with intrinsic images. In Proceedings of the IEEE International Conference on Computer Vision, pages 3514–3522, 2015.

[119] Fangchang Ma and Sertac Karaman. Sparse-to-dense: Depth prediction from sparse depth samples and a single image. In 2018 IEEE International Conference on Robotics and Automation (ICRA), pages 4796–4803, 2018.

[120] Xinjing Cheng, Peng Wang, and Ruigang Yang. Depth estimation via affinity learned with convolutional spatial propagation network. In Proceedings of the European Conference on Computer Vision (ECCV), pages 103–119, 2018.

[121] Xiaotian Chen, Xuejin Chen, and Zheng-Jun Zha. Structure-aware residual pyramid network for monocular depth estimation. arXiv preprint arXiv:1907.06023, 2019.

[122] Bo Li, Chunhua Shen, Yuchao Dai, Anton Van Den Hengel, and Mingyi He. Depth and surface normal estimation from monocular images using regression on deep features and hierarchical CRFs. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 1119–1127, 2015.

[123] Minhyeok Heo, Jaehan Lee, Kyung-Rae Kim, Han-Ul Kim, and Chang-Su Kim. Monocular depth estimation using whole strip masking and reliability-based refinement. In European Conference on Computer Vision, pages 36–51, 2018.

[124] Ishit Mehta, Parikshit Sakurikar, and PJ Narayanan. Structured adversarial training for unsupervised monocular depth estimation. In 2018 International Conference on 3D Vision (3DV), pages 314–323. IEEE, 2018.

[125] Ibraheem Alhashim and Peter Wonka. High quality monocular depth estimation via transfer learning. arXiv preprint arXiv:1812.11941, 2018.

[126] Huan Fu, Mingming Gong, Chaohui Wang, Kayhan Batmanghelich, and Dacheng Tao. Deep ordinal regression network for monocular depth estimation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 2002–2011, 2018.

[127] Xiaoyang Guo, Hongsheng Li, Shuai Yi, Jimmy Ren, and Xiaogang Wang. Learning monocular depth by distilling cross-domain stereo networks. In European Conference on Computer Vision, pages 484–500, 2018.

[128] Jia-Wang Bian, Zhichao Li, Naiyan Wang, Huangying Zhan, Chunhua Shen, Ming-Ming Cheng, and Ian Reid. Unsupervised scale-consistent depth and ego-motion learning from monocular video. arXiv preprint arXiv:1908.10553, 2019.

[129] Jin Han Lee, Myung-Kyu Han, Dong Wook Ko, and Il Hong Suh. From big to small: Multi-scale local planar guidance for monocular depth estimation. arXiv preprint arXiv:1907.10326, 2019.

[130] Lukas Liebel and Marco Körner. MultiDepth: Single-image depth estimation via multi-task regression and classification. arXiv preprint arXiv:1907.11111, 2019.

[131] Vladimir Nekrasov, Thanuja Dharmasiri, Andrew Spek, Tom Drummond, Chunhua Shen, and Ian Reid. Real-time joint semantic segmentation and depth estimation using asymmetric annotations. In 2019 International Conference on Robotics and Automation (ICRA), pages 7101–7107. IEEE, 2019.

[132] Jiaxiong Qiu, Zhaopeng Cui, Yinda Zhang, Xingdi Zhang, Shuaicheng Liu, Bing Zeng, and Marc Pollefeys. DeepLiDAR: Deep surface normal guided depth prediction for outdoor scene from sparse LiDAR data and single color image. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 3313–3322, 2019.

[133] Wei Yin, Yifan Liu, Chunhua Shen, and Youliang Yan. Enforcing geometric constraints of virtual normal for depth prediction. In Proceedings of the IEEE International Conference on Computer Vision, pages 5684–5693, 2019.

[134] Zhicheng Fang, Xiaoran Chen, Yuhua Chen, and Luc Van Gool. Towards good practice for CNN-based monocular depth estimation. In Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision, pages 1091–1100, 2020.

[135] Kourosh Sartipi, Tien Do, Tong Ke, Khiem Vuong, and Stergios I Roumeliotis. Deep depth estimation from visual-inertial SLAM. arXiv preprint arXiv:2008.00092, 2020.

[136] Ke Xian, Jianming Zhang, Oliver Wang, Long Mai, Zhe Lin, and Zhiguo Cao. Structure-guided ranking loss for single image depth prediction. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 611–620, 2020.

[137] Andrea Pilzer, Dan Xu, Mihai Puscas, Elisa Ricci, and Nicu Sebe. Unsupervised adversarial depth estimation using cycled generative networks. In 2018 International Conference on 3D Vision (3DV), pages 587–595. IEEE, 2018.

[138] Xiaojuan Qi, Renjie Liao, Zhengzhe Liu, Raquel Urtasun, and Jiaya Jia. GeoNet: Geometric neural network for joint depth and surface normal estimation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 283–291, 2018.

[139] Huangying Zhan, Ravi Garg, Chamara Saroj Weerasekera, Kejie Li, Harsh Agarwal, and Ian Reid. Unsupervised learning of monocular depth estimation and visual odometry with deep feature reconstruction. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 340–349, 2018.

[140] Vincent Casser, Soeren Pirk, Reza Mahjourian, and Anelia Angelova. Depth prediction without the sensors: Leveraging structure for unsupervised learning from monocular videos. In Proceedings of the AAAI conference on Artificial Intelligence, volume 33, pages 8001–8008, 2019.

[141] Sara Elkerdawy, Hong Zhang, and Nilanjan Ray. Lightweight monocular depth estimation model by joint end-to-end filter pruning. arXiv preprint arXiv:1905.05212, 2019.

[142] Xiaohan Fei, Alex Wong, and Stefano Soatto. Geo-supervised visual depth prediction. IEEE Robotics and Automation Letters, 4(2):1661–1668, 2019.

[143] Clément Godard, Oisin Mac Aodha, Michael Firman, and Gabriel J Brostow. Digging into self-supervised monocular depth estimation. In Proceedings of the IEEE/CVF International Conference on Computer Vision, pages 3828–3838, 2019.

[144] Anurag Ranjan, Varun Jampani, Lukas Balles, Kihwan Kim, Deqing Sun, Jonas Wulff, and Michael J Black. Competitive collaboration: Joint unsupervised learning of depth, camera motion, optical flow and motion segmentation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 12240–12249, 2019.

[145] Fabio Tosi, Filippo Aleotti, Matteo Poggi, and Stefano Mattoccia. Learning monocular depth estimation infusing traditional stereo knowledge. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 9799–9809, 2019.

[146] Jamie Watson, Michael Firman, Gabriel J Brostow, and Daniyar Turmukhambetov. Self-supervised monocular depth hints. In Proceedings of the IEEE International Conference on Computer Vision, pages 2162–2171, 2019.

[147] Nikolaos Zioulis, Antonis Karakottas, Dimitrios Zarpalas, Federico Alvarez, and Petros Daras. Spherical view synthesis for self-supervised 360 depth estimation.

In 2019 International Conference on 3D Vision (3DV), pages 690–699. IEEE, 2019.

[148] Vitor Guizilini, Rares Ambrus, Sudeep Pillai, Allan Raventos, and Adrien Gaidon. 3D packing for self-supervised monocular depth estimation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 2485–2494, 2020.

[149] Marvin Klingner, Jan-Aike Termöhlen, Jonas Mikolajczyk, and Tim Fingscheidt. Self-supervised monocular depth estimation: Solving the dynamic object problem by semantic guidance. arXiv preprint arXiv:2007.06936, 2020.

[150] Kuo-Shiuan Peng, Gregory Ditzler, and Jerzy Rozenblit. Edge-guided occlusion fading reduction for a light-weighted self-supervised monocular depth estimation. arXiv preprint arXiv:1911.11705, 2019.

[151] Chang Shu, Kun Yu, Zhixiang Duan, and Kuiyuan Yang. Feature-metric loss for self-supervised learning of depth and egomotion. arXiv preprint arXiv:2007.10603, 2020.

[152] Feng Xue, Guirong Zhuo, Ziyuan Huang, Wufei Fu, Zhuoyue Wu, and Marcelo H Ang Jr. Toward hierarchical self-supervised monocular absolute depth estimation for autonomous driving applications. arXiv preprint arXiv:2004.05560, 2020.

[153] Pierluigi Zama Ramirez, Matteo Poggi, Fabio Tosi, Stefano Mattoccia, and Luigi Di Stefano. Geometry meets semantics for semi-supervised monocular depth estimation. In Asian Conference on Computer Vision, pages 298–313, 2018.

[154] Ali Jahani Amiri, Shing Yan Loo, and Hong Zhang. Semi-supervised monocular depth estimation with left-right consistency using deep neural network. In 2019 IEEE International Conference on Robotics and Biomimetics (ROBIO), pages 602–607, 2019.

[155] Amir Atapour-Abarghouei and Toby P Breckon. Real-time monocular depth estimation using synthetic data with domain adaptation via image style transfer. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 2800–2810, 2018.

[156] Chuanxia Zheng, Tat-Jen Cham, and Jianfei Cai. T2Net: Synthetic-to-realistic translation for solving single-image depth estimation tasks. In Proceedings of the European conference on computer vision (ECCV), pages 767–783, 2018.

[157] Shanshan Zhao, Huan Fu, Mingming Gong, and Dacheng Tao. Geometry-aware symmetric domain adaptation for monocular depth estimation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 9788–9798, 2019.

[158] Fu-En Wang, Yu-Hsuan Yeh, Min Sun, Wei-Chen Chiu, and Yi-Hsuan Tsai. BiFuse: Monocular 360 depth estimation via bi-projection fusion. In Proceedings

of the IEEE Conference on Computer Vision and Pattern Recognition, pages 462–471, 2020.

[159] Arun CS Kumar, Suchendra M Bhandarkar, and Mukta Prasad. DepthNet: A recurrent neural network architecture for monocular depth prediction. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, pages 283–291, 2018.

[160] Haokui Zhang, Chunhua Shen, Ying Li, Yuanzhouhan Cao, Yu Liu, and Youliang Yan. Exploiting temporal consistency for real-time video depth estimation. In Proceedings of the IEEE International Conference on Computer Vision, pages 1725–1734, 2019.

[161] Rui Wang, Stephen M Pizer, and Jan-Michael Frahm. Recurrent neural network for (Un-) supervised learning of monocular video visual odometry and depth. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 5555–5564, 2019.

[162] Ruibo Li, Ke Xian, Chunhua Shen, Zhiguo Cao, Hao Lu, and Lingxiao Hang. Deep attention-based classification network for robust depth prediction. In Asian Conference on Computer Vision, pages 663–678. Springer, 2018.

[163] Yuru Chen, Haitao Zhao, and Zhengwei Hu. Attention-based context aggregation network for monocular depth estimation. arXiv preprint arXiv:1901.10137, 2019.

[164] Hongwei Zou, Ke Xian, Jiaqi Yang, and Zhiguo Cao. Mean-variance loss for monocular depth estimation. In 2019 IEEE International Conference on Image Processing (ICIP), pages 1760–1764. IEEE, 2019.

[165] Wenfeng Song, Shuai Li, Ji Liu, Aimin Hao, Qinping Zhao, and Hong Qin. Contextualized CNN for scene-aware depth estimation from single RGB image. IEEE Transactions on Multimedia, 22(5):1220–1233, 2019.

[166] Dalila Sanchez-Escobedo, Xiao Lin, Josep R Casas, and Montse Pardas. HybridNet for depth estimation and semantic segmentation. In 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pages 1563–1567. IEEE, 2018.

[167] Peng Wang, Xiaohui Shen, Zhe Lin, Scott Cohen, Brian Price, and Alan L Yuille. Towards unified depth and semantic prediction from a single image. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 2800–2809, 2015.

[168] Omid Hosseini Jafari, Oliver Groth, Alexander Kirillov, Michael Ying Yang, and Carsten Rother. Analyzing modular CNN architectures for joint depth prediction and semantic segmentation. In 2017 IEEE International Conference on Robotics and Automation (ICRA), pages 4620–4627, 2017.

[169] Akhil Gurram, Onay Urfalioglu, Ibrahim Halfaoui, Fahd Bouzaraa, and Antonio M López. Monocular depth estimation by learning from heterogeneous datasets. In 2018 IEEE Intelligent Vehicles Symposium (IV), pages 2176–2181. IEEE, 2018.

[170] Jianbo Jiao, Ying Cao, Yibing Song, and Rynson Lau. Look deeper into depth: Monocular depth estimation with semantic booster and attention-driven loss. In Proceedings of the European Conference on Computer Vision (ECCV), pages 53–69, 2018.

[171] Yi-Yu Hsieh, Wei-Yu Lin, Dong-Lin Li, and Jen-Hui Chuang. Deep learning-based obstacle detection and depth estimation. In 2019 IEEE International Conference on Image Processing (ICIP), pages 1635–1639. IEEE, 2019.

[172] Saddam Abdulwahab, Hatem A Rashwan, Miguel Angel Garcia, Mohammed Jabreel, Sylvie Chambon, and Domenec Puig. Adversarial learning for depth and viewpoint estimation from a single image. IEEE Transactions on Circuits and Systems for Video Technology, 30(9):2947–2958, 2020.

[173] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 3431–3440, 2015.

[174] Joseph Redmon and Ali Farhadi. YOLOv3: An incremental improvement. arXiv preprint arXiv:1804.02767, 2018.

[175] Eduardo Romera, José M Alvarez, Luis M Bergasa, and Roberto Arroyo. ERFNet: Efficient residual factorized ConvNet for real-time semantic segmentation. IEEE Transactions on Intelligent Transportation Systems, 19(1):263–272, 2017.

[176] Linda Wang, Mahmoud Famouri, and Alexander Wong. DepthNet Nano: A highly compact self-normalizing neural network for monocular depth estimation. arXiv preprint arXiv:2004.08008, 2020.

[177] Reza Mahjourian, Martin Wicke, and Anelia Angelova. Unsupervised learning of depth and ego-motion from monocular video using 3D geometric constraints. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 5667–5675, 2018.

[178] Fangchang Ma, Guilherme Venturelli Cavalheiro, and Sertac Karaman. Self-supervised sparse-to-dense: Self-supervised depth completion from LiDAR and monocular camera. In 2019 IEEE International Conference on Robotics and Automation (ICRA), pages 3288–3295, 2019.

[179] Yilun Zhang, Ty Nguyen, Ian D Miller, Steven Chen, Camillo J Taylor, Vijay Kumar, et al. DFineNet: Ego-motion estimation and depth refinement from sparse, noisy depth input with RGB guidance. arXiv preprint arXiv:1903.06397, 2019.

[180] Vitor Guizilini, Rui Hou, Jie Li, Rares Ambrus, and Adrien Gaidon. Semantically-guided representation learning for self-supervised monocular depth. arXiv preprint arXiv:2002.12319, 2020.

[181] Adrian Johnston and Gustavo Carneiro. Self-supervised monocular trained depth estimation using self-attention and discrete disparity volume. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 4756–4765, 2020.

[182] Nikolaus Mayer, Eddy Ilg, Philip Hausser, Philipp Fischer, Daniel Cremers, Alexey Dosovitskiy, and Thomas Brox. A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 4040–4048, 2016.

[183] Vignesh Prasad and Brojeshwar Bhowmick. SfMLearner++: Learning monocular depth & ego-motion using meaningful geometric constraints. In 2019 IEEE Winter Conference on Applications of Computer Vision (WACV), pages 2087–2096. IEEE, 2019.

[184] Maria Klodt and Andrea Vedaldi. Supervising the new with the old: learning SfM from SfM. In Proceedings of the European Conference on Computer Vision, pages 698–713, 2018.

[185] Raul Mur-Artal and Juan D Tardós. ORB-SLAM2: An open-source SLAM system for monocular, stereo, and RGB-D cameras. IEEE Transactions on Robotics, 33(5):1255–1262, 2017.

[186] Sudheendra Vijayanarasimhan, Susanna Ricco, Cordelia Schmid, Rahul Sukthankar, and Katerina Fragkiadaki. SfM-Net: Learning of structure and motion from video. arXiv preprint arXiv:1704.07804, 2017.

[187] Qi Dai, Vaishakh Patil, Simon Hecker, Dengxin Dai, Luc Van Gool, and Konrad Schindler. Self-supervised object motion and depth estimation from video. arXiv preprint arXiv:1912.04250, 2019.

[188] Wang Zhao, Shaohui Liu, Yezhi Shu, and Yong-Jin Liu. Towards better generalization: joint depth-pose learning without posenet. arXiv preprint arXiv:2004.01314, 2020.

[189] Alex Kendall, Matthew Grimes, and Roberto Cipolla. PoseNet: A convolutional network for real-time 6-DoF camera relocalization. In Proceedings of the IEEE International Conference on Computer Vision, pages 2938–2946, 2015.

[190] Yuliang Zou, Zelun Luo, and Jia-Bin Huang. DF-Net: Unsupervised joint learning of depth and flow using cross-task consistency. In Proceedings of the European conference on computer vision (ECCV), pages 36–53, 2018.

[191] Zhichao Yin and Jianping Shi. GeoNet: Unsupervised learning of dense depth, optical flow and camera pose. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 1983–1992, 2018.

[192] Anjie Wang, Zhijun Fang, Yongbin Gao, Songchao Tan, Shanshe Wang, Siwei Ma, and Jenq-Neng Hwang. Adversarial learning for joint optimization of depth and ego-motion. IEEE Transactions on Image Processing, 29:4130–4142, 2020.

[193] Yasin Almalioglu, Muhamad Risqi U Saputra, Pedro PB de Gusmao, Andrew Markham, and Niki Trigoni. GANVO: Unsupervised deep monocular visual odometry and depth estimation with generative adversarial networks. In 2019 International conference on robotics and automation (ICRA), pages 5474–5480, 2019.

[194] Jun Liu, Qing Li, Rui Cao, Wenming Tang, and Guoping Qiu. MiniNet: An extremely lightweight convolutional neural network for real-time unsupervised monocular depth estimation. ISPRS Journal of Photogrammetry and Remote Sensing, 166:255–267, 2020.

[195] Hu Tian and Fei Li. Semi-supervised depth estimation from a single image based on confidence learning. In ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pages 8573–8577. IEEE, 2019.

[196] Rongrong Ji, Ke Li, Yan Wang, Xiaoshuai Sun, Feng Guo, Xiaowei Guo, Yongjian Wu, Feiyue Huang, and Jiebo Luo. Semi-supervised adversarial monocular depth estimation. IEEE Transactions on Pattern Analysis and Machine Intelligence, 42(10):2410–2422, 2019.

[197] Vitor Guizilini, Jie Li, Rares Ambrus, Sudeep Pillai, and Adrien Gaidon. Robust semi-supervised monocular depth estimation with reprojected distances. In Conference on Robot Learning, pages 503–512. PMLR, 2020.

[198] Min Yue, Guangyuan Fu, Ming Wu, and Hongqiao Wang. Semi-supervised monocular depth estimation based on semantic supervision. Journal of Intelligent and Robotic Systems, 100(2):455–463, 2020.

[199] Jiahao Pang, Wenxiu Sun, Jimmy SJ Ren, Chengxi Yang, and Qiong Yan. Cascade residual learning: A two-stage convolutional neural network for stereo matching. In Proceedings of the IEEE International Conference on Computer Vision, pages 887–895, 2017.

[200] Alessio Tonioni, Matteo Poggi, Stefano Mattoccia, and Luigi Di Stefano. Unsupervised domain adaptation for depth prediction from images. IEEE transactions on pattern analysis and machine intelligence, 42(10):2396–2409, 2019.

[201] Yongcheng Jing, Yezhou Yang, Zunlei Feng, Jingwen Ye, Yizhou Yu, and Mingli Song. Neural style transfer: A review. IEEE Transactions on Visualization and Computer Graphics, 26(11):3365–3385, 2019.

[202] Yiyi Liao, Lichao Huang, Yue Wang, Sarath Kodagoda, Yinan Yu, and Yong Liu. Parse geometry from a line: Monocular depth estimation with partial laser observation. In 2017 IEEE international conference on robotics and automation (ICRA), pages 5059–5066. IEEE, 2017.

[203] Maximilian Jaritz, Raoul De Charette, Emilie Wirbel, Xavier Perrotton, and Fawzi Nashashibi. Sparse and dense data with CNNs: Depth completion and semantic segmentation. In 2018 International Conference on 3D Vision (3DV), pages 52–60. IEEE, 2018.

[204] Tsun-Hsuan Wang, Fu-En Wang, Juan-Ting Lin, Yi-Hsuan Tsai, Wei-Chen Chiu, and Min Sun. Plug-and-play: Improve depth prediction via sparse data propagation. In 2019 International Conference on Robotics and Automation (ICRA), pages 5880–5886. IEEE, 2019.

[205] Yun Chen, Bin Yang, Ming Liang, and Raquel Urtasun. Learning joint 2D-3D representations for depth completion. In Proceedings of the IEEE/CVF International Conference on Computer Vision, pages 10023–10032, 2019.

[206] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. ORB-SLAM: a versatile and accurate monocular SLAM system. IEEE transactions on robotics, 31(5):1147–1163, 2015.

[207] Vaishakh Patil, Wouter Van Gansbeke, Dengxin Dai, and Luc Van Gool. Dont forget the past: Recurrent depth estimation from monocular video. IEEE Robotics and Automation Letters, 5(4):6813–6820, 2020.

[208] Nícolas dos Santos Rosa, Vitor Guizilini, and Valdir Grassi. Sparse-to-continuous: Enhancing monocular depth estimation using occupancy maps. In 2019 19th International Conference on Advanced Robotics (ICAR), pages 793–800, 2019.

[209] Yunhan Zhao, Shu Kong, Daeyun Shin, and Charless Fowlkes. Domain decluttering: simplifying images to mitigate synthetic-real domain shift and improve depth estimation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 3330–3340, 2020.

[210] Ekim Yurtsever, Jacob Lambert, Alexander Carballo, and Kazuya Takeda. A survey of autonomous driving: Common practices and emerging technologies. IEEE Access, 8:58443–58469, 2020.

[211] Sangyun Oh, Hye-Jin S Kim, Jongeun Lee, and Junmo Kim. RRNet: Repetition-reduction network for energy efficient depth estimation. IEEE Access, 8:106097–106108, 2020.

[212] Xiaochuan Yin, Xiangwei Wang, Xiaoguo Du, and Qijun Chen. Scale recovery for monocular visual odometry using depth estimated with deep convolutional neural fields. In Proceedings of the IEEE International Conference on Computer Vision, pages 5870–5878, 2017.

[213] Hongcheng Luo, Yang Gao, Yuhao Wu, Chunyuan Liao, Xin Yang, and Kwang-Ting Cheng. Real-time dense monocular SLAM with online adapted depth prediction network. IEEE Transactions on Multimedia, 21(2):470–483, 2018.

[214] Nan Yang, Rui Wang, Jorg Stuckler, and Daniel Cremers. Deep virtual stereo odometry: Leveraging deep depth prediction for monocular direct sparse odometry. In Proceedings of the European Conference on Computer Vision (ECCV), pages 817–833, 2018.

[215] Raul de Queiroz Mendes, Eduardo Godinho Ribeiro, Nicolas dos Santos Rosa, and Valdir Grassi Jr. On deep learning techniques to boost monocular depth estimation for autonomous navigation. arXiv preprint arXiv:2010.06626, 2020.

[216] Jeff Michels, Ashutosh Saxena, and Andrew Y Ng. High speed obstacle avoidance using monocular vision and reinforcement learning. In International Conference on Machine Learning, pages 593–600, 2005.

[217] H Alvarez, Lina María Paz, Jürgen Sturm, and Daniel Cremers. Collision avoidance for quadrotors with a monocular camera. In Experimental Robotics, pages 195–209. Springer, 2016.

[218] Punarjay Chakravarty, Klaas Kelchtermans, Tom Roussel, Stijn Wellens, Tinne Tuytelaars, and Luc Van Eycken. CNN-based single image obstacle avoidance on a quadrotor. In 2017 IEEE International Conference on Robotics and Automation (ICRA), pages 6369–6374. IEEE, 2017.

[219] Zhenghong Zhang, Mingkang Xiong, and Huilin Xiong. Monocular depth estimation for UAV obstacle avoidance. In 2019 4th International Conference on Cloud Computing and Internet of Things (CCIOT), pages 43–47. IEEE, 2019.

[220] Jiahao Lin, Hai Zhu, and Javier Alonso-Mora. Robust vision-based obstacle avoidance for micro aerial vehicles in dynamic environments. arXiv preprint arXiv:2002.04920, 2020.

[221] Timo Scharwächter and Uwe Franke. Low-level fusion of color, texture and depth for robust road scene understanding. In 2015 IEEE Intelligent Vehicles Symposium (IV), pages 599–604. IEEE, 2015.

[222] L.Oyuki RoJas-Perez, Roberto Munguia-Silva, and Jose Martinez-Carranza. Real-time landing zone detection for UAVs using single aerial images. In Proceedings of the International Micro-Air Vehicles Conference, 2018.

[223] Ping Li, Matthew Garratt, Andrew Lambert, and Shanggang Lin. Metric sensing and control of a quadrotor using a homography-based visual inertial fusion method. Robotics and Autonomous Systems, 76:1–14, 2016.

[224] Jiefei Wang, Matthew Garratt, Sreenatha Anavatti, and Shanggang Lin. Real-time visual odometry for autonomous MAV navigation using RGB-D

camera. In 2015 IEEE International Conference on Robotics and Biomimetics (ROBIO), pages 1353–1358. IEEE, 2015.

[225] Jakob Engel, Thomas Schöps, and Daniel Cremers. LSD-SLAM: Large-scale direct monocular SLAM. In European Conference on Computer Vision, pages 834–849. Springer, 2014.

[226] Shing Yan Loo, Ali Jahani Amiri, Syamsiah Mashohor, Sai Hong Tang, and Hong Zhang. CNN-SVO: Improving the mapping in semi-direct visual odometry using single-image depth prediction. In 2019 International Conference on Robotics and Automation (ICRA), pages 5218–5223. IEEE, 2019.

[227] Larry Matthies, Roland Brockers, Yoshiaki Kuwata, and Stephan Weiss. Stereo vision-based obstacle avoidance for micro air vehicles using disparity space. In In 2014 IEEE International Conference on Robotics and Automation (ICRA), pages 3242–3249, 2014.

[228] Roland Brockers, Anthony Fragoso, Brandon Rothrock, Connor Lee, and Larry Matthies. Vision-based obstacle avoidance for micro air vehicles using an egocylindrical depth map. In International Symposium on Experimental Robotics, pages 505–514, 2016.

[229] Nicolai Wojke and Marcel Häselich. Moving vehicle detection and tracking in unstructured environments. In 2012 IEEE International Conference on Robotics and Automation, pages 3082–3087. IEEE, 2012.

[230] Dominic Zeng Wang, Ingmar Posner, and Paul Newman. What could move? finding cars, pedestrians and bicyclists in 3D laser data. In 2012 IEEE International Conference on Robotics and Automation, pages 4038–4044, 2012.

[231] Andrea Cherubini, Fabien Spindler, and Francois Chaumette. Autonomous visual navigation and laser-based moving obstacle avoidance. IEEE Transactions on Intelligent Transportation Systems, 15(5):2101–2110, 2014.

[232] Junjie Hu, Yan Zhang, and Takayuki Okatani. Visualization of convolutional neural networks for monocular depth estimation. In Proceedings of the IEEE/CVF International Conference on Computer Vision, pages 3869–3878, 2019.

[233] Tom van Dijk and Guido de Croon. How do neural networks see depth in single images? In Proceedings of the IEEE/CVF International Conference on Computer Vision, pages 2183–2191, 2019.

[234] Philippe Weinzaepfel, Jerome Revaud, Zaid Harchaoui, and Cordelia Schmid. DeepFlow: Large displacement optical flow with deep matching. In Proceedings of the IEEE International Conference on Computer Vision, pages 1385–1392, 2013.

[235] Edgar Simo-Serra, Eduard Trulls, Luis Ferraz, Iasonas Kokkinos, Pascal Fua, and Francesc Moreno-Noguer. Discriminative learning of deep convolutional feature point descriptors. In Proceedings of the IEEE International Conference on Computer Vision, pages 118–126, 2015.

[236] Eddy Ilg, Nikolaus Mayer, Tonmoy Saikia, Margret Keuper, Alexey Dosovitskiy, and Thomas Brox. FlowNet 2.0: Evolution of optical flow estimation with deep networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 2462–2470, 2017.

[237] Anurag Ranjan and Michael J Black. Optical flow estimation using a spatial pyramid network. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 4161–4170, 2017.

[238] Deqing Sun, Xiaodong Yang, Ming-Yu Liu, and Jan Kautz. PWC-Net: CNNs for optical flow using pyramid, warping, and cost volume. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 8934–8943, 2018.

[239] Sunggoo Jung, Sunyou Hwang, Heemin Shin, and David Hyunchul Shim. Perception, guidance, and navigation for indoor autonomous drone racing using deep learning. IEEE Robotics and Automation Letters, 3(3):2539–2544, 2018.

[240] Alessandro Giusti, Jérôme Guzzi, Dan C Cireşan, Fang-Lin He, Juan P Rodríguez, Flavio Fontana, Matthias Faessler, Christian Forster, Jürgen Schmidhuber, Gianni Di Caro, et al. A machine learning approach to visual perception of forest trails for mobile robots. IEEE Robotics and Automation Letters, 1(2):661–667, 2015.

[241] Nikolai Smolyanskiy, Alexey Kamenev, Jeffrey Smith, and Stan Birchfield. Toward low-flying autonomous MAV trail navigation using deep neural networks for environmental awareness. In 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 4241–4247. IEEE, 2017.

[242] Bruna G Maciel-Pearson, Samet Akçay, Amir Atapour-Abarghouei, Christopher Holder, and Toby P Breckon. Multi-task regression-based learning for autonomous unmanned aerial vehicle flight control within unstructured outdoor environments. IEEE Robotics and Automation Letters, 4(4):4116–4123, 2019.

[243] Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne. Imitation learning: A survey of learning methods. ACM Computing Surveys (CSUR), 50(2):1–35, 2017.

[244] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. Nature, 518(7540):529–533, 2015.

[245] Matthew Hausknecht and Peter Stone. Deep recurrent Q-learning for partially observable MDPs. In 2015 AAAI Fall Symposium Series, 2015.

[246] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double Q-learning. In Proceedings of the AAAI conference on artificial intelligence, volume 30, 2016.

[247] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In International Conference on Machine Learning, pages 1995–2003. PMLR, 2016.

[248] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971, 2015.

[249] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347, 2017.

[250] Chao Wang, Jian Wang, Xudong Zhang, and Xiao Zhang. Autonomous navigation of UAV in large-scale unknown complex environment with deep reinforcement learning. In 2017 IEEE Global Conference on Signal and Information Processing (GlobalSIP), pages 858–862. Ieee, 2017.

[251] Jiaqi Xiang, Qingdong Li, Xiwang Dong, and Zhang Ren. Continuous control with deep reinforcement learning for mobile robot navigation. In 2019 Chinese Automation Congress (CAC), pages 1501–1506. IEEE, 2019.

[252] Gabriel Moraes Barros and Esther Luna Colombini. Using soft actor-critic for low-level UAV control. arXiv preprint arXiv:2010.02293, 2020.

[253] Daniel Dugas, Juan Nieto, Roland Siegwart, and Jen Jen Chung. NavRep: Unsupervised representations for reinforcement learning of robot navigation in dynamic human environments. In 2021 IEEE International Conference on Robotics and Automation (ICRA), pages 7829–7835. IEEE, 2021.

[254] Linh Kästner, Teham Buiyan, Xinlin Zhao, Lei Jiao, Zhengcheng Shen, and Jens Lambrecht. Towards deployment of deep-reinforcement-learning-based obstacle avoidance into conventional autonomous navigation systems. arXiv preprint arXiv:2104.03616, 2021.

[255] Steven M LaValle et al. Rapidly-exploring random trees: A new tool for path planning. Technical Report, 1998.

[256] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. IEEE transactions on Systems Science and Cybernetics, 4(2):100–107, 1968.

[257] Edsger W Dijkstra. A note on two problems in connexion with graphs. Numerische mathematik, 1(1):269–271, 1959.

[258] Xueqin Huang, Han Deng, Wei Zhang, Ran Song, and Yibin Li. Towards multi-modal perception-based navigation: A deep reinforcement learning method. IEEE Robotics and Automation Letters, 6(3):4986–4993, 2021.

[259] Linhai Xie, Sen Wang, Andrew Markham, and Niki Trigoni. Towards monocular vision based obstacle avoidance through deep reinforcement learning. arXiv preprint arXiv:1706.09829, 2017.

[260] Abhik Singla, Sindhu Padakandla, and Shalabh Bhatnagar. Memory-based deep reinforcement learning for obstacle avoidance in UAV with limited environment knowledge. IEEE Transactions on Intelligent Transportation Systems, 22(1):107–118, 2019.

[261] Deepak-George Thomas, Daniil Olshanskyi, Karter Krueger, Tichakorn Wongpiromsarn, and Ali Jannesari. Interpretable UAV collision avoidance using deep reinforcement learning. arXiv preprint arXiv:2105.12254, 2021.

[262] Zhihan Xue and Tad Gonsalves. Monocular vision obstacle avoidance UAV: A deep reinforcement learning method. In 2021 2nd International Conference on Innovative and Creative Information Technology (ICITech), pages 1–6. IEEE, 2021.

[263] Lingping Gao, Jianchuan Ding, Wenxi Liu, Haiyin Piao, Yuxin Wang, Xin Yang, and Baocai Yin. A vision-based irregular obstacle avoidance framework via deep reinforcement learning. In 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 9262–9269. IEEE, 2021.

[264] Brian Yamauchi. A frontier-based approach for autonomous exploration. In Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA'97.'Towards New Computational Principles for Robotics and Automation', pages 146–151. IEEE, 1997.

[265] Delong Zhu, Tingguang Li, Danny Ho, Chaoqun Wang, and Max Q-H Meng. Deep reinforcement learning supervised autonomous exploration in office environments. In 2018 IEEE International Conference on Robotics and Automation (ICRA), pages 7548–7555. IEEE, 2018.

[266] Farzad Niroui, Kaicheng Zhang, Zendai Kashino, and Goldie Nejat. Deep reinforcement learning robot for search and rescue applications: Exploration in unknown cluttered environments. IEEE Robotics and Automation Letters, 4(2):610–617, 2019.

[267] Haoran Li, Qichao Zhang, and Dongbin Zhao. Deep reinforcement learning-based automatic exploration for navigation in unknown environment. IEEE Transactions on Neural Networks and Learning Systems, 31(6):2064–2076, 2019.

[268] Vu Phi Tran, Matthew A Garratt, Kathryn Kasmarik, Sreenatha G Anavatti, and Shadi Abpeikar. Frontier-led swarming: Robust multi-robot coverage of unknown environments. Swarm and Evolutionary Computation, 75:101171, 2022.

[269] Tauã M Cabreira, Lisane B Brisolara, and Ferreira Jr Paulo R. Survey on coverage path planning with unmanned aerial vehicles. Drones, 3(1):4, 2019.

[270] Olimpiya Saha, Guohua Ren, Javad Heydari, Viswanath Ganapathy, and Mohak Shah. Deep reinforcement learning based online area covering autonomous robot. In 2021 7th International Conference on Automation, Robotics and Applications (ICARA), pages 21–25. IEEE, 2021.

[271] Bruna G Maciel-Pearson, Letizia Marchegiani, Samet Akcay, Amir Atapour-Abarghouei, James Garforth, and Toby P Breckon. Online deep reinforcement learning for autonomous UAV navigation and exploration of outdoor environments. arXiv preprint arXiv:1912.05684, 2019.

[272] Claudio Piciarelli and Gian Luca Foresti. Drone patrolling with reinforcement learning. In Proceedings of the 13th International Conference on Distributed Smart Cameras, pages 1–6, 2019.

[273] Nathan Koenig and Andrew Howard. Design and use paradigms for Gazebo, an open-source multi-robot simulator. In 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), volume 3, pages 2149–2154. IEEE, 2004.

[274] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. AirSim: High-fidelity visual and physical simulation for autonomous vehicles. In Field and Service Robotics, pages 621–635. Springer, 2018.

[275] Johannes Meyer, Alexander Sendobry, Stefan Kohlbrecher, Uwe Klingauf, and Oskar von Stryk. Comprehensive simulation of quadrotor UAVs using ROS and Gazebo. In International Conference on Simulation, Modeling, and Programming for Autonomous Robots, pages 400–411. Springer, 2012.

[276] Fadri Furrer, Michael Burri, Markus Achtelik, and Roland Siegwart. RotorSa modular Gazebo MAV simulator framework. In Robot Operating System (ROS), pages 595–625. Springer, 2016.

[277] pixhawk.org. jMAVSim. `https://pixhawk.org/dev/hil/jmavsim`.

[278] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In Conference on Robot Learning, pages 1–16. PMLR, 2017.

[279] Epic Games. Unreal Engine 4. `https://www.unrealengine.com`, 2019.

[280] Andrew Svanberg Hamilton. Rural Australia. `https://www.unrealengine.com/marketplace/en-US/product/rural-australia?lang=en-US`, 2021.

[281] Lorenz Meier, Dominik Honegger, and Marc Pollefeys. PX4: A node-based multithreaded open source robotics framework for deeply embedded platforms. In 2015 IEEE international conference on robotics and automation (ICRA), pages 6235–6240. IEEE, 2015.

[282] AirSim. Creating and Setting Up Unreal Environment. `https://microsoft.github.io/AirSim/unreal_custenv/#creating-and-setting-up-unreal-environment`.

[283] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. PyTorch: An imperative style, high-performance deep learning library. Advances in Neural Information Processing Systems, 32, 2019.

[284] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In Proceedings of the 22nd ACM International Conference on Multimedia, pages 675–678, 2014.

[285] Frank Seide and Amit Agarwal. CNTK: Microsoft's open-source deep-learning toolkit. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 2135–2135, 2016.

[286] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. TensorFlow: Large-scale machine learning on heterogeneous distributed systems. arXiv preprint arXiv:1603.04467, 2016.

[287] Rami Al-Rfou, Guillaume Alain, Amjad Almahairi, Christof Angermueller, Dzmitry Bahdanau, Nicolas Ballas, Frédéric Bastien, Justin Bayer, Anatoly Belikov, Alexander Belopolsky, et al. Theano: A python framework for fast computation of mathematical expressions. arXiv e-prints, pages arXiv–1605, 2016.

[288] Xingshuai Dong, Matthew A Garratt, Sreenatha G Anavatti, and Hussein A Abbass. MobileXNet: An efficient convolutional neural network for monocular depth estimation. IEEE Transactions on Intelligent Transportation Systems, 23(11):20134–20147, 2022.

[289] Dan Xu, Wei Wang, Hao Tang, Hong Liu, Nicu Sebe, and Elisa Ricci. Structured attention guided convolutional neural fields for monocular depth estimation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 3917–3925, 2018.

[290] Xinchen Ye, Shude Chen, and Rui Xu. DPNet: Detail-preserving network for high quality monocular depth estimation. Pattern Recognition, page 107578, 2020.

[291] Tien-Ju Yang, Andrew Howard, Bo Chen, Xiao Zhang, Alec Go, Mark Sandler, Vivienne Sze, and Hartwig Adam. NetAdapt: Platform-aware neural network adaptation for mobile applications. In European Conference on Computer Vision, pages 285–300, 2018.

[292] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 1–9, 2015.

[293] Joseph Bullock, Carolina Cuesta-Lázaro, and Arnau Quera-Bofarull. XNet: A convolutional neural network (CNN) implementation for medical X-Ray image segmentation suitable for small datasets. In Medical Imaging 2019: Biomedical Applications in Molecular, Structural, and Functional Imaging, volume 10953, page 109531Z, 2019.

[294] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. ImageNet large scale visual recognition challenge. International Journal of Computer Vision, 115(3):211–252, 2015.

[295] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. arXiv preprint arXiv:1511.07122, 2015.

[296] Zbigniew Wojna, Vittorio Ferrari, Sergio Guadarrama, Nathan Silberman, Liang-Chieh Chen, Alireza Fathi, and Jasper Uijlings. The devil is in the decoder: Classification, regression and GANs. International Journal of Computer Vision, pages 1–13, 2019.

[297] Go Irie, Takahito Kawanishi, and Kunio Kashino. Robust learning for deep monocular depth estimation. In 2019 IEEE International Conference on Image Processing (ICIP), pages 964–968, 2019.

[298] Seema Kumari, Ranjeet Ranjhan Jha, Arnav Bhavsar, and Aditya Nigam. AutoDepth: Single image depth map estimation via residual CNN encoder-decoder and stacked hourglass. In 2019 IEEE International Conference on Image Processing (ICIP), pages 340–344. IEEE, 2019.

[299] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. In Proceedings of the IEEE/CVF International Conference on Computer Vision, pages 1026–1034, 2015.

[300] Xinghui Dong, Christopher J Taylor, and Tim F Cootes. Defect detection and classification by training a generic convolutional neural network encoder. IEEE Transactions on Signal Processing, 68:6055–6069, 2020.

[301] Ilija Radosavovic, Raj Prateek Kosaraju, Ross Girshick, Kaiming He, and Piotr Dollár. Designing network design spaces. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 10428–10436, 2020.

[302] Lei He, Guanghui Wang, and Zhanyi Hu. Learning depth from single images with deep neural network embedding focal length. IEEE Transactions on Image Processing, 27(9):4676–4689, 2018.

[303] Praful Hambarde and Subrahmanyam Murala. S2DNet: Depth estimation from single image and sparse samples. IEEE Transactions on Computational Imaging, 6:806–817, 2020.

[304] Chanho Eom, Hyunjong Park, and Bumsub Ham. Temporally consistent depth prediction with flow-guided memory units. IEEE Transactions on Intelligent Transportation Systems, 21(11):4626–4636, 2019.

[305] John Paul T Yusiong and Prospero C Naval. AsiANet: Autoencoders in autoencoder for unsupervised monocular depth estimation. In 2019 IEEE Winter Conference on Applications of Computer Vision (WACV), pages 443–451. IEEE, 2019.

[306] Jonas Uhrig, Nick Schneider, Lukas Schneider, Uwe Franke, Thomas Brox, and Andreas Geiger. Sparsity invariant CNNs. In 2017 international conference on 3D Vision (3DV), pages 11–20. IEEE, 2017.

[307] Heiko Hirschmuller. Stereo processing by semiglobal matching and mutual information. IEEE Transactions on Pattern Analysis and Machine Intelligence, 30(2):328–341, 2007.

[308] Anirban Roy and Sinisa Todorovic. Monocular depth estimation using neural regression forest. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 5506–5514, 2016.

[309] Xingshuai Dong, Matthew A Garratt, Sreenatha G Anavatti, Hussein A Abbass, and Junyu Dong. Lightweight monocular depth estimation with an edge guided network. In 2022 17th International Conference on Control, Automation, Robotics and Vision (ICARCV), pages 204–210. IEEE, 2022.

[310] Xinchen Ye, Shude Chen, and Rui Xu. DPNet: Detail-preserving network for high quality monocular depth estimation. Pattern Recognition, 109:107578, 2021.

[311] Jiaming Sun, Zehong Shen, Yuang Wang, Hujun Bao, and Xiaowei Zhou. LoFTR: Detector-free local feature matching with transformers. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 8922–8931. IEEE, 2021.

[312] Xiaohan Tu, Cheng Xu, Siping Liu, Renfa Li, Guoqi Xie, Jing Huang, and Laurence Tianruo Yang. Efficient monocular depth estimation for edge devices in internet of things. IEEE Transactions on Industrial Informatics, 17(4):2821–2832, 2020.

[313] Michael Rudolph, Youssef Dawoud, Ronja Güldenring, Lazaros Nalpantidis, and Vasileios Belagiannis. Lightweight monocular depth estimation through guided decoding. arXiv preprint arXiv:2203.04206, 2022.

[314] Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In 2017 IEEE international conference on robotics and automation (ICRA), pages 3357–3364. IEEE, 2017.

[315] Hongge Xu, Jing Samantha Pan, Xiaoye Michael Wang, and Geoffrey P Bingham. Information for perceiving blurry events: Optic flow and color are additive. Attention, Perception, & Psychophysics, 83:389–398, 2021.

[316] Simon Bultmann, Jan Quenzel, and Sven Behnke. Real-time multi-modal semantic fusion on unmanned aerial vehicles with label propagation for cross-domain adaptation. Robotics and Autonomous Systems, page 104286, 2022.

[317] Aqeel Anwar and Arijit Raychowdhury. Autonomous navigation via deep reinforcement learning for resource constraint edge nodes using transfer learning. IEEE Access, 8:26549–26560, 2020.

[318] Daniel J Butler, Jonas Wulff, Garrett B Stanley, and Michael J Black. A naturalistic open source movie for optical flow evaluation. In European Conference on Computer Vision, pages 611–625. Springer, 2012.

[319] Wei Li, Xing Wang, Xin Xia, Jie Wu, Xuefeng Xiao, Min Zheng, and Shiping Wen. SepViT: Separable vision transformer. arXiv preprint arXiv:2203.15380, 2022.

[320] Carlos Campos, Richard Elvira, Juan J Gómez Rodríguez, José MM Montiel, and Juan D Tardós. ORB-SLAM3: An accurate open-source library for visual, visual–inertial, and multimap SLAM. IEEE Transactions on Robotics, 37(6):1874–1890, 2021.

[321] Vivien Potó, József Árpád Somogyi, Tamás Lovas, and Árpád Barsi. Laser scanned point clouds to support autonomous vehicles. Transportation Research Procedia, 27:531–537, 2017.

[322] Xiaoshui Huang, Guofeng Mei, Jian Zhang, and Rana Abbas. A comprehensive survey on point cloud registration. arXiv preprint arXiv:2103.02690, 2021.

[323] Paul J Besl and Neil D McKay. Method for registration of 3-D shapes. In Sensor fusion IV: control paradigms and data structures, volume 1611, pages 586–606. Spie, 1992.

[324] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Open3D: A modern library for 3D data processing. arXiv preprint arXiv:1801.09847, 2018.