Learning-Based Monocular Vision Obstacle Detection and Avoidance for UAV Navigation in Urban Airspace

Yuhang Zhang

School of Mechanical & Aerospace Engineering

A thesis submitted to the Nanyang Technological University
in partial fulfillment of the requirements for the degree of
Master of Engineering

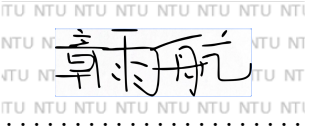
2023

Statement of Originality

I hereby certify that the work embodied in this thesis is the result of original research, is free of plagiarised materials, and has not been submitted for a higher degree to any other University or Institution.

17/03/2023

Date



Yuhang Zhang

Supervisor Declaration Statement

I have reviewed the content and presentation style of this thesis and declare it is free of plagiarism and of sufficient grammatical clarity to be examined. To the best of my knowledge, the research and writing are those of the candidate except as acknowledged in the Author Attribution Statement. I confirm that the investigations were conducted in accord with the ethics policies and integrity standards of Nanyang Technological University and that the research data are presented honestly and without prejudice.

27/03/2023

Date

Prof. Low Kin Huat

Authorship Attribution Statement

This thesis contains material from 2 papers accepted at conferences in which I am listed as an author.

Chapter 4 is published as Y. Zhang, Q. Yu, K. H. Low and C. Lv, "A Self-Supervised Monocular Depth Estimation Approach Based on UAV Aerial Images," 2022 IEEE/AIAA 41st Digital Avionics Systems Conference (DASC), Portsmouth, VA, USA, 2022, pp. 1-8, doi: 10.1109/DASC55683.2022.9925733.

The contributions of the co-authors are as follows:

- Prof. Low Kin Huat provided the initial project direction and edited the manuscript drafts.
- Asst. Prof Lv Chen assisted with experimental design and draft revisions.
- I co-designed the study and performed all the laboratory work at the Air Traffic Management Research Institute, including theoretical design, dataset processing, and data analysis.
- Mr. Qing Yu assisted with preprocessing the UAV dataset and coding for the depth conversion module.

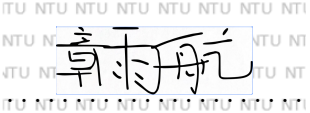
Chapter 5 is published as Y. Zhang, K. H. Low and C. Lv, "Partially-Observable Monocular Autonomous Navigation for UAV through Deep Reinforcement Learning." (Accepted by AIAA AVIATION 2023 Forum)

The contributions of the co-authors are as follows:

- Prof. Low Kin Huat provided the initial project direction and edited the manuscript drafts.
- Asst. Prof Lv Chen assisted with experimental design and draft revisions.
- I co-designed the study and performed all the laboratory work at the Air Traffic Management Research Institute, including theoretical design, simulation experiment, and data analysis.

17/03/2023

Date



Yuhang Zhang

Acknowledgements

I would like to express my sincerest gratitude to my supervisor, Prof. Low Kin Huat, for his invaluable guidance, support, and encouragement throughout my academic journey. His expertise, vision, and patience have been instrumental in shaping my research work and developing my skills as a researcher. His insightful comments, constructive feedback, and constant motivation have inspired me to strive for excellence and to push the boundaries of knowledge.

I would also like to extend my deepest appreciation to my co-supervisor, Prof. Lv Chen, for his valuable input, encouragement, and support. His expertise, dedication, and commitment have been critical in shaping my research work and in guiding me through the challenges of academic research. His guidance and mentorship have been invaluable in my academic growth and personal development.

I am deeply grateful to my friends in the lab, especially Dai Wei, Deng Chao, Pang Bizhao, Huxinting, Zhang Mingcheng, and Yu Qing, for their invaluable support, encouragement, and camaraderie. Their friendship, laughter, and shared experiences have made my academic journey a memorable and enjoyable one. Their intellectual curiosity, passion, and hard work have also inspired me to strive for excellence and to embrace challenges with enthusiasm and perseverance.

I would like to extend a special thanks to Yan Chao for his invaluable guidance and support throughout my research journey. From the very beginning, Yan Chao played a crucial role in helping me to define my research direction and set achievable goals. His extensive knowledge, expertise, and guidance have been instrumental in my development as a researcher.

I would also like to thank my parents for their unconditional love, support, and sacrifice. Their constant encouragement, guidance, and belief in me have been the driving force behind my academic success, and I am forever grateful for their love and support.

Finally, I would like to thank my girlfriend, Xu Wenfang, for her unwavering love, support, and encouragement. Her patience, understanding, and belief in me have been a constant source of inspiration. I am deeply grateful for her love and support. Her unwavering presence in my life has been a source of strength and motivation, and I am grateful for her love and support.

Once again, I would like to express my sincere appreciation to everyone who has contributed to my academic journey. I hope to honor their support and encouragement by pursuing excellence and making meaningful contributions to society.

"Everything negative - pressure, challenges - is all an opportunity for me to rise."

—Kobe, Bryant

To my dear family

Abstract

Unmanned aerial vehicles (UAVs), also known as drones, have gained considerable interest among academics in recent years, which significantly increases the demand for autonomous navigation systems for UAVs. In this study, autonomous navigation is addressed as a two-stage problem. The first stage is the obstacle detection phase, during which the UAVs leverage their onboard sensors to perceive the surroundings and detect obstacles. The second stage is the obstacle avoidance phase, where the UAVs make decisions to continue operations without colliding with anything while approaching the target.

This study proposes a self-supervised learning-based depth prediction framework as the criterion for obstacle proximity detection. The framework consists of two subnetworks: the depth and pose estimation modules. The former predicts depth values, while the latter evaluates the object's translation and rotation matrix. The framework leverages the subnetworks' outputs to reconstruct the projection relationships of pixels between adjacent frames to optimize the training process. To address the challenge of UAVs operating in large-scale environments, a modified loss function that combines photometric loss and edge-aware smoothness is presented to increase the depth prediction module's accuracy.

This study also presents a deep reinforcement learning-based obstacle avoidance system that takes depth maps as inputs to generate evasive maneuvers. The system works independently from priori maps and models while it learns the collision-free trajectory from interacting with the environment. This study deploys value-based algorithms to learn the nonlinear mapping between the continuous inputs and discrete output motion commands. Furthermore, the proposed system is validated on different simulation scenarios. It is capable of completing the obstacle avoidance task with relatively optimal trajectories in varied environments, demonstrating its superior effectiveness and transferability.

Keywords: unmanned aerial vehicles, urban airspace, obstacle detection and avoidance, self-supervised learning, deep reinforcement learning.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Unmanned Aerial Vehicles (UAVs), also known as drones, have attracted significant interest from researchers over the past years. UAVs refer to aircraft that are operated remotely without a pilot onboard. Compared to traditional manned aircraft, UAVs are more flexible, faster to execute, more repeatable, and more cost-effective [1], leading to them becoming a strong competitor to traditional data measurement platforms. Since UAVs can be flown in treacherous terrains and inaccessible areas together with real-time data acquisition and fast processing, they have been employed as platforms for military missions such as reconnaissance and precision strikes for over 20 years.

Recently, due to their flexibility and versatility, UAVs' applications in other scenarios have gained wider attention. More specifically, the employment of UAVs in civilian applications is developing rapidly. Examples include but are not limited to traffic surveillance [2], air pollution monitoring [3], parcel delivery [4], and fire detection [5]. The introduction of UAVs has brought revolutionary changes to many industries. For instance, UAVs equipped with sensors can be utilized to measure the Air Quality Index (AQI), which takes 8 pollutants into consideration [6]. Compared with traditional stationary sensors, UAVs are capable of detecting pollutant concentrations at different altitudes, bringing repeatability and lightweight performance to save space and costs for stationary sensors. Moreover, UAVs can also be employed in healthcare supplies transportation to reduce emergency patients'

deaths caused by traffic jams. A UAV can arrive at the scene of sudden illness faster than artificial medical assistance in 32 percent of the cases [7], and releasing the emergency medication from a low altitude can further speed up the medical treatment.

The global UAV market tends to grow by leaps and bounds as researchers gain a better understanding of their capabilities and deployment scenarios. By 2030, there are expected to be 14.2 million UAVs in the global airspace, and the global market is anticipated to be worth approximately 92 billion USD [8]. To cope with the large volume of aircraft and the increasing complexity of the air traffic system, innovative and multidimensional air traffic management systems are required for each country's safe and efficient operations of UAVs in the National Air Space (NAS), especially in the very low-level (VLL) urban airspace. Some of the world's major powers have proposed incipient management systems to deal with the issues triggered by the exponential growth of UAVs. In the US, the Federal Aviation Administration (FAA) is the authority to set the regulations for aviation operations and UAV missions. The FAA cooperates with the National Aeronautics and Space Administration (NASA) to develop Unmanned Aircraft System (UAS) Traffic Management (UTM) [9] as the control system for UAV deployment in US airspace. European Aviation Safety Agency (EASA) coordinates with the Civil Aviation Administration of China (CAAC) to develop a civil UAS Operation Management System (UOMS) [10], which is also utilized for supervising UAV traffic activities and services.

Despite commercial and non-commercial benefits brought out by the expansion of UAV employment, it is predictable that the large-scale deployment of UAVs will still pose an inevitable danger, especially in complex urban environments. For instance, in October 2021, dozens of UAVs suddenly crashed into the ground during a celebratory light show in Hangzhou, China [11]. Although there were no casualties, the UAVs hit some vehicles, causing considerable financial damage. In April 2022, an out-of-control UAV used for tourist photography crashed into the roof of a local landmark building in Italy [12], resulting in damage and repercussions. In addition to citizens outdoors, when encountering UAVs that lose control, even individuals who stay indoors are not entirely safe from danger. In June 2022, a remotely controlled aircraft suddenly became unresponsive and collided with a nearby hotel in Sydney [13]. The impact shattered the window's glass and scratched the tourist

in the room, simultaneously creating non-negligible panic in the crowd. The incidents are prone to happen repeatedly and on a large scale in the future without well-designed traffic management systems, and the safety of citizens' lives and property will be seriously threatened. For instance, according to a recent medical malpractice report [14], when a medium-sized unmanned helicopter with 325-mm long blades crashes, the impact can easily puncture the injured person's eyeballs, immediately reducing his vision and causing permanent and irreversible damage. Hence, how to ensure the safe operation of UAVs is a worthy area of research. As low-altitude airspace continues to open up to UAVs, one of the most challenging issues is how to avoid collision with an obstacle or another UAV, which can be triggered by a myriad of factors such as equipment failure, path conflict, or extreme weather conditions. Therefore, an essential capability for sustainable UAV operations is obstacle detection and avoidance (ODA), which requires UAVs to operate safely and efficiently in congested airspace while minimizing the probability of collision with surrounding obstacles.

As an autonomous operation system can empower the UAV to perform reactive evasive maneuvers encountering obstacles, which means that aircraft can make real-time decisions based on the available airspace and traffic conditions, the shortcomings of the traditional air traffic management system's inadequate response to pop-out hazards, such as falling aircraft and workers at height outdoors, can be bridged. By dynamically adjusting their flight trajectories, autonomous aircraft can optimize traffic flow and reduce the chances of collisions. Moreover, autonomous aircraft can communicate and share important information with each other and with ground-based systems. They can exchange data regarding their position, speed, flight plans, and other relevant parameters. This communication enables coordinated decision-making, ensuring smooth operations and effective traffic management in shared airspace. Consequently, a robust and computationally practicable ODA system is an integral part of every existing UAV traffic management framework. Furthermore, given that the UAV market in Singapore is proliferating and Singapore has not yet developed an intelligent ODA system for UAVs in VLL urban airspace. Consequently, this research aims to address the absence of an intelligent ODA system for UAVs in highly populated urban airspace. By leveraging cutting-edge learning-based and computer vision techniques, the proposed system in this research will endow UAVs with real-time ODA capabilities, ensuring the safety of UAV operations and diminishing the likelihood of collisions simultaneously.

1.2 Research Scope

ODA is a broad topic that can be analyzed from multiple perspectives. In this research, it is treated as a two-stage problem: perception and action. Perception, also called obstacle detection (OD), is the first stage of an ODA system. In this phase, UAVs perceive the surrounding environment and detect obstacles through the onboard sensors, providing preliminary information for the subsequent acting stage. Sensors can be categorized into two major parts [15]: active and passive. Active sensors can obtain information by utilizing their own source to release measurement media such as light or sound waves. In comparison, passive sensors can only receive information from the object to be measured, such as reflected sunlight. Considering the cost and payload limitations of UAVs, a camera, one of the passive sensors, is employed in this thesis to complete perception tasks. Subsequently, RGB images are leveraged to estimate the distance between UAVs and the surrounding environment.

Action, also called obstacle avoidance (OA), is the second stage of an ODA system. In this phase, UAVs continue approaching their intended destination while avoiding obstacles. OA can also be categorized into two major parts [16]: global path planning and reactive OA Global path planning refers to generating reference paths to connect starting and ending spots after considering all the environment configurations. In contrast, reactive OA stands for the timely avoidance maneuver after detecting an obstacle. This thesis focuses on reactive OA because it can respond promptly to sudden local changes in complex urban airspace.

Compared with fixed-wing UAVs, which cannot perform complicated maneuvers like hovering, rotary UAVs are more suitable for congested VLL urban airspace. Therefore, this thesis studies the issue of reactive ODA for rotary UAVs with a camera onboard in VLL urban airspace. Moreover, since traditional algorithms utilized for ODA are relatively computationally inefficient, accuracy-limited, and sensitive to noises, artificial intelligence (AI) approaches that consist of both regression and decision-making problems are utilized to build ODA structures. Compared with traditional approaches, these AI approaches usually require a large number of training cycles to converge, resulting in an improvement in accuracy, generalization, and robustness. To summarize, the research scope of this thesis lies in

the learning-bassed monocular vision autonomous navigation approach for UAVs in congested airspace.

The overall research framework is depicted in Fig. 1.1, which highlights the key components and techniques deployed in the ODA system for UAVs. The green blocks represent the completed works, and the blocks with the asterisk sign are the novel contributions proposed by the author. The system is a two-staged process integrating the latest advancements in AI and computer vision. In the OD phase, an self-supervised monocular vision-based depth estimation approach is proposed to estimate the depth of objects in the environment. This depth information is subsequently leveraged to perceive obstacles in the UAV's flight trajectory. For the OA phase, depth information is utilized to guide the UAV to perform evasive maneuvers when encountering obstacles. The combination of monocular depth estimation (MDE) and reactive OA provides an efficient and reliable solution for UAVs' ODA in populated environments.

1.3 Main Contributions

The main contributions of this thesis are threefold:

- Develop a self-supervised learning-based MDE algorithm suitable for estimating the distance between UAVs and their surroundings to detect obstacles. The primary contributions are the construction of the depth and pose estimation network, as well as the introduction of an innovative weighted loss function. Introducing these elements enhances the model's performance in large-scale and textureless environments. The model is validated on a dataset containing multiple outdoor scenes captured by UAVs and outperforms other cutting-edge models. One conference paper was published relying on the results.

- Complete the building of simulation scenes based on Gazebo / Ros, which allows the author to validate the system's effectiveness in multiple but controlled environments before deploying it in real-world scenarios. The primary contribution is building the software-in-the-loop simulation scene consisting

of crowded obstacles. This scene can meet the verification needs of various tasks, including navigating through narrow spaces and steering around obstacles.

- Develop a system derived from deep reinforcement learning (DRL) algorithms to generate real-time action commands for UAVs to avoid imminent obstacles. The primary contribution is the design of an OA module with depth images as inputs instead of RGB images. This framework integrates depth information with goal position and maps them directly to the speed control commands of the UAV. The real-timing and generalization of this framework are verified in the simulation scene, and one conference paper was published based on the obtained results.

The successful completion of this research can result in the development of an intelligent onboard tactical ODA system for UAVs operating in VLL urban airspace. This system can provide real-time ODA capabilities, allowing for safe and seamless navigation in complex environments. Furthermore, implementing this system can also significantly affect the UAV industry in Singapore. The capacity to operate in densely populated areas with numerous obstacles opens up the possibility for UAVs to be applied in various scenarios, including parcel delivery, inspection, and monitoring.

1.4 Organization of the Thesis

The thesis is organized as follows:

- Chapter 1 provides an overview of the research background, motivation, and scope, together with the main contributions.
- Chapter 2 presents a thorough review of the prevailing literature on ODA, which mainly lies in MDE and single-UAV OA solutions. For MDE, motion-based and deep learning-based approaches are the review's focus. The review mainly covers traditional multistep and RL-based approaches for single-UAV OA solutions.

- Chapter **??** presents the background knowledge essential to the challenges addressed in the later chapters. The foundational information for MDE includes the basics of imaging, depth estimation techniques, and principles of multilayer perceptron and convolutional neural networks. Additionally, the background knowledge for single-UAV OA is outlined, including the Markov decision process, the principles of RL, and value-based algorithms.

- Chapter 3 presents a self-supervised MDE algorithm with a weighted photometric loss function. The main contents of this chapter include the design of networks for depth and pose estimation, the implementation of an automatic mask, the development of the weighted loss function, and a comparative analysis of the results with other MDE models.

- Chapter 4 proposes an RL-based OA system, which takes depth maps as input to generate motion control commands. The main contents of this chapter include the design of the policy network, the establishment of multiple simulation scenarios based on Gazebo/ROS, the training and validation of the model, and a comparative analysis of flight trajectory and other crucial indicators.

- Chapter 5 summarizes the content of the full text and provides an outlook on future research directions.

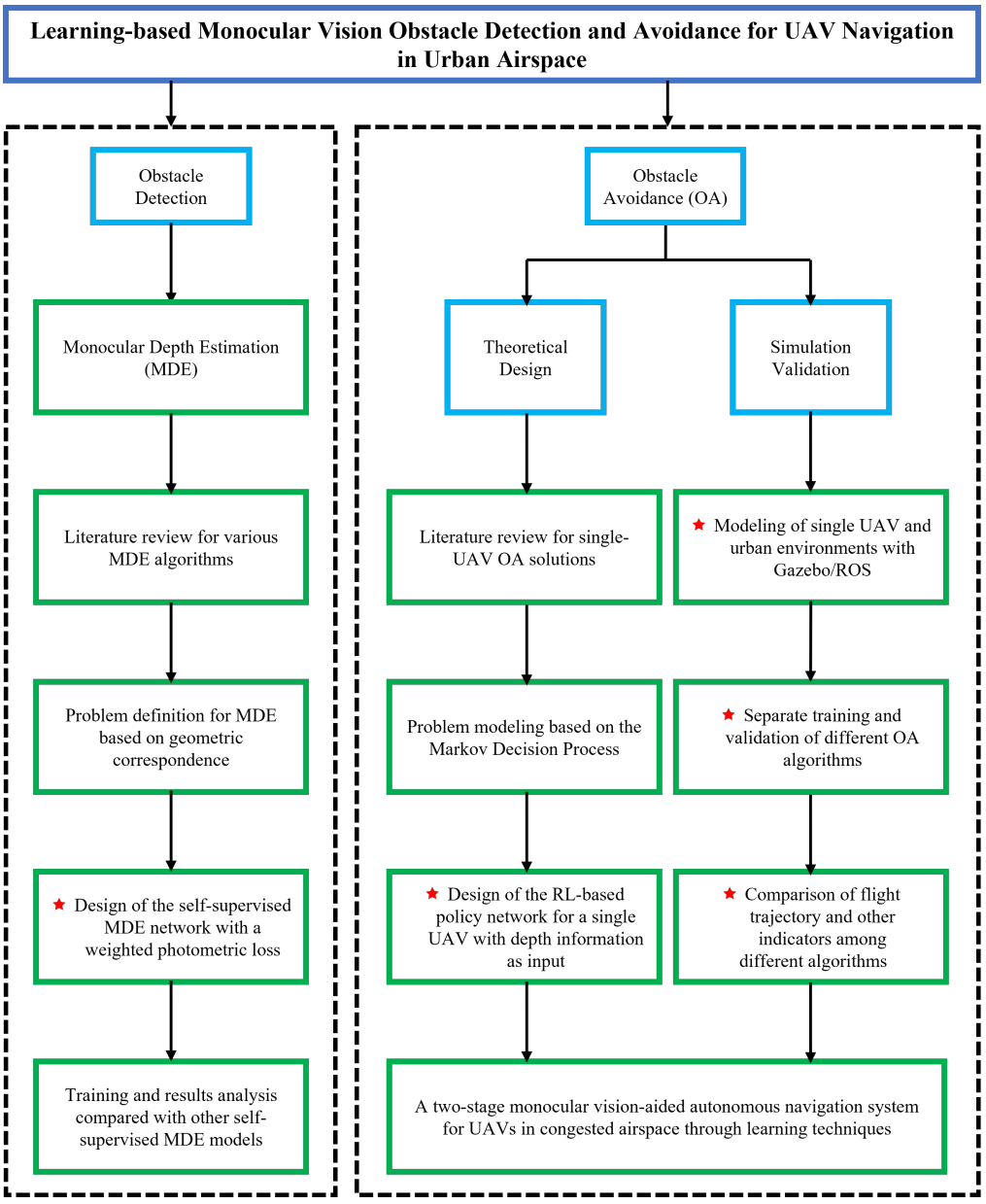


FIGURE 1.1: Overall research plan.

Chapter 2

Literature Review

The overview of the literature review is shown in Fig. 2.1, which includes two sections. The first section is dedicated to the review of MDE techniques, including both traditional and learning-based methods. The traditional methods, such as Structure from Motion, are thoroughly examined. Meanwhile, the learning-based methods are classified into supervised and self-supervised based on the supervision signal used for the training process. The review of these approaches provides a comprehensive understanding of the most advanced techniques in the MDE domain.

The second part of the literature review is centered on examining OA algorithms for a single UAV. This section takes into consideration the challenges posed by both static and dynamic obstacles during UAV navigation. The traditional and reinforcement learning (RL)-based approaches are presented in this section as potential solutions for OA in a single UAV scenario. Traditional approaches are usually cascaded or dependent on a pre-existing map. Meanwhile, RL-based approaches are map-free and model-free. The review of these approaches offers a thorough grasp of the current advanced techniques in the area of OA for a single UAV.

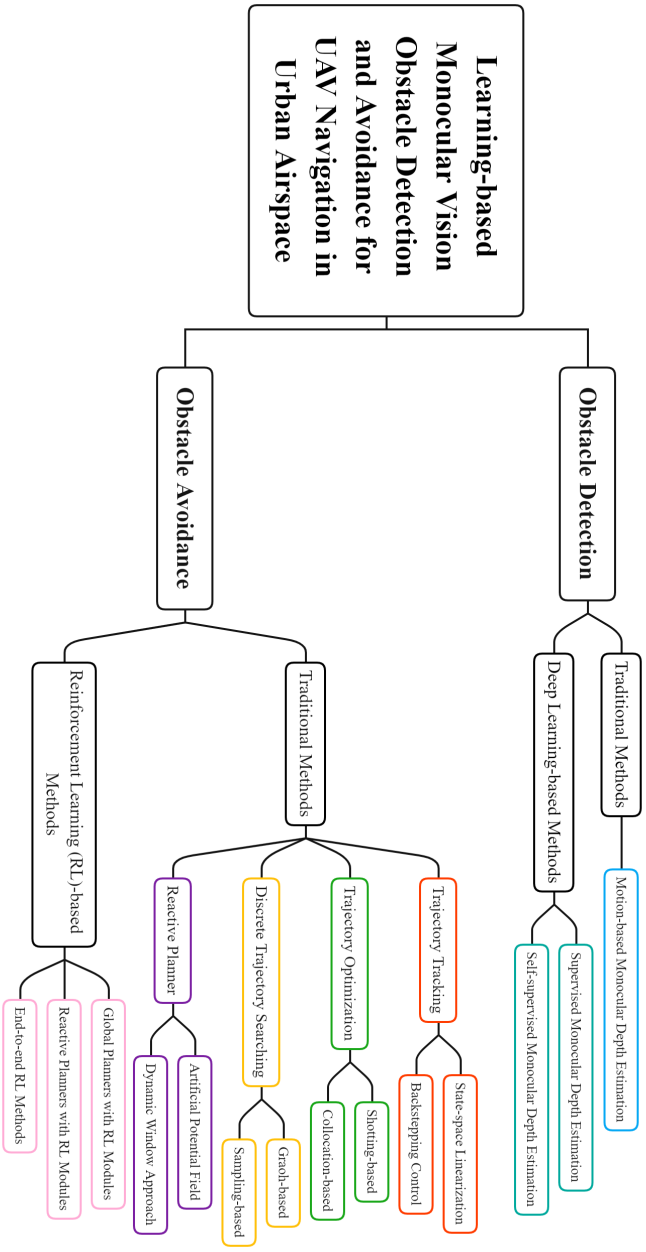


FIGURE 2.1: Overall literature review.

2.1 Monocular Depth Estimation

Previous studies have tried to estimate depth information from stereo images [17–20]. These images offer spatial data that can be utilized to calculate the depth of each pixel. Despite being more accurate than monocular sequences, stereo depth estimation demands more resources, including camera calibration, and the accuracy of the estimation can be affected by the baseline distance between the two lenses [21]. This makes it more resource-intensive compared to MDE.

MDE, on the other hand, is a technique for estimating depth information from a single image, which is especially crucial for small-sized UAVs because of their weight and size limitations. The objective of MDE is to predict the metric depth of each pixel from a single input image. Commonly, there are two types of resolutions to this issue: traditional motion-based techniques and learning-based techniques. The subsequent sections of the literature review will focus on exploring the relevant works from these two categories.

2.1.1 Motion-based Monocular Depth Estimation

Motion-based depth estimation techniques, such as Structure from Motion (SfM), utilize a collection of images captured from varying viewpoints to recover the structure of a scene and estimate depth information [22].

The pipeline of SfM usually starts with feature extraction and matching, where interest points in the images are detected and compared across different views. The next step is geometric verification, where the extracted matches are filtered to remove outliers that do not adhere to the geometric constraints of the scene. The remaining matches are then deployed to track the 3D point cloud map, thereby the depth map conversion can be completed relying on it [23]. This method of MDE has been extensively researched in this subject and has its roots in conventional computer vision techniques.

Li [24] developed an approach for depth estimation from monoscopic video by combining SfM with the Depth From Cues technique. This approach switches between two alternatives depending on the scene's complexity, aiming to produce more effective results. Despite its potential to produce promising depth maps

in certain conditions, this approach is limited by camera motion constraints and environmental degeneration.

Prakash [25] presented a sparse depth estimation approach based on SfM and a multi-scale keypoint detector. A monocular video sequence is fed into the framework to facilitate training, which performs feature matching between consecutive frames and leverages two-view geometry to determine sparse depth values, resulting in fast and robust performance.

In reference [26], a multi-sensor data fusion-aided SfM framework is presented to generate depth maps with a higher degree of precision. It aligns the correspondence between the sparse point cloud and lidar scan, realizing a transformation from 3D sensor data to uncalibrated RGB images. As an auxiliary module, the introduction of 3D sensor datasets is conducive to updating the parameters to be estimated and improving the roughness of original SfM approaches.

In conclusion, one of the significant benefits of SfM is its ability to handle intricate camera motions, such as rotations and translations, to produce depth maps with high precision in stable surroundings. Additionally, SfM is efficient for real-time applications as it has a fast processing speed.

However, SfM also has some disadvantages that make it less suitable for certain types of applications. For example, SfM is limited in handling scenes with low texture, which can result in incorrect depth estimates. Additionally, SfM requires the use of multiple images, which can be resource-intensive and may not be feasible in some applications, such as when being applied to small-sized UAVs.

Another disadvantage of SfM is that it can be prone to errors in camera calibration, which can impact the precision of the depth map. The estimation accuracy is also affected by other environmental factors such as occlusions, reflections, and shadows. SfM techniques typically yield sparse depth maps, which fall short of the requirements for UAV operations that demand dense depth maps for executing complex maneuvers.

2.1.2 Learning-based Monocular Depth Estimation

The application of deep learning (DL) in MDE has seen remarkable advancements in image processing in recent times. These advancements have enabled the creation of learning-based depth estimation models that are divided into two primary groups according to the availability of ground truth data. These groups are supervised models, trained with annotated data, and self-supervised models, where the training process utilizes unannotated data to learn depth estimation.

Learning-based MDE approaches have several advantages over traditional methods. One of the main benefits is their ability to embrace vast volumes of data to grasp intricate patterns and relationships in images, which can lead to more accurate depth predictions. Additionally, learning-based approaches are adaptable and can be readily customized for various image types, such as those captured in low-light conditions [27] or distorted images [28]. They are also robust to changes in camera parameters and can handle significant variations in the scale of items in the scene. Furthermore, the end-to-end training process of learning-based methods enables them to handle complex combinations of image features and to capture non-linear relationships between pixels and depth. This allows them to perform well in scenarios with fine-grained details, such as textures and patterns.

A. Supervised Monocular Depth Estimation

MDE dependent on supervised learning entails training a deep neural network on a sizeable annotated dataset of images and their corresponding depth maps. Thereafter, the trained network can be employed to generate the depth map of a fresh, unseen image. The pipeline for supervised learning-based MDE is depicted in Fig. 2.2 and comprises several steps. The initial step involves preprocessing the image, which may include resizing, normalization, and augmentation. The second step involves feeding the image into the neural network, which outputs a depth map. The supervision signal utilized to optimize the training process is the discrepancy between the estimated value and the annotated ground truth, as demonstrated by Eq. 2.1. The supervised learning-based MDE approach is highly effective when a large annotated dataset is available and has shown groundbreaking performance on various baseline datasets.

$$\mathcal{L}_2\left(d, d^*\right) = \frac{1}{N}\sum_{i}^{N} \|d - d^*\|_2^2 \tag{2.1}$$

where d represents the estimated value, d^* refers to the annotated ground truth.

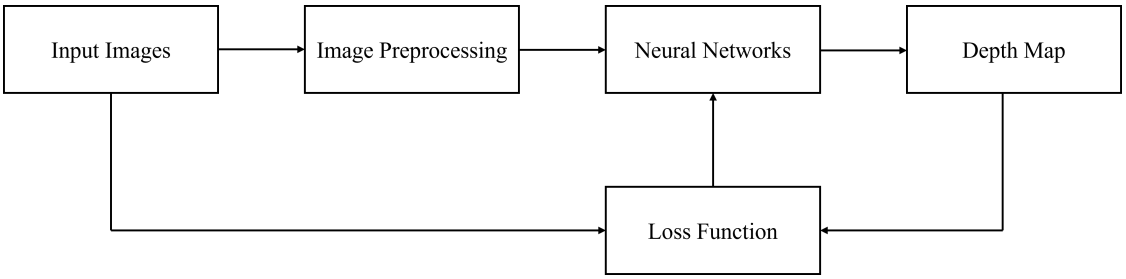


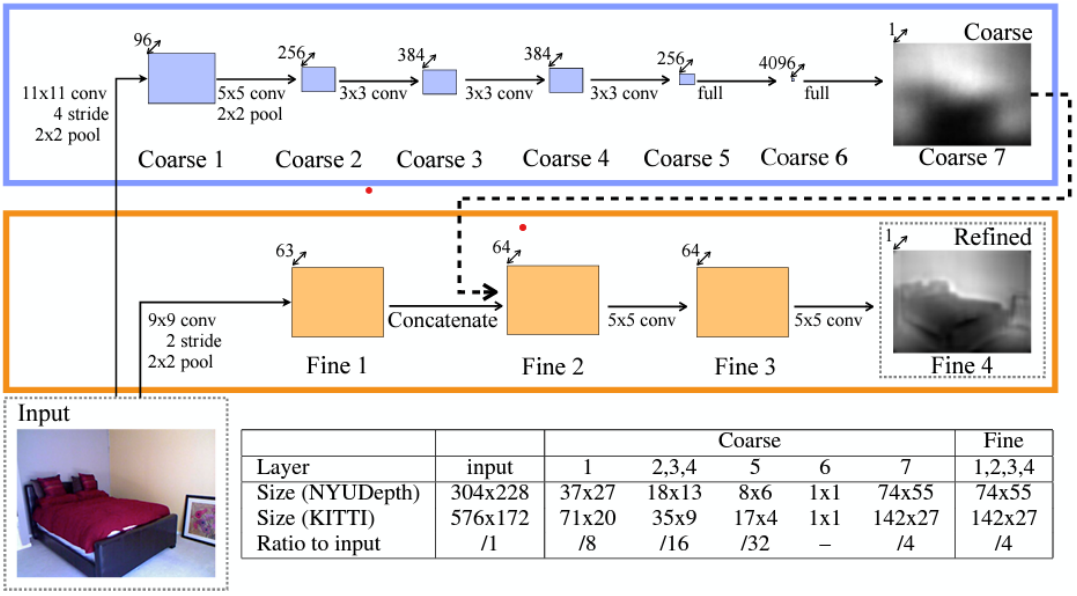
FIGURE 2.2: Pipeline for supervised MDE.

Eigen [29] was the first to utilize a supervised learning strategy for MDE, according to the authors' knowledge. The model's architecture is shown in Figure 2.3. It employs two CNNs (the overall rough network and the local fine-grained network) to handle the ambiguity and uncertainty from the overall scale. Additionally, the loss function includes the difference generated by different scales to minimize the error between the prediction and actual depth, leading to an impressive performance on the KITTI [30], and NYU [31] datasets.

Eigen [32] also designed a single multi-scale convolutional network that can perform multiple tasks, such as depth prediction and semantic labeling, with minimal modifications. This network lowers the gradient error between the forecasted and reference values in the horizontal and vertical directions. It is capable of producing outputs in real-time, achieving a frequency of approximately 30Hz, as evidenced by the results.

Such an approach works well on images with short-range depth values and richly textured backgrounds. Supervised-based solutions for aerial images, on the other hand, are relatively troublesome owing to the absence of fixed structures (degrees of freedom for the UAV are much more than self-driving vehicles).

To bridge the gap, Miclea and Nedevschi [33] bring out a CNN incorporating a novel scene digging module and an innovative softmax transformation layer to achieve greater convergence in aerial scenarios. This architecture introduces an optimum feature encoder with a creative image interpolation submodule. It combines ordinal



		Coarse					Fine
Layer	input	1	2,3,4	5	6	7	1,2,3,4
Size (NYUDepth)	304x228	37x27	18x13	8x6	1x1	74x55	74x55
Size (KITTI)	576x172	71x20	35x9	17x4	1x1	142x27	142x27
Ratio to input	/1	/8	/16	/32	–	/4	/4

FIGURE 2.3: The network architecture.

regression (that better accounts for areas with more minor texture variation) and classification (that performs better on stand-alone objects) as loss functions to train the model. The model is trained using images from the MidAir [34] dataset and validated on synthetically generated and authentic aerial images. Although the model shows promising performance in the validation scenarios, it is still not general enough to other real-life aerial scenarios.

B. Self-Supervised Monocular Depth Estimation

Self-supervised models for MDE have emerged as a promising solution for overcoming the limitations of supervised methods, particularly the dependence on large amounts of annotated data. Unlike supervised models, self-supervised models do not require ground truth depth information to be annotated. They use the information available in the RGB images alone to train the model, making self-supervised models more scalable and accessible, especially for applications where annotated data is difficult to obtain. The self-supervised models' training process leverages the scene's inherent geometric structure, such as the relative position of objects, to complete depth estimation. This feature facilitates them to be trained on a variety of datasets, enabling them to be more versatile and flexible than supervised models.

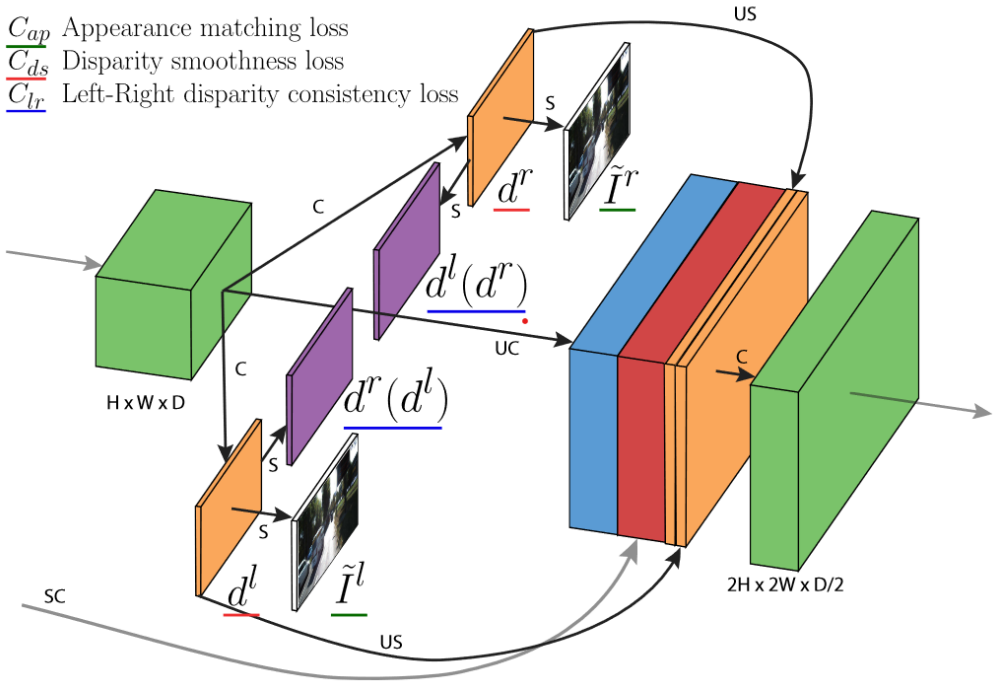


FIGURE 2.4: The overall loss module of Monodepth.

As far as the author is aware, Garg [35] introduced the first self-supervised framework for the MDE problem. In this framework, stereo pairs with pre-determined camera movement are employed to exploit the non-linear functional relationship between the image and depth map. A color deviation between the original and reconstructed images is employed to monitor updating the network weights.

Godard [36] presented a novel CNN network to enhance the accuracy of depth images. This work uses calibrated stereo pairs that include left and right color images for training. Both disparities (left-to-right and right-to-left) can be inferred simultaneously from the left input image. Moreover, an error in disparity consistency of a stereo pair is proposed as the loss function to enforce mutual consistency between the left-view disparity and the projected right-view disparity map. The overall loss module is shown in Fig. 2.4. Although this method is unsuitable for training on single-view datasets directly, it still outperforms many other supervised methods.

Godard [37] tried to handle occlusions and reduce visual artifacts in self-supervised methods robustly. Figure. 2.5 illustrates the complete architecture of this model. In his method, a minimum reprojection loss and an automatic filter are proposed to remove occluded pixels and items with the identical velocity as the sensor.

Moreover, a multi-scale estimation method is presented to aid in the network's faster convergence. This method performs well on both monocular videos and stereo pairs.

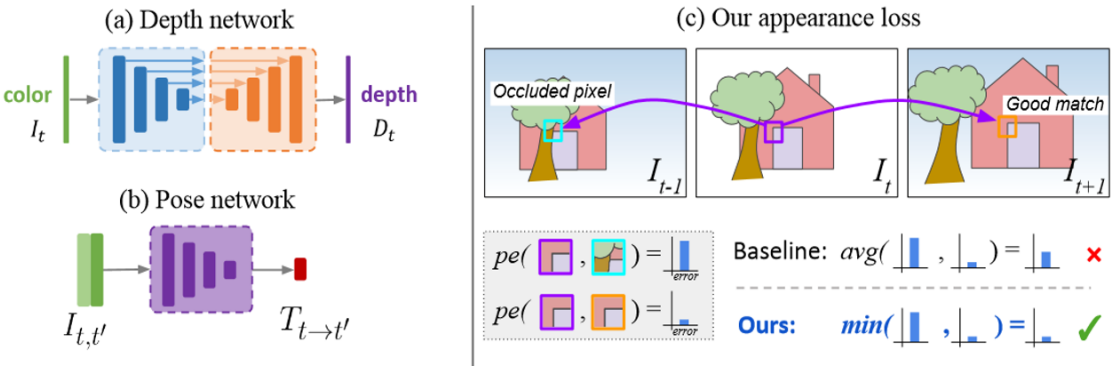


FIGURE 2.5: The overall framework of Monodepth2.

Although a large proportion of approaches handle the MDE problem in ground-related scenarios (for automotive and indoor), learning-based depth estimation for aerial environments is still an unexplored field, with many new algorithms to explore. Several researchers [38][39] are attempting to create datasets by using a simulator to generate synthetic images, which are categorized into two parts (low altitude range and high altitude range). While such datasets can provide an outdoor reference environment, there is an overall mismatch with the natural atmospheric environment (from the aspects of illumination and environment appearance). Such a research gap highlights the challenges of the aerial scenario, most originating from the unconstructed ego-motion and complex backgrounding texture.

2.2 Obstacle Avoidance for Single Agent

The primary objective of OA is to guarantee the effectiveness and security of navigation through an environment. The OA phase of UAV navigation is responsible for integrating global or local information to generate an optimal, obstacle-free route between the starting location and destination. The OA system comprises two components: global path planning and reactive OA.

Global path planning is responsible for generating a path that is optimal, collision-free, practical, and dynamically viable based on the existing global map. It takes

into account the global environment, including the layout of the terrain, the presence of obstacles, and any other environmental factors that may impact the UAV's path.

Reactive OA, on the other hand, is responsible for guiding the UAV to perform real-time evasive maneuvers in response to dynamic obstacles in the local environment. It operates in real-time and is designed to respond to obstacles encountered by the UAV during its flight, such as suddenly out-of-control UAVs, workers outside windows of high-rise buildings, or other unforeseen obstacles.

A typical OA system is hierarchical and multi-layered in series [40], which is also required to be customized for specific application scenarios. In this system, global path planning is independent of reactivate obstacle avoidance, both of which need separate designs for different scenarios. Therefore, traditional OA solutions do not adapt well to unconstructed and dynamic environments. With the rapid development of RL , RL-based techniques are leveraged to obtain a more robust collision-free path without preliminary map data. Overall, this section provides a literature review of OA solutions from the perspective of both traditional and learning-based approaches.

2.2.1 Traditional Collision Avoidance Solutions

Before introducing the pipeline of the traditional OA solutions, it is compulsory to generate a map that represents the global environmental configurations. The accuracy and level of detail of a priori map directly affects the performance of subsequent trajectory planning. Examples of commonly used structured maps include but are not limited to metric map (such as occupancy grid map [41], point cloud map [42]), topological map [43], and semantic map [44]. The typical hierarchical pipeline of the traditional OA solutions can be divided into submodules: discrete trajectory searching, trajectory optimization, trajectory tracking, and reactive planner [45]. These four submodules cooperate to perform global path planning tasks and simultaneously reactivate obstacle avoidance. The literature on traditional approaches to implementing these submodules is reviewed in this section.

A. Discrete Trajectory Searching

Discrete trajectory searching (DTS) is a method for generating an executable route made up of multiple sparse waypoints from the start to the end points. DTS works on the basis of configuration space and leverages the same trajectory planning algorithm to handle the planning issue for geometrically and kinematically diverse autonomous robots. Several different types of algorithms can be utilized to solve the traditional DTS problem, such as the graph-based algorithm and the sampling-based algorithm.

i. Graph-based Algorithms

Graph-based approaches define the possible locations of UAVs and obstacles as a configuration space and overlay a grid on that space. After recognizing each configuration as a grid point, the algorithm aims to generate a tree-like and obstacle-free path connecting the initial and goal points.

Dijkstra [46] was the first to propose a graph-based path planning algorithm. Dijkstra's algorithm exhaustively calculates the minimum distance from each node to the target node. This complete and greedy algorithm indeed yields optimal solutions, but is often computationally complex and lacks directionality.

To speed up the calculation, [47] introduces a heuristic module to measure the distance (usually the Euclidean distance or Manhattan distance) between the current search location and the target point, which is also defined as the A* algorithm. The introduction of the heuristic module makes the methods more directional and also computationally speedy. The flowchart of the A* algorithm is presented in Fig. 2.6.

Stentz [48] proposes a dynamic A* (D*) algorithm to tackle the route planning issue in an uncertain environment. Compared with the A*, D* starts the path search from the goal position to the initial position. The appearance of a new obstacle in the trajectory only affects the path planning between the current position and the obstacle in real time. Hence D* can better adapt to the dynamic environment and reduce the computational cost.

In reference [49], Koenig presents a Lifelong Planning A* (LPA*) algorithm. LPA* combines traditional A* with the incremental search to avoid the recalculation

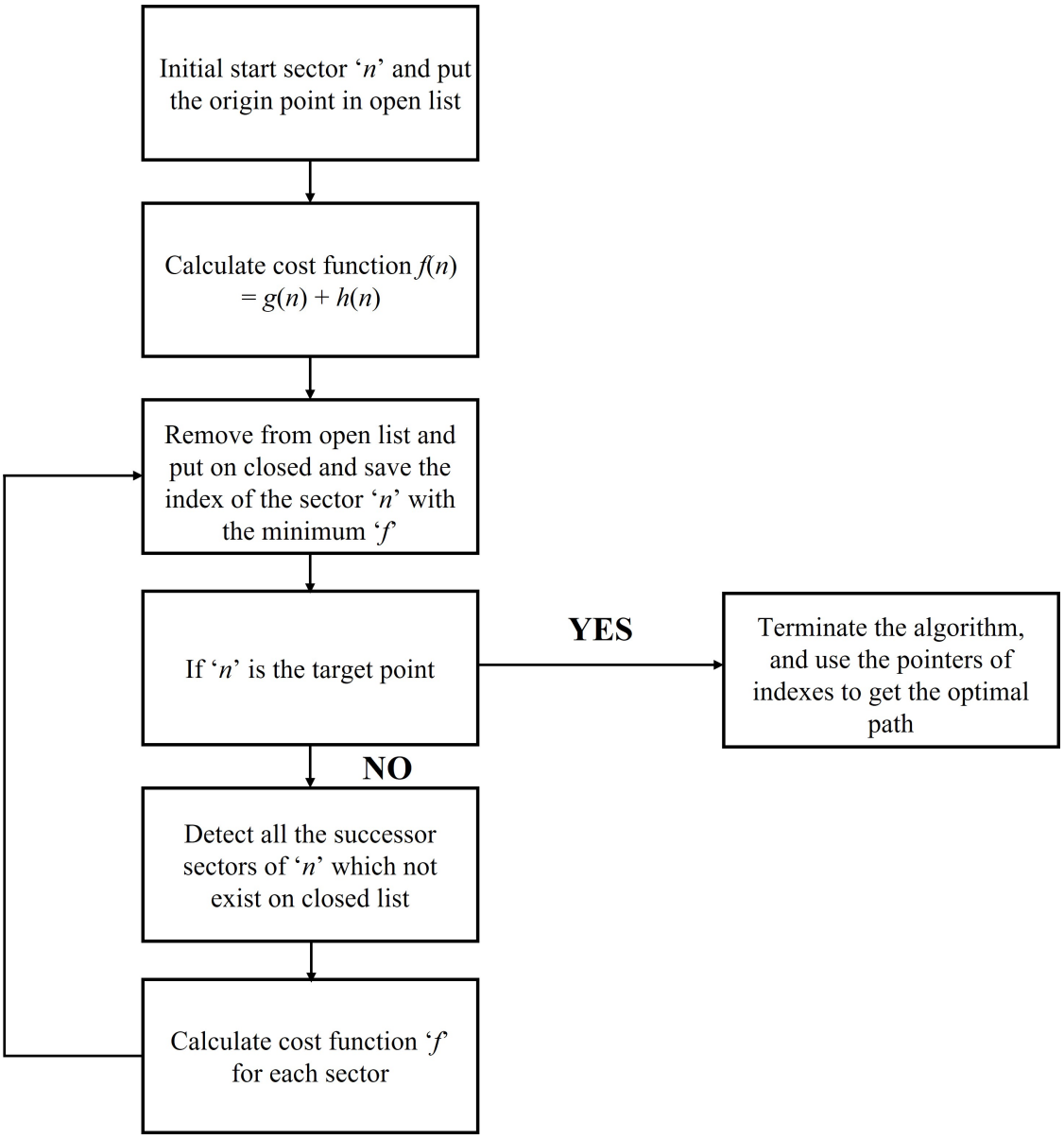


FIGURE 2.6: The flowchart of A* algorithm.

of global path planning resulting from changes in environmental preconditions, speeding up planning in similar path cases.

Koenig develops a D* lite algorithm in [50]. D* lite is an incremental heuristic algorithm for handling path planning challenges in unknown environments. D* lite works based on the assumption that all unknown regions are free space, and incrementally completes the planning task between the starting point (real-time position) and the fixed target point.

Although these brute-force methods can usually generate feasible and optimal results, many nodes are produced during the pathfinding process, which requires a significant amount of computational memory to store. The Jump Point Search (JPS) was developed by Harabor [51] to tackle the issue of excessive memory consumption caused by redundant nodes. JPS introduces a forced neighbor pruning rule while maintaining the basic principle of A*. It starts pathfinding from the straight and diagonal directions of the point in the open list. Since only nodes recognized as jump points are added to the open list, the JPS algorithm excludes other points that may also be on the optimal path but are unnecessary.

Graph-based algorithms make a non-negligible contribution to generating optimal and executable paths. Nevertheless, they are also associated with several drawbacks that hinder their extensive application. One of the primary limitations is the high computational complexity of constructing and searching the graph. This process can be time-consuming and computationally expensive, particularly in open and complicated environments. Furthermore, the geometry of the environment may not be adequately reflected by graph-based techniques, which can result in less-than-ideal solutions. Also, graph-based techniques might not be appropriate for real-time applications due to their processing complexity.

ii. Sampling-based Algorithms

The uniqueness and core of sampling-based planning (SBP) lies in the random sampling in the configuration space. SBP algorithms randomly generate child nodes on the map starting from the parent node in the workspace, and connect the child nodes to the parent node for obstacle detection. Only the child nodes that do not trigger collision expand until an origin-destination path is generated. Probabilistic road map (PRM) and rapid-exploring random tree (RRT) are recognized as the two most commonly used sampling-based algorithms.

The PRM [52] consists of two phases. The first one is the learning phase. In this phase, the algorithm randomly samples nodes in the configuration space to approximate the object's motion and saves the nodes located in the free space. Subsequently, vertices are linked to eliminate infeasible paths and generate a collision-free roadmap. The query phase follows the learning phase to search for the executable and optimal path on the basis of the Dijkstra algorithm and roadmap information. PRM algorithms can effectively reduce computation time while avoiding obstacles.

Algorithm 1 Probabilistic Road MAP Algorithm

Input:

n : the size of the roadmap in terms of nodes

k : the number of neighboring configurations to consider for each state

Output:

A roadmap $G = (V, E)$

$V \leftarrow \varnothing$
$E \leftarrow \varnothing$
while $|V| < n$ **do**
    repeat
        $q \leftarrow$ a random configuration in $\mathcal{Q}$
    until q is collision-free
    $V \leftarrow V \cup \{q\}$
end while
for all $q \in V$ **do**
    $N_q \leftarrow$ the k closest neighbors of q chosen from V according to dist
    for all $q' \in N_q$ **do**
        if $(q, q') \notin E$ **and** $\Delta(q, q') \neq$ NIL **then**
            $E \leftarrow E \cup \{(q, q')\}$
        end if
    end for
end for

However, the smooth execution of PRM algorithms must be based on a sufficient number of iterations. If the number of sampled nodes is not large enough, pre-computing a roadmap can be time-consuming or unworkable. The pseudocode of the PRM algorithm is shown in Algo. 3.

The RRT is a tree-growing planning algorithm introduced by Lavalle in [53]. The RRT completes the pathfinding process by creating a tree-like graph. The RRT algorithm starts by setting the initial point as the root node and randomly sampling nodes in the motion space. The closest node in the tree to the randomly sampled point is then selected for obstacle detection. The point is moved towards the randomly sampled point direction with a predetermined step size, and collision-free path detection is performed. If a new point is generated, it is included in the tree-like graph. This process continues until the target point becomes a child node of the tree structure. The pseudocode for the RRT algorithm is presented in Fig. 2.

Karaman [54] introduces RRT* to ensure the asymptotically optimal output path generation. In this reference, a searching algorithm along a minimum-cost and

Algorithm 2 Rapid-exploring Random Tree Algorithm

Input:
$\mathcal{M}$: Global Configuration
x_{init}: Initial Position
x_{goal}: Goal Position
Output: : A path $\mathcal{T}$ from x_{init} to x_{goal}

```
T.init()
for i = 1 to n do
    x_rand ← Sample(M)
    x_near ← Near (x_rand , T)
    x_new ← Steer (x_rand , x_near , Step Size )
    E_i ← Edge (x_new , x_near )
    if CollisionFree (M, E_i) then
        T.addNode(x_new)
        T.addEdge(E_i)
    end if
    if x_new = x_goal then
        Success()
    end if
end for
```

random geometric graph (RGG) is introduced during the expansion phase of the child node. Instead of choosing the nearest parent node, the improved algorithm selects the node with the minimum cost from the starting point as the parent node and rewires the tree.

Reference [55] develops the RRT*-SMART algorithm to compensate for the sluggish convergence rate of RRT*. Instead of leveraging random sampling, it employs biased sampling as the beacon for path optimization, which guides the road search process along the periphery of the obstacle. Similarly, Arslan [56] presents the RRT# algorithm to improve convergence speed. It constructs a spanning tree-like structure extending from the initial point. The tree structure not only keeps the current optimal path option but also stores the cost-come values of all vertices, both of which can fully exploit step-by-step information for each iteration.

Gayle [57] proposes a Reactive Deforming Roadmap (RDR) framework to improve PRM's inability to handle dynamic obstacles and changing environments. RDR applies internal and external forces to simultaneously perform local modification and global maintenance, which can minimize the number of global roadmap updates while reacting to dynamic obstacles.

Reference [58] presents a small-step retraction planner (SSRP) to enhance the unsatisfactory performance of PRM in narrow passages scenarios. SSRP combines two algorithms (Optimist and Pessimist) with merits and drawbacks in terms of speed and reliability to retract barely colliding configurations for the fast generation of waypoints within narrow passages.

The fact that sampling-based methods are computationally effective and capable of handling high-dimensional state spaces makes them ideal for real-time applications. Additionally, they can handle complex and non-linear environments, and provide a high degree of flexibility in terms of defining obstacles and other constraints. However, sampling-based methods also have several drawbacks, including the demand for a high number of samples in complicated situations and sensitivity to sample density. The number of samples required to cover the state space completely increases exponentially with the number of dimensions, resulting in the curse of dimensionality.

B. Trajectory Optimization

Since most of the DTS approaches are based on geometric constraints for the pathfinding process, in many cases, the final trajectories generated by DTS algorithms are not optimal or near-optimal. Also, these trajectories are not smooth enough or even non-executable with respect to the actual results.

Therefore, trajectory optimization (TO) exists to take into account multiple constraints (e.g., kinetic constraints, time allocation constraints) to make the path planning module more realistic and feasible. The purpose of TO is to determine a practical and ideal route between the starting and desired state. This approach is usually formulated as a non-linear problem [59], representing the path as a sequence of points or a continuous curve. The constraints are OA missions and any other physical or operational limitations. The optimization process seeks to minimize a cost function that captures the desired trade-off between path length, smoothness, and other performance metrics. TO is highly effective when the environment is known in advance, and the obstacles are static. However, it can be computationally expensive, especially in real-time applications, where many samples and complex optimization algorithms are needed to ensure the optimal path.

To address this challenge, various approaches, such as approximation [60] and heuristics [61], have been developed to make trajectory optimization more computationally efficient. Furthermore, integrating DTS and TO together can empower the planner to produce a smooth, kino-dynamically viable, risk-free, and implementable trajectory that fulfills multiple optimality metrics (efficiency optimal, energy consumption optimal, etc.).

C. Trajectory Tracking

Trajectory tracking (TT) refers to the process of forcing the UAV to fly along a path parameterized in terms of time and space. Specifically speaking, the UAV must arrive at the pre-specified location at a pre-determined time in the TT phase, which intrinsically integrates the spatial and temporal distribution relationships into a whole. Furthermore, TT also conducts research on the achievement of a trade-off between tracking fidelity and stability.

Typical TT approaches are subject to control tasks. They mainly focus on modeling onboard dynamical equation-based controllers. In that way, they can be used to supervise the operations of vehicles that must follow the reference trajectory rigorously. Conventional TT control methods consist of state-space linearization [62], geometric tracking controller [63], backstepping control[64], and sliding mode control[65].

These classical algorithms perform well in simple scenarios, but their widespread use also requires breaking through some limitations. For instance, UAVs own more complex degrees of freedom, which can introduce non-linear or uncertain terms. The complexity of mathematical modeling can substantially rise as a result of these factors. Furthermore, the unstable environments (such as congested airspace and multi-UAV interaction environment) in realistic applications also constrain the implementation of these approaches. Consequently, these standard TT algorithms should also have robustness and potential for local replanning.

Reference [66] proposes an enhanced sliding mode control method to increase the anti-disturbance capability of mobile robots. The core of the algorithm is the proposal of an adaptive law based on a non-linear model and considering the robot's dynamical uncertainty. Also, it leverages discrete projection mapping to readjust the parameters, and the velocity tracking error is significantly reduced.

Gavilan [67] designs this route-following guidance principle from another perspective. A model predictive control (MPC) strategy depending on the optimal computation of a particular cost is presented to solve the online TT problem for UAVs. This model designs a modified navigation low to iterate on the route prediction and refinement process, ultimately achieving robustness and time synchronization even in harsh weather conditions.

Reference [68] copes with the non-linear model predictive control (NMPC) problem with dynamic OA. A solution for optimum control relying on the proximal averaged newton is applied for the trajectory calculation phase, coupling with a penalty consisting of dynamic obstacles for the navigation phase. This combination allows the optimizer takes the dynamic barrier as an input, empowering the system to perform TT tasks in several different dynamic environments.

D. Reactive Planner

After connecting DTS, TO, and TT modules in tandem, most autonomous vehicles can already perform obstacle avoidance tasks. However, many unknown and uncertain factors still exist in the real world. For example, UAVs in operation must reasonably account for the sudden appearance of pedestrians and other UAVs when passing around the corners of buildings. Therefore, an intelligent OA system must be able to respond to unexpected situations promptly, and a reactive planner serves this requirement.

UAVs can receive real-time assistance from the reactive planner to help them avoid impediments in their flight trajectories. They are designed to react quickly to environmental changes and generate safe and efficient trajectories in real time. The main objective of a reactive planner is to ensure the UAV's safety in the face of unexpected or dynamically changing obstacles. The reactive planner operates in real-time and generates a corresponding response according to the current position of the UAV and identified obstructions. It typically uses a combination of algorithms such as potential fields [69], motion primitives [70], or other motion planning techniques to generate reactive motion commands. A reactive planner is handy in environments where the UAV operates in close proximity to obstacles, where the obstacles are moving, or in other highly dynamic situations.

Di [69] implements Artificial Potential Field (APF) method in the local planning scheme. The essence of APF is modifying the robot's coordinate by virtue of the

attractive and repulsive fields generated by the target goal and obstacles encountered accordingly. However, traditional APF methods tend to be trapped in local minima and challenging to find the optimal solution when the local obstacles are dense. Reference [69] also introduces a left-turning potential (LTP) field method to resolve the abovementioned limitations. LTP instructs the robot to escape the local optimum trap by calculating the magnitude and direction of each force field, which is uncomplicated and symmetrical.

Minguez [71] develops a nearness diagram method (ND) to command the motion planner in unstructured and changing environments. ND focuses on the extraction of spatial information from a high level. It describes the environment from the obstacles to the central point and robot periphery, respectively. Then, ND deploys this information to judge the safety situation of the robots and produce motion commands with five different motion laws.

Reactive planners can also provide resolutions for OA systems in changing environments populated with dynamic obstacles. The dynamic window approach (DWA) is a rather classic example.

Seder develops the DWA approach in [72] to fine-tune the graph-based global planner. DWA is a velocity sampling-based optimization algorithm that allows for the robot's dynamics and kinematic constraints. DWA samples in the space of translational and rotational velocities. Subsequently, it simulates the trajectory at different sampling values within the range of maximum acceleration and minimum deceleration.

DWA can select the optimal result by evaluating different trajectories utilizing the evaluation function. The DWA approach is often employed in conjunction with global planning algorithms to improve the mobility and flexibility of the OA system.

2.2.2 Learning-based Collision Avoidance Solutions

The investigation of conventional OA techniques has been reasonably exhaustive and detailed. These works do have some restrictions, however. For instance, global planning and reactive planning modules can function independently, which means they demand separate designs from developers. In addition, the complexity of the

entire system is prone to rise due to the multiple constraints that must be considered to guarantee the viability and optimality of the final produced trajectory. Traditional OA systems also have many serial sub-modules, each of which must go through a separate, time-consuming tuning process.

Recently, as RL technology has continued to advance, its distinctive benefits have also been emphasized in creating OA systems. On the one hand, RL-based methods can learn how to generate motion-planning commands straight from interaction data between robots and their surroundings. On the other hand, end-to-end RL-based OA systems can operate without a priori map, and the integration of the global and reactive planners can be achieved.

RL-based techniques are more adaptable and flexible since the quality of the geometric map in conventional approaches directly influences the outcome trajectories. This section examines the pertinent literature concerning deploying RL approaches as a supplemental module and developing end-to-end RL-based OA systems.

A. Global Planners with RL Submodule

Many scholars have focused on integrating RL algorithms and global planner to address the challenges encountered in conventional global path planning approaches, including limitations of the static environment and large-scale pathfinding problem, etc.

To get swing-free trajectories for UAVs with a suspended load, Faust [73] creates an approximate value iteration (AVI) RL-based method, which divides the route-finding process into two phases: learning and generation. A value function based on a greedy strategy is acquired during the learning phase. This value function can then be transferred to other road circumstances by some criterion in the generation phase. Simulation results and experiments indicate that adding the RL module is conducive to UAVs in static environments generating various pathways with minimal residual oscillations.

Subsequently, this research is expanded to the circumstance considering the kinematic constraints of the UAV suspended load system. This comprehensive approach [74] leverages the PRM algorithm as the global route-seeking method, cooperating with the RL-based algorithm to generate a risk-free trajectory with restricted load adjustment for the quadrotor. The RL agent utilizes the state and action space

restrictions to design the route tracking module. This coupling composition not only expedites the convergence of a single policy but also enables the expansion of the state and action space across tasks.

For the indoor long-distance navigation tasks across large-scale complex maps, Faust further presents a hierarchical PRM-RL approach in reference [75]. Unlike constructing waypoints based on straightforward configuration space linear connectivity, the PRM-RL algorithm first utilizes Rl agents for the local point-point navigation, dividing the whole configuration space into several parts. The connectivity, as assessed by the RL agent, is then used to produce the optimal motion commands.

The application of RL in global planners offers several advantages. For instance, it can handle dynamic environments, where obstacles can appear and disappear in real time. It can also handle partial observability, where the agent is difficult to have full knowledge of its surroundings. Moreover, the efficiency of the global planning algorithms and the robustness of the RL agent can be integrated to give the whole structure the capability to escape the local minima trap and maintain stability in a dynamic environment [75].

B. Reactive Planners with RL Submodule

Reactive planners in OA are responsible for reacting quickly and efficiently to environmental changes, which requires a high level of flexibility and adaptability. To achieve this level of performance, many researchers have proposed incorporating RL techniques into reactive planners. They aim to provide robots with the potential to undertake evasive maneuvers in the face of unexpected impediments while being robust to external disturbances.

Patel [76] introduces a hybrid DWA-RL algorithm to speed the response time of conventional local planners to moving obstacles. This algorithm creates a time-astute, low-dimensional observation space by embedding the DWA into the low-level space generator and leveraging the velocity sets together with their corresponding cost values. With this algorithm, robots can promptly obtain trajectories that are both dynamically viable and spatially sensitive.

To boost the effectiveness of conventional reactive planners in uncharted areas with limited global knowledge, Chang [77] proposes an adaptive Q-Learning-based DWA

structure. By increasing the number of DWA evaluation functions, the algorithm lessens the detrimental effect of some elements (such as robot sluggishness or dense obstacles) to increase the navigation ability of the DWA. This strategy balances efficiency and computational cost with a statisfying success rate even in the dynamic environments.

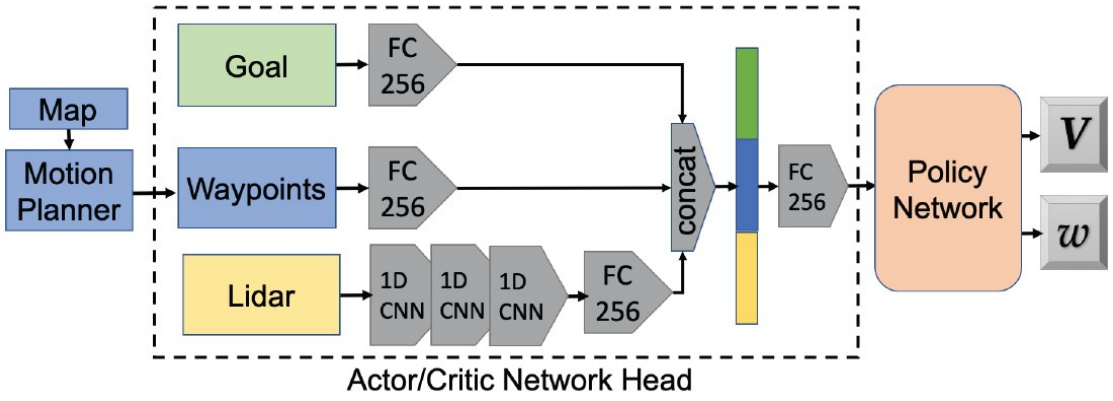


FIGURE 2.7: The architecture of network designed for safe navigation among pedestrians.

Besides, another area of focus for certain researchers is the utilization of RL algorithms to predict uncertainty risk levels and behavioral patterns in the surrounding environment (like the off-course UAVs and pedestrians).

A hybrid RL-based framework is proposed in [78] to learn policies that guarantee robots can execute evasive maneuvers while heading toward the desired destination in congested indoor environments. The network architecture is depicted in Fig. 2.7. The planning and RL modules make up the framework. On the basis of a prior map, the planning module is responsible for producing workable route points, which are then fed into the policy network combined with the sensor data and target state. The policy network in the RL module is responsible for learning various pedestrian-robot interaction patterns based on a soft actor-critic (SAC) method. According to experimental results, the RL module can successfully assist the robot in learning a range of interacting maneuvers, such as decelerating, meandering in place, or backing up.

C. End-to-end RL-based Obstacle Avoidance Solutions

Notably, in the studies mentioned in the preceding sections, RL algorithms are only employed as an enhancement or supplement to conventional OA solutions. The concentration of review in this section is on end-to-end RL-based OA resolutions.

Since these mapless resolutions achieve the integration of the global and reactive planners, researchers do not need to take geometric maps' effect on overall planning quality into consideration. Agent-level and sensor-level approaches are the two categories into which RL-based map-free OA resolutions can be further subdivided. The agent-level approaches are founded on a pre-defined state evaluation procedure that offers direct access to the environment's upper-level state data. The robot can acquire the optimum planning tactic faster since agent-level models are more convenient to train. However, these approaches assume flawless perception [40] or require additional communication channels to exchange state information. The adaptability and applicability of agent-level approaches are constrained by these limitations.

As for the sensor-level approaches, the non-linear mapping from the unprocessed sensor data to the output strategies is directly established, which is recognized as an end-to-end technique. As a result, compared to agent-level approaches, sensor-level approaches are more scalable, adaptable, and have greater sim-to-real capabilities. The approaches for end-to-end and sensor-level RL-based route planning are described in this part. Sensor-level RL motion planning techniques are further classified into two groups: laser range finder (LRF) based methods and visual-based approaches, in accordance with the general research trends and widely leveraged robot perception techniques.

i. Laser Range Finder-based Methods

Map building, depth estimation, autonomous driving, and other applications employ LRF technology extensively. Several cutting-edge RL-based OA resolutions with LRF techniques are presented in this section.

Researchers present the intrinsic curiosity module in [79] to enable RL agents to receive the intrinsic reward, in contrast to some strategies that specify the reward type of navigation in advance. The planner gains greater generalization skills in the 2D virtual world due to this exploring approach.

Similarly, Shi [80] shows a novel A3C framework with an intrinsic curiosity module to produce motion commands directly from unprocessed laser data. They validate the feasibility of the framework by successfully applying it to an authentic mixed scenario from the physical engine.

The hazard assessment and avoidance issue in a static simulated environment with moving impediments is taken into consideration by Wang [81]. The LRF's raw sensor data is sent into the collision-free planning module, which then generates a 5-dimensional force vector. The geographical stream and the temporal stream are the two streams that make up the policy network in their planner, which is significant to notice. The two streams are responsible for extracting time series information and geometric discrepancy from the input, respectively. According to experimental findings, this strategy can take immediate avoidance measures when confronted with dynamic impediments in a 2D simulation environment.

ii. Vision Sensors-based Methods

Vision sensors are also frequently employed as perceptual modules for autonomous robots in addition to Lidar sensors. Some investigations have discovered that perception and decision-making tasks can potentially be performed directly on the unprocessed visual data input. Convolutional neural networks (CNNs) and the DRL architecture are ideally suited for this purpose.

In [82], DeepMind researchers suggest using the NavA3C method to train the agent to discover its destination and navigate through several 3D mazes. It consists of several auxiliary submodules, including depth estimation and closure detection. This study successfully expedites the robot's learning of navigational techniques by utilizing additional experience from auxiliary activities such as ambient closure detection.

Li puts out a DRL-aided goal-oriented visual navigation strategy for interior settings in [83]. In contrast to earlier efforts on visual navigation, this approach has a higher capacity to generalize to different contexts and a stronger potential to be transferred to actual scenarios with only minor modifications.

The assumption behind the aforementioned visual-based motion planning techniques is that the static target is known. In active dynamic target tracking settings, researchers from Tencent AI Lab and Peking University have been collaborating on

route planning tasks for the purpose of dynamic goal tracking [84]. They leverage LSTM technology to store the movement patterns of the target point and update the trajectory in real-time. Additionally, This study also enhances the sim-to-real capacity of the model by environment detail augmentation, thus further strengthening the model's robustness and scalability.

Chapter 3

Self-Supervised Monocular Depth Estimation

Self-supervised MDE is one of the directions that have made significant progress in depth estimation in recent years. It requires only a single camera to acquire continuous images to train a depth estimation model. This technique is not supposed to necessitate a significant quantity of labeled datasets with corresponding ground truth compared with previous conventional approaches, which tremendously reduces the difficulty of acquiring datasets. After optimization in different studies, this technique has achieved comparable relative depth prediction effectiveness on the KITTI dataset [30] as models stemming from supervised learning [29].

While most existing depth prediction methods are tailored for autonomous driving, this work proposes a modified self-supervised framework specifically designed for UAV applications. The modification details are explained, followed by the experimental results and analysis based on a dataset of UAV application scenarios, including depth value conversion and error analysis. This self-supervised depth estimation architecture is to satisfy the obstacle detection function of UAVs in VLL urban complex airspace.

3.1 Methodology

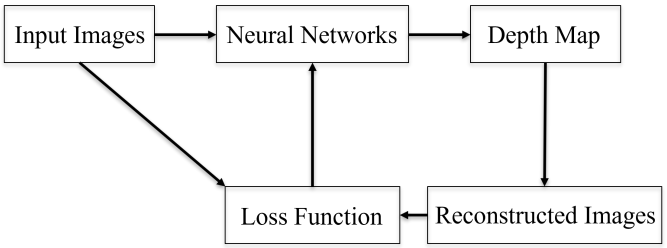


FIGURE 3.1: Flowchart explaining the model workflow

A. Overview

Considering the possibility of acquiring a sufficient amount of ground truth, which is also expensive. This chapter introduces a self-supervised model to solve the MDE problem. The training process of self-supervised architecture is solely governed by the information available in the RGB input image(s). The pipeline of self-supervised depth estimation approaches can be depicted in Fig. 3.1. Self-supervised methods take the geometric restrictions of objects in adjacent frames as the supervision indicator. More specifically, self-supervised models always take a monocular video sequence as input (in which one image is the target image and another is the reference image), along with camera intrinsics and the spatial transformations between frames. The focus of the training goal is straightforward. After getting the depth map predicted by the network, points generated based on the inferred depth in the target image can be warped to the reference image to get the reconstructed image combined with the camera movement information (see Eq. 3.1). The proof of Eq. 3.1 is depicted in Appendix. A.1. Due to the inaccurate depth prediction results, there is a photometric error between the original and reconstructed images. The network can be trained based on the photometric error instead of the ground truth.

$$P_n = \mathbf{K} T_{n-1 \to n} D_{n-1}\left(P_{n-1}\right) \mathbf{K}^{-1} P_{n-1}, \tag{3.1}$$

where P_n represents the pixel on current frame I_n, and P_{n-1} denotes the matching pixel of P_n on the previous frame I_{n-1}. $\mathbf{K}$ is the internal parameter of the camera, which is assumed to be a known constant. $T_{n-1 \to n}$ stands for the self movement matrix between I_{n-1} and I_n, and $D_{n-1}\left(P_{n-1}\right)$ refers to the depth value at pixel P_{n-1}.

This chapter proposes a model for jointly inferring a depth map and the camera motion information between the continuous frames from an unlabeled video sequence. The model comprises two parts: a subnetwork designed for depth prediction and a subnetwork designed for pose estimation. Figure. 3.2 depicts the overall framework of our model. We denote the input frame of the depth estimation subnetwork as the target image I_t, and images at the previous moment or later as reference images I_{t-1} or I_{t+1}. The depth estimation subnetwork aims to generate the depth map D_t from I_t. The pose estimation subnetwork aims to output the self-motion matrix $T_{t\to(t-1)}$ or $T_{t\to(t+1)}$ by taking a pair of target and reference images (I_{t-1}, I_t), or (I_t, I_{t+1}) as input.

Subsequently, the reference images (I_{t-1} or I_{t+1}) can be warped into the target image (I_t) to reconstruct the target view (see Fig. 3.3). It can be seen from the Fig. 3.3 that after projecting the pixel point in I_t to I_{t-1}, a bilinear sampler is employed to calculate the 2d coordinates of the projected pixel point in the I_{t-1} to reconstruct the image. The differences between target and reconstructed images are considered reprojection loss. By utilizing the reprojection loss between the target image and reconstructed image as supervision, the entire model is able to be trained in a self-supervised manner.

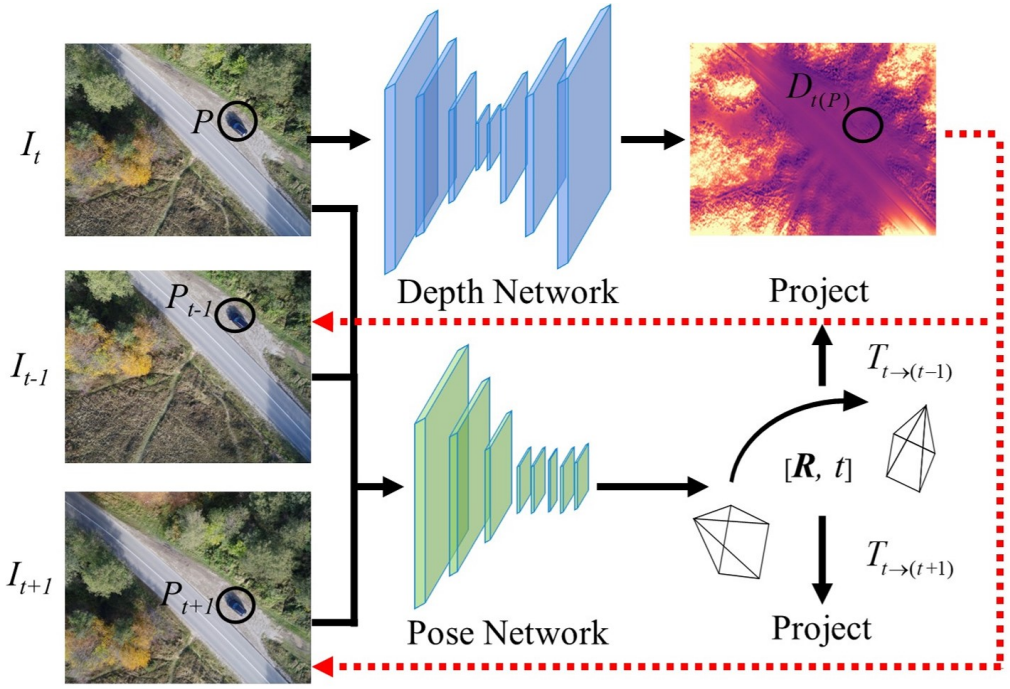


FIGURE 3.2: Overview of the pipeline of our self-supervised depth estimation model

B. Bilinear Interpolation

As shown in Fig. A.3, the disparity map depicts the geometric correlation between the pixel points of the left and right views of the stereo images captured simultaneously. The disparity map allows pixel points from the left view to be mapped to the right view based on the corresponding points. The mapping relationship can be expressed as follows when the disparity value is an integer.

$$\tilde{I}_r(u,v) = I_l\left(u - D_r^{u,v}, v\right), \tag{3.2}$$

where the reconstructed right view is denoted by $\tilde{I}_r$, with u and v representing the horizontal and vertical pixel coordinates, respectively. I_l is the input left view, and $D_r^{u,v}$ represents the disparity value at row u and column v.

However, during the mapping process, the predicted disparity is a continuous floating point number, which means that the mapped coordinates may not align with pixel locations, resulting in non-integer values for $u - D_r^{u,v}$. In such cases, the pixel mapping relationship cannot be expressed simply as in Eq. 3.2. To optimize pixel values at non-pixel locations, the bilinear interpolation method is often leveraged in image processing. This method is advantageous because of its differentiability, which is essential for backpropagation during the training process. Therefore, this chapter employs bilinear interpolation to optimize pixel correspondence in stereo images.

The bilinear interpolation involves linearly interpolating the coordinates in the X and Y directions. Figure. 3.3 shows a coordinate with point P_{t-1} and its four surrounding points P_{t-1}^1, P_{t-1}^2, P_{t-1}^3, and P_{t-1}^4. The aim is to estimate the value of point P dependent on the known value of the four neighbors. To achieve this, let the midpoint of P_{t-1}^1, P_{t-1}^2 and P_{t-1}^3, P_{t-1}^4 be C_1 and C_2, respectively. The values of these two points are obtained leveraging linear interpolation in the X-direction:

$$f\left(C_1\right) \approx \frac{x_2 - x}{x_2 - x_1} f\left(P_{t-1}^1\right) + \frac{x - x_2}{x_2 - x_1} f\left(P_{t-1}^2\right), \tag{3.3}$$

$$f\left(C_2\right) \approx \frac{x_2 - x}{x_2 - x_1} f\left(P_{t-1}^3\right) + \frac{x - x_2}{x_2 - x_1} f\left(P_{t-1}^4\right), \tag{3.4}$$

where x_1, x, and x_2 are the horizontal coordinates of P_{t-1}^1, P_{t-1}, and P_{t-1}^2, respectively.

Then, similarly, the value of P_{t-1} can be obtained by linear interpolation in the Y direction based on C_1, C_2:

$$f(P_{t-1}) \approx \frac{y_2 - y}{y_2 - y_1} f\left(C_2\right) + \frac{y - y_1}{y_2 - y_1} f\left(C_1\right), \tag{3.5}$$

where y_1, y, and y_2 are the vertical coordinates of C_2, P_{t-1}, and C_1, respectively.

Combining Eqs. 3.3, 3.4, and 3.5 to get:

$$f(P_{t-1}) \approx \frac{1}{(x_2 - x_1)(y_2 - y_1)} \begin{bmatrix} x_2 - x & x - x_1 \end{bmatrix} \begin{bmatrix} f\left(P_{t-1}^3\right) & f\left(P_{t-1}^1\right) \\ f\left(P_{t-1}^4\right) & f\left(P_{t-1}^2\right) \end{bmatrix} \begin{bmatrix} y_2 - y \\ y - y_1 \end{bmatrix}. \tag{3.6}$$

The bilinear interpolation method in this chapter is performed for the pixel coordinates, assuming that the coordinates of the four neighbors are $(1,0)$, $(1,1)$, $(2,0)$, and $(2,1)$. As a result, Eq. 3.6 can be simplified as:

$$f(x,y) \approx \begin{bmatrix} 1 - x & x - 1 \end{bmatrix} \begin{bmatrix} f(1,0) & f(1,1) \\ f(2,1) & f(2,2) \end{bmatrix} \begin{bmatrix} 2 - y \\ y - 1 \end{bmatrix}. \tag{3.7}$$

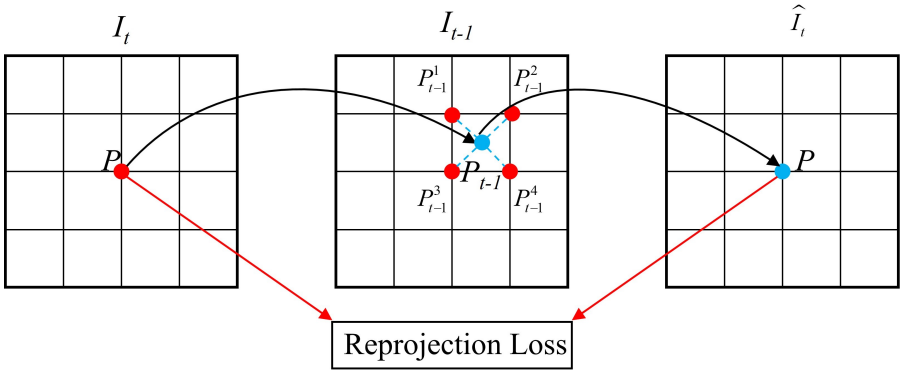


FIGURE 3.3: An illustration of bilinear interpolation approach.

C. Analysis of Monocular UAV Depth Estimation Problem

A monocular UAV refers to a UAV that utilizes only a single camera in a fixed direction for perceptual ODA. Most vision-based UAVs are equipped with at least one camera in either the forward or downward direction. Since downward-facing cameras are commonly employed to provide partial auxiliary information [85] or under simplified assumptions, such as translation without yaw [86], variations between images captured by downward-facing cameras are primarily influenced by translation and rotation in the x-y plane in UAV application scenarios. Consequently, the scenario variations between consecutive frames are typically not as ample as those in forward-facing environments, making pose and depth estimation more challenging compared to forward-facing scenarios. To enhance the model's generalization and capability to extend to multiple scenarios, this study conducts the training process based on a dataset acquired by a downward-facing camera and subsequently applies it to the forward OA scenario for validation.

Monocular vision might not be able to compute depth information through geometric disparity as accurately and directly as stereo vision. Nevertheless, it can be leveraged to anticipate significant depth information in the ODA scenarios combined with self-supervised learning techniques. In-flight UAVs equipped with monocular vision can capture consecutive video frames that suit the objectives and requirements of self-supervised MDE. Moreover, a training model based on UAVs' downward depth estimation scenes could strengthen the precision of forward depth estimation in UAV ODA missions.

Unlike the road driving environment of the KITTI [30] dataset leveraged in the related research, the depth estimation in this study mainly focuses on UAV scenarios. Compared to the road driving scenario consisting primarily of structured objectives such as light poles, pedestrians, and buildings, the UAV scenario mainly covers structured and unstructured objectives such as forests, bushes, automobiles, and roadways. Furthermore, the UAV has only one downward-facing monocular camera, which has a different focal length, single pixel size, and resolution than the camera launched in the self-driving automobiles, resulting in the different camera's internal reference matrix K. As a result, directly training the model with the KITTI dataset can negatively impact its accuracy. In order to reduce depth estimation errors, a dataset comprising UAV-specific camera shots that capture continuous motion information is required. The WildUAV [87] dataset, which

includes various landscapes, such as forests, plains, and wilderness, recorded deploying $4K$ cameras mounted on UAVs is leveraged in this study to perform MDE tasks.

The self-supervised depth estimation model learns the relationship between object color, texture, structure, and relative depth, which is subject to dataset bias and requires a corresponding scene dataset to improve estimation accuracy. However, occlusion and violation of point presence assumptions in both images can occur, which needs to be minimized during training. Besides, the practical situation will have the limit of depth estimation distance under the influence of resolution and noise, which cannot be learned to estimate more distant scenes. Moreover, too prominent obstacles may cover the entire image frame, leading to an immense content gap between adjacent frames. Especially when there is excessive rotational motion between two frames, pixels are prone to fail to establish correspondence. Therefore, comprehensive consideration of different optimization methods is required to strengthen training and reduce the impact of inherent defects on UAV OA.

D. Depth Estimation Subnetwork

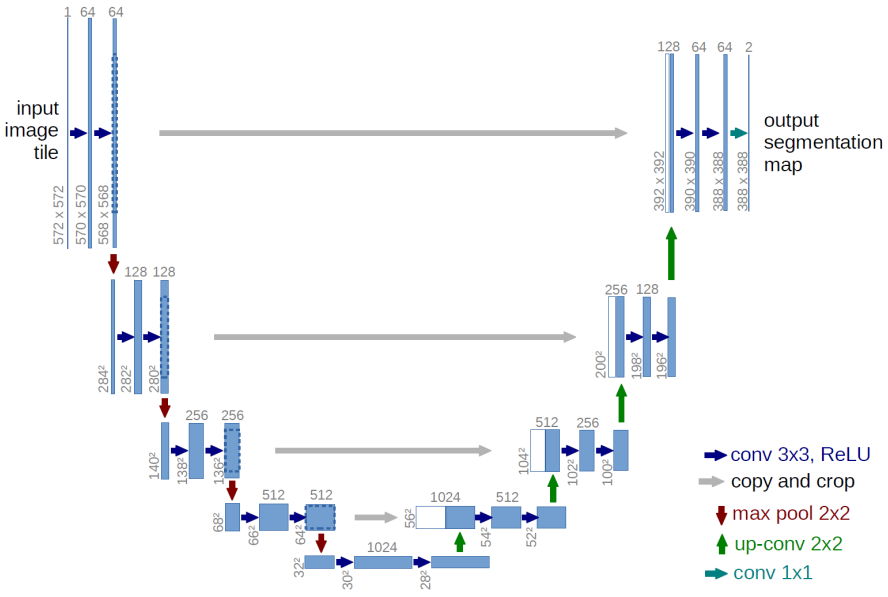


FIGURE 3.4: U-Net architecture.

Since U-Net [88] has the capability to localize each pixel on the image and distinguish specific patterns precisely, it is commonly utilized for pixel-level prediction

of depth estimation. Fig. 3.4 depicts the network architecture of U-Net, with the number of channels shown on top of each block. Blue blocks denote feature maps with multiple channels, while the image size is specified at the lower left corner. White blocks represent copied feature maps from the shrinking stage. The arrows denote different operations.

Based on this, a modified version of this typical structure is deployed to create the depth subnetwork. The architecture of the depth subnetwork is shown in Fig. 3.6(a). Similar to the traditional U-net, the network consists of an encoder, a decoder, and four skip connections. The encoder comprises convolution blocks, batch normalization layers, and a Maxpooling layer, all of which extract different sizes of depth features from input images.

Unlike the traditional U-Net, these convolution blocks are built based on ResNet [89], which develops a deep residual learning approach to tackle the problem of gradient explosion or disappearance during backpropagation. It specifically allows these layers to stack the output directly to the next layer without passing through the weight layer, which is also defined as residual mapping. The deep residual learning approach centers around the residual block, illustrated in Fig. 3.5, which incorporates the desired underlying mapping as $\mathcal{H}(\mathbf{x})$ and the activation function as $\mathcal{F}(\mathbf{x})$. Without the skip connection, input x undergoes weight multiplication, followed by the addition of a bias term during network optimization.

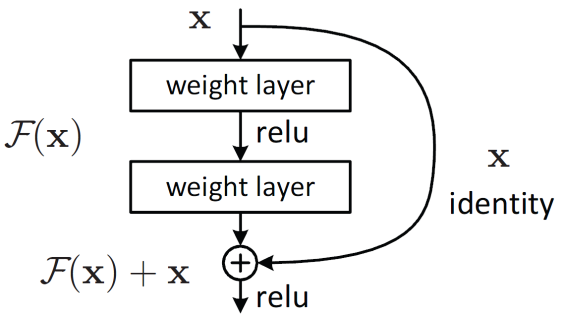


FIGURE 3.5: Diagram of the residual block.

Then $\mathcal{H}(\mathbf{x})$ can be represented as:

$$\mathcal{H}(\mathbf{x}) = \mathcal{F}(\mathbf{w}\mathbf{x} + \mathbf{b}). \tag{3.8}$$

However, with the introduction of the shortcut shown in the above figure, the desired underlying mapping can be recast into

$$\mathcal{H}(\mathbf{x}) = \mathcal{F}(\mathbf{x}) + \mathbf{x}. \tag{3.9}$$

The main purpose of introducing skip connections is to allow the network to adaptively determine whether or not to perform identity mapping, which is an integral part of nonlinear function modelling, even though it is only an ordinary linear mapping. As mentioned before, identity shortcut connections are capable of converting the original multiplication operations into addition operations, avoiding introducing extra parameters and decreasing computational complexity simultaneously. As the model's complexity increases, modelling an identity mapping deploying stacked layers becomes challenging. To speed up the training process, ResNet proposes optimizing the difference between the desired and identity mapping, since it is easier to push the residual to zero than to fit a feasible identity mapping with nonlinear layers.

Unlike the traditional ResNet50 backbone, the output channels of each convolution block of the proposed model are 32, 64, 128, 256, and 512, removing the convolution block related to 1024 output channels to accelerate depth estimation. The decoder comprises the same convolution blocks as the encoder, upsampling layers, and a Softmax layer to restore the extracted features to the original size. Among them, the upsampling layers serve to gradually scale up the image to its original resolution while simultaneously reducing the number of channels in the output result. The output channels of the first four convolution blocks in the decoder are 512, 256, 128, and 64, respectively, and the output channel of the last convolution blocks is 1. Skip connections combine the information in the shallower and the deeper layers, making the network leverage fine-grained features learned in the encoder to predict depth in the decoder. In more detail, the coarse information in the encoder is resized and stacked with the decoder to accelerate training. The detailed architectures of the encoder and decoder of the proposed depth estimation subnetwork are depicted in Table 3.1 and Table 3.2.

E. Multi-Scale Upsampling

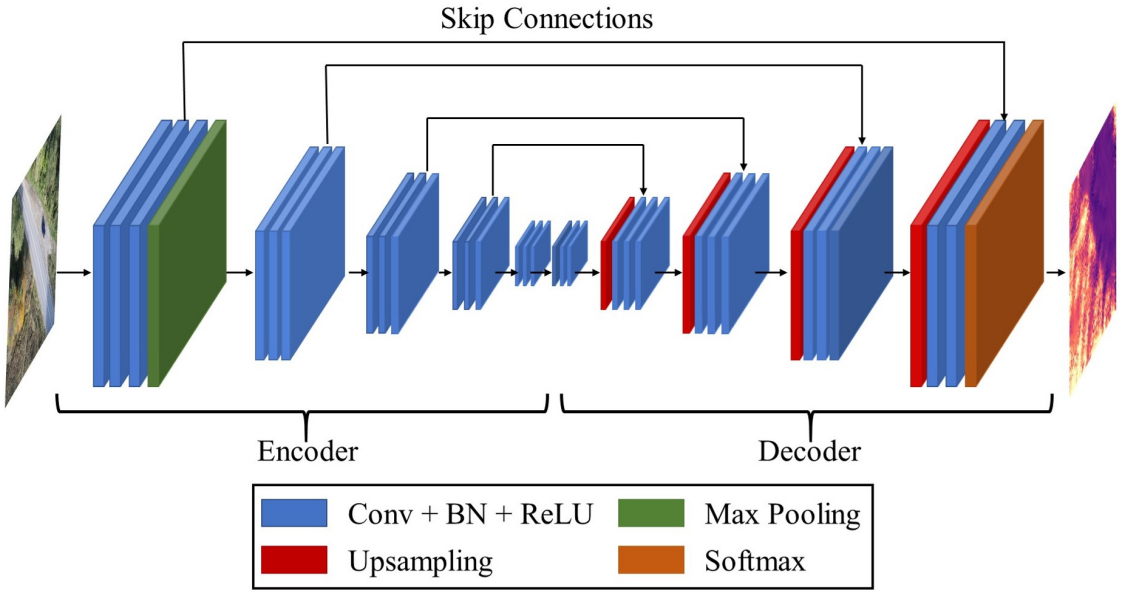
TABLE 3.1: Detailed architecture of encoder

Name of Layer	Kernel Size	Stride	Number of Channels	Repeat Times
Conv 1	7×7	2	3/32	3
Pool 1	3×3	2	-	1
Resblock1	$\begin{bmatrix} 1 \times 1 \\ 3 \times 3 \\ 1 \times 1 \end{bmatrix}$	1	32/64	3
Resblock2	$\begin{bmatrix} 1 \times 1 \\ 3 \times 3 \\ 1 \times 1 \end{bmatrix}$	1	64/128	3
Resblock3	$\begin{bmatrix} 1 \times 1 \\ 3 \times 3 \\ 1 \times 1 \end{bmatrix}$	1	128/256	3
Resblock4	$\begin{bmatrix} 1 \times 1 \\ 3 \times 3 \\ 1 \times 1 \end{bmatrix}$	1	256/512	3

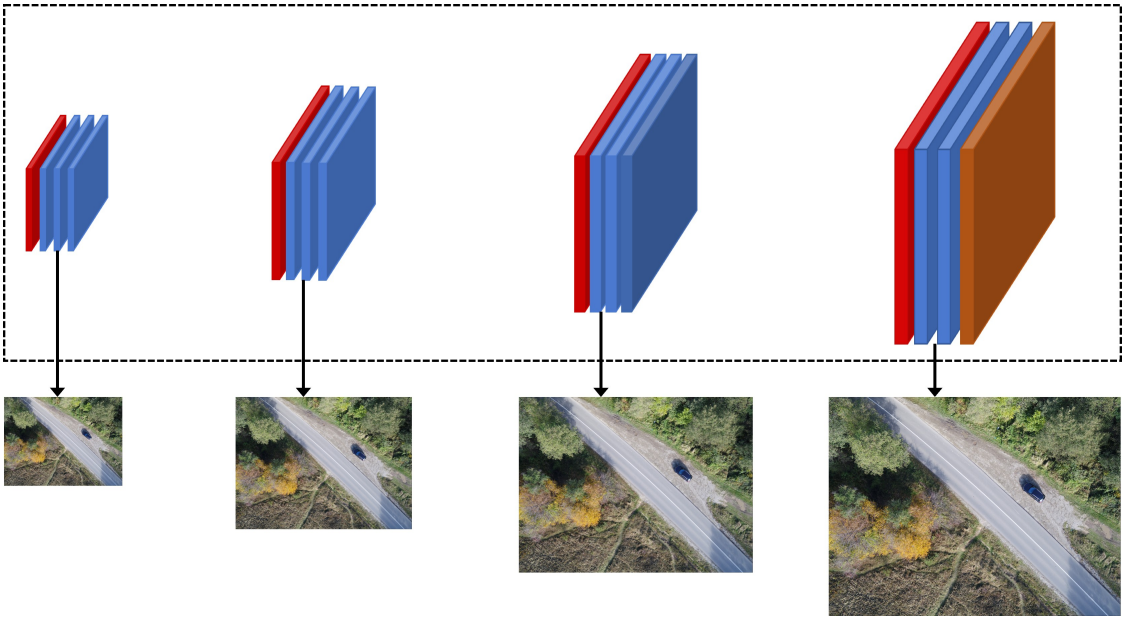
To comprehensively combine the features extracted from all decoder layers, which are conducive to predicting the depth map more accurately, a multi-scale upsampling estimation method is proposed to calculate the weighted loss sum in all decoder layers. More specifically, it is shown in Fig. 3.6(b) that images with four different resolutions are produced during the decoder. Losses obtained at different scales in the decoder can be calculated compared with their corresponding reconstructed images, respectively. Therefore, the total loss is the combination of each photometric error at the resolution of each decoder layer. Compared with the deeper layers, shallower layers of the network provide less contour information related to the depth map. The final loss is a weighted sum of losses in each layer, and the weights are supposed to increase gradually from shallower to deeper layers.

F. Pose Estimation Subnetwork

As for the pose estimation subnetwork, a modified ResNet50 is also utilized as the encoder. The encoder accepts two adjacent frames(six channels) as input to extract features, and the output channels of the convolution block in the encoder increase



(a) Architecture of the depth estimation network.



(b) Reconstructed images at different resolutions in the decoder.

FIGURE 3.6: Framework of the depth estimation network.

TABLE 3.2: Detailed architecture of decoder

Name of Layer	Kernel Size	Stride	Number of Channels	Repeat Times
Conv 2	3×3	2	512/512	3
Upconv 1	3×3	2	512/256	1
Conv 3	3×3	1	512/256	3
Upconv 2	3×3	2	256/128	1
Conv3	3×3	1	256/128	3
Upconv 3	3×3	2	128/64	1
Conv 4	3×3	1	128/64	3
Upconv 4	3×3	2	64/32	1
Conv 5	3×3	1	64/32	1
Conv 6	3×3	1	32/1	1

from 16 to 512 exponentially. The decoder comprises five convolution blocks followed by a global average pooling layer, and the final output channels of the decoder are 6, corresponding to the 6-DoF relative camera pose. The architecture of the pose estimation subnetworks is shown in Fig. 3.7.

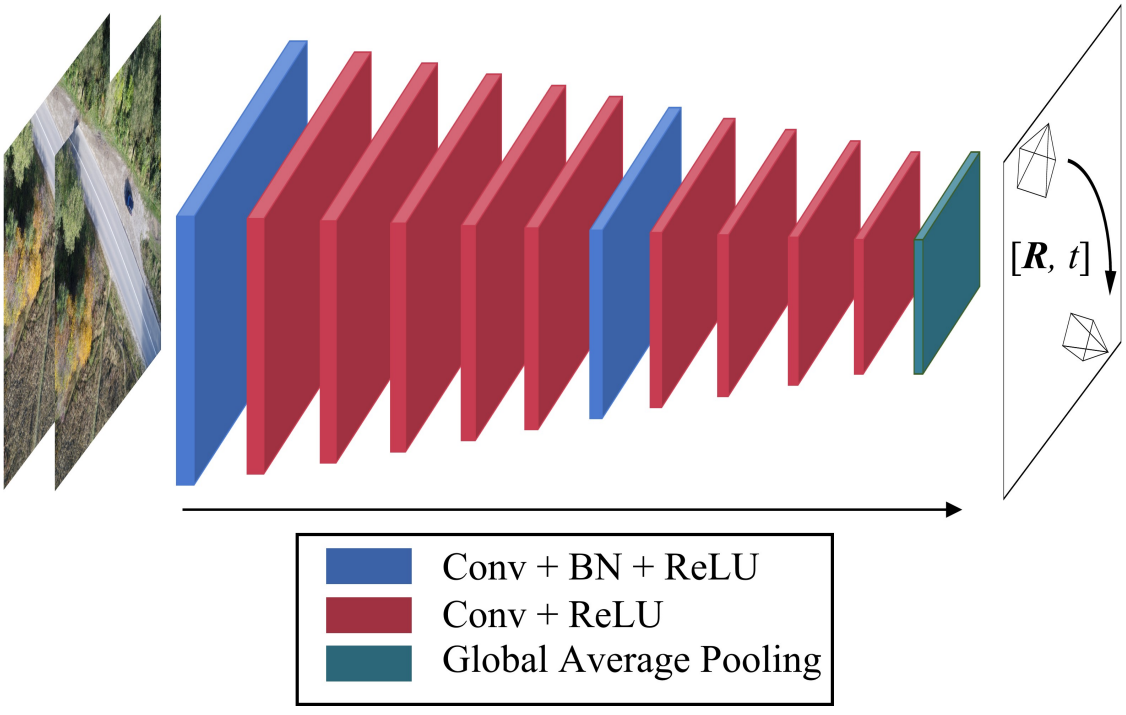


FIGURE 3.7: Architecture of the pose estimation network.

G. Automatic Mask

The network proposed in this chapter must meet the prerequisites of a moving camera and a stationary background. However, these assumptions are difficult to meet in real-world application scenarios. For example, prediction can degenerate significantly when UAVs are required to perform hovering operations in texture-less environments or there is a moving object in the current scene. To solve this problem, we employ an automatic mask μ to exclude the loss of pixels where the reprojection error of the reconstructed image $I_{r\to t}$ is higher than that of the unwarped reference image I_r. Since a static camera or an object moving at a similar velocity as the camera always produces identical pixels between adjoining frames, this formulation can filter out these pixels that break down the previous assumptions.

$$\mu = \left[\min_r pe\left(I_t, I_{r\to t}\right) < \min_r pe\left(I_t, I_r\right)\right], \tag{3.10}$$

where [ ] is the Iverson bracket, then μ is 1 when the conditions are determined to be ture.

H. Loss Function

Our model is guided to update the network parameters in the direction of minimizing photometric reprojection error. We denote the relative pose between the target image I_t, and the reference image I_r, (I_{t-1} or I_{t+1},) as $T_{t\to r}$. Our model tends to establish a dense depth map D_t that generates the minimum photometric reprojection error L_p. Initially, the reconstructed image $I_{r\to t}$ can be obtained utilizing the following formulation:

$$I_{r\to t} = I_r \left\langle \text{proj}\left(D_t, T_{t\to r}, K\right)\right\rangle, \tag{3.11}$$

where proj() are the resulting 2D coordinates of the projected depth D_t in I_r, $\langle\ \rangle$ is the bilinear sampling operator, and K is the camera intrinsic matrix.

Minimum photometric reprojection loss L_p is adopted as one of the loss indicators between I_t and $I_{t\to r}$, the formulation can be expressed as:

$$L_p = \min_r pe\left(I_t, I_{r\to t}\right), \tag{3.12}$$

where pe represents the photometric reprojection loss and $\min_r$ denotes the minimum value of pe.

Inspired by reference [90], the most frequently leveraged loss functions in DL are $L1$ and $L2$ losses. Among them, $L1$ loss serves to minimize the absolute difference between the model's predicted values $\tilde{y}$ and true values y. It can be mathematically represented as $L1 = \|y - \tilde{y}\|_1$. $L1$ loss is known for its robustness and adaptability to outliers, allowing for multiple solutions, but its non-derivable property may lead to unstable solutions. Meanwhile, $L2$ loss is responsible for minimizing the squared error between the model's predicted values $\tilde{y}$ and true values y. It can be mathematically represented as $L2 = \|y - \tilde{y}\|_2^2$. $L2$ loss is sensitive to outliers. This sensitivity to outliers can result in instability and divergence of the training.

Considering the effect of a large amount of noise and outliers at the early stage of training, we build pe based on $L1$ loss and Structural Similarity Index (SSIM) following the methods in [37]. Among them, SSIM is an indicator leveraged to measure the degree of similarity between two images. As an image has a nonnegligible structural property, there is a strong correlation between adjacent pixels in the image. This correlation between pixels is an essential carrier of the scene's structural information. The most sensitive noise signals in the human visual system are local luminance, color contrast, and scene structure in the field of view. SSIM well captures the characteristics of the human visual system that is sensitive to scene structure. SSIM extracts spatial scene information from the image from the perspective of image composition, making this index an independent attribute of scene brightness and contrast, which is not considered by conventional methods of image similarity comparison. SSIM consists of three aspects: the image's local brightness, local contrast, and scene structure. The mean value represents the similarity of local brightness, the standard deviation reflects the similarity of contrast, and the covariance represents the similarity of scene structure. Therefore, SSIM can be expressed as follows:

$$\mathrm{SSIM}(\tilde{I}, I) = L(\tilde{I}, I) \cdot C(\tilde{I}, I) \cdot S(\tilde{I}, I), \tag{3.13}$$

where $L(\tilde{I}, I)$, $C(\tilde{I}, I)$, and $S(\tilde{I}, I)$ refer to the similarity of local brightness, local contrast, and scene structure, respectively, which can be mathematically depicted below:

$$L(\tilde{I}, I) = \frac{2\mu_I \mu_{\tilde{I}} + C_1}{\mu_I^2 + \mu_{\tilde{I}}^2 + \tilde{I}}, \tag{3.14}$$

$$C(\tilde{I}, I) = \frac{2\sigma_I \sigma_{\tilde{I}} + C_2}{\sigma_I^2 + {\sigma_{\tilde{I}}}^2 + C_2}, \tag{3.15}$$

$$S(\tilde{I}, I) = \frac{\sigma_{\tilde{I}I} + C_3}{\sigma_I \sigma_{\tilde{I}} + C_3}. \tag{3.16}$$

Here, I is the original input image, $\tilde{I}$ is the reconstructed image, μ_I, $\mu_{\tilde{I}}$ refer to the mean values of I, $\tilde{I}$, respectively, σ_I, $\sigma_{\tilde{I}}$ represent the variances of I, $\tilde{I}$, respectively, $\sigma_{\tilde{I}I}$ denotes the covariance of I, $\tilde{I}$, C_1, C_2, and C_3 are constants. Typically, $C_3 = 2C_2$ to protect the denominator from being 0, therefore SSIM can be simplified as:

$$\text{SSIM} = \frac{(2\mu_I \mu_{\tilde{I}} + C_1)(2\sigma_{\tilde{I}I} + C_2)}{\left(\mu_I^2 + \mu_{\tilde{I}}^2 + C_1\right)\left(\sigma_I^2 + \sigma_{\tilde{I}}^2 + C_2\right)}. \tag{3.17}$$

Inspired by reference [91], constants C_1 and C_2 are preset as 0.01^2 and 0.03^2, respectively.

Following the methods in [37], pe can be calculated based on $L1$ distance and SSIM:

$$pe\left(I_{r\to t}, I_t\right) = \frac{\alpha}{2}\left(1 - \text{SSIM}\left(I_{r\to t}, I_t\right)\right) + (1-\alpha)\left\|I_{r\to t} - I_t\right\|_1, \tag{3.18}$$

where α is 0.85 following the setting of reference [37], $\left\|I_{r\to t} - I_t\right\|_1$ represents the $L1$-norm, the value range of SSIM is $[0, 1]$, and the degree of similarity between the reconstructed and original images increases as the value approaches 1.

The depth values of neighboring pixels in an image are prone to be identical or close to each other, which also does not change dramatically at a certain point. In order to smooth the depth transition of edge pixels, this chapter also adds depth smoothing loss to the estimation network, which can be obtained by calculating the disparity gradient change. By reducing this depth smoothing loss, the accuracy and quality of the output depth map can be further improved. Therefore, the edge-aware smoothness loss L_s is introduced to form the overall loss function together with the photometric reprojection loss:

$$L_s = |\partial_x d_t^*| \, e^{-|\partial_x I_t|} + |\partial_y d_t^*| \, ^{-|\partial_y I_t|} , \tag{3.19}$$

where $d_t^* = d_t / \bar{d}_t$ is the inverse depth normalized by the mean value.

In Eq. 3.19, the edge-aware smoothness loss includes two parts: gradient changes in the disparity map and the original RGB image. Typically, there are massive gaps in sharp edge areas, including color, contrast, and shape, which means that the gradient here varies considerably. Therefore, $e^{-|\partial_x I_t|}$ and $e^{-|\partial_y I_t|}$ exist to prevent the effect of edge regions on the overall loss function while constraining gradient changes in smoothed regions, thus ensuring that continuous regions in the original image can maintain to be smoothed in the disparity map.

Since the features extracted in the low-resolution layers of the decoder reveal less about the contour of the depth map, which may lead to inaccurate network prediction, assigning the same weights to the low-resolution and high-resolution layers tends to result in the divergence of the network. Therefore, the overall loss function can be written as follows:

$$L = \sum_i w_i \left(\mu L_{p,i} + \lambda L_{s,i} \right) \tag{3.20}$$

where subscript i represents the layers of the decoder at different resolutions, w_i denotes the weights determined by the resolution, and λ is the smoothing coefficient.

3.2 Simulation Results

A. Dataset

WildUAV [87] is employed as the training dataset. WildUAV raw dataset contains over 1500 high-resolution (5280 $\times$ 3956) images, and their corresponding depth ground truth data is provided jointly with camera intrinsics. The scenarios in the captured images contain both unstructured forest areas with various landforms and structured objects such as driveways, vehicles, and pedestrians. Some samples of the dataset are shown in Fig. 3.8.

Although WildUAV comprises high-resolution images with rich information, the total number of images is insufficient to avoid overfitting. Therefore, images are pre-processed before training. Ten sub-images are cropped from the center part for each image because the center part typically contains richer texture information than the boundary part. Subsequently, every sub-image is linked with its adjacent frame to get ten sequences. Then data segmentation is performed on these sequences to obtain the final training and validation sets. The segmentation approaches deployed are as follows:

- Flip the image horizontally with a 50% probability.
- Adjust the γ value of the image with a 50% probability, γ value is varied in the range of $[0.8, 1.2]$.
- Adjust the brightness of the image with a 50% probability, brightness value is varied in the range of $[0.5, 2.0]$.
- Take a random number from $[0.8, 1.2]$, and multiply the values of original RGB channels by this number to get a new image color.

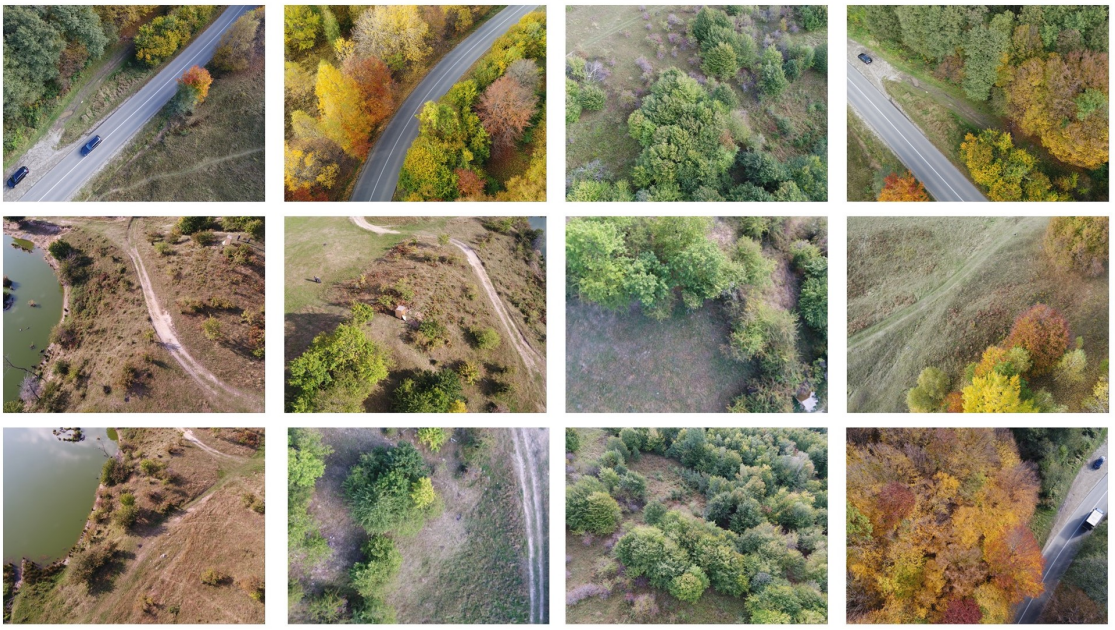


FIGURE 3.8: Samples of the dataset

Eventually, the training set comprises 11961 images, while the validation set includes 1329 images and the test set contains 2100 images.

B. Implementation Details

Only monocular image sequences are utilized as the training set, and the network is trained in the development environment of Ubuntu 20.04, Pytorch 1.8.2, and CUDA 11.3. The details of the parameter setting are as follows: we set $\lambda = 0.001$, $w = \{0.125, 0.25, 0.5, 1\}$ accordingly. The training process of the network involves 30 epochs with a batch size of 8. During the first 25 epochs, the learning rate is set to 2×10^{-4} and then switched to 10^{-4} for the remaining epochs. The training on a computer with four NVIDIA T4 Tensor Core GPUs, each with 2560 CUDA cores and 16 GB of memory, takes approximately 6 hours.

C. Performance Evaluation Metrics

To quantitatively examine the prediction results, several commonly used evaluation metrics in prior works are employed as indicators of judging the performance of models. Their corresponding formulas can be written as follows:

$$\text{Abs Rel} = \frac{1}{N}\sum_{i=1}^{N}\frac{|D_i - D_i^*|}{D_i^*}, \tag{3.21}$$

$$\text{Sq Rel} = \frac{1}{N}\sum_{i=1}^{N}\frac{|D_i - D_i^*|^2}{D_i^*}, \tag{3.22}$$

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}|D_i - D_i^*|^2}, \tag{3.23}$$

$$\text{LG RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}|\lg D_i - \lg {D_i}^*|^2}, \tag{3.24}$$

$$\text{Accuracy} : \max\left(\frac{D_i}{D_i^*}, \frac{D_i^*}{D_i}\right) = \delta < T. \tag{3.25}$$

In the above formulas, N is the total number of pixels in the target image, D_i and D_i^* denote the estimated depth and the depth ground truth, and T is the artificially set thresholds $\{1.25, 1.25^2, 1.25^3\}$.

D. Analysis of Results

Estimation results are compared with Monodepth [36] and Monodepth2 [37], both of which are unsupervised monocular depth estimation models showing great results on the KITTI [30] dataset. The estimation results are compared in Fig. 3.9.

It can be seen from Fig. 3.9 that the proposed model produces a sharper depth map than the comparative models. Compared with the depth ground truth, the proposed model performs better on the depth estimation of the drive lanes, while Monodepth and Monodepth2 show weaker performance in predicting their depth. Since the two monocular depth estimation models leveraged for comparison apply shallower networks than ours, they are prone to fail when encountering environments with less texture and longer distances in UAV application scenarios. Combined deeper networks with the weighted loss, the proposed model can better adapt to textureless environments and more accurately predict depth information for densely wooded areas and sparse driveways. It can be speculated that the introduction of deeper networks and weighted loss brings more promising prediction results.

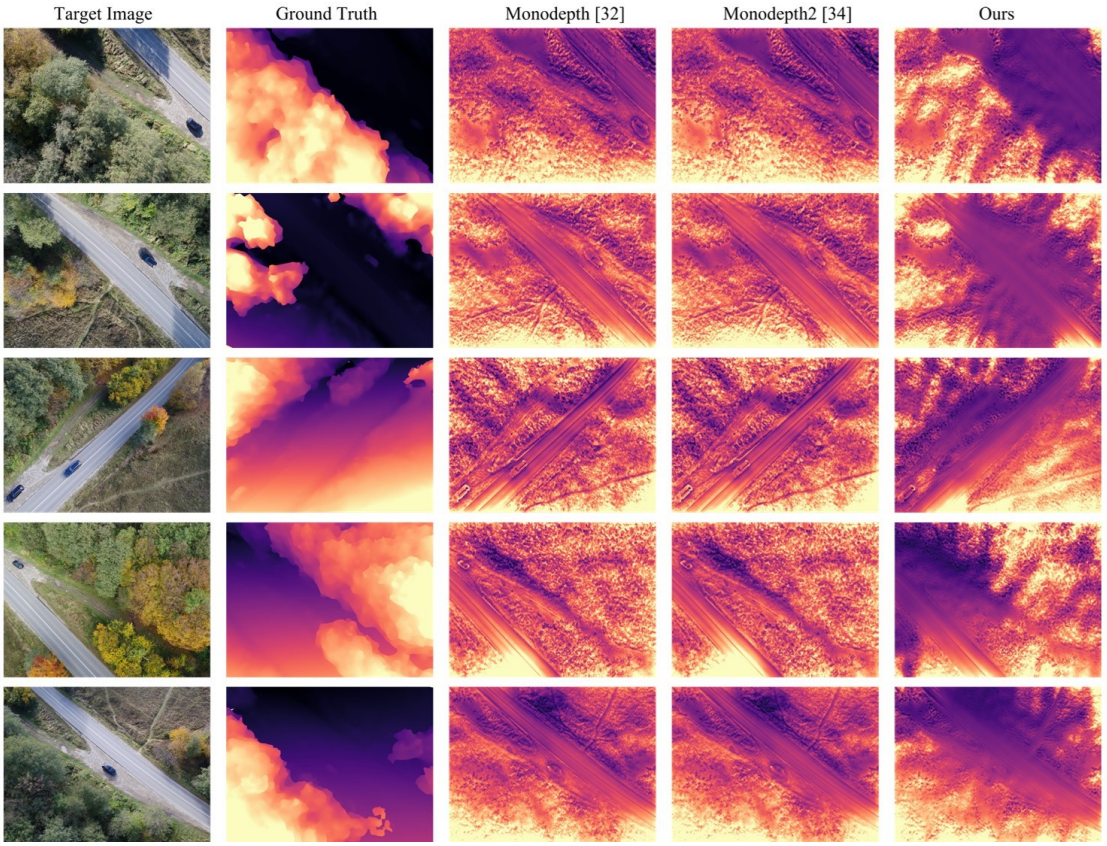


FIGURE 3.9: Estimation results on the WildUAV dataset

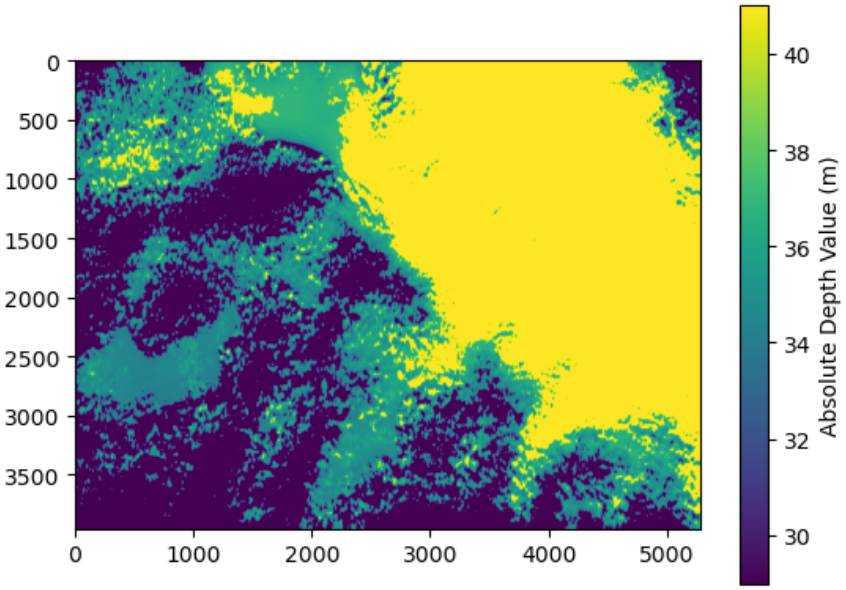
The above results can only qualitatively judge the model's performance. In order to quantitatively analyze the estimated results, the absolute predicted values are required to compare with the ground truth. Since the last layer of the depth estimation subnetwork deploys a Sigmoid activation function, the values of the final depth map are floating point numbers between 0 and 1, which means they can only represent the relative depth relationship instead of absolute depth values between individual pixels. The distance measurement range must be specified before converting the depth map values into the actual distances. As the actual depth values of the dataset deployed for training are between $2m$ and $98m$, the minimum distance is set to $1m$, and the maximum distance is set to $100m$ considering the associated error. The relative depth value can also be defined as the disparity, roughly considered the inverse of the depth. As mentioned in Eq. A.6, disparity values can be converted into absolute depth values following the method presented in [37]:

$$D = 1/[(a - b)\sigma + b], \tag{3.26}$$

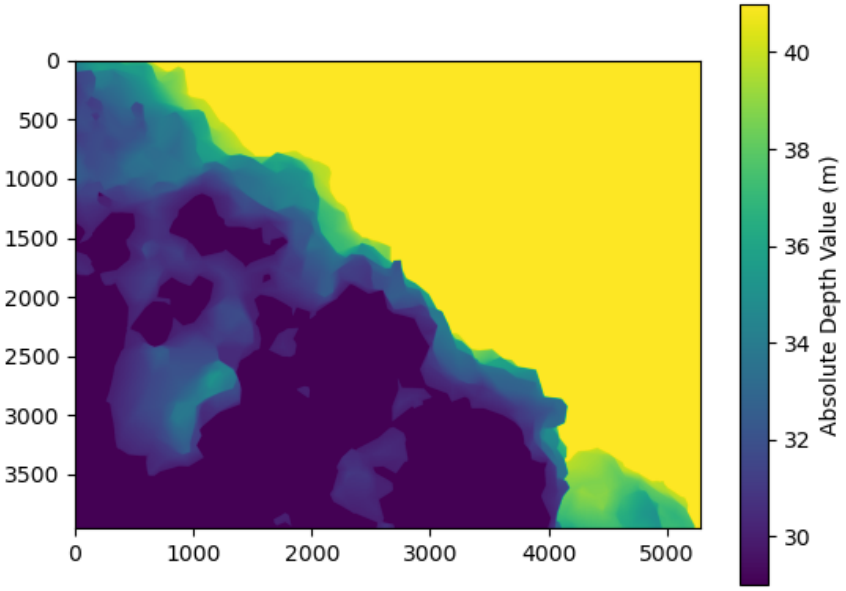
where D refers to the absolute depth value, σ represents the disparity value, a denotes the reciprocal of the maximum distance, and b is the reciprocal of the minimum distance.

After completing the conversion process, the results can be leveraged to compare with the ground truth. The comparison result of one example can be depicted in Fig. 3.10.

It can be concluded from Fig. 3.10 that the predicted and actual results' depth values are both between $30m$ and $45m$, and their distributions are relatively similar. Additionally, given the significant variation in depth span ranging from $10m$ to $80m$ in this scenario, this chapter considers the model's errors within this range to investigate the performance variance across different depth levels. It can be concluded from the results that errors are basically around 10 percent, which is tolerable in practical applications [92, 93]. Furthermore, the model exhibits greater inaccuracy in its predictions at larger scales, whereas it is relatively accurate at smaller scales. The existence of this discrepancy could be associated with the low contrast and less texture present in the larger-scale environment. Errors of the proposed model are depicted in Table 3.3.



(a) Depth distribution of predicted result



(b) Depth distribution of ground truth

FIGURE 3.10: Comparison of predicted result and ground truth

TABLE 3.3: Errors of the proposed model

Actual Dist(m)	**Estimated Dist(m)**	**Absolute Err(m)**	**Relative Err(%)**
84	75.67	8.33	9.92
78	71.18	6.82	8.74
63	70.81	7.81	12.40
50	43.66	6.34	12.68
41	45.53	4.53	11.05
37	31.91	5.09	13.76
22	19.59	2.41	10.95
8	8.37	0.37	4.63
Average Err(m)		5.21	

TABLE 3.4: Configurations of each monocular depth estimation model

Method	**Loss**		
	Photometric	Smoothness	Weighted
Monodepth [36]	✓	✓	✗
Monodepth2 [37]	✓	✓	✗
Ours w/o wt	✓	✓	✗
Ours w/ wt	✓	✓	✓

TABLE 3.5: Performance of each monocular depth estimation model

Method	**Errors**				**Accuracy (%)**		
	Abs Rel	Sq Rel	RMSE	RMSE log	$\delta < 1.25$	$\delta < 1.25^2$	$\delta < 1.25^3$
Monodepth [36]	0.187	1.503	6.133	0.25	0.795	0.928	0.969
Monodepth2 [37]	0.174	1.435	5.786	0.221	0.831	0.939	0.974
Ours w/o wt	0.182	1.449	5.775	0.232	0.828	0.934	0.972
Ours w/ wt	**0.149**	**1.219**	**5.246**	**0.216**	**0.857**	**0.941**	**0.978**

To further illustrate the performance of the model, the proposed model is also evaluated without introducing weight loss, and Table 3.4 reveals the configuration of each model. In this table, Ours w/ wt represents the model with weighted loss applied, and Ours w/o wt represents the model without weighted loss applied. Table 3.5 reveals the estimation results of each model. In this table, better performance is characterized by more minor errors and higher accuracy. The results show that weighted loss positively impacts the training process because the model with weighted loss owns the lowest error and highest accuracy among the four models.

Therefore, it can be inferred that after assigning a weight to the loss of shallower layers in the decoder, which provide less contour information of the depth map, the total loss function is less contaminated, leading to improved convergence of the model.

3.3 Conclusion and Discussion

This chapter proposes a self-supervised depth estimation model to forecast the depth map from datasets captured by UAVs at low altitudes. This model comprises depth and pose estimation subnetworks which are trained jointly to forecast depth given a series of monocular images. This model leverages a modified loss function that weights the different layers' photometric loss according to their corresponding resolutions. Subsequently, this model is trained on a low-altitude UAV dataset, and evaluation results are compared with other self-supervised MDE models (Monodepth and Monodepth2) to manifest its superiority. The results demonstrate that the model produces surpassing results by introducing weighted reconstruction loss. This self-supervised MDE model can contribute to the OD phase of an ODA system.

In addition to the accuracy of the model prediction, we also considered the performance of the model in terms of other hardware constraints in a comprehensive manner. Firstly, regarding the lens angle-of-view, in our simulations, the camera provided by Gazebo captures distortion-free images with a fixed angle of view. Hence we expect to consider the effect of lens angle of view and distortion on model performance in future physical experiments without over-considering it in ideal simulation conditions. Secondly, the camera's frequency and resolution also primarily affect the model's computational speed. Since the UAV processes the following image input only after executing an evasive maneuver during the subsequent obstacle avoidance module, the difference in camera resolution observed in our simulations mainly impacts computational speed rather than the overall model performance. Consequently, to optimize computational efficiency, the camera is configured to capture images at a resolution of 160×120 pixels and a frame rate of $30Hz$. Lastly, in terms of computational resource consumption, the proposed model can achieve inference at approximately 30Hz on the CPU, which is most feasible to process images in simulated experiments. However, considering that

practical applications may require real-time inference for more demanding inputs such as video streams, the model's performance in physical experiments still requires further validation.

The model proposed in this chapter is capable of providing a priori information to the OA part. During UAV operations, UAVs will measure the distance between the airframe and the surrounding environment in real-time by virtue of onboard sensors (monocular camera) and the proposed model. The nature of the UAV's inability to take sharp turns places greater demands on the application of the ODA system. Hence it is necessary to design a reasonable reaction distance for UAVs in advance, which is also defined as a minimum turn radius [16]. A minimum turn radius refers to the minimum distance at which UAVs can make an evasive maneuver. When the distance between UAVs and the surrounding objects is less than a minimum turn radius, the OA task is regarded as a failure. UAVs need to reperform the evasive maneuver. Otherwise, UAVs can maintain their current heading.

The model proposed in this chapter is trained under a realistic scenario-based dataset. However, the subsequent OA task in this study is performed in a simulated environment, where the illumination and texture conditions are synthetic and relatively unrealistic. Some factors in the simulated environment may affect the accuracy of the model's depth estimation of obstacles, which may require additional consideration in subsequent studies.

- Blurred Image. Significant rotation or shaking during image capture can result in blurring of the entire image, leading to a significant reduction in image structure similarity and potentially resulting in substantial estimation error.

- Strong illumination source interference or insufficient light. Simulated environments may differ significantly from real-life circumstances, resulting in large black areas in the obstacle's texture that make it difficult to estimate depth based on color and structural similarity. These areas tend to trigger significant errors. Furthermore, intense illumination sources, such as direct illumination of a certain intensity, can also lead to interference.

- Too proximate obstacles. The proposed model performs better in predicting results on small-scale scenarios compared to large-scale ones. However,

objects that are too close (less than 0.5 m away) may fall out of the imaging range and create outrageous changes in adjacent frames during training, thereby increasing the learning difficulty.

- Dense or homogeneous objects. When the background is too densely packed or has a homogeneous color-texture structure, the introduced edge-aware smoothness loss during training may cause the depth estimation model to perceive the partially differentiated background as a whole or neglect specific details.

Chapter 4

Depth Information-Based Obstacle Avoidance through Deep Reinforcement Learning

Designing a reactive OA strategy for UAVs leveraging DRL techniques, such as DQN and its derivatives, is the initial task in this chapter. The approach utilizes depth images acquired from the model introduced in Chapter 4 and target point data as input states for the UAV, which are then fed into a policy network. The network generates a corresponding action stemming from the input, evaluates the action's quality through the environment's feedback, and ultimately constructs a policy mapping that connects the environment to performed evasive maneuvers via network training. Subsequently, this chapter builds a simulation platform for training and validation based on ROS/Gazebo, which contains different obstacle scenarios and a quadrotor to investigate the viability of the presented OA system. The main contributions are as follows:

- An end-to-end framework is presented to solve the ODA problem for UAVs, with limited observation information provided by monocular cameras.
- A DRL-based policy network is designed to complete the direct mapping between the input depth images and UAV's action space (linear and angular velocity).

- A simulation platform, which contains a variety of scenarios and a quadrotor, is established based on ROS/Gazebo to train the DRL model and validate the feasibility of the proposed ODA system.

4.1 Methodology

A. Problem Definition

In the case of the OA problem for UAV navigation, the flight strategy must be determined based on environmental information to ensure safe arrival at the destination. These problems exhibit a common feature where the agent is supposed to to make a decision and take a feasible action according to the current circumstances to achieve its objective. RL [94] is one of the prominent algorithms for an intelligent agent to perceive the environment and make autonomous decisions.

Figure. 4.1 illustrates a diagrammatic representation of a reinforcement learning (RL) system, which involves agents possessing perception and behavior capabilities, such as UAVs or autonomous robots. The agent receives observations of the environment's state, determines a policy, and carries out an action resting on the observation. The environment is the scenario in which the interaction with the intelligent agent occurs. In this study, UAVs observe the environment through onboard sensors and select the following action in accordance with the observation information and its policy. After executing the action, the environment rewards the action and offers a new state for the following phase. Simultaneously, the UAV updates its policy according to the feedback from the environment.

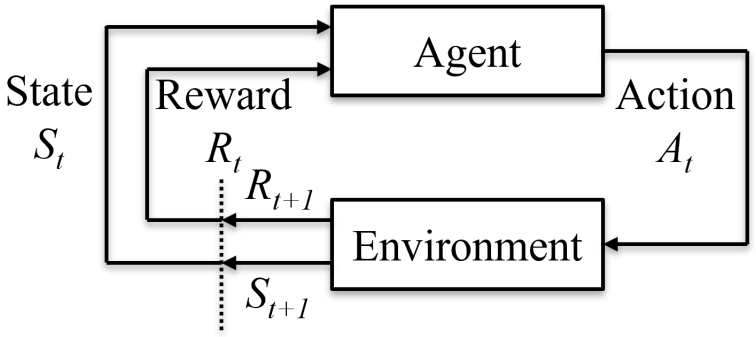


FIGURE 4.1: Relationship between agent and environment.

A policy is a mapping relationship between the current status of the environment and action. Policies are divided into two categories: deterministic and stochastic, where a deterministic policy outputs the specific action selected, and a stochastic policy determines actions considering a probability distribution.

A return refers to the feedback that an agent receives as a result of executing an action. It is generally calculated by pre-determined functions, which define an evaluation criterion for the goodness of the action. When the agent receives favorable feedback for an action it performs, it will tend to carry out that action the next time it encounters a similar situation. Conversely, it will avoid performing an action that is unfavorable to itself. The agent adapts its policy and gains the capacity to distinguish between various states through ongoing interactions with the environment. The primary goal of RL is to discover the best course of action that maximizes the expected return.

The main objective of this chapter is to address the reactive OA problem of UAVs by proposing an end-to-end control framework leveraging DRL to enhance the intelligence of autonomous OA. Figure. 4.2 depicts the reactive OA framework based on DRL, where the UAV senses the environment by deploying onboard sensors, primarily a monocular camera, to capture environmental information. The state expression of the environment is then extracted as input to the network, and through network training, a policy mapping is constructed between the environment and OA behaviors.

To ensure a safe operation, the UAV must be conscious of all available information in the external environment, which is defined as the state s. The state refers to a property related to the environment and benefits the UAV's navigation. The state s directly determines the UAV's action a implementation and reward value r generation. The action selected affects the observation information obtained by the UAV, implying that the specific movement options of the UAV result in a change in its location, direction, observation information, and reward feedback. The UAV is required to adopt an action counting on the state at each time step t during navigation tasks. Additionally, the action performed in the previous moment impacts subsequent states and therefore the following decision-making process of the UAV. Successive state sequences are typically probabilistic considering the performed action, indicating that autonomous ODA comprises a series of decision-making

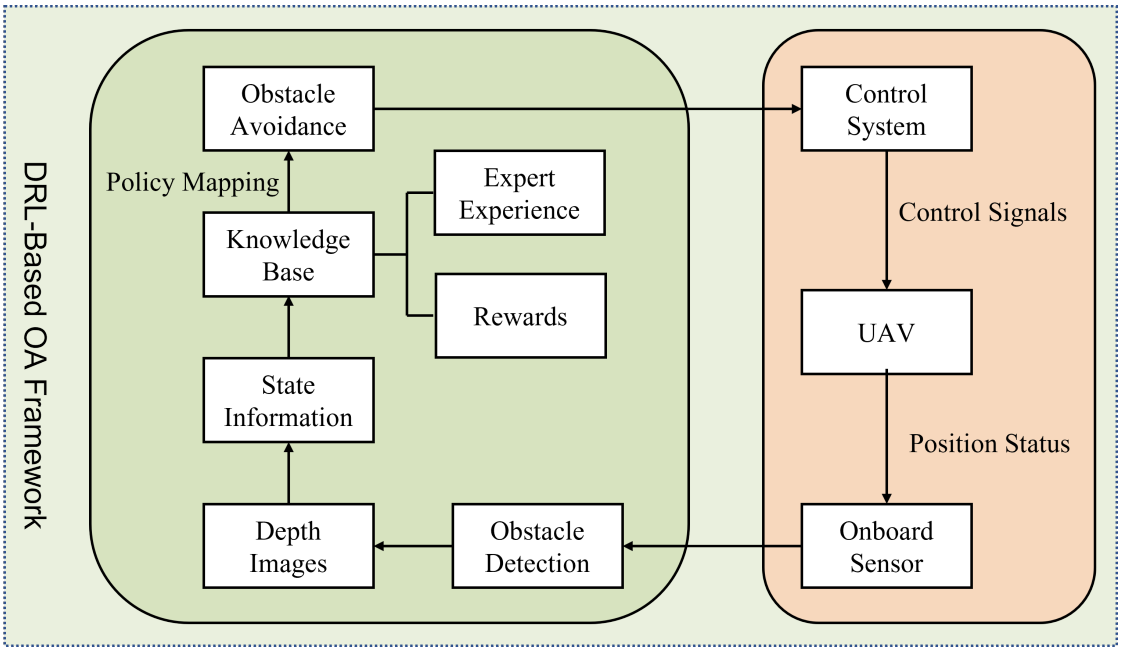


FIGURE 4.2: Framework of the reactive OA system through DRL.

problems under uncertainty. The overall framework of the proposed ODA system in this chapter for UAVs is depicted in Fig. 4.3.

Although previous studies have explained that monocular images contain a wealth of information about the environmental state, onboard monocular cameras only provide a forward view and are not qualified indicators of the complete state of the environment. Although depth maps can be inferred from monocular images leveraging the model proposed in Chapter 4, the decision module of the UAV can only get one evaluation of the state information and rely on it to select action at the next moment. Considering the above, this chapter proposes a partially observable Markov decision process (POMDP) to describe the UAV's OA problem.

This chapter models UAVs' end-to-end autonomous OA process as a Markov decision process. The UAV leverages depth images as its state inputs to the end-to-end OA system, and the output is a motion control command. After the UAV executes the corresponding action according to the command, its attitude will change so that the UAV will enter a new state and be rewarded accordingly, and a time series $(s_1, a_1, r_1, s_2, a_2, r_2, \ldots, a_{t-1}, s_t)$ will be generated. Therefore, the autonomous OA problem for UAVs can be solved by RL-based methods. Meanwhile, since the

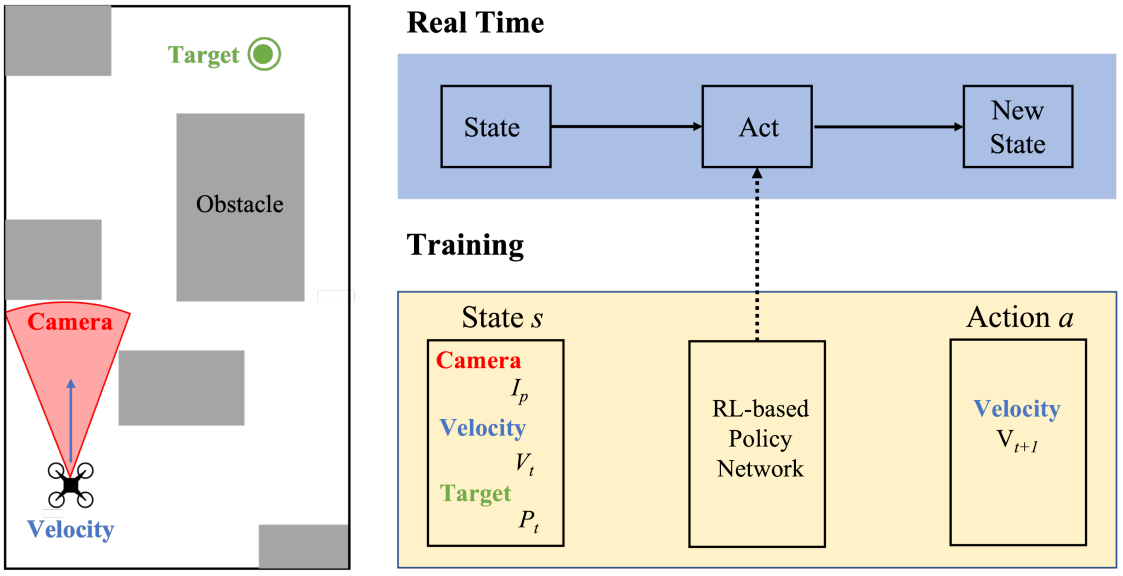


FIGURE 4.3: Framework of the proposed ODA system.

network's inputs are images, CNNs can be leveraged to extract features from images and speed up the training process of the network. In summary, DRL methods can be deployed to tackle UAVs' autonomous OA problem.

B. Partially Observable Markov Decision Process

The POMDP framework consists of six elements denoted by the tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \Omega, \mathcal{O})$. $\mathcal{S}$ represents the environment state space. $\mathcal{A}$ is the action space of the agent that contains a set of motion commands. $\mathcal{P}$ is the state transition model, which indicates the probability distribution of state evolution according to the chosen action, and can be expressed as $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$. $\mathcal{R}$ is the reward function that provides immediate feedback on the actions taken by the UAV. For example, if a UAV avoids an obstacle by choosing an action a in the current state s, it is considered beneficial, and the reward value is positive. Conversely, if the selected action results in a collision or goes out of bounds, the corresponding reward value is negative. Ω defines the observation space, and an observation $o \in \Omega$ reflects the actual state s. $\mathcal{O} : \mathcal{S} \times \mathcal{A} \times \Omega \rightarrow [0, 1]$ represents the observation probability distribution based on the state and action spaces. Specifically, at each time step t, after executing an action $a_t \in \mathcal{A}$ in the current state $s_t \in \mathcal{S}$, the environment transitions to a state s_{t+1} with probability $P(s_{t+1} \mid s_t, a_t)$. Then, the UAV obtains an observation $o_t \in \Omega$ based on s_{t+1} with probability $P(o_t \mid s_{t+1}, a_t)$.

In general, the OA problem can be transformed into an optimal policy-searching problem, which can be represented as $\pi^* : \Omega \rightarrow \mathcal{A}$. The optimal policy endows the UAV capability to adopt an action at each time step t for yielding the maximum expected value $\mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}\left(s_t, a_t\right)\right]$ of the cumulative reward value for all moments, where $\gamma \in (0, 1)$ is the discount factor.

Defining the sets $\mathcal{S}$, $\mathcal{A}$, Ω and the functions $\mathcal{R}$, $\mathcal{P}$, $\mathcal{O}$ is a prerequisite for seeking the optimal policy. Functions $\mathcal{P}$ and $\mathcal{O}$ must be established depending on environmental factors, such as wind, turbulence, and the UAV's dynamics, both of which are challenging to acquire in real-world scenarios. Since this chapter emphasizes the OA task for a single agent and UAVs must have the capability to navigate in unstructured and unknown environments. It mainly works on designing the state space $\mathcal{S}$, action space $\mathcal{A}$, and reward function $\mathcal{R}$ during the problem modeling process. For the single UAV, the state space consists of two main parts: depth images s_i generated by the monocular camera combined with the depth estimation model and the target point coordinates s_t. In order to obtain more accurate and richer observation information, four consecutive depth images are stacked as partial state space information. In other words, s_i contains four adjacent frames of t, $t-1$, $t-2$, $t-3$, and s_t is a two-dimensional coordinate.

C. Basic Q-Learning Algorithm

The approach employed in this chapter is derived from the Q-Learning algorithm [95], which replaces the Q-value table with a deep network. The Q-Learning algorithm proposed by Watkins is a value-based and model-free RL technique that leverages transient strategies. The iterative process of the algorithm employs the state-action value function as the estimation function. Value iteration updates all Q values in each iteration. However, in practical scenarios, it can only operate on a limited number of samples, as traversing all states and actions is challenging. The Q-Learning algorithm proposes an approach for updating the Q-value, which can be defined as:

$$Q_t\left(S_t, a_t\right) = Q_{t-1}\left(S_t, a_t\right) + \alpha\left[r\left(S_t, a_t\right) + \gamma \max_a \left(Q_{t-1}\left(S_{t+1}, a\right)\right) - Q_{t-1}\left(S_t, a_t\right)\right], \tag{4.1}$$

where α is the learning factor, $r\left(S_t, a_t\right)$ denotes the reward at moment t, γ represents the discount factor.

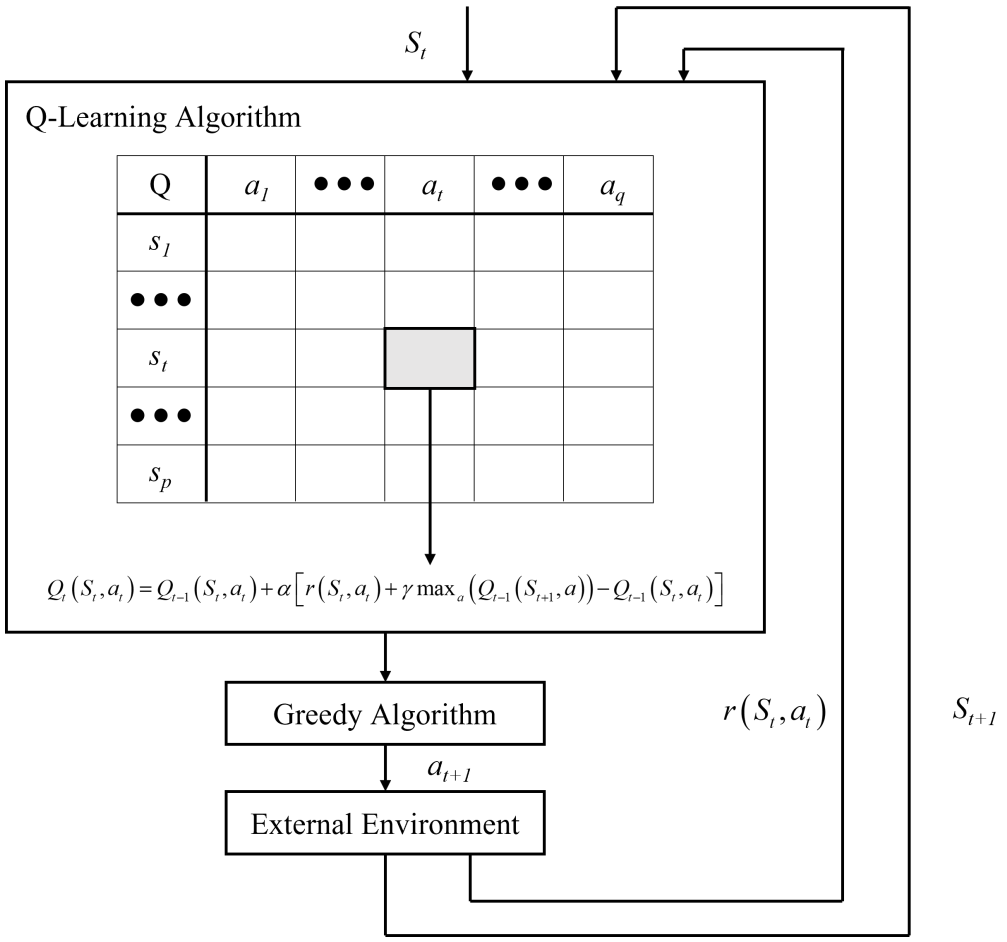


FIGURE 4.4: Framework of Q-Learning Algorithm.

It can be concluded from Eq. 4.1 that the value assigned to an element in the Q matrix can be calculated by adding the value of the corresponding element in the reward matrix R to the maximum reward value of all possibly executed actions in the next state, multiplied by the discount factor γ. Furthermore, When a specific state is given, a ε greedy strategy is leveraged to select the action, and the action selection strategy is denoted as:

$$a_{t+1} = \begin{cases} \arg\max_{a'} Q\left(S_{t+1}, a'\right) & \text{with a probability of } 1-\varepsilon \\ \text{Random Action} & \text{with a probability of } \varepsilon \end{cases}, \tag{4.2}$$

where ε is the random probability factor for action selection.

The framework of the Q-Learning algorithm is depicted in Fig. 4.4. It indicates that it is necessary to design the perceptual state s, the action executed a, and the received reward r, which are leveraged continuously to update the tabular

Algorithm 3 Basic Q-Learning Algorithm

Input:

α : learning factor
γ : discount factor
r: reward function
S_t: state space
a_t: action space
N: number of iterations
δ: convergence threshold

Output:

$Q(S, a)$: Convergent action-value estimation function

Initialize $Q(S, a)$ to 0, let moment t be 0, set α and γ
for n **in** range(N) **do**
    Observe S_t and select an a_t at moment t according to the ϵ greedy strategy
    Execute a_t, calculate $r(S_t, a_t)$, and observe S_{t+1} at $t+1$ moment
    Select $\arg\max_{a_{t+1}} Q\left(S_{t+1}, a_{t+1}\right)$ based on S_{t+1} and $Q(S, a)$ at t moment
    Update $Q^*\left(S_t, a_t\right)$ (Eq. 4.1) and calculate error $e = \left|Q^*\left(S_t, a_t\right) - Q\left(S_t, a_t\right)\right|$
    $S_t = S_{t+1}$
    if $e < \delta$ **then**
        return $Q(S, a)$
    else
        continue
    end if
end for
return $Q(S, a)$

Q-function and eventually reach a convergent policy. In summary, based on the description of the basic Q-learning algorithm, its algorithmic flow is shown in Alg. 3.

D. Deep Q-Learning Algorithm

The basic Q-Learning algorithm can obtain the Q-values of actions by referencing the table when receiving the current state as input. This algorithm can only satisfy the case where the state space dimension is small. As the dimensionality of the state and action spaces increases (e.g., image inputs are high-dimensional representation information), the utilization of tables cannot record Q-values for all states, and dimensional disaster is prone to occur, thus reducing the accuracy and safety of control. The Deep Q-Learning (DQN) algorithm [96] is a variation of the traditional Q-Learning algorithm, which differs in the following three aspects. Firstly, it employs a deep CNN to estimate the action-value function. Secondly, it

leverages the experience replay mechanism to train the model. Finally, it independently sets up a target network to handle the temporal difference (TD) error. Fig. 4.5 below depicts the flow of the DQN algorithm.

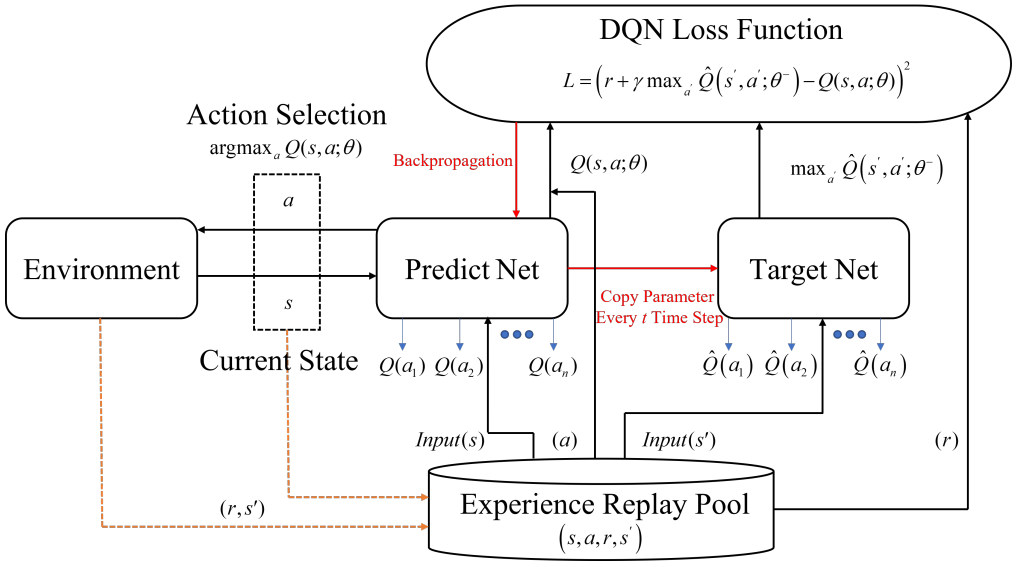


FIGURE 4.5: Flowchart of Deep Q-Learning Algorithm.

A collection of basis functions and their accompanying parameters are multiplied to estimate the action-value function leveraging linear approximation. The DQN network, in contrast, deploys a nonlinear approximation strategy and a neural network to approximate the action-value function. The action-value function is written as $Q(s,a;\theta)$, and the parameters corresponding to it are the network weights, indicated by θ in this chapter.

The experience replay mechanism of DQN is akin to human memory, where optimal strategies are learned by summarizing experiences from memories. It is accomplished by storing samples obtained from each agent's interaction with the environment in an experience pool, from which a random set of samples is chosen for training. The motivation for introducing the experience replay mechanism is that DL requires samples to be independent of each other, while the before and after states of agents in RL are correlated. The strong correlation between training samples may result in failures. By randomly selecting samples, the experience replay mechanism can effectively break the correlation between data, resulting in more stable training.

Algorithm 4 Deep Q-Learning Algorithm

Input:

$f(s;\theta)$: deep CNN
Mem: buffer memory
M: iteration episodes
γ discount factor
θ: network parameters of the predict net
θ^{-}: network parameters of the target net

Output:

θ_i network parameters

Preset buffer memory Mem to capacity N
Preset $f(s;\theta)$ with randomly initialized weights
for episode = 1 to M **do**
    Initialize the perceptual state s_1, calculate the Q-value $f(s_1;\theta)$
    for $t = 1$ to T **do**
        if *Random number* $< \epsilon$ **then**
            Perform a random action a_t
        else
            Perform action $a_t = \text{argmax}_a\, f(s_t, a;\theta)$
        end if
        Perform action a_t, yield feedback reward r_t and acquire s_{t+1}
        Add transition (s_t, a_t, r_t, s_{t+1}) to Mem as a sequence
        Randomly draw n sequences (s_j, a_j, r_j, s_{j+1}) from Mem
        Calculate $y_i = \begin{cases} r_j & \text{for terminal } s_{j+1} \\ r_j + \gamma \max_{a'} f(s_{j+1}, a';\theta) & \text{for non-terminal } s_{j+1} \end{cases}$
        Execute a gradient descent on $(y_i - f(s_j, a_j;\theta))^2$ with the set batch size
        Update θ
        Update θ^{-} every t step
    end for
end for

When the action-value function is approximated leveraging a neural network, the parameter θ is updated at each iteration through the gradient descent method:

$$\theta_{t+1} = \theta_t + \alpha \left[r + \gamma \max_a Q(s', a', \theta) - Q(s, a, \theta) \right] \nabla Q(s, a, \theta), \tag{4.3}$$

where $r + \gamma \max_a Q(s', a', \theta)$ refers to the TD target. When calculating the action-value function of the TD target, the parameters leveraged are identical to those leveraged to calculate the approximation function. There is instability in training due to the correlation between data. To solve this issue, the DQN algorithm defines the parameters for the TD target as θ^{-}, which are updated after a fixed number of steps. On the other hand, the parameters for the approximated function are

denoted as θ and updated after each iteration. Therefore, the update process of the action-value function can be expressed as:

$$\theta_{t+1} = \theta_t + \alpha \left[r + \gamma \max_a Q\left(s', a', \theta^-\right) - Q(s, a, \theta) \right] \nabla Q(s, a, \theta). \tag{4.4}$$

The neural network in the DQN algorithm is trained deploying target Q values as labeled data. The loss function measures the discrepancy between predicted outputs and labeled samples, and the aim of the training is to optimize this loss value. The loss function of DQN is formulated as:

$$L(w) = E\left[\left(r + \gamma \max Q\left(s', a', \theta^-\right) - Q(s, a, \theta) \right)^2 \right]. \tag{4.5}$$

Alg. 4 illustrates the DQN algorithm, which primarily establishes the mapping between the UAVs' continuous perception state and discrete decision actions. The algorithm takes the image captured by the UAV as input, utilizes deep CNN to reduce its dimensionality, and fits the action-value function to obtain the optimal Q-policy value for selecting the optimal evasive actions.

E. Double Deep Q-Learning Algorithm

The DQN method employs the Q-Learning framework, however, it still has bootstrapping, which is a flaw in the Q-Learning algorithm itself. In bootstrapping, the projected value function is greater than the actual value function. This problem is mostly caused by the maximizing operation used in the Q-Learning technique to update the value function. If the same magnitude consistently overestimates the value function at every point, the ideal strategy stays the same. But, in practice, the bootstrapping is not consistent, which results in an inferior approach rather than an excellent one. The ultimate goal of RL is to find the best possible policy, therefore even an exaggerated value function is not supposed to have an impact on the outcome.

The Double DQN (DDQN) [97] algorithm aims to address the bootstrapping issue in the DQN algorithm, as illustrated in the flowchart in Fig .5 and detailed in Alg .5. The key step in DDQN is the split of action selection and evaluation for target Q-value calculation, which is performed using separate Q-value networks. In the

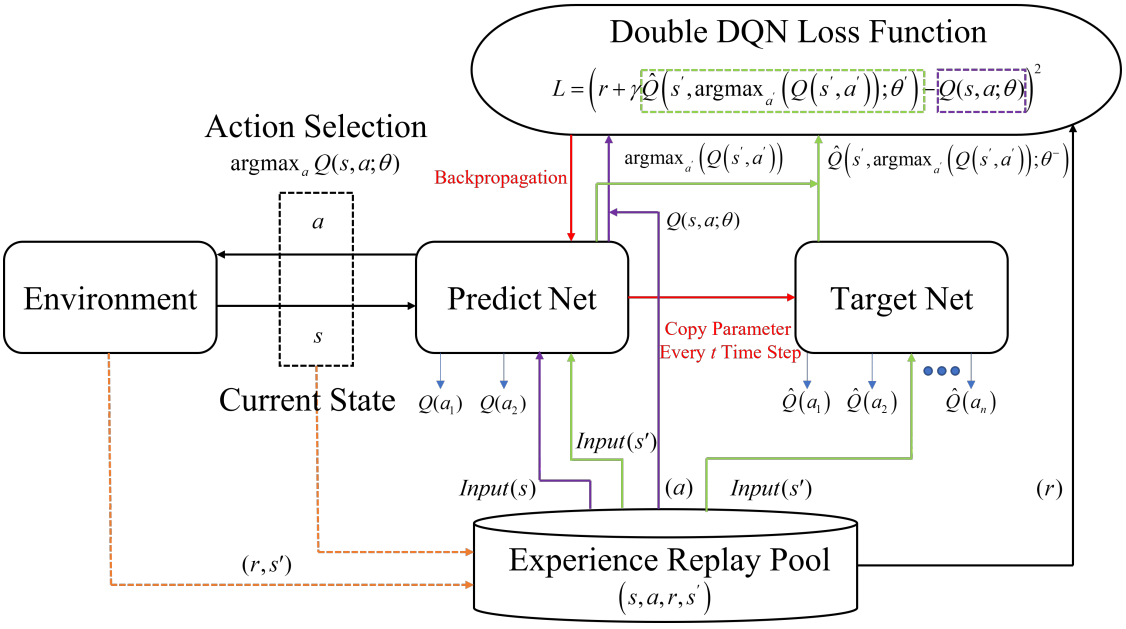


FIGURE 4.6: Flowchart of Double Depp Q-Learning Algorithm.

DQN algorithm, the target Q-value is calculated as shown below:

$$r + \gamma \max_{a} Q\left(s', a'; \theta^{-}\right). \tag{4.6}$$

The approach taken by the DDQN algorithm to determine the maximum Q-value for each action in the target Q-network is dissimilar. Firstly, it identifies the action that corresponds to the maximum Q-value in the current Q-network using the following method:

$$a_{\max} = \operatorname{argmax}_{a} Q\left(s', a'; \theta\right). \tag{4.7}$$

After selecting this action, it is leveraged in the target network to calculate the target Q-value, which can be expressed as:

$$r + \gamma Q\left(s', \operatorname{argmax}_{a} Q\left(s', a'; \theta\right); \theta^{-}\right). \tag{4.8}$$

Algorithm 5 Double Deep Q-Learning Algorithm

Input:

$f(s;\theta)$: deep CNN
θ: network parameters of the predict net
θ^{-}: network parameters of the target net
Mem: buffer memory
M: iteration episodes
γ discount factor

Output:

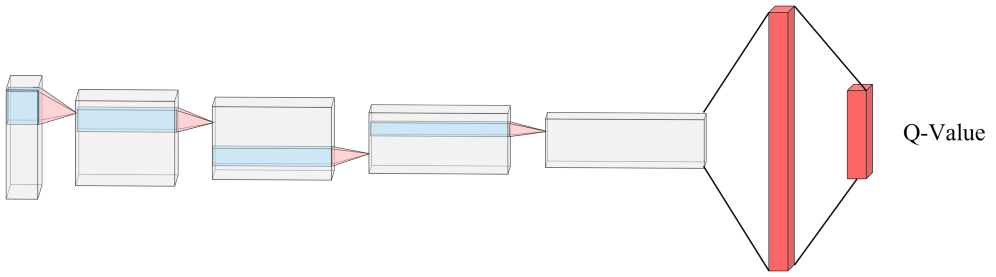
θ_i network parameters

Preset buffer memory Mem to capacity N
Preset $f(s;\theta)$ with randomly initialized weights
for episode = 1 to M **do**
    Initialize the perceptual state s_1, calculate the Q-value $f(s_1;\theta)$
    for $t = 1$ to T **do**
        if *Random number* $< \epsilon$ **then**
            Perform a random action a_t
        else
            Perform action $a_t = \text{argmax}_a f(s_t, a;\theta)$
        end if
        Perform action a_t, yield feedback reward r_t and acquire s_{t+1}
        Add transition (s_t, a_t, r_t, s_{t+1}) to Mem as a sequence
        Randomly draw n sequences (s_j, a_j, r_j, s_{j+1}) from Mem
        Calculate $y_j = \begin{cases} r_j & \text{for terminal } s_{j+1} \\ r_j + \gamma \max_a f\left(s_{j+1}, \text{argmax}_a f\left(s_{j+1}, a';\theta\right);\theta^{-}\right) & \text{else} \end{cases}$
        Execute a gradient descent on $\left(y_j - f\left(s_j, a_j;\theta\right)\right)^2$ with the set batch size
        Update θ
        Update θ^{-} every t step
    end for
end for

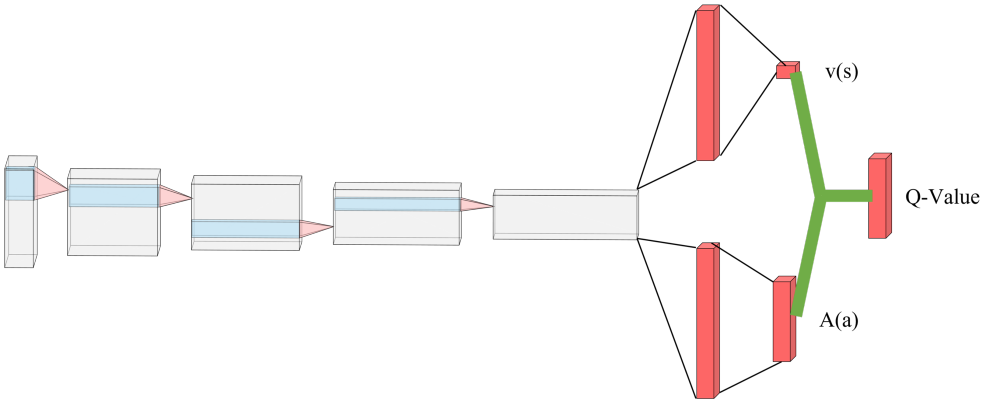
F. Dueling Deep Q-Learning Algorithm

For many vision-based DRL tasks, the value functions of various state-action pairs are distinct. However, for some states, the magnitude of the value function is independent of the action taken. Based on this observation, Ref. [98] proposed a dueling network structure as a DQN model, as illustrated in Fig. 4.7. The first network model is the classical DQN structure, while the second model is a dueling network structure that bifurcates the abstract features extracted from the convolutional layer into two branches. The state-value function, which expresses the value of the environment's static state, is represented by the top path. The state-dependent action-advantage function, on the other hand, is represented by

the lower route and indicates the added value delivered by choosing a certain action. The Q-value of each action is then calculated by reintegrating these two routes.



(a) Schematic diagram of DQN algorithm.



(b) Schematic diagram of Dueling DQN algorithm.

FIGURE 4.7: Comparison of DQN and Dueling DQN algorithms.

The Dueling DQN consists of two fully connected layers, one for estimating the state-value function and the other for estimating the action-advantage function. The state value function is denoted as $v(s;\theta,\beta)$, the action dominance function is denoted as $A(s,a;\theta,\alpha)$, and the Q value is the sum of the state value and action advantage:

$$Q(s,a) = v(s;\theta,\beta) + A(s,a;\theta,\alpha), \tag{4.9}$$

where θ, β, and α correspond to the parameters of the convolutional layer and two-branch fully connected layers, respectively. In practice, the action advantage function is generally set to the difference between the individual action advantage function and the average of all action advantage functions in a state. It helps maintain the relative ordering of the advantage functions for each action in that state, narrows the range of Q-values, reduces the redundancy of the degrees of freedom, and improves the algorithm's stability. The equation can be depicted as

Algorithm 6 Dueling Deep Q-Learning Algorithm

Input:

$f(s;\theta)$: deep CNN
θ: network parameters of the predict net
θ^-: network parameters of the target net
Mem: buffer memory
M: iteration episodes
γ discount factor

Output:

θ_i network parameters

Preset buffer memory Mem to capacity N
Preset $f(s;\theta)$ with randomly initialized weights
for episode = 1 to M **do**
    Initialize the perceptual state s_1, calculate the Q-value $f(s_1;\theta)$ via **dueling network**
    for $t = 1$ to T **do**
        if *Random number* $< \epsilon$ **then**
            Perform a random action a_t
        else
            Perform action $a_t = \text{argmax}_a f(s_t, a; \theta)$
        end if
        Perform action a_t, yield feedback reward r_t and acquire s_{t+1}
        Add transition (s_t, a_t, r_t, s_{t+1}) to Mem as a sequence
        Randomly draw n sequences (s_j, a_j, r_j, s_{j+1}) from Mem
        Set $y_j = \begin{cases} r_j & \text{for terminal } s_{j+1} \\ r_j + \gamma \max_a f\left(s_{j+1}, \text{argmax}_a f\left(s_{j+1}, a'; \theta\right); \theta^-\right) & \text{else} \end{cases}$
        Execute a gradient descent on $\left(y_j - f\left(s_j, a_j; \theta\right)\right)^2$ with the set batch size
        Update θ
        Update θ^- every t step
    end for
end for

follows:

$$Q(s,a) = v(s;\theta,\beta) + \left(A(s,a;\theta,\alpha) - \frac{1}{|A|} \sum_{a'} A\left(s,a';\theta,\alpha\right) \right). \tag{4.10}$$

F. End-To-End Obstacle Avoidance Modeling Based on Depth Images

The problem to be solved in this chapter can be modeled as a POMDP to achieve autonomous OA based on depth images through DRL. The approach is considered end-to-end as it does not require intermediate steps, such as building a map. It can output direct command-level actions by taking depth images and target point

information as inputs. The algorithm can update and optimize its strategy based on the feedback obtained after executing the corresponding action and getting new images from the environment, until it learns the optimal strategy to reach the destination. Details of the modeling are shown below.

i. State Space Definition

The issue of autonomous OA in an unexplored environment can be depicted as:

$$v_t = f\left(x_t, m_t\right). \tag{4.11}$$

The mapping function f that the DRL network seeks takes the following inputs:

(1) Image information x_t: Perceiving the surrounding environment for memorization and OA. This chapter leverages a monocular RGB camera installed on the UAV in the virtual simulation environment as the onboard sensor to estimate depth from captured images. Specifically, the state inputs include a stack of four continuous frames from the camera, which consist of images captured after four consecutive performed evasive maneuvers.

(2) Target Information m_t: Specifying the destination point and directing UAV's motions.

Obtaining the location information of both the UAV and the destination is crucial for autonomous navigation, alongside sensing obstacle proximity. In Fig. 4.8, an array $[v_t, \omega_t, d_t, \theta_t]$ shows the relative position information between the UAV and the destination. The UAV's linear and angular velocity at the moment t are represented by v_t and ω_t, respectively, while d_t denotes the distance of the UAV relative to the desired goal at t moment. The value of θ refers to the deviation between the UAV's yaw angle and the angle formed by the line connecting the UAV and the desired goal concerning the x-axis coordinate system. The UAV's relative yaw angle θ and distance d_t to the desired target determine the goal information. This information enables the UAVs to navigate toward the target direction once they are trained.

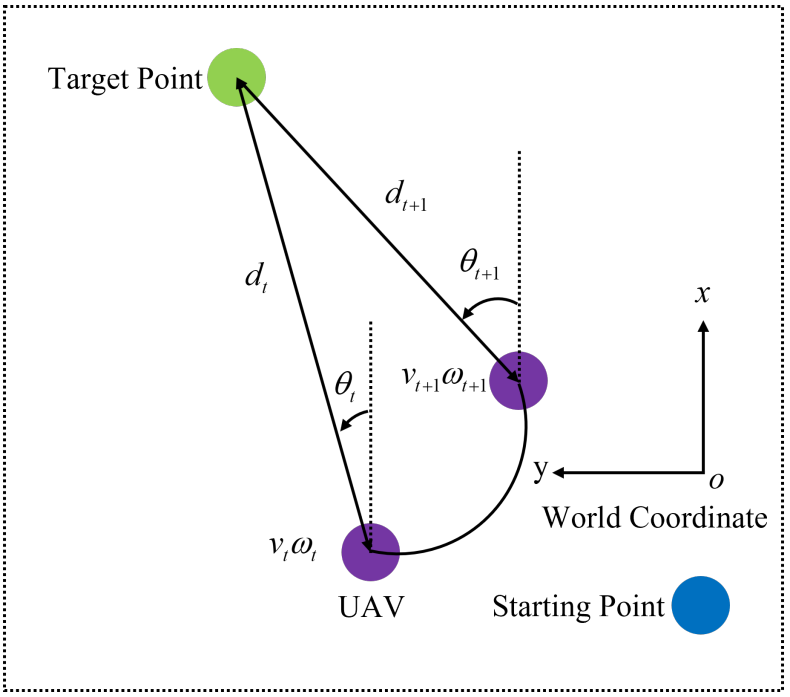


FIGURE 4.8: Information about the UAV's position relative to the destination.

ii. Action Space Definition

The action space in DQN and its extended algorithms consists of discrete sequences of actions, as illustrated in Table 4.1. This chapter sets a constant flight altitude and linear velocity of 0.5 m/s for the UAV. The algorithm's output governs the UAV's angular velocity or the yaw angle to control the UAV's flight direction.

TABLE 4.1: Action Space Definition

Action Number	Linear Velocity	Angular Velocity
0	0.5 m/s	$-\pi/4$ rad/s
1	0.5 m/s	$-\pi/8$ rad/s
2	0.5 m/s	0 rad/s
3	0.5 m/s	$\pi/8$ rad/s
4	0.5 m/s	$\pi/4$ rad/s

iii. Design of Reward Function

The reward function for the OA problem in this chapter is outlined in Table 4.2. If the UAV comes within 0.5 m of an obstacle, it is recognized as colliding with it and receives a reward of -100. Similarly, It also receives a reward of -100 if it flies out of the designated area. However, reaching the destination yields a reward of 1000. Additionally, executing an action results in a reward of $10 \times (d_{t-1} - d_t)$. A positive value is assigned if the action results in decreasing the distance between the UAV and the desired goal. A penalty of -0.2 is imposed on each action performed to incentivize the UAV to reach the destination through the shortest possible trajectory while avoiding obstacles.

TABLE 4.2: Reward Function

Scenario	Reward
Within 0.5 m of an obstacle	-100
For each action performed	$10 \times (d_{t-1} - d_t) - 0.2$
Within 1 m of the target	1000
Out of bounds	-100

iv. Network Architecture

Low-dimensional target and movement information are often obscured by high-dimensional images. Therefore, it is necessary to pre-process high-dimensional images. In this chapter, image features are extracted leveraging CNNs first, and then the low-dimensional image features are fed into the decision layer together with other information. Initially, the inputs are four stacked consecutive depth images with the dimension of 160×120. Subsequently, four convolutional layers are leveraged to extract image features. The first convolutional layer's kernel size, stride, and output channels are 8×6, 8×8, and 32, respectively. Thereby the first layer's output dimension is $20 \times 15 \times 32$. Similarly, the output dimension of the following layers are $7 \times 5 \times 64$, $4 \times 3 \times 64$, and $2 \times 2 \times 64$, respectively. Eventually, image features and information about the desired destination are integrated as the input to the network. The overall framework of the end-to-end autonomous OA strategy is depicted in Fig. 4.9, and the architecture of the decision-making module is presented in Fig. 4.10.

The initial four layers of the above-described end-to-end network are in charge of receiving observable visual elements from the incoming data and extracting

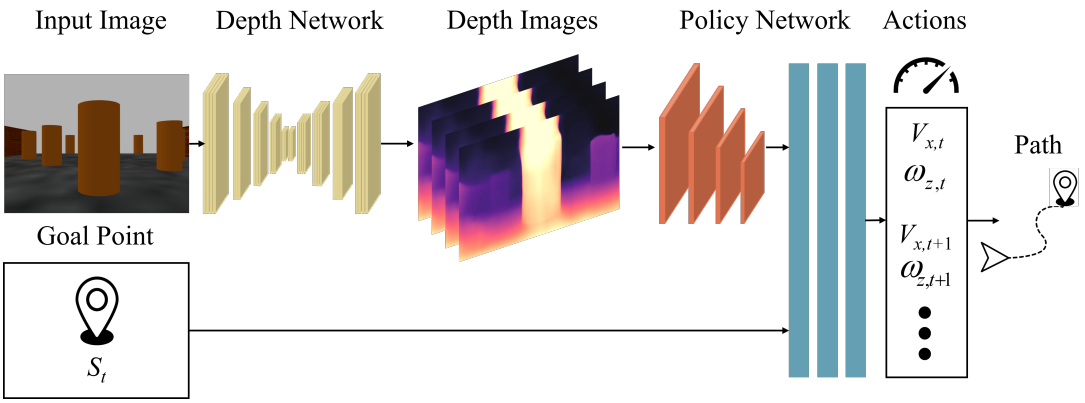


FIGURE 4.9: Information about the UAV's position relative to the target point.

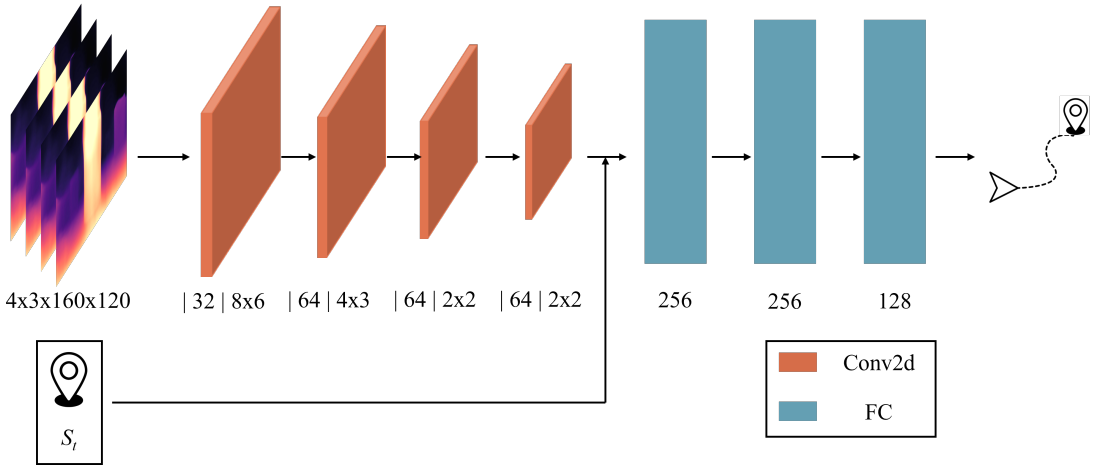


FIGURE 4.10: Information about the UAV's position relative to the target point.

features. During training, these layers' parameters are updated along with the two fully connected layers, considerably increasing the experience required to carry out OA tasks in the environment.

Table 4.3 depicts the algorithm's parameter configuration. The maximum number of training iterations is 15000, with a limit of 80 steps per episode, indicating that the current episode will terminate when the step count surpasses this limit, with no further penalty incurred. The learning rate of the network is 0.0001, with a Q-value discount factor γ of 0.99, and the greedy strategy termination parameter is 0.1. The ε parameter in the ε greedy method reduces linearly from an initial value of 1.0 based on the number of training steps, with a training step $timestep = 10000$.

$$\varepsilon = \varepsilon - \frac{1.0}{timestep}. \tag{4.12}$$

TABLE 4.3: Network Parameter Configuration.

Paramater	Value
Discount Factor	0.99
Learning Rate	0.0001
Greedy Policy Termination Parameter	0.1
Maximum Training Episode	15000
Experience Pool Size	60000
Maximum Training Step	80
Batch Size	64
Number of Stacked Frames	4

G. Simulation Environment

FIGURE 4.11: Quadrotor leveraged for simulation experiment.

This chapter builds simulation scenarios for training and validation based on Gazebo/Ros. Gazebo is an open-source 3D simulation software widely deployed in robotics research and education. It permits users to build and simulate complex scenarios involving robots, their environment, sensors, actuators, and physics-based models. Gazebo offers a high-fidelity simulation environment that enables engineers and researchers to test and validate their algorithms and systems in a secure, cost-effective, and repeatable manner. It supports various sensors and platforms,

including UAVs, ground robots, and manipulators, and can interface with various programming languages and tools like Robot Operating System (ROS).

The UAV in this chapter is set to fly at a fixed altitude, which makes the simulation scenarios two-dimensional. In this section, several scenarios are designed for model training and validation. Scenarios deployed for training are depicted in Fig. 4.13, which consists of walls and cylindrical obstacles. The scene's dimensions are $15m \times 30m$. The objective of the UAV is to start from one end of the scene and reach the other end. If the UAV successfully reaches the end or collides along the way, it is reset to the starting point to reperform the task. The small-scale scenario is relatively narrow and crowded, making it challenging for the UAV to perform the OA task.

Gazebo is a powerful simulation environment for drones, which relies on different ROS nodes to simulate the movement patterns of the UAV. These nodes can be assigned to various components, such as the onboard camera and the flight control module. During the simulated flight, these nodes generate corresponding ROS topics and can publish and subscribe data to each other. For instance, the camera node may publish received observations of the surroundings. The flight control module may subscribe to the data published by the camera to adjust its current velocity and orientation accordingly. By exchanging data in this way, the different nodes cooperate together to complete the simulated flight in the virtual environment.

To illustrate the operation of Gazebo, Fig. 4.12 shows the ROS nodes and ROS topics included in the virtual environment created in this chapter. These nodes and topics enable the UAV to interact with the simulated environment, rendering it a valuable tool for developing and validating UAV applications. Here, $/gazebo$ is mainly responsible for providing kinematic and dynamic parameters of various objects, $/command/motor$ is in charge of publishing commands to control the motor operation, $/motor/status$ handles feeding the current engine parameters to Gazebo, $/front\_cam/camera/image$ publishes images captured by the camera, and $/cmd\_vel$ facilitates to adjust the UAV's movement status. These topics and nodes work in coordination to regulate the motion of the UAV in a close-loop manner.

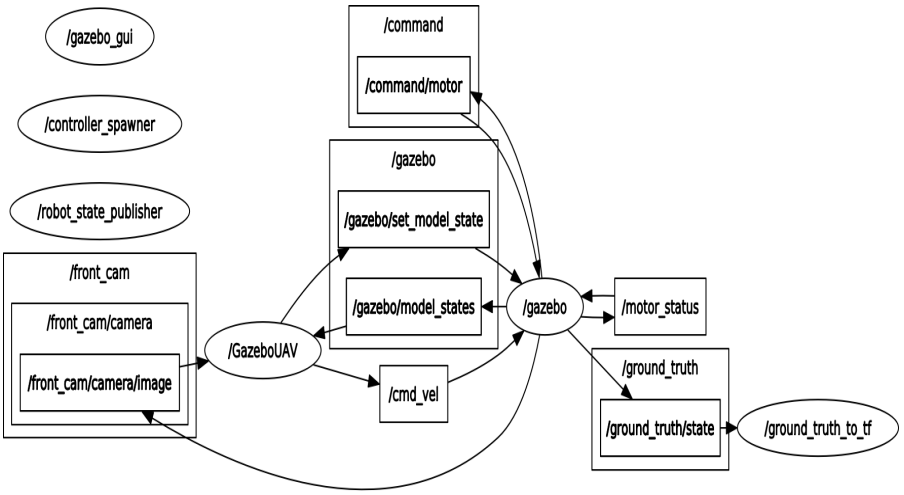


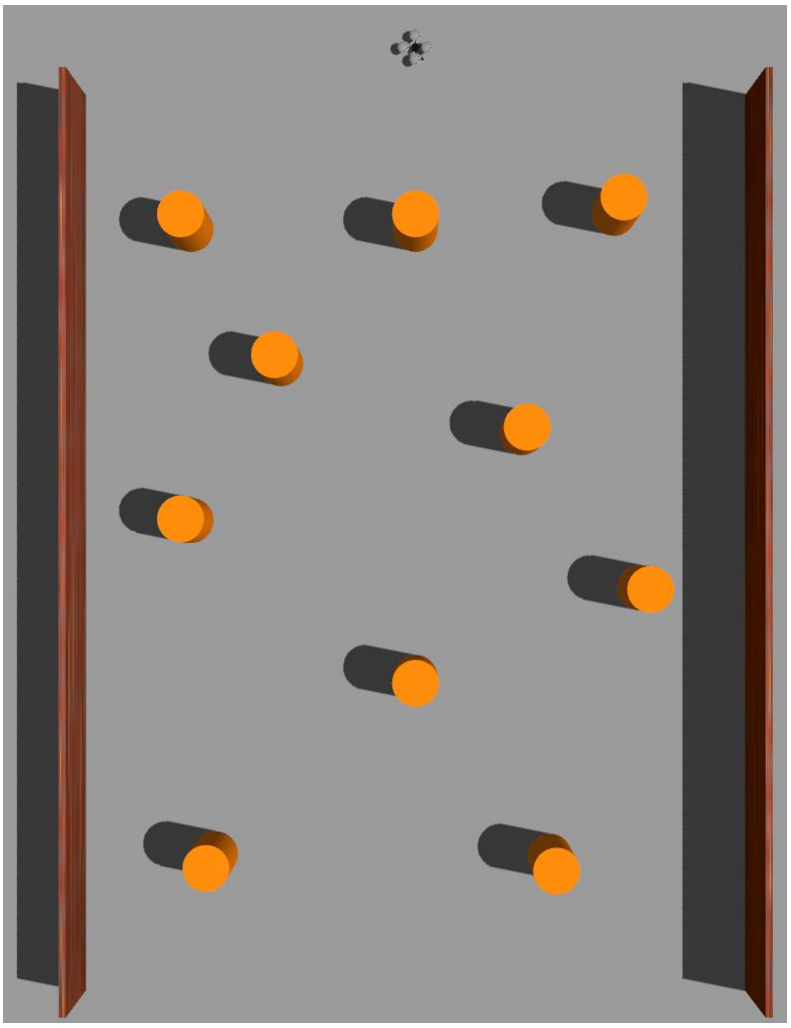
FIGURE 4.12: Relationship of ROS nodes.

4.2 Simulation Results

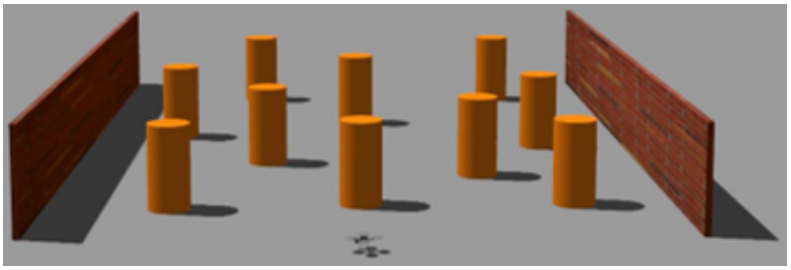
In this chapter, a depth estimation model is utilized to anticipate the depth of RGB images captured by monocular cameras in order to achieve environment perception. The simulated scenario includes a wall, several cylindrical columns, and a low-contrast background, and the outcomes of the depth prediction are illustrated in Fig. 4.14.

It can be concluded that the model is capable of accurately perceiving the proximity of obstacles when they are nearby, as shown in (a) and (b), but its performance may be compromised in certain situations, such as (c). As the UAV's sensing range is limited, the model struggles to generate precise depth maps and detect obstacles when the obstacles are far away and the background lacks texture and contrast. Since distant obstacles have little impact on the UAV's safe flight, this limitation of the model can be tolerated, and the depth maps it produces are suitable for OA tasks in relatively sparsely large-scale regions (compared with small-scale populated scenarios such as urban streets).

Once the depth map is obtained, it is possible to combine it with the UAV location information to train the neural network and generate a hazard-free flight trajectory. During training, the cumulative reward obtained in each episode is an important metric for evaluating the model's performance. To compare the performance of the three algorithms, this chapter records the cumulative reward during the training



(a) Top view of the scenario.



(b) Front view of the scenario.

FIGURE 4.13: Simulation scenario built in Gazebo.

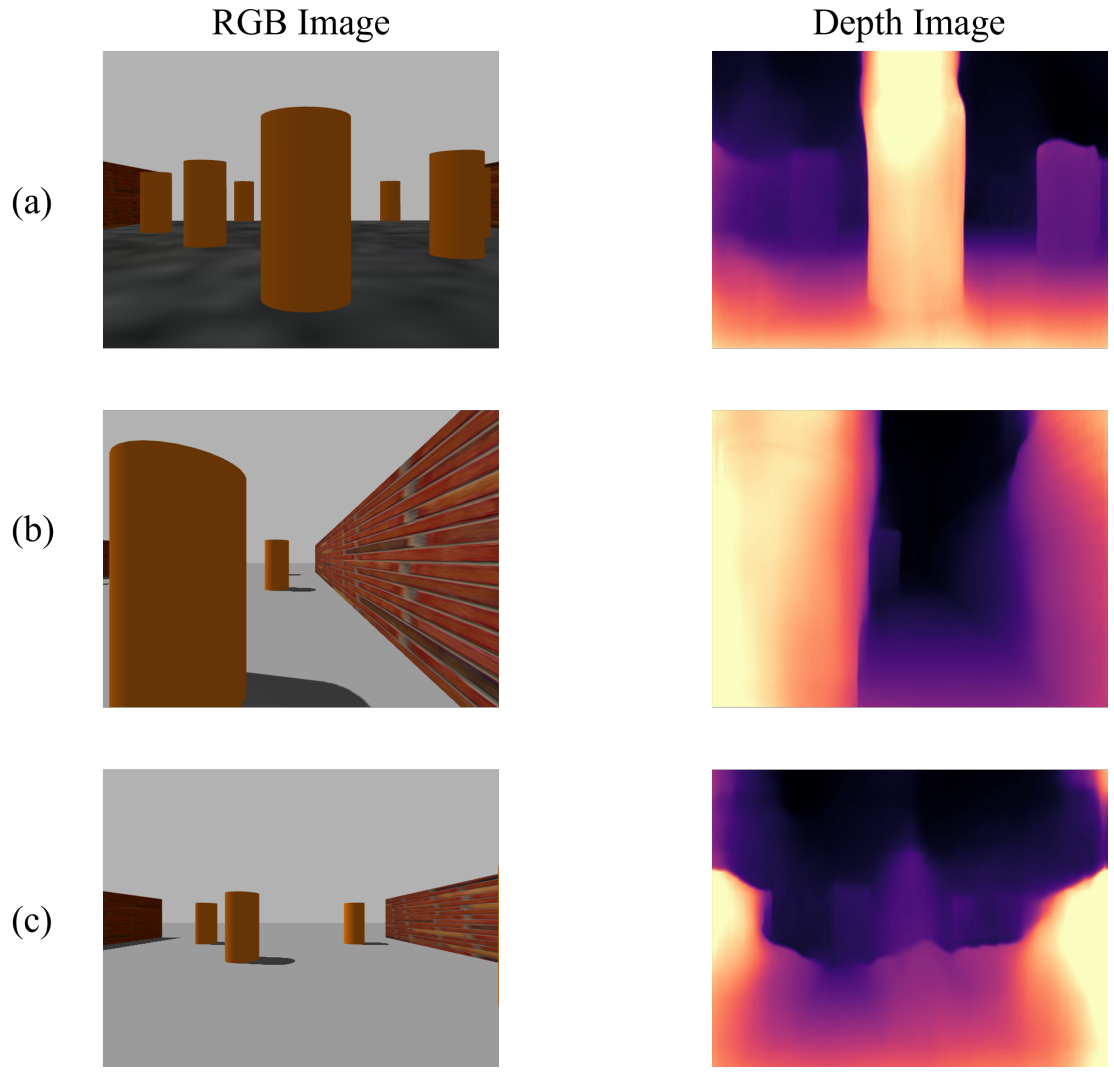


FIGURE 4.14: Results of depth estimation based on the simulated environment.

of each of the three algorithms and summarizes their results in one figure for comparative analysis, as shown in the following figures. Figures. 4.15, 4.16, and 4.17 show the cumulative rewards after 15000 training episodes for DQN, Double DQN, and Dueling DQN algorithms, where the solid lines represent the mean rewards values within a specific window, and the shaded areas illustrate the distribution of rewards around these mean values.

The convergence behavior of three algorithms, namely DQN, Double DQN, and Dueling DQN, has been investigated by analyzing their reward curves. Generally, all three algorithms exhibit an upward trend in the reward curves. Specifically, the

DQN algorithm requires approximately 3000 training episodes to converge gradually and stabilize after approximately 8000 episodes. Similarly, the Double DQN algorithm displays gradual convergence after about 1000 episodes, and stabilization occurs after about 3000 episodes. In contrast, the Dueling DQN algorithm takes roughly 2000 episodes to converge gradually and stabilizes after about 3000 episodes. A comparison of these three algorithms is presented in Fig. 4.18.

Upon comparing these algorithms, it can be observed that the DQN algorithm exhibits the slowest convergence speed. The average reward within a specific window stabilizes between 700 and 800, basically the same as the Double DQN algorithm, lower than the Dueling DQN algorithm. Although the Double DQN algorithm's average reward also stabilizes between 700 and 800, it converges fastest among the three algorithms, which is attributed to the solved bootstrapping issue. In comparison, the Dueling DQN algorithm converges almost as promptly as Double DQN, with a negligible difference in convergence speed. However, it performs the best among the three algorithms, with the average reward finally stabilizing at around 800, nearly 8 percent higher than the other two algorithms.

Overall, these results indicate that the Dueling DQN algorithm comprehensively

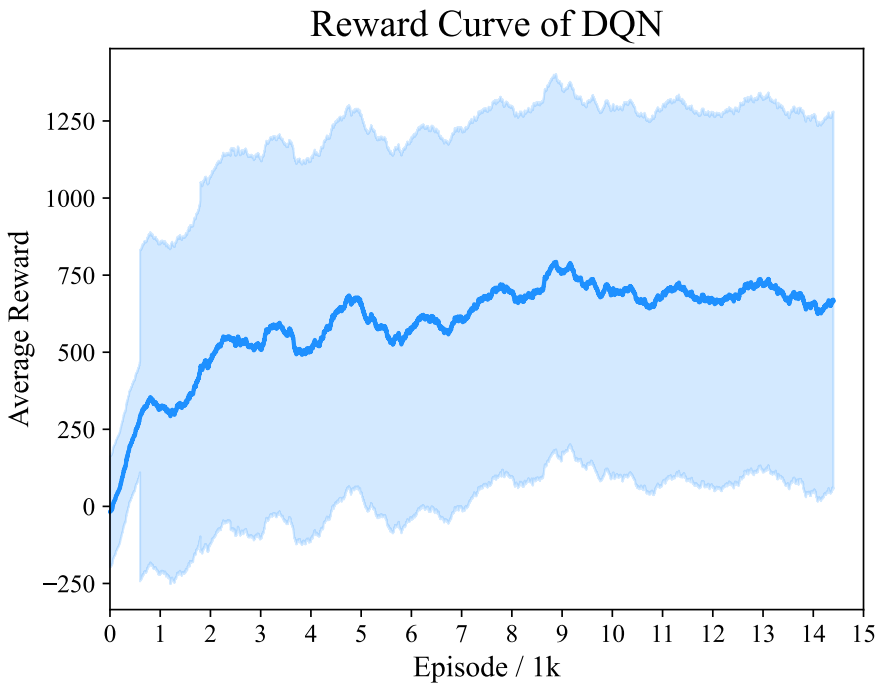


FIGURE 4.15: Reward curve of DQN algorithm.

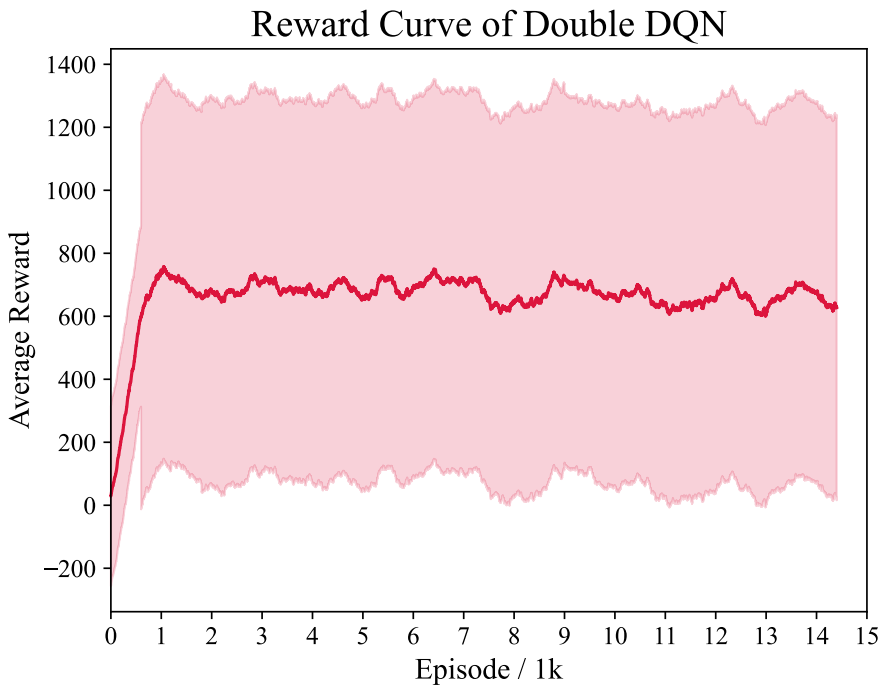


FIGURE 4.16: Reward curve of Double DQN algorithm.

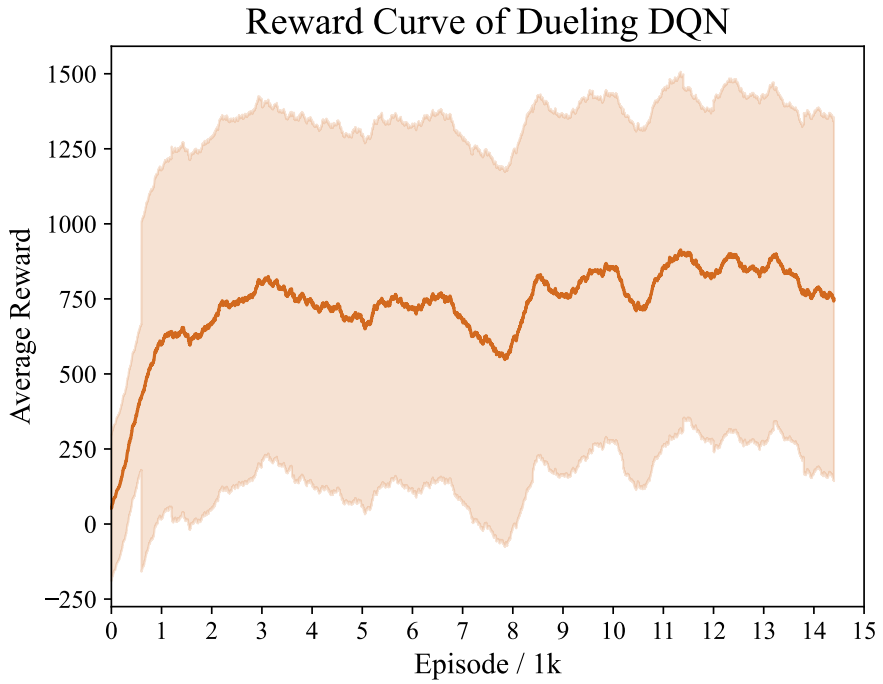


FIGURE 4.17: Reward curve of Dueling DQN algorithm.

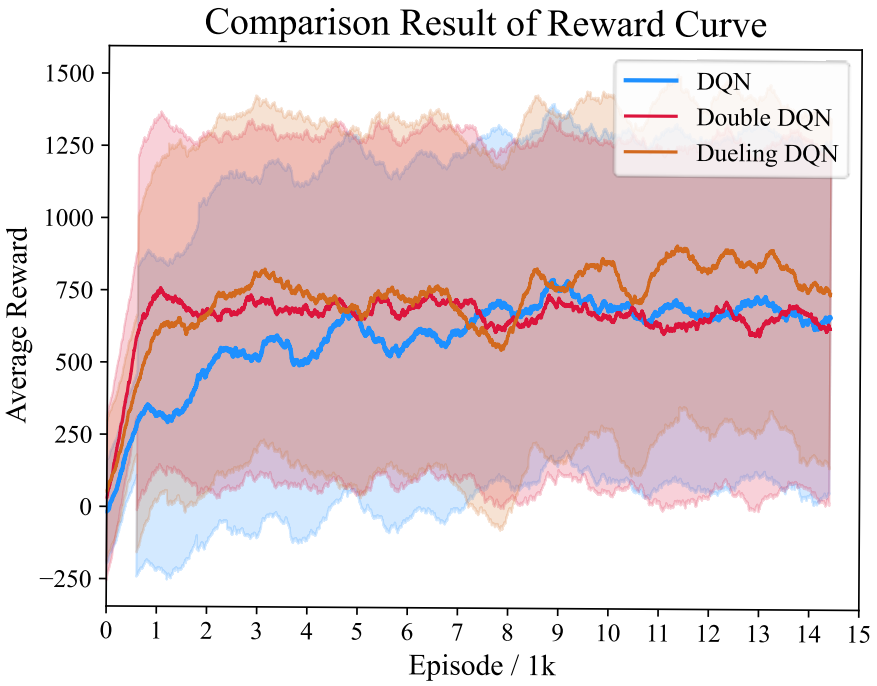


FIGURE 4.18: Reward curve comparison of DQN, Double DQN, and Dueling DQN.

outperforms the other two algorithms, considering the average reward and convergence speed. The Double DQN algorithm also performs reasonably well and converges faster than DQN.

The reward curve serves as a crucial indicator during the training process. Once the model is trained, its capability to generate collision-free trajectories is assessed through multiple scenarios in the validation process. The success rate of such flights is recorded, and the flight trajectory is plotted to compare the performance differences between different models. This chapter presents the validation results for three alternate simulated environments: a scenario with random obstacle initial point locations, a crowded scenario with a regular and neat arrangement, and a horizontal scenario with a more consistent and concentrated arrangement. Flight trajectories corresponding to these three scenarios are depicted in Figs. 4.19, 4.20, and 4.21, respectively.

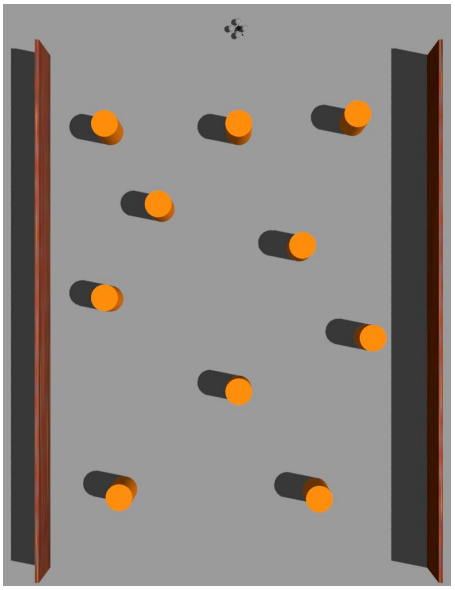
This chapter conducts 100 validation flights for each of the three scenarios, recording a flight as successful when it reaches the target point without collision. The

success rate of OA, action command generation frequency, and the average number of steps required to complete the flight are recorded as performance indicators and presented in Table 4.4, Table 4.5, and Table 4.6.

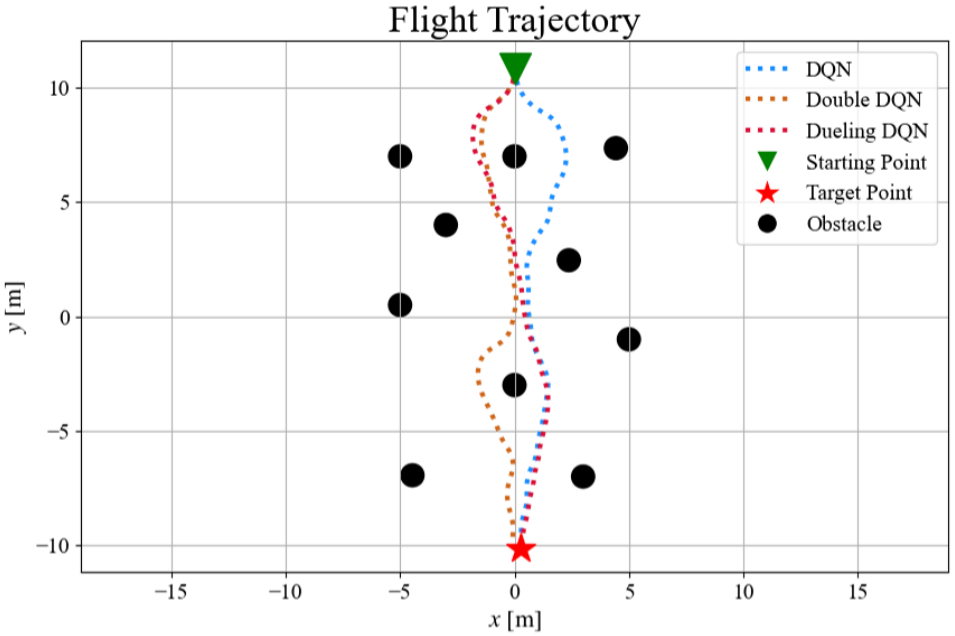
The figures reveal that the three algorithms generate similar collision-free trajectories for scene-crowded and scene-horizontal. Notably, in scene-crowded, all three algorithms opt to navigate through narrow obstacles rather than bypassing them. This suggests that the action execution penalty introduced to minimize the number of UAV movement steps enables it to select a more optimal path to reach the target point. For scene-random, each algorithm generates a different path, but the differences are mainly related to the selection of steering direction when facing an obstacle directly ahead. Since turning left or right can be viewed as symmetric selections, the three trajectories can be considered essentially similar, with no significant differences.

The tables reveal that the success rates of all three algorithms for different scenarios range between 70 and 80 percent, which is consistent with the average reward stabilizing around 800 during training. Notably, the Dueling DQN algorithm achieved the highest success rate among the three algorithms, as well as the shortest average number of steps required to complete a flight. This result can be attributed to the fact that the action-advantage function leveraged in the Dueling DQN algorithm better distinguishes between the rewards generated by the current state and the available actions, enabling it to select the better optimal action in a given scenario more effectively than the other two algorithms. Although the Dueling DQN algorithm generates action commands slightly less frequently than the other two algorithms, this has minimal impact on the UAV application scenario, which is less densely populated with obstacles than the unmanned vehicle application scenario, leading to a slight frequency of action command generation acceptable.

Although DRL approaches can learn navigation and OA strategies under conditional constraints, they also suffer two limitations. The first limitation is that MDE is subject to severe inaccuracies when the underlying assumptions are violated or when there are significant deviations in scene features. This can significantly impact the success rate of the UAV OA. The second limitation arises from the policy model resting on gradient optimization and reward forward propagation, which results in limited perceptual time memory.

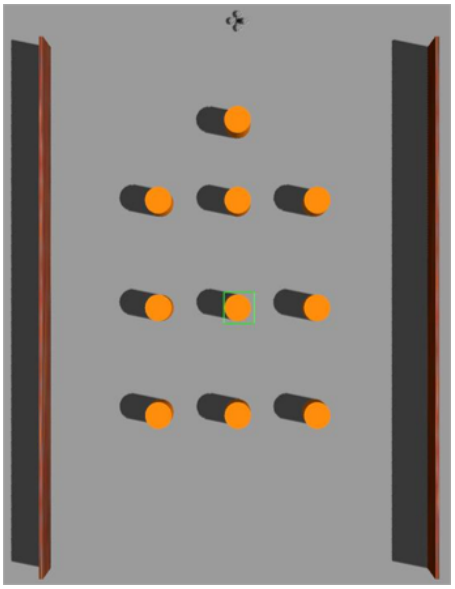


(a) Top view of the scenario.

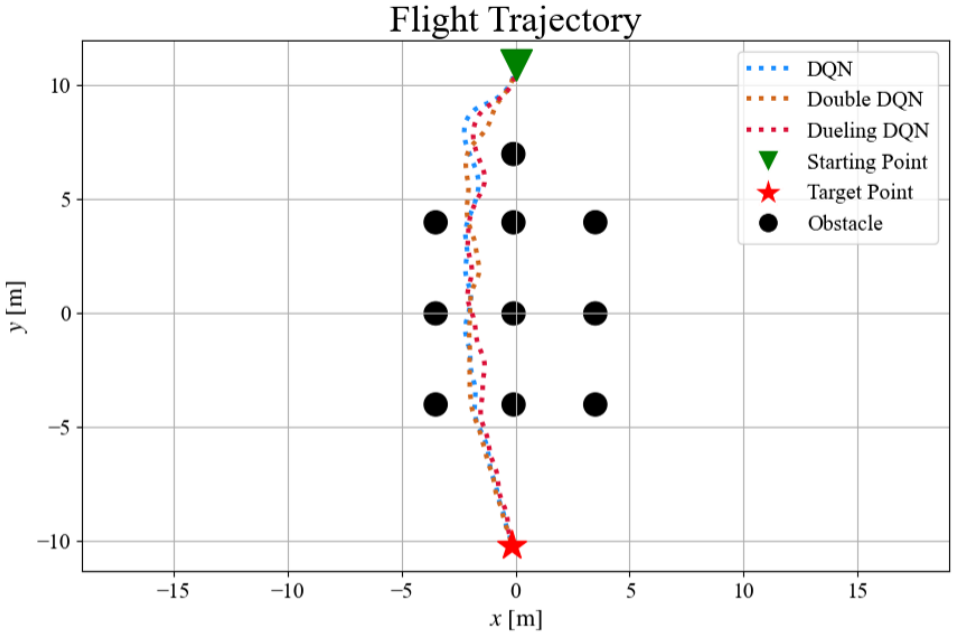


(b) Collision-free trejectory.

FIGURE 4.19: Collision-free trajectory of scene-random.

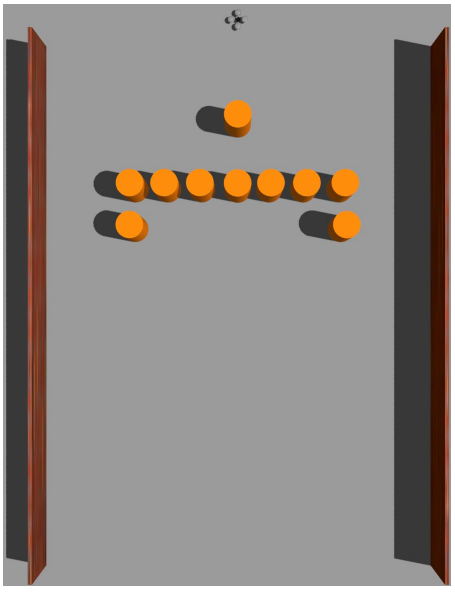


(a) Top view of the scenario.

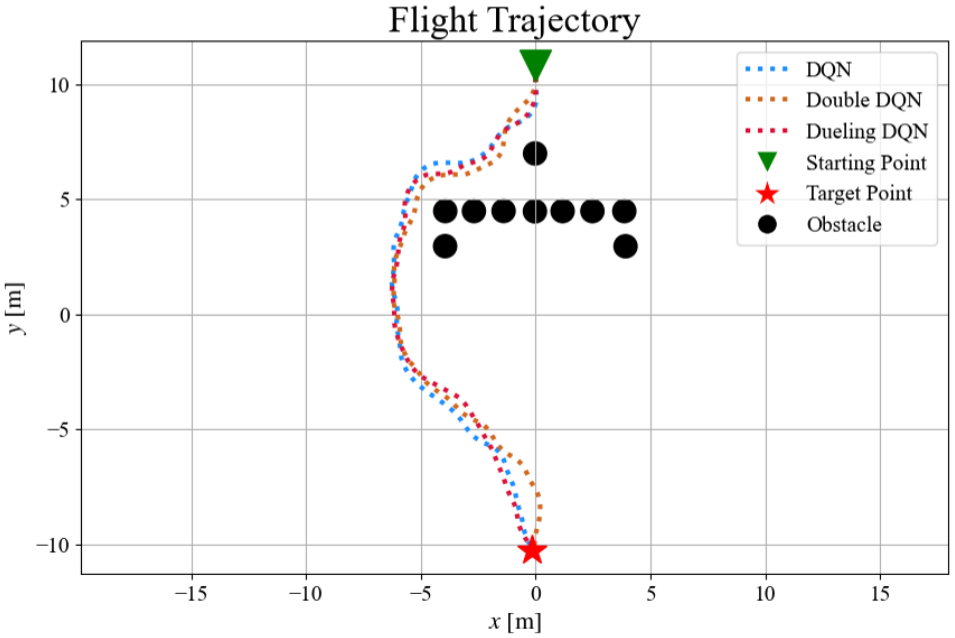


(b) Collision-free trejectory.

FIGURE 4.20: Collision-free trajectory of scene-crowded.



(a) Top view of the scenario.



(b) Collision-free trejectory.

FIGURE 4.21: Collision-free trajectory of scene-horizontal.

In some regions, such as recessed areas depicted in Fig. 4.21(a), the negative reward of an obstacle cannot be forward transmitted out, or the UAV step is too small to escape within a limited number of times. In such cases, the UAV cannot get out after entering the area. Better methods to solve this problem include long-term memory, comprehensive map modeling, and retraining a more appropriate RL policy model for the scenario. However, these methods are challenging to implement in practical small-scale real-time OA. The idea of simulated annealing can help jump out of the local optimal region by increasing the chance of random exploration. However, the long-term exploration of feasible path behavior is impractical in terms of energy consumption. Therefore, when trapped in the optimal region, leveraging the policy-based temporary goal method is prone to increase the possibility of the UAV escaping from the local optimal region.

TABLE 4.4: Obstacle Avoidance Performance in Scene-Random

Algorithm \ Indicator	Success Rate (%)	Frequency (HZ)	Average Steps
DQN	77	33	69
Double DQN	79	33	67
Dueling DQN	82	31	64

TABLE 4.5: Obstacle Avoidance Performance in Scene-Crowded

Algorithm \ Indicator	Success Rate (%)	Frequency (HZ)	Average Steps
DQN	72	33	77
Double DQN	73	33	76
Dueling DQN	77	31	74

When the UAV hovers in a particular area for a long time, and it is determined that there is no path to move forward to the target, it is considered to be caught in a local optimal area. However, consecutive depth perceptions can provide hidden escape information. In this case, a temporary target point is selected relying on the

TABLE 4.6: Obstacle Avoidance Performance in Scene-Horizontal

Algorithm \ Indicator	Success Rate (%)	Frequency (HZ)	Average Steps
DQN	79	33	87
Double DQN	80	33	86
Dueling DQN	84	31	83

behavior value of the policy network, ranging from 60 to 90 degrees on either side of a straight line perpendicular to the desired destination and UAV's position, after which it is reset back to the original desired destination after a random duration T, which is related to the size of the area to be escaped. This method may help alleviate the issue of UAVs being trapped in recessed areas.

4.3 Conclusion and Discussion

This chapter introduces a depth image-based OA system that differs from conventional methods by not relying on pre-existing maps or models to generate collision-free trajectories. Instead, the system leverages a neural network to model the nonlinear mapping between depth map input and UAV motion commands. Additionally, the chapter presents an RL-based approach for the agent to gradually learn to avoid obstacles through the trial-and-error process rather than depending on pre-calculated collision-free trajectories. This approach results in a system with lower computational requirements, improved generalization, and better adaptability to pop-out obstacles.

This chapter provides a focused analysis of the UAV OA problem, with an emphasis on mapping continuous state inputs to discrete action outputs. To achieve this, this chapter employs three value-based algorithms (DQN, Double DQN, and Dueling DQN) to model the action-value function. These algorithms score all possible functions to be executed in the current state and subsequently output the action with the highest value. If the UAV continues to execute the actions with the highest value, these action sequences can form a collision-free flight trajectory that leads to the endpoint. Furthermore, to validate the proposed model, the chapter creates three different scenarios deploying Gazebo. The results demonstrate that all three

algorithms have the potential to facilitate the UAV to complete the OA task with a probability of nearly 80 percent. This success suggests that the proposed model has some degree of feasibility and transferability.

The models proposed in this chapter have been validated only in a simulated environment, and their sim-to-real capabilities cannot be established due to the significant gap between the two environments. For instance, simulated environments offer relatively stable illumination and climate conditions, resulting in seldom disruptions to the camera inputs. In contrast, the real environment is subject to unpredictable climate and illumination variations, rendering the model trained in a stable simulated environment inapplicable to real-life scenarios. Furthermore, the UAV's dynamics control model in the simulated environment is more stable and less affected by external disturbances. However, the UAV's motion is frequently affected by external factors such as intense turbulence, which can lead to abrupt changes in its motion state, causing image distortion and excessive occlusion, thereby significantly affecting its real-time OA capability. Additionally, while this chapter only considers OA in static environments, dynamic obstacles in practical applications cannot be ignored. Future research should focus on improving the UAV's sim-to-real capabilities and handling dynamic obstacles.

Chapter 5

Conslusion and Future Work

This study proposes a system that enables UAVs to recognize and avoid obstacles by leveraging learning-based monocular vision. The process of automatic navigation for UAVs is divided into two phases: OD and OA. OD involves observing the surrounding environment through an onboard monocular camera and detecting approaching obstacles. OA involves utilizing RGB images obtained from the camera to execute avoidance maneuvers that facilitate the UAV to reach the target point while circumventing obstacles. The study provides a comprehensive theoretical design and simulation verification of the automatic navigation system, covering introductory and theoretical knowledge, experiment design, and results analysis.

By successfully developing a functional autonomous navigation system that relies on monocular vision, this study lays the foundation for future research to build upon. Initially, the model presented in this study has been verified solely in a simulated environment, and their capability to transfer to real-world scenarios cannot be established due to the substantial divergence between the two settings. Simulated environments provide relatively stable illumination and weather conditions, resulting in minimal disruptions to the camera inputs. Conversely, real-world conditions are unpredictable, with variations in weather and illumination that render the models trained in a stable simulated environment inadequate for real-world applications. Moreover, the UAV's dynamics control model in the simulated environment is more stable and less susceptible to external disturbances. However, the UAV's motion in actual scenarios is frequently impacted by external factors such as

turbulence, which can lead to sudden modifications in its motion state, causing image distortion and excessive occlusion, thereby significantly impacting its real-time OA capability. Therefore, conducting physical flight experiments is an essential aspect of future work. Specifically, the trained model will be deployed to transmit control signals to the UAV's flight control module. The model's performance will be further validated by examining the UAV's OA capacity in a real environment. Furthermore, it is crucial to overcome the issue of reducing the sim-to-real gap in UAV navigation as part of future research efforts. By narrowing this gap, the applicability of UAVs in real-world scenarios can be significantly enhanced. Given the intricate nature of an actual UAV navigation system, addressing key challenges such as reducing motor response time, optimizing battery consumption, and improving OA capabilities in three-dimensional space are pressing concerns that require immediate attention. Although it is acknowledged that definitive solutions to these issues have not been provided in this study, extensive investigation of recent advancements in these research areas has been undertaken through References [92, 99–101], which serve as a catalyst for conducting further research.

Afterward, future research should focus on developing automatic navigation systems to better adapt the model's performance to dynamic obstacles in real environments. With the complexity of the modern urban airspace environment, pop-out obstacles may appear in the original collision-free flight trajectory of the UAV, causing it challenging for the original self-navigation system based on static environment training to handle these situations. Therefore, there is a demand to improve the capacity of UAVs to cope with dynamic obstacles. Additionally, automatic navigation systems for fleets present a promising avenue of research to cater to the growing UAV application scenarios. It entails devising resolutions to acquire positional information of surrounding environments and other UAVs in the fleet from visual inputs to accomplish OA tasks while preserving the position fleet formation. This field of study holds immense potential in sectors like cargo transportation, but it also poses significant challenges that warrant further exploration originating from this study.

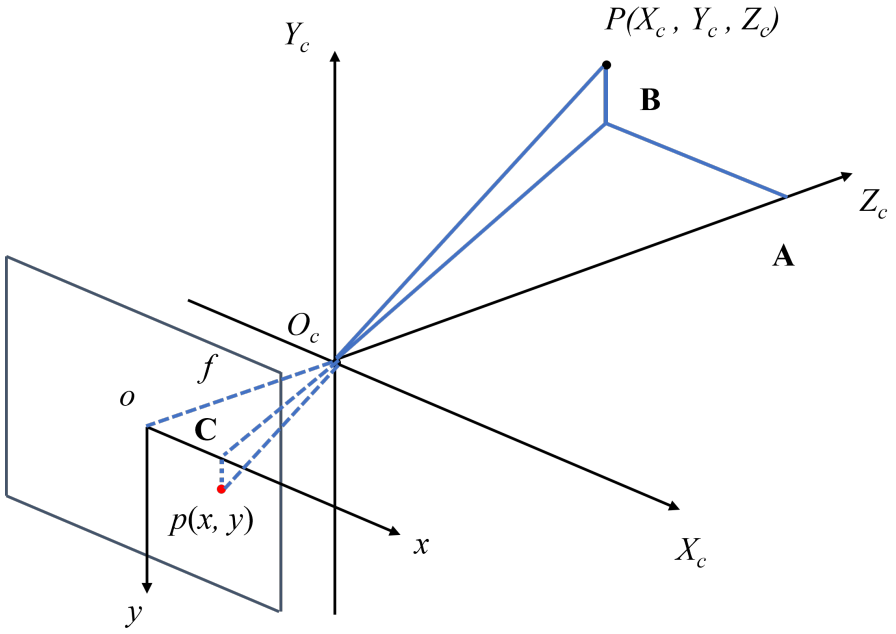
Appendix A

Preliminary for Chapter 3

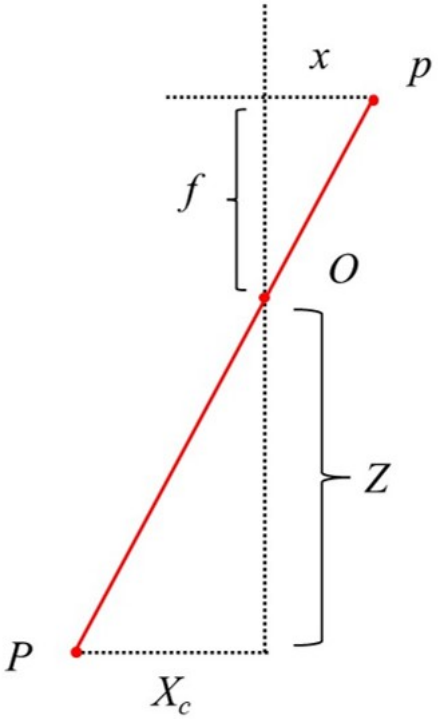
A.1 Image-Forming Principle

The imaging principle refers to the scientific concept behind the formation of images by capturing light from an object and transforming it into an image that can be perceived by the human eye or captured by an imaging device. The foundation of the imaging principle is the interaction of light with objects, which produces an image. Specifically, light enters the lens of an imaging device, passes through the aperture, and reaches the image sensor, where it is transformed into an electrical signal, which is then processed to form the final image.

A geometric model may be used to explain how the camera converts coordinate points in the three-dimensional environment (measured in meters) into pixels in the two-dimensional picture plane. The simplest model, known as the pinhole model, depicts the connection between the light beam that goes through a pinhole and the image that is projected on the rear of the pinhole. This study selects a straightforward pinhole camera model to structure this mapping relationship. The modeling process can be depicted in Fig. A.1(a). Consider the camera coordinate system as $O_c - X_c - Y_c - Z_c$, where O_c represents the camera's optical center or the pinhole. Let P be a real-world spatial point, which after projecting through the optical center O_c, falls on the physical imaging plane $o - x - y$, resulting in the imaging point p. The spatial coordinates of P are denoted as $[X_c, Y_c, Z_c]$, and the coordinates of p as $[x, y, z]$. The physical imaging plane is assumed to be at a distance of f (focal length) from O_c. According to the triangle similarity relation



(a) Principle of pinhole imaging.



(b) Diagram of similar triangle.

FIGURE A.1: Schematic diagram of camera imaging principle.

in Fig. A.1(b), the scaling relationship between camera coordinates and imaging coordinates can be obtained. Equation. A.1 omits the minus sign because it is equal to placing the imaging plane symmetrically in front of the camera. Removing the minus sign makes the formula more straightforward and allows for a more realistic model to be created.

$$\frac{Z_c}{f} = \frac{X_c}{x} = \frac{Y_c}{y}. \tag{A.1}$$

To describe how the sensor transforms the sensed light into picture pixels, the results on the imaging plane must be sampled and quantized because the final outputs of the camera are pixels. Let $o - u - v$ be a pixel plane fixed on the physical plane, and let $[u, v]$ denote the pixel coordinates of the projection point p. The pixel plane is usually translated and scaled with respect to the imaging plane. The relationship between the two planes is shown in Fig. A.2. The scaling factors for the u- and v-axis are denoted by α and β, respectively, and the origin is translated by $[cx, cy]$. Therefore, the mapping between the coordinates of the spatial point P and the pixel coordinates $[u, v]$ can be expressed as Eq. A.2, where $f_x = \alpha f$ and $f_y = \beta f$.

$$\begin{cases} u = f_x \frac{X_c}{Z} + c_x \\ v = f_y \frac{Y_c}{Z} + c_y \end{cases}. \tag{A.2}$$

Equation. A.2 can be written in matrix form, and the pixel coordinates can be written as homogeneous coordinates. The final expression is shown in Eq. A.3, where $\boldsymbol{K}$ represents the camera intrinsics, which is an inherent parameter matrix of the camera and is considered to remain unchanged after being produced.

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \frac{1}{Z_c} \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X_c \\ Y_c \\ Z_c \end{pmatrix} \overset{\text{def}}{=} \frac{1}{Z_c} \boldsymbol{K} \boldsymbol{P}. \tag{A.3}$$

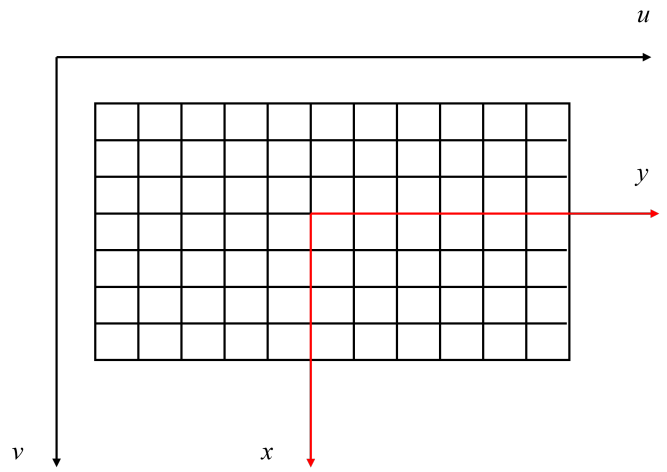


FIGURE A.2: Diagram of the relationship between the imaging and pixel plane.

This study draws inspiration from traditional stereo matching techniques, which rely on left and right images to determine depth values [102], to propose self-supervised learning depth estimation methods. As shown in Fig. A.3, the conventional stereo-based approaches utilize two cameras with optical centers O_L and O_R, respectively. A point P in the real world is projected onto the imaging planes of the left and right cameras as P_L and P_R, respectively. The left-right image disparity at P is calculated as $D = X_L - X_R$, where X_L and X_R are the distances of P_L and P_R from the left edge of the imaging plane, respectively. The baseline distance B between the optical centers and the focal length f are also depicted. The depth d of point P is the distance from P to the baseline. These principles serve as the foundation for the self-supervised MDE techniques proposed in this study.

Let the length of the left and right imaging planes be l. Similar triangle relationship can be obtained from $\Delta PP_LP_R \sim \Delta PO_LO_R$:

$$\frac{d-f}{d} = \frac{P_LP_R}{O_LO_R}, \tag{A.4}$$

hence:

$$\frac{d-f}{d} = \frac{B-\left(X_L-\frac{l}{2}\right)-\left(\frac{l}{2}-X_R\right)}{B}, \tag{A.5}$$

simplify Eq. A.5 to get:

$$d = \frac{B \times f}{D}. \tag{A.6}$$

The derivation process above indicates that the object depth and disparity have an inverse relationship when the camera's baseline and focal length parameters are known. With the disparity map, the left view can be obtained from the right view through mapping in the stereo images, and the image reconstruction can be completed.

Similar to the approaches based on stereo pair-wise images, the presented self-supervised model is trained with a monocular sequence and tested on a single image. Specifically, after obtaining the depth map and transformation matrix between adjacent frames, all pixel points on the current frame can be reprojected onto its adjacent frames, which is also defined as the warping process. The pixel coordinates of p_{n-1} can be acquired using through

$$P_{n-1} \sim \mathbf{K} T_{n \to n-1} D_n \left(P_n\right) \mathbf{K}^{-1} P_n, \tag{A.7}$$

where I_n denote the current frame and P_n be a pixel on it, while P_{n-1} corresponds to the matching pixel on the previous frame I_{n-1}. The internal parameter of the camera is represented by the constant matrix $\mathbf{K}$. $T_{n \to n-1}$ denotes the self-movement matrix between I_{n-1} and I_n. Additionally, $D_n\left(P_n\right)$ represents the depth value at the pixel P_n.

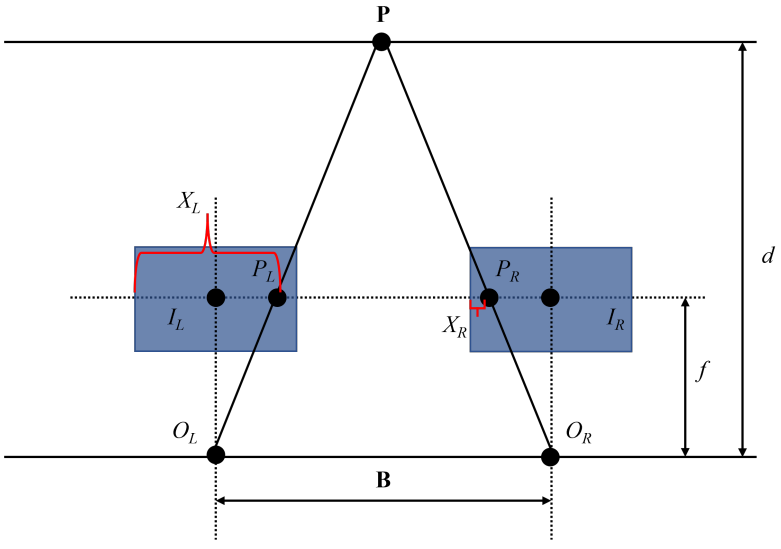


FIGURE A.3: The fundamental idea behind stereo matching techniques for depth estimation, where I_L and I_R are stereo pictures acquired in pairs by the left and right cameras, respectively.

A.2 Multilayer Perceptron

An MLP, short for Multilayer Perceptron, is a form of an artificial neural network composed of multiple interconnected layers. It is also defined as a feedforward neural network, as the extracted features can only flow in one direction along the forward propagation without loops. A simple diagram of the MLP is depicted in Fig. A.4. The structure of an MLP is comprised of an input layer, one or multiple hidden layers, and an output layer. Each circle in the hidden layer in the above figure represents a neuron node, and the individual nodes are usually fully connected to each other. These nodes serve to extract desired features from the input vector to model a function that can execute either classification or regression operations on input vectors. The output of the hidden layer is obtained by applying an activation function f to $w_1X + b_1$, where w_1 represents the weight of the fully connected layer, b_1 is the bias, and X is the input layer vector. The activation function is a mathematical operation to offer more nonlinearity to the output of a node. Since practical problems often involve complex functions, relying solely on linear activation functions in a MLP will result in outputs limited to linear combinations of vectors, which falls short of practical requirements. Therefore, nonlinear activation functions such as sigmoid, hyperbolic tangent, and rectified linear unit (ReLU) are frequently employed in MLPs.

In this study, several MLPs are leveraged to complete regression tasks. In order to ensure that the output is closest to the actual value, gradient descent is usually deployed to update the weights and biases. All weights and biases are required to be initialized first. Subsequently, the gradient is calculated by computing the error between the predicted and actual values, which supervises updating weights and biases to penalize the error. When the error is sufficiently small, this MLP can be considered feasible.

A.3 Convolutional Neural Network

A Convolutional Neural Network (CNN) is a DL algorithm specifically optimized for image recognition and visual tasks. CNNs are suitable for processing spatial data and usually contain five parts: an input layer, a convolutional layer, a pooling layer, a fully connected layer, and an output layer. The structure of a traditional

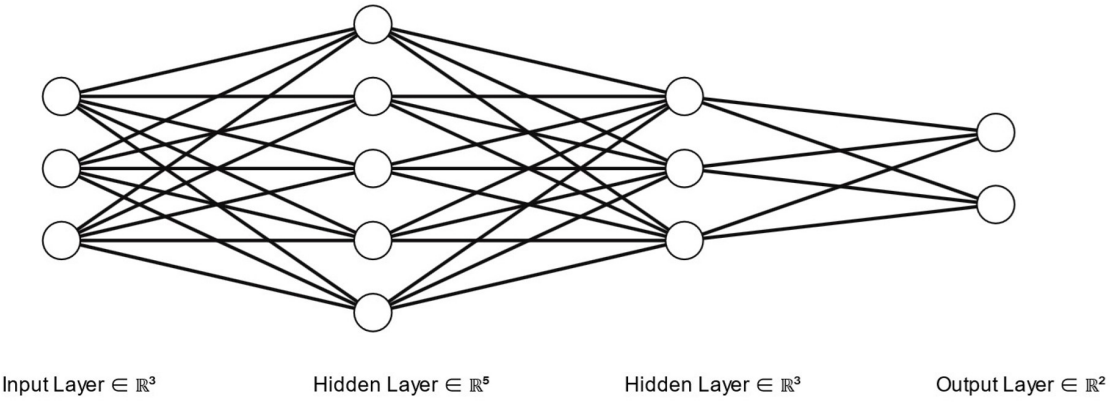


FIGURE A.4: Diagram of a multilayer perceptron.

CNN is shown in Fig. A.5 below. Here, the convolution layer maps the unprocessed data to the feature space of intermediate layers and extracts the image's local features while ensuring its spatial continuity. Moreover, the pooling layer reduces the dimensionality of the features by means of average pooling or maximum pooling, which can reduce the computational burden and has rotational invariance. The fully connected layer can integrate the previously learned local information and map the captured distributed feature representations into the sample token space, which is leveraged to connect the final feature map and classifier.

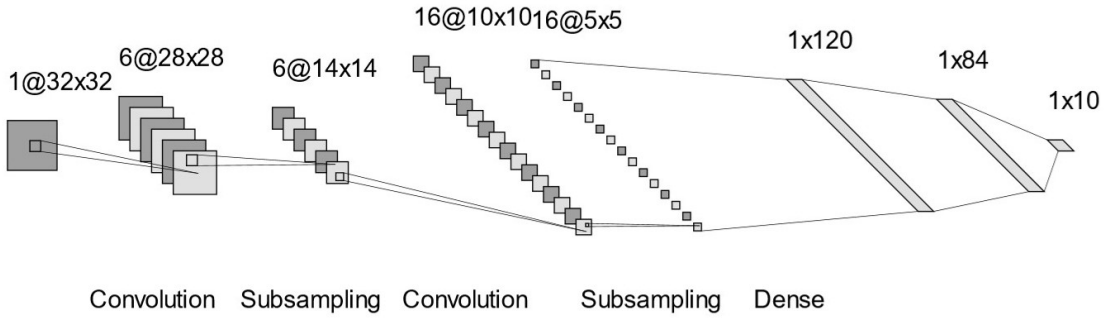


FIGURE A.5: Basic modules of a convolutional neural network.

The working principle of a CNN is shown in Eq. A.8.

$$y_{ij}^{(l+1)} = \sum_{a=1}^{A} \sum_{b=1}^{B} \omega_{ab}^{(l)} \alpha_{(ki+a)(kj+b)}^{(l)} + \beta^{(l)}, \tag{A.8}$$

where $\alpha_{mn}^{(l)}$ and $y_{ij}^{(l+1)}$ are the input and output of the lth convolution layer, $\omega_{ab}^{(l)}$ denotes the convolution kernel and its receptive field size is $A \times B$, k represents the stride, and $\beta^{(l)}$ refers to the bias.

CNNs have a unique set of characteristics, including local connections, shared weights, and pooling operations, that help to decrease computational demands while increasing the model's resistance to transformations such as translation and scaling. Additionally, the deep structure of CNNs provides them with powerful feature learning capabilities, which enables them to extract more robust features and better align with the desired objective function.

Appendix B

Preliminary for Chapter 4

B.0.1 Markov Decision Process

Throughout the training process, the agent constantly engages with the environment to generate a series of states, actions, and rewards.$(s_1, a_1, r_1, s_2, a_2, r_2, \ldots, a_{t-1}, s_t)$, defined as a general sequential decision process and is usually described by a Markovian Decision Process (MDP) [103]. MDP is a mathematical framework used to describe decision-making problems in RL, which suggests that the current state s at time t is solely dependent on the previous state s_{t-1} and action a_{t-1}, disregarding other historical states. This concept is depicted in Eq. B.1.

$$P\left[s_t \mid s_{t-1}\right] = P\left[s_t \mid s_1, \ldots, s_{t-1}\right]. \tag{B.1}$$

The MDP framework comprises four elements denoted by the tuple (S, A, R, P). The state space S refers to the set of possible states that the agent can be in at any given time. The action space A represents the set of actions that the agent can take in each state. The state transition function P, denoted as $p\left(s_{t+1} \mid s_t, a_t\right)$, describes the probability that the agent will transition to state s_{t+1} after taking action a_t in state s_t. Finally, the reward function R provides feedback to the agent about its performance after taking an action a_t in state s_t.

The objective of RL is to discover the best course of an action sequence that maximizes the expectation of weighted cumulative reward during the MDP. According to the previous description, a policy accountable for generating the best course of

an action sequence can be defined as a nonlinear mapping from states to actions, or as a probability distribution of actions given a group of states, usually denoted by π:

$$\pi(a \mid s) = P\left[\, \mathrm{A}_t = a \mid S_t = \mathrm{s}\right]. \tag{B.2}$$

The complete MDP is described as follows: initially, the state s_0 in which the agent is initialized, and then the action a_0 is selected resting on the policy function $a_0 \sim \pi\left(a \mid s_0\right)$. Subsequently, the new state s_1 is obtained by relying on the state transition function $s_1 \sim p\left(s \mid s_0, a_0\right)$ to get a reward $r_1 = R_{s_0 s_1}^{a_0}$ for the current step. Iterations of this operation continue until the termination state s_T is reached, where the entire trajectory $\tau = (s_0, a_0, s_1, a_1, \ldots, s_T)$ can be obtained, and the joint probability of the trajectory can be expressed as:

$$p(\tau) = p\left(s_0\right) \prod_{t=0}^{T-1} p\left(a_{t-1} \mid s_{t-1}\right) \cdot p\left(s_t \mid s_{t-1}, a_{t-1}\right). \tag{B.3}$$

For each trajectory, the function that encompasses all single-step rewards is the cumulative reward function, $R = f\left(r_0, r_1, \ldots, r_T\right)$. This function can be either a T-step cumulative reward function $R = \sum_{t=0}^{T-1} r_t$ or a γ-discounted reward function $R = \sum_{t=0}^{T-1} \gamma^t r_t$. The expected cumulative reward under a given task can be defined as:

$$E_R = E\left[\sum_{t=0}^{T-1} \gamma^t r_t\right]. \tag{B.4}$$

The agent aims to discover the most effective approach that leads to the highest anticipated cumulative reward:

$$\pi^* = \max_{\pi} E^{\pi}\left[\sum_{t=0}^{T-1} \gamma^t r_t\right]. \tag{B.5}$$

Usually, the cumulative reward is suitable means to express the value of the present state. The expected cumulative reward employing the strategy π with starting state s is specified as the state value function:

$$V^{\pi}(s) = E^{\pi}\left[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s\right]. \tag{B.6}$$

With s as the starting state and a as the action executed deploying policy π after perceiving the starting state, the expected cumulative reward for deploying policy π is defined as the state-action value function:

$$Q^{\pi}(s,a) = E^{\pi}\left[\sum_{t=0}^{\infty}\gamma^{t} r_t \mid s_0 = s, a_0 = a\right]. \tag{B.7}$$

The Bellman equation for the state value function can be obtained from the definition of the state value function and Eq. B.6:

$$\begin{aligned} V(s) &= E\left[G_t \mid S_t = s\right] \\ &= E\left[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots \mid S_t = s\right] \\ &= E\left[R_{t+1} + \gamma\left(R_{t+2} + \gamma R_{t+3} + \ldots\right) \mid S_t = s\right] \\ &= E\left[R_{t+1} + \gamma V\left(S_{t+1}\right) \mid S_t = s\right] \end{aligned}. \tag{B.8}$$

Similarly, the Bellman equation for the state-action value function can be obtained as follows:

$$\begin{aligned} Q^{\pi}(s,a) &= E\left[r_{t+1} + \lambda r_{t+2} + \lambda^2 r_{t+3} + \ldots \mid (s,a)\right] \\ &= E_{s'}\left[r_t + \lambda\left(Q^{\pi}\left(s', a'\right)\right) \mid (s,a)\right] \end{aligned}. \tag{B.9}$$

Thus the optimal state value function and the state-action value function can be obtained as:

$$V^{*}(s) = \max_{a}\left[R_{t+1} + \gamma E\left[V^{*}\left(\ \mathrm{S}'\right)\right]\right], \tag{B.10}$$

$$Q^{*}(s,a) = \max_{\pi} Q^{\pi}(s,a). \tag{B.11}$$

The optimal policy can then be determined dependent on the value function acquired:

$$\pi^{*}(a \mid s) = \begin{cases} 1 & a = \underset{a \in A}{\mathrm{argmax}} Q(s,a) \\ 0 & \text{otherwise} \end{cases}. \tag{B.12}$$

List of Publications

- **Y. Zhang**, Q. Yu, K. H. Low and C. Lv, "A Self-Supervised Monocular Depth Estimation Approach Based on UAV Aerial Images," 2022 IEEE/AIAA 41st Digital Avionics Systems Conference (DASC), Portsmouth, VA, USA, 2022, pp. 1-8, doi: 10.1109/DASC55683.2022.9925733.
- **Y. Zhang**, K. H. Low and C. Lv, "Partially-Observable Monocular Autonomous Navigation for UAV through Deep Reinforcement Learning." (Accepted by AIAA AVIATION 2023 Forum)

Bibliography

[1] Claudia Stöcker, Rohan Bennett, Francesco Nex, Markus Gerke, and Jaap Zevenbergen. Review of the current state of uav regulations. *Remote sensing*, 9(5):459, 2017. 1

[2] Anuj Puri. A survey of unmanned aerial vehicles (uav) for traffic surveillance. *Department of computer science and engineering, University of South Florida*, pages 1–29, 2005. 1

[3] Oscar Alvear, Nicola Roberto Zema, Enrico Natalizio, and Carlos T Calafate. Using uav-based systems to monitor air pollution in areas with poor accessibility. *Journal of Advanced Transportation*, 2017, 2017. 1

[4] Rajesh Gupta, Arpit Shukla, Parimal Mehta, Pronaya Bhattacharya, Sudeep Tanwar, Sudhanshu Tyagi, and Neeraj Kumar. Vahak: A blockchain-based outdoor delivery scheme using uav for healthcare 4.0 services. In *IEEE INFOCOM 2020-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 255–260. IEEE, 2020. 1

[5] Luis Merino, Fernando Caballero, JR Martinez-de Dios, and Aníbal Ollero. Cooperative fire detection using unmanned aerial vehicles. In *Proceedings of the 2005 IEEE international conference on robotics and automation*, pages 1884–1889. IEEE, 2005. 1

[6] Adarsh Sudarsanan, Amalu S Panicker, Karthik J, Sachu Pradeep, Mohamed Samshad, Arun C, and Praveen Raj R S. Air pollution detection using uav. In *International journal of innovative research in technology*, volume 8 of *1*. Ahmedabad: Solaris Publication, jun 2021. 1

[7] Michael Rucker. The potential of drones providing health services. URL `https://www.verywellhealth.com/potential-of-drones-providing-health-services-4018989`. 2

[8] David Daly. A not-so-short history of unmanned aerial vehicles (uav), May 2022. URL `https://consortiq.com/uas-resources/short-history-unmanned-aerial-vehicles-uavs`. 2

[9] Parimal Kopardekar, Joseph Rios, Thomas Prevot, Marcus Johnson, Jaewoo Jung, and John E Robinson. Unmanned aircraft system traffic management

(utm) concept of operations. In *AIAA Aviation and Aeronautics Forum (Aviation 2016)*, number ARC-E-DAA-TN32838, 2016. 2

[10] Rakesh Shrestha, Inseon Oh, and Shiho Kim. A survey on operation concept, advancements, and challenging issues of urban air traffic management, Jan 2021. URL `https://www.frontiersin.org/articles/10.3389/ffutr.2021.626935/full`. 2

[11] Wynn Wang. Dozens of drones fall from the sky during light show in china. URL `https://www.scmp.com/video/china/3151232/dozens-drones-fall-sky-during-light-show-china`. 2

[12] Livia Borghese. Tourists crash drones into italy landmarks in rome and pisa, Apr 2022. URL `https://edition.cnn.com/travel/article/tourist-drone-incidents-rome-pisa/index.html`. 2

[13] Darren Linton. Drone crash in sydney hotel injures guest, Jun 2022. URL `https://www.sheppnews.com.au/national/drone-crash-in-sydney-hotel-injures-guest/`. 2

[14] Vincent YW Lee, David TL Liu, Gloria Leung, Y Luo, and Philip TH Lam. Devastating projectile injury of the eye caused by a remote-controlled toy helicopter. *Hong Kong medical journal*, 15(6):492–493, 2009. 3

[15] Jawad N Yasin, Sherif AS Mohamed, Mohammad-Hashem Haghbayan, Jukka Heikkonen, Hannu Tenhunen, and Juha Plosila. Unmanned aerial vehicles (uavs): Collision avoidance systems and approaches. *IEEE access*, 8:105139–105155, 2020. 4

[16] Anusha Mujumdar and Radhakant Padhi. Evolving philosophies on autonomous obstacle/collision avoidance of unmanned aerial vehicles. *Journal of Aerospace Computing, Information, and Communication*, 8(2):17–41, 2011. 4, 58

[17] 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. 11

[18] AN Rajagopalan, Subhasis Chaudhuri, and Uma Mudenagudi. Depth estimation and image restoration using defocused stereo pairs. *IEEE transactions on pattern analysis and machine intelligence*, 26(11):1521–1525, 2004.

[19] 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.

[20] Fabio Tosi, Filippo Aleotti, Matteo Poggi, and Stefano Mattoccia. Learning monocular depth estimation infusing traditional stereo knowledge. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9799–9809, 2019. 11

[21] Xingshuai Dong, Matthew A Garratt, Sreenatha G Anavatti, and Hussein A Abbass. Towards real-time monocular depth estimation for robotics: A survey [-5pt]. *IEEE Transactions on Intelligent Transportation Systems*, 2022. 11

[22] Shimon Ullman. The interpretation of structure from motion. *Proceedings of the Royal Society of London. Series B. Biological Sciences*, 203(1153): 405–426, 1979. 11

[23] Johannes L Schonberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4104–4113, 2016. 11

[24] Ping Li, Dirk Farin, Rene Klein Gunnewiek, et al. On creating depth maps from monoscopic video using structure from motion. In *Proc. of IEEE Workshop on Content Generation and Coding for 3D-television*, pages 508–515, 2006. 11

[25] 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. 12

[26] Li Ding and Gaurav Sharma. Fusing structure from motion and lidar for dense accurate depth map estimation. In *2017 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 1283–1287. IEEE, 2017. 12

[27] Minhyeok Lee, Sangwon Hwang, Chaewon Park, and Sangyoun Lee. Edgeconv with attention module for monocular depth estimation. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 2858–2867, 2022. 13

[28] Varun Ravi Kumar, Senthil Yogamani, Markus Bach, Christian Witt, Stefan Milz, and Patrick Mäder. Unrectdepthnet: Self-supervised monocular depth estimation using a generic framework for handling common camera distortion models. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 8177–8183. IEEE, 2020. 13

[29] David Eigen, Christian Puhrsch, and Rob Fergus. Depth map prediction from a single image using a multi-scale deep network. *Advances in neural information processing systems*, 27, 2014. 14, 35

[30] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *2012 IEEE conference on computer vision and pattern recognition*, pages 3354–3361. IEEE, 2012. 14, 35, 40, 53

[31] 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. 14

[32] David Eigen and Rob Fergus. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In *Proceedings of the IEEE international conference on computer vision*, pages 2650–2658, 2015. 14

[33] Vlad-Cristian Miclea and Sergiu Nedevschi. Monocular depth estimation with improved long-range accuracy for uav environment perception. *IEEE Transactions on Geoscience and Remote Sensing*, 60:1–15, 2021. 14

[34] 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. 15

[35] 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. 16

[36] 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. 16, 53, 56

[37] 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. 16, 48, 49, 53, 54, 56

[38] Kyle Julian, John Mern, and Rachael Tompa. Uav depth perception from visual images using a deep convolutional neural network. In *Tech. Rep.* 2017. 17

[39] 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 0–0, 2018. 17

[40] Zichen He, Jiawei Wang, and Chunwei Song. A review of mobile robot motion planning methods: from classical motion planning workflows to reinforcement learning-based architectures. *arXiv preprint arXiv:2108.13619*, 2021. 18, 31

[41] ByeoungDo Kim, Chang Mook Kang, Jaekyum Kim, Seung Hi Lee, Chung Choo Chung, and Jun Won Choi. Probabilistic vehicle trajectory prediction over occupancy grid map via recurrent neural network. In *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, pages 399–404. IEEE, 2017. 18

[42] Yuquan Xu, Vijay John, Seiichi Mita, Hossein Tehrani, Kazuhisa Ishimaru, and Sakiko Nishino. 3d point cloud map based vehicle localization using stereo camera. In *2017 IEEE intelligent vehicles symposium (IV)*, pages 487–492. IEEE, 2017. 18

[43] Olaf Booij, Bas Terwijn, Zoran Zivkovic, and B Krose. Navigation using an appearance based topological map. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 3927–3932. IEEE, 2007. 18

[44] Carlos Nieto-Granda, John G Rogers, Alexander JB Trevor, and Henrik I Christensen. Semantic map partitioning in indoor environments using regional analysis. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1451–1456. IEEE, 2010. 18

[45] Lun Quan, Luxin Han, Boyu Zhou, Shaojie Shen, and Fei Gao. Survey of uav motion planning. *IET Cyber-systems and Robotics*, 2(1):14–21, 2020. 18

[46] Edsger W Dijkstra. A note on two problems in connexion with graphs. In *Edsger Wybe Dijkstra: His Life, Work, and Legacy*, pages 287–290. 2022. 19

[47] 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. 19

[48] Anthony Stentz. Optimal and efficient path planning for partially known environments. In *Intelligent unmanned ground vehicles*, pages 203–220. Springer, 1997. 19

[49] Sven Koenig, Maxim Likhachev, and David Furcy. Lifelong planning a. *Artificial Intelligence*, 155(1-2):93–146, 2004. 19

[50] Sven Koenig and Maxim Likhachev. Dˆ* lite. *Aaai/iaai*, 15:476–483, 2002. 20

[51] Daniel Harabor and Alban Grastien. Online graph pruning for pathfinding on grid maps. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 25, pages 1114–1119, 2011. 21

[52] Lydia E Kavraki, Petr Svestka, J-C Latombe, and Mark H Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation*, 12(4):566–580, 1996. 21

[53] Steven M LaValle et al. Rapidly-exploring random trees: A new tool for path planning. 1998. 22

[54] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The international journal of robotics research*, 30(7):846–894, 2011. 22

[55] Jauwairia Nasir, Fahad Islam, Usman Malik, Yasar Ayaz, Osman Hasan, Mushtaq Khan, and Mannan Saeed Muhammad. Rrt*-smart: A rapid convergence implementation of rrt. *International Journal of Advanced Robotic Systems*, 10(7):299, 2013. 23

[56] Oktay Arslan and Panagiotis Tsiotras. Use of relaxation methods in sampling-based algorithms for optimal motion planning. In *2013 IEEE International Conference on Robotics and Automation*, pages 2421–2428. IEEE, 2013. 23

[57] Russell Gayle, Avneesh Sud, Ming C Lin, and Dinesh Manocha. Reactive deformation roadmaps: motion planning of multiple robots in dynamic environments. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3777–3783. IEEE, 2007. 23

[58] Mitul Saha, Jean-Claude Latombe, Yu-Chi Chang, and Friedrich Prinz. Finding narrow passages with probabilistic roadmaps: The small-step retraction method. *Autonomous robots*, 19(3):301–319, 2005. 24

[59] John T Betts. Survey of numerical methods for trajectory optimization. *Journal of guidance, control, and dynamics*, 21(2):193–207, 1998. 24

[60] Igor Mordatch and Emo Todorov. Combining the benefits of function approximation and trajectory optimization. In *Robotics: Science and Systems*, volume 4, page 23, 2014. 25

[61] Zhigang Xu, Yu Wang, Guanqun Wang, Xiaopeng Li, Robert L Bertini, Xiaobo Qu, and Xiangmo Zhao. Trajectory optimization for a connected automated traffic stream: Comparison between an exact model and fast heuristics. *IEEE Transactions on Intelligent Transportation Systems*, 22(5): 2969–2978, 2020. 25

[62] Michael A Henson and Dale E Seborg. Feedback linearizing control. In *Nonlinear process control*, volume 4, pages 149–231. Prentice-Hall Upper Saddle River, NJ, USA, 1997. 25

[63] Taeyoung Lee, Melvin Leok, and N Harris McClamroch. Geometric tracking control of a quadrotor uav on se (3). In *49th IEEE conference on decision and control (CDC)*, pages 5420–5425. IEEE, 2010. 25

[64] Tarek Madani and Abdelaziz Benallegue. Control of a quadrotor mini-helicopter via full state backstepping technique. In *Proceedings of the 45th IEEE Conference on Decision and Control*, pages 1515–1520. IEEE, 2006. 25

[65] Samir Bouabdallah and Roland Siegwart. Backstepping and sliding-mode techniques applied to an indoor micro quadrotor. In *Proceedings of the 2005 IEEE international conference on robotics and automation*, pages 2247–2252. IEEE, 2005. 25

[66] Benoit Belobo Mevo, Mohamad R Saad, and Raouf Fareh. Adaptive sliding mode control of wheeled mobile robot with nonlinear model and uncertainties. In *2018 IEEE Canadian Conference on Electrical & Computer Engineering (CCECE)*, pages 1–5. IEEE, 2018. 25

[67] Francisco Gavilan, Rafael Vazquez, and Eduardo F Camacho. An iterative model predictive control algorithm for uav guidance. *IEEE transactions on aerospace and electronic systems*, 51(3):2406–2419, 2015. 26

[68] Björn Lindqvist, Sina Sharif Mansouri, Ali-akbar Agha-mohammadi, and George Nikolakopoulos. Nonlinear mpc for collision avoidance and control of uavs with dynamic obstacles. *IEEE robotics and automation letters*, 5(4): 6001–6008, 2020. 26

[69] Wang Di, Li Caihong, Guo Na, Song Yong, Gao Tengteng, and Liu Guoming. Local path planning of mobile robot based on artificial potential field. In *2020 39th Chinese Control Conference (CCC)*, pages 3677–3682. IEEE, 2020. 26, 27

[70] Olov Andersson, Oskar Ljungqvist, Mattias Tiger, Daniel Axehill, and Fredrik Heintz. Receding-horizon lattice-based motion planning with dynamic obstacle avoidance. In *2018 IEEE Conference on Decision and Control (CDC)*, pages 4467–4474. IEEE, 2018. 26

[71] Javier Minguez and Luis Montano. Nearness diagram navigation (nd): A new real time collision avoidance approach. In *Proceedings. 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2000)(Cat. No. 00CH37113)*, volume 3, pages 2094–2100. IEEE, 2000. 27

[72] Marija Seder and Ivan Petrovic. Dynamic window based approach to mobile robot motion control in the presence of moving obstacles. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 1986–1991. IEEE, 2007. 27

[73] Aleksandra Faust, Ivana Palunko, Patricio Cruz, Rafael Fierro, and Lydia Tapia. Learning swing-free trajectories for uavs with a suspended load. In *2013 IEEE International Conference on Robotics and Automation*, pages 4902–4909. IEEE, 2013. 28

[74] Aleksandra Faust, Ivana Palunko, Patricio Cruz, Rafael Fierro, and Lydia Tapia. Aerial suspended cargo delivery through reinforcement learning. *Department of Computer Science, University of New Mexico, Tech. Rep*, 151, 2013. 28

[75] Aleksandra Faust, Kenneth Oslund, Oscar Ramirez, Anthony Francis, Lydia Tapia, Marek Fiser, and James Davidson. Prm-rl: Long-range robotic navigation tasks by combining reinforcement learning and sampling-based planning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5113–5120. IEEE, 2018. 29

[76] Utsav Patel, Nithish Kumar, Adarsh Jagan Sathyamoorthy, and Dinesh Manocha. Dynamically feasible deep reinforcement learning policy for robot navigation in dense mobile crowds. *arXiv preprint arXiv:2010.14838*, 2020. 29

[77] Lu Chang, Liang Shan, Chao Jiang, and Yuewei Dai. Reinforcement based mobile robot path planning with improved dynamic window approach in unknown environment. *Autonomous Robots*, 45(1):51–76, 2021. 29

[78] Claudia Pérez-D'Arpino, Can Liu, Patrick Goebel, Roberto Martín-Martín, and Silvio Savarese. Robot navigation in constrained pedestrian environments using reinforcement learning. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1140–1146. IEEE, 2021. 30

[79] Deepak Pathak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell. Curiosity-driven exploration by self-supervised prediction. In *International conference on machine learning*, pages 2778–2787. PMLR, 2017. 31

[80] Haobin Shi, Lin Shi, Meng Xu, and Kao-Shing Hwang. End-to-end navigation strategy with deep reinforcement learning for mobile robots. *IEEE Transactions on Industrial Informatics*, 16(4):2393–2402, 2019. 32

[81] Yuanda Wang, Haibo He, and Changyin Sun. Learning to navigate through complex dynamic environment with modular deep reinforcement learning. *IEEE Transactions on Games*, 10(4):400–412, 2018. 32

[82] Piotr Mirowski, Razvan Pascanu, Fabio Viola, Hubert Soyer, Andrew J Ballard, Andrea Banino, Misha Denil, Ross Goroshin, Laurent Sifre, Koray Kavukcuoglu, et al. Learning to navigate in complex environments. *arXiv preprint arXiv:1611.03673*, 2016. 32

[83] 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. 32

[84] Wenhan Luo, Peng Sun, Fangwei Zhong, Wei Liu, Tong Zhang, and Yizhou Wang. End-to-end active object tracking and its real-world deployment via reinforcement learning. *IEEE transactions on pattern analysis and machine intelligence*, 42(6):1317–1332, 2019. 33

[85] Friedrich Fraundorfer, Lionel Heng, Dominik Honegger, Gim Hee Lee, Lorenz Meier, Petri Tanskanen, and Marc Pollefeys. Vision-based autonomous mapping and exploration using a quadrotor mav. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4557–4564. IEEE, 2012. 40

[86] Jesus Tordesillas and Jonathan P How. Panther: Perception-aware trajectory planner in dynamic environments. *IEEE Access*, 10:22662–22677, 2022. 40

[87] Horaţiu Florea, Vlad-Cristian Miclea, and Sergiu Nedevschi. Wilduav: Monocular uav dataset for depth estimation tasks. In *2021 IEEE 17th International Conference on Intelligent Computer Communication and Processing (ICCP)*, pages 291–298. IEEE, 2021. 40, 50

[88] 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. 41

[89] 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

[90] Katarzyna Janocha and Wojciech Marian Czarnecki. On loss functions for deep neural networks in classification. *arXiv preprint arXiv:1702.05659*, 2017. 48

[91] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004. 49

[92] Jiajun Ou, Xiao Guo, Ming Zhu, and Wenjie Lou. Autonomous quadrotor obstacle avoidance based on dueling double deep recurrent q-learning with monocular vision. *Neurocomputing*, 441:300–310, 2021. 54, 96

[93] Minwoo Kim, Jongyun Kim, Minjae Jung, and Hyondong Oh. Towards monocular vision-based autonomous flight through deep reinforcement learning. *Expert Systems with Applications*, 198:116742, 2022. 54

[94] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018. 62

[95] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8:279–292, 1992. 66

[96] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013. 68

[97] 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. 71

[98] 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. 73

[99] Alessandro Devo, Jeffrey Mao, Gabriele Costante, and Giuseppe Loianno. Autonomous single-image drone exploration with deep reinforcement learning and mixed reality. *IEEE Robotics and Automation Letters*, 7(2):5031–5038, 2022. 96

[100] 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.

[101] Yunlong Song, Kexin Shi, Robert Penicka, and Davide Scaramuzza. Learning perception-aware agile flight in cluttered environments. *arXiv preprint arXiv:2210.01841*, 2022. 96

[102] Yue Ming, Xuyang Meng, Chunxiao Fan, and Hui Yu. Deep learning for monocular depth estimation: A review. *Neurocomputing*, 438:14–33, 2021. 100

[103] Martin L Puterman. Markov decision processes. *Handbooks in operations research and management science*, 2:331–434, 1990. 105