Faculty of Applied Sciences,
Department of Electrical Engineering
and Computer Science

Reliable Monocular Depth Estimation for Unmanned Aerial Vehicles

PhD Dissertation

Michaël Fonder

supervised by

Prof. Marc Van Droogenbroeck

May 2023

Jury Members

Prof. Bernard Boigelot, University of Liège, Belgium (President)

Dr. Anthony Cioppa, University of Liège, Belgium

Prof. Christophe De Vleeschouwer, Catholic University of Louvain, Belgium

Prof. Damien Ernst, University of Liège, Belgium

Dr. Silvio Giancola, KAUST University, Saudi Arabia

Prof. Matthia Sabatelli, University of Groningen, The Netherlands

Prof. Marc Van Droogenbroeck, University of Liège, Belgium (Supervisor)

Cover picture

Sunrise on a mountain range, Tasermiut Fjord, Greenland - © Michaël Fonder

Abstract

In recent years, unmanned aerial vehicles (UAVs), also known as drones, have become increasingly popular. In particular, small UAVs, such as quadcopters, have gained attention due to their agility and maneuverability, making them ideal for tasks such as aerial photography, surveying, and delivery. However, remote controlling such vehicles requires specific skills that are scarce and expensive to hire, which creates a demand for automated or assisted flight.

Planning a motion or a trajectory in an uncontrolled environment while avoiding undesired collisions can hardly be achieved without knowing the distance to potential obstacles in the way. However, the dedicated sensors typically used in robotics for estimating distances can hardly be used on small UAVs because of their size, weight, or power requirement. As drones are usually equipped with one or several cameras for various purposes, using the visual information to infer distances to objects in the environment emerged as a compelling, though challenging, option to replace these sensors.

Depth estimation, the process of determining the distance from a camera to an object in the environment, is a computer vision task that has varied applications. When used for autonomous piloting application, depth estimation methods need to fulfill a series of requirements that are not necessarily needed for other applications, and that were left unanswered has a whole. This dissertation aims at addressing reliable monocular depth estimation for UAVs.

Similar to many other computer vision fields, the state of the art in depth estimation has been led by methods based on deep learning, which require large datasets to be trained on. In the first part of this work, we note a lack of dataset suitable to train and test depth estimation methods for outdoor UAV applications, and introduce Mid-Air, a new multipurpose synthetic dataset of low altitude drone flights in unstructured environments. The ground-truth data available with this new dataset makes it useful not only for depth estimation, but also for various other computer vision tasks such as visual odometry, semantic segmentation, or even surface normal estimation.

In the other parts of this work, we address the challenge of depth estimation itself. First, we proceed to identify the weaknesses of existing depth estimation methods when considering a major requirement for UAV applications that is the ability to handle a wide range of environments, even unseen ones. We use this analysis to propose a new method, called M4Depth, that makes use of a notion of visual parallax, that we define, to avoid the weaknesses identified in other methods. Second, we consider the reliability requirement on depth estimation. To this extent, we investigate the use of uncertainty as a mean to anticipate erroneous data in the depth estimates. We then present a new method, called M4Depth+U, that upgrades M4Depth to jointly estimate depth and its uncertainty, and show that the obtained uncertainty is indeed representative of the error on the depth estimates.

Our tests on several datasets and in various conditions, including zero-shot cross-dataset transfer, show that our methods are robust to visual changes, and generalize better than existing methods, while being more computationally efficient. With these results, M4Depth+U emerges as an excellent and reliable joint depth and uncertainty estimator, and shows that it has the properties expected from a depth estimation method targeting UAV applications.

Acknowledgements

This PhD thesis has been quite a journey, sometimes challenging and difficult, but also enjoyable and rewarding at other moments. I would probably not have been able to bring this journey to its completion without the people who surrounded me during this period. I would like to take some time to thank them before diving into this dissertation.

First and foremost, I would like to give some credit to Marc Van Droogenbroeck thanks to whom I got introduced to the field of computer vision, and who invited me to extend my stay at the University after my Master's thesis. You have always been available for research-related help or advice when I needed it. You also fully supported me and gave me the liberty to pursue my research as I wanted. Thank you for that. Thank you, Bernard Boigelot, for accepting to join my jury as its president. I lost count of the hours we spent in your office talking about topics that were completely unrelated to the reason why I dropped in in the first place. These talks have always been interesting, and I will miss them after leaving Montefiore. It has also been a real pleasure to work with you as a teaching assistant.

Life is full of surprises. One I certainly didn't see coming was the opportunity to sail across the Atlantic Ocean to Greenland as part of my job for the Katabata project. I am immensely grateful to Damien Ernst, who carried this project, for trusting in me for its concretization. The adventure has been life-changing in many aspects, and was the starting point of many things that will certainly outlast my thesis. In addition, it offered me a break from the office at the time when I needed it the most. Thank you to Julien, Lucas, Thomas, Tobias, and Sophie, the crew of the Unu Mondo Expedition 2020, for making these three months aboard the Northabout unforgettable.

Work would not be enjoyable without nice colleagues, and fortunately, the Montefiore Institute is full of them. I first want to thank my closest colleagues, Adrien, Anaïs, Anthony, Marc, Renaud, and Sébastien for the many good moments and interesting talks we shared together. Thank you to Benoît, Benjamin, Gaspard, Matthia, Pascal, Simon, and Yann for these chats and debates that were often way beyond research, and sometimes deep and personal. Also, thank you to both the regular and occasional Rikiki card game players who contributed to nice and memorable lunch breaks. Finally, thank you to all the colleagues I could not mention here for brevity reasons; you all contributed positively to my stay at the Montefiore Institute, and I am grateful to you for that.

Outside of work, I also had the chance to be surrounded by amazing friends. Thank you to my friends from the Big Band for having been there all along, for your incredible support, and for all the invaluable moments we shared together. Thank you to Anne-Claire, Auriane, and all the other people of Choice for the deep discussions and reflections about life; they really helped me to move forward as a person. Lastly, thank you to all my successive flatmates for having made home a lively and enjoyable place to be. Special thanks go to Amaury, Jérôme, Jiyeon, Rashmi, and Romain for having made COVID lockdowns much more bearable than they could have been.

Finally, my deepest gratitude goes to my family. Thank you Maman and Papa for everything, especially for your unconditional love and support. You created a wonderful environment that allowed me to grow and become who I am now. The family would obviously not be the same without my siblings, Alice, Denis, and Martin, who happen to also be some of my closest friends. Thank you for being the wonderful people you are. Thank you to my four grandparents for also being particularly supportive and proud of my choices. Mamy and Daddy, thank you for having housed me for a few months between two apartments. It has been a great opportunity to get to know you better, and I have many good memories from my short stay with you. A last special thanks goes to Clara who joined me during the last stretch of my thesis. Your love and your presence by my side have been the best uplift I could hope for to wrap up this work.

Fundings

This work was partly supported by the Walloon region (Service Public de Wallonie Recherche, Belgium) under grant n°2010235 (ARIAC by DigitalWallonia4.ai)

Contents

Contents

Contents

Acronyms

AuSE	Area under the Sparsification Error
BNN	Bayesian Neural Network
CNN	Convolutional Neural Network
DoF	Degrees of Freedom
GNSS	Global navigation satellite system
GPU	Graphical Processing Unit
IMU	Inertial Measurement Unit
IR	Infrared
LiDAR	Light Detection and Ranging
LoD	Level of Detail
MAP	Maximum A Posteriori
MVD	Multi-View Depth
MVS	Multi-View Stereo
PSCV	Parallax-Sweeping Cost Volume
RMSE	Root-Mean-Square Error
SAE	Society of Automobile Engineers
SfM	Structure from Motion
SLAM	Simultaneous Localization and Mapping
SNCV	Spatial Neighborhood Cost Volume
ToF	Time of Flight
UAV	Unmanned Aerial Vehicle
VO	Visual Odometry

1 Introduction

Summary

Overview

What is depth estimation? Why do we need it for small unmanned aerial vehicles (UAVs)? Why focus specifically on small UAVs? What are the challenges? These are legitimate questions one may ask oneself when beginning the reading of this document. In this chapter, we answer these questions by explaining the context in which this thesis emerged and the motivations that led to our specific research topic, *i.e.,* reliable monocular depth estimation for small UAVs.
In addition, this chapter introduces the content of this thesis with a small summary of each of the following chapters, and it highlights the original contributions associated to these chapters.

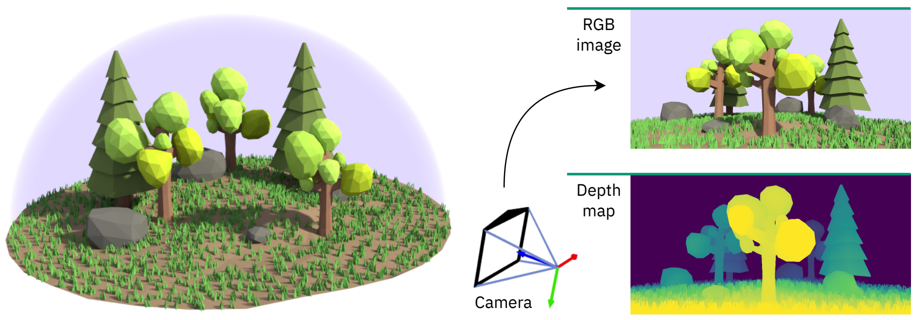


Figure 1.1: Illustration of the concept of depth estimation. Depth estimation is the task of inferring depth, a specific notion of distance separating the objects of the scene to a camera, for each pixel of an image captured by this camera. Brighter colors in the depth map correspond to smaller depth values.

1.1 Depth and unmanned aerial vehicles

Planning a motion or a trajectory in an uncontrolled environment while avoiding undesired collisions can hardly be achieved without knowing the distance to potential obstacles in the way. With the development of autonomous vehicles, the need for accurate and reliable distance estimation becomes increasingly important. In the nature, vision appeared to be particularly effective for this purpose. Similarly, in robotics, vision is one of the few options available to estimate distances.

Unmanned aerial vehicles (UAVs), also known as drones, have become increasingly popular in recent years due to their versatility and potential applications. Small UAVs, such as quadcopters, have gained attention due to their agility and maneuverability, making them ideal for tasks such as aerial photography, surveying, and delivery. However, their size, weight, and power constraints make it difficult to use dedicated distance sensors, which are often too heavy, bulky, or power-hungry for such small vehicles. As a result, there is a strong interest in using on-board mounted cameras as an alternative to these dedicated sensors.

Therefore, depth estimation, the process of determining the distance from a camera to an object in the environment, has become an attractive solution on small UAVs. However, achieving accurate depth estimation from a camera is a challenging problem due to the lack of information about absolute depth in the image. The development of deep learning algorithms has led to significant advances in this field, making it possible to estimate depth from images with reasonable accuracy. Despite these advancements, there are still many unanswered challenges, especially when considering UAV applications. These challenges include the ability to produce estimates that can be trusted, the ability to handle a wide range of environments, even unseen ones, the ability to handle a wide range of lighting conditions, and the need for efficient computational performance.

In this thesis, we identify the shortcomings of the state of the art and propose original contributions to address them. We first note a lack of suitable datasets to train deep neural networks for this task and propose, therefore, a new multi-modal dataset called Mid-Air to address this issue. We then propose a lightweight depth estimation method designed to work in all types of environments, even unseen ones, which shows robustness to visual changes and achieves state-of-the-art performance in several experiments. Finally, we address the requirement for reliability of the depth estimates by adapting our method to output its own uncertainty in addition to depth. Our experiments show that our method keeps its strengths while producing uncertainty estimates that are well correlated with the real error on the depth estimates.

In the following parts of this chapter, we first explain the background in which this work started. We then explain the motivations that led to the topic of monocular depth estimation for small UAVs. Next, we give an overview of the other chapters of this document and highlight the original contributions associated to them. Finally, we list the scientific publications linked to this thesis.

1.2 Background

The motivations underlying choices and hypotheses made in this thesis require introducing some concepts about UAVs, autonomous piloting, and deep learning. In this section, we introduce and discuss the key concepts on which this thesis is based.

1.2.1 The popularization of small UAVs

Small UAVs, also known as drones, are small-scale remote-controlled or autonomous flying vehicles. They are usually equipped with a variety of sensors that enable them to navigate and perform complex tasks with precision and accuracy. The emergence of drones as a popular and versatile technology can be traced back to the late 2000s when advancements in materials science, electronics, sensors, and navigation technologies made them increasingly accessible and affordable, leading to a wide range of new possible applications.

Industrial applications. Although the development of drones has been driven by military and surveillance applications, they have been found useful for several other applications, often revolving around aerial imagery, that call for different and specific solutions. For example, they efficiently replace human eyes for building and structure inspections [88] which otherwise require expensive means such as full-fledged helicopters, cranes, or professional climbers. This allows for a better and more frequent monitoring of infrastructures, and therefore for a better planning of maintenance requirements. When equipped with multi-spectral cameras, drones are also valuable assets for crop monitoring in agriculture [64]. Indeed, periodic imaging of the fields can be used to detect problematic areas, which allows taking appropriate and localized actions, therefore leading to a better use of resources such as water, fertilizers, and pesticides. Aside imaging-related tasks, drones are also considered for applications in logistic chains, where they can be used for remote and automated parcel deliveries [21].

On-board sensors

Drones are vehicles loaded with a lot of different sensors. While the exact type and number of sensors can vary depending on the targeted application, the vast majority of drones rely on a common set of sensors for their flight and mission. Hereafter, we list and explain the purpose of these sensors.

Attitude. Maintaining the stability of an airborne vehicle and imposing it a specific motion can only be achieved by controlling its pose in space, called its *attitude*. Controlling the attitude of an aircraft requires having feedback on its state. The sensors used in drones for this purpose are grouped within a block called the Inertial Measurement Unit (IMU). An IMU usually includes three different 3-axis sensors that are a gyroscope, an accelerometer, and a magnetometer. The gyroscope measures the angular velocity, the accelerometer the linear acceleration, and the magnetometer the magnetic field. When fused, the measurements of the three sensors can give an accurate estimation of the attitude of the vehicle.

While the IMU can be used alone to stabilize and control an aircraft, it is not reliable for long-term spatial localization. Indeed, the successive integration of errors due to sensor biases and noises leads to drifts in the estimates that are impossible to correct only with the information provided by the IMU.

Positioning. The absolute spatial position of a vehicle on Earth can be known thanks to Global Navigation Satellite Systems (GNSS) such as GPS, Galileo, GLONASS, and BeiDou. Each system has specific characteristics and requires its own dedicated receiver. However, integrated circuit manufacturers have designed single chips that embed receivers for multiple systems to take advantage of the satellites from multiple constellations, therefore increasing the accuracy of the localization estimate. As opposed to an IMU, the position given by a GNSS is absolute and does not suffer from error accumulation issues. Without using an external signal for correction, the positioning error of a GNSS receiver can be expected to be lower than 3m, and falls to 1m for more expensive devices or for static receivers. By default, GNSS receivers update their position once every second.

GNNS-based positioning is only possible when the receiver has a direct line of sight with at least four satellites, its accuracy increasing with the number of visible satellites. This is a major drawback since any masking of the sky, and therefore of visible satellites, degrades the accuracy of the positioning. Even worse, the positioning becomes impossible when there is no direct line of sight between the receiver and

Terminology: Image vs frame 1

Image is the term used to refer to the 2-D array of (grayscale or color) pixels captured by a camera. When images are captured at regular time intervals, they make a video stream. A frame specifically refers to one of the temporally ordered images making a video stream.

satellites. Hence, GNSS-based positioning is not possible, or heavily degraded, in places where the sky is largely obstructed such as in indoors, in forests, or in canyons. While natural canyons are really specific edge cases, urban canyons are much more frequent and are known to be challenging for GNSS-based positioning [18].

Cameras. Drones used for aerial imaging tasks are obviously equipped with the imaging sensor required for their task. However, cameras are useful for many other purposes than the primary targeted application. Indeed, the visual information provided by a camera can be used by the drone during flights. For instance, cameras can be used to compensate for the weaknesses of attitude and positioning sensors. Furthermore, visual information can be exploited to extract information that is simply not available with any other sensors and used for high-level flight control tasks such as object tracking or obstacle avoidance. As a result, drones are often equipped with one or several additional cameras whose sole purpose is to provide additional information to their flight controller.

Vision for precise motion and localization estimation

Despite their weaknesses, an IMU and a GNSS receiver can be used in conjunction with a camera to get a precise estimate of the motion and localization of the vehicle at a high frequency. Two types of algorithms, visual odometry (VO) and Simultaneous Localization and Mapping (SLAM), can be used for this purpose.

Visual odometry. Visual Odometry (VO) algorithms aim at recovering the path of the vehicle incrementally pose after pose, and at ensuring the local consistency of the estimated motion [4, 122]. Their principle is to detect and track several keypoints from a limited number of past frames recorded by the camera up to the latest one, and to optimize the camera pose estimated for each frame by using epipolar geometry. Because of their iterative nature and limited temporal memory, visual odometry algorithms can be lightweight enough to run in real time on small embedded computers.

VO algorithms can work with visual data alone. In this case, translations can only be estimated up to an unknown scaling factor because all the information about the absolute scale of the 3-D scene captured by a camera is lost when it gets projected into a 2-D image. Using IMU and GNSS measurements allows solving this ambiguity

Epipolar geometry 2

A point in space seen by a camera is projected at specific coordinates on the camera sensor plane that depends on the camera parameters and the position of the point with respect to the camera. The epipolar geometry is a set of geometric relations that describes the constraints linking the projection coordinates of a point in space seen by two cameras to the relative pose in space of these two cameras.

accurately, in addition to reducing the error on the estimated poses. However, the incremental nature of VO algorithms makes the localization estimate prone to drift over long time windows when corrections thanks to GNSS data are not possible.

SLAM. As for VO, the purpose of Simultaneous Localization and Mapping (SLAM) algorithms is to accurately estimate the path of a vehicle [2, 130]. However, the approach used to achieve this purpose is different. Instead of trying to optimize the pose of the camera with respect to a few past frames, SLAM algorithms localize themselves within a sparse 3-D point map they build. Each point of this map encodes some specific visual features that were detected and matched in several camera frames, and that were projected in the 3-D space using epipolar geometry. As opposed to VO that forgets features seen previously, features making the points map of a SLAM algorithm are kept in memory during the whole run of the algorithm.

When a new frame is recorded, feature points are extracted from the image and compared to features in the map. The camera pose is then optimized using the image features that match points of the map. SLAM algorithms keep some specific frames, called *keyframes*, in memory to build and update their map. A frame is selected as a keyframe if the corresponding camera pose meets a set of requirements with respect to the camera pose of the other keyframes. Each time a new keyframe is added in memory, both the point map and the estimated trajectory are updated using a process called *bundle adjustment* that jointly minimizes the error on the estimated trajectory and the error on the 3-D reprojection of observed feature points. The global map used in pair with bundle adjustment allows correcting drift when the camera closes a loop, *i.e.,* when it passes in places seen previously, as the algorithm should match the latest detected features to features already present in the map.

Being able to correct incremental drift is the key advantage of SLAM over VO. However, it is worth noting that SLAM algorithms lose this advantage when trajectories do not make loops, or when bundle adjustment cannot be made reliably in case of poor feature detection and matching. Moreover, the task of querying and updating the map gets quickly computationally expensive as the number of tracked points grows. Hence, maintaining real-time operations can only be achieved by limiting the number of points in the map, which requires a strategy to discard the less relevant points. While not being an issue for applications in well delimited places such as indoors, this requirement reduces the comparative interest of SLAM over VO in large environments where discarding points in the map could lead to a significantly degraded drift correction ability.

1.2.2 A demand for autonomous piloting

The interest for assisted and automated piloting exists for all types of vehicles. UAVs are no exception to this interest, and automated flight is already widely used for applications in the open sky, where the automation simply consists in following GNSS waypoints. However, automation is more complex when drones fly closer to the ground and must navigate around obstacles, as this requires dynamic trajectory adaptations. This is particularly relevant for tasks such as structure inspection, which requires a close and detailed analysis of structures from multiple perspectives. Similarly, automated tracking for video capture and the final leg of automated drone deliveries also require drones to operate among obstacles. Such low-altitude flights require an in-depth understanding of the environment and a precise trajectory planning to avoid collisions. In this regard, autonomous flight shares similarities with autonomous driving. As cars have existed for longer than small UAVs, research in autonomous driving got a head start over autonomous flight, which is beneficial for autonomous flight as some progress made for autonomous cars can be transferred to UAVs.

The first steps towards autonomous driving were achieved in the late 1980s, well before the emergence of small UAVs, with the first demonstrations of a car able to follow a road while relying solely on on-board sensors [9]. In a pioneering work of 1989, Pomerleau [111] proposed an autonomous car control algorithm based on neural networks. In the 2000s, the DARPA contributed to crystallize research in autonomous vehicles thanks to its Grand Challenges [20]. Finally, the research in this field exploded in the 2010s thanks to the rise of deep learning. In 2022, full driving automation remained to be achieved. Indeed, at this time, several car manufacturers were waiting for regulatory approval for level 3 technology on a scale defined by the Society of Automobile Engineers (SAE) that ranges from 0 (no driving assistance) to 5 (full driving automation)*, while some level 4 vehicles were tested and driving in specific and well-defined areas [69].

As research and experiments progressed, the task of fully automated driving revealed its whole complexity. This led researchers to split the global task into numerous distinct sub-tasks that have to be successfully addressed individually to tackle fully automated driving itself. These sub-tasks can be organized in three categories that can be transferred to all autonomous vehicles, including drones.

1. Low-level tasks. Safely piloting a vehicle is not possible without having accurate knowledge of its state and position with respect to its direct surroundings. This

*Details on this scale can be found in a survey from Halin *et al.* [47].

requires to gather and process measurements from multiple sensors, which is done by low-level tasks. For example, the pose and motion of the vehicle can be estimated thanks to odometry or SLAM algorithms that make use of cameras, IMU sensors, and GNSS receivers. Autonomous vehicles also have to be equipped with a mix of distance sensors that allow them to gather spatial information on their surroundings. Correctly processing the readings of these sensors is crucial since they are parts of the collision warning system.

2. Intermediate tasks. Taking piloting decisions requires having an in-depth understanding of the environment in which the vehicle moves. The purpose of intermediate tasks is to extract high-level information that can be used to take piloting decisions from the sensors of the car. Most of them are vision related and involve image analysis. For example, autonomous cars have to follow traffic regulations, which implies being able to reliably detect and recognize road signs. This calls for object detection and classification algorithms. Similarly, all other road users have to be detected, recognized and precisely located. Ideally, their motion should also be estimated to allow for anticipation in higher level tasks. This in turn calls for instance and semantic segmentation methods, as well as for tracking algorithms.

3. High-level tasks. Finally, high-level tasks are the ones in charge of aggregating all the available information in order to take piloting actions. This implies developing control strategies that ensure the safety of the vehicle when moving towards the desired estimation. For high-level tasks to be successful, lower-level ones need to be addressed properly and reliably, as their performance could directly impact decisions made by high level-tasks.

1.2.3 The rise of deep learning methods

The progress made in autonomous vehicles has been tightly linked to the one made in deep learning. If the theoretical basis for neural networks was already laid in the 1980s, they only gained traction in the 2010s when training and inference speeds became several order of magnitude faster thanks to the use of Graphical Processing Units (GPUs) in computers. The arrival of AlexNet [70], a Convolutional Neural Network (CNN) for image classification, in 2012 triggered an unprecedented interest in deep learning for computer vision tasks. The release of large datasets and the advances in optimization techniques have led deep learning methods to outperform other methods for many computer vision tasks, such as image classification, object detection, semantic segmentation, and image captioning, just to name a few. More generally, deep learning emerged as a powerful and versatile tool as it has also led to breakthroughs in many other fields, such as natural language processing, speech recognition, and game playing.

Deep learning methods are a class of artificial neural networks that are capable of learning from large amounts of data to perform various tasks, including computer vision. These methods consist of multiple layers of neurons, where each neuron performs a nonlinear transformation of its inputs. The output of one layer serves as the input to the next layer, and so on, until the final output is produced. The key feature of deep learning methods is their ability to automatically learn hierarchical representations of data, which allows them to capture complex patterns and relationships in the data.

In practice, a deep neural network is defined by its architecture, a set of parameters connected in a specific way. The parameters are usually initialized with random values, and need to be tuned to get the desired output from a given input during an iterative process referred to as the network training. At each step of this process, an input is sampled from a dataset that is expected to represent the task the network has to perform. This input is used to generate an output based on the value of the network parameters at this step. A loss value that reflects the quality of the generated output is then computed, a high value reflecting a poor output quality. For example, the loss value can simply be the error between the output produced by the network and the expected output when input-output pairs are available in the training dataset. The network parameters are updated based on the loss value using a process called *backpropagation*. This process involves computing the gradient of the loss value with respect to the model parameters, and then updating the parameters in the direction that reduces the loss value.

Computing the loss value by using the expected output is called *supervised learning* and is not always possible because pairs of input-output training samples are sometimes not available. However, obtaining a relevant feedback to train the network remains possible without having access to the expected output values. This is what is done in semi-supervised, unsupervised and reinforcement learning, for example.

A dependence on large collections of data

One of the main advantages of deep neural networks over other methods is their ability to learn and generalize from large amounts of data. Unlike traditional machine learning algorithms that rely on hand-crafted features, deep learning models can automatically learn features and representations directly from the raw data, leading to improved performance. However, one of the shortcomings of deep learning methods is their vulnerability to out-of-distribution data. Since the network is trained on a specific dataset, it may not generalize well to examples that differ from the training data. This can lead to inaccurate predictions or even complete failure of the method. For example, a network trained on daytime images of pedestrians may not perform well on nighttime images or images with heavy rain or fog.

Avoiding this issue requires training the network on a collection of data that is diverse enough to faithfully represent the full range of inputs that the method could encounter for its targeted application. Therefore, the years 2010 have seen a surge in the number of available datasets targeting various applications in parallel to the development of deep learning methods. Many large datasets dedicated to autonomous driving tasks have been released during this decade [61], while only a few datasets dedicated to UAVs were released during the same period. Each dataset targets a well-defined subset of tasks by providing the annotated ground-truth data required for the training of neural networks [74]. The parameters differentiating datasets targeting the same tasks can be the location, the weather or the road condition where the data is recorded as well as the type and placement of sensors used to record the data [61, 74, 79, 84].

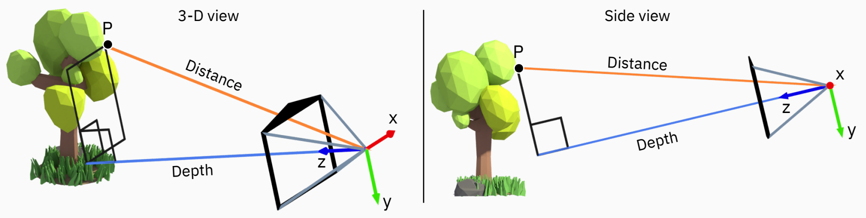
In addition to large datasets, researchers have proposed various techniques such as data augmentation, domain adaptation, and transfer learning to overcome a lack of training data for the targeted task. Data augmentation involves artificially increasing the size of the training set by applying transformations to the existing data, such as rotations, translations, and scaling for images. Domain adaptation and transfer learning involve using knowledge from a related but different dataset to improve the performance on the target dataset.

For some applications, it is impossible to collect data that faithfully represent the full range of inputs that the method could encounter. This is, for example, the case of computer vision tasks with drones because they can see a variety of scenes that would be hard to fully capture. In such case, methods should be designed to be robust and perform well on unseen data that may have a different distribution than the training data. Methods can be tested on purpose on data that have a different distribution than the training set. This is called *zero-shot cross-dataset transfer,* and it allows grasping the behavior of a method when exposed to out-of-distribution data.

1.3 Motivations

As for all autonomous vehicles, flight automation requires addressing low-level tasks before starting to think about higher level automation. Accurate distance estimation is especially important for drones because they can move in all directions and do not need to follow a predetermined path. Unlike road vehicles, drones do not have to detect and follow visual instructions such as road signs, which can limit the possible trajectories. Moreover, drones have more freedom of motion, making interaction with other vehicles easier than road vehicles which must share their space with many other users. Therefore, distance becomes one of the most critical pieces of information required for planning flight trajectories and avoiding obstacles. However, gathering information on distance to surrounding objects is more complex for small UAVs than for other vehicles, and was still an open challenge when we began this thesis. As awareness to distance to the surroundings is a low-level task, it should be solved before addressing higher-level tasks. This is why we decided to address this challenge in this thesis.

Terminology: Distance vs depth 3



Depth is a notion of distance used in computer vision. As illustrated in the figure hereinabove, the distance separating a point P in space to a camera is the Euclidean distance between this point and the camera origin, which is defined as its focal point. The depth of the point P is the z coordinate of its position when it is expressed within the camera referential. With this definition, the distance of a point to a camera is greater or equal to its depth. Computer vision methods often prefer to work with depths instead of distances, as it allows for a simplification of geometrical operations involving camera projections.

In this section, we first explain why distance sensors such as the ones used in other autonomous vehicles are not adapted for small UAVs. We then detail how vision provides multiple cues about distance and how cameras can be a relevant alternative to dedicated distance sensors. We then list the properties that a depth estimation method targeting small UAV application should have. Finally, we cover works that address the task of estimating distance, or depth in computer vision, with a single camera. We present them in three categories, and highlight their limitations when considering the task of depth estimation for autonomous flight applications.

1.3.1 Physical constraints of UAVs vs distance sensors

Ground autonomous vehicles are often loaded with a significant set of various distance sensors. While some sensors such as ultrasonic distance sensors or radars can only give the distance to the closest object within a given detection cone, only the ones based on light time of flight are able to create accurate 3-D maps of their environment. These sensors, which consist of an infrared (IR) light emitter and an IR sensitive receiver, derive the distance to an object from the time taken by a modulated light pulse to make the round-trip to the sensor after having bounced on the object [39]. The two existing methods for scanning distances on multiple dimensions, illustrated in Fig. 1.2, are time-of-flight (ToF) cameras [48] and LiDARs [92]. Since the former do not have any moving parts, they are more compact, lighter and cheaper to manufacture compared to the latter. However, ToF cameras have a significantly lower maximum range than LiDARs because their IR signal quickly fades with distance, as opposed to the laser-based technology.

One of the differences between ground vehicles and aerial vehicles is the trade-

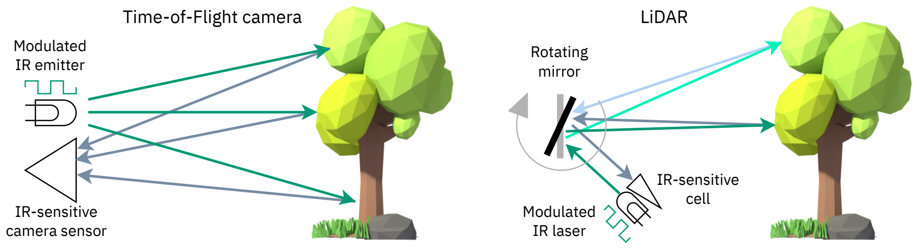


Figure 1.2: Comparative illustration of ToF cameras and LiDARs working principles. Both sensors derive the distance to objects in the environment from the time of flight of an infrared (IR) light pulse. A ToF camera floods the scene with a single modulated IR pulse, and an IR sensitive camera sensor detects the time of arrival of the bounced IR rays for each pixel of its sensor. On the other hand, a LiDAR projects a laser IR pulse on a moving mirror and waits for the light ray to bounce back to the mirror and hit the IR-sensitive cell. A LiDAR sensor scans the scene by taking several measurements with different mirror orientations, which creates rolling shutter artifacts that ToF cameras do not have.

off between size and weight. Unlike ground vehicles, any additional weight to the payload of an aircraft requires adjusting its aerodynamics by increasing its lifting area and size. Furthermore, adding weight to an aircraft significantly increases its energy consumption, which ultimately reduces its flight autonomy for a given energy input.

For drones used in low-altitude applications, size is a critical constraint. They must be small enough to access all the necessary locations for the intended purpose. Furthermore, major regulatory bodies have imposed rules and regulations that vary depending on the weight of the vehicle, with lighter drones having less strict regulations applicable to them. As a result, this has created a general incentive for drone manufacturers to reduce the size and weight of their products.

Despite constant improvements, LiDAR units are heavy, bulky, and expensive when compared to drones. In 2023, available 3-D LiDARs still weight more than 500gr while costing at least 1,000 USD [52, 91]. In comparison, the upper weight limit for the most permissive category drones in the European Union is set to 250gr, and prosumer drones can be found for less than 2,000 USD. Equipping a drone with just one LiDAR unit significantly reduces the margin for an additional payload while steeply increasing the price of the vehicle. ToF cameras are sometimes used as an alternative to LiDARs on autonomous vehicles for collision avoidance. However, their limited measuring range (10 meters for the best ones) makes them poorly suited for trajectory planning.

In conclusion, sensors that would typically be used for distance sensing on ground autonomous vehicles are not adapted for use on small UAVs when costs and weight need to be limited. Therefore, alternatives to these sensors need to be found before considering to addressing tasks of higher level for autonomous flight with these vehicles.

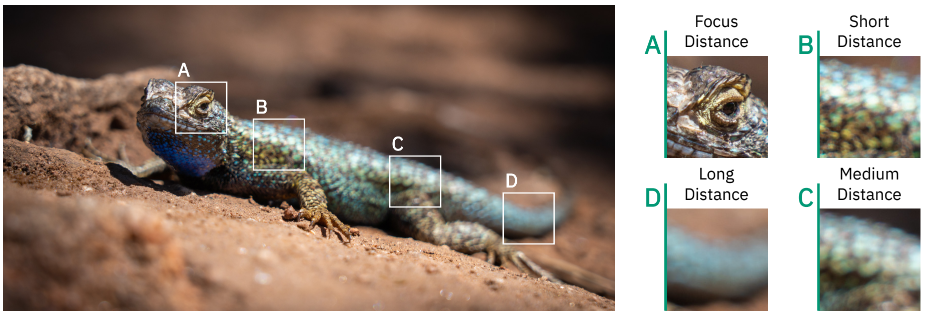


Figure 1.3: Illustration on how defocus can be used to infer depth. The further away a point is from the focus distance, the higher is the defocus blur.

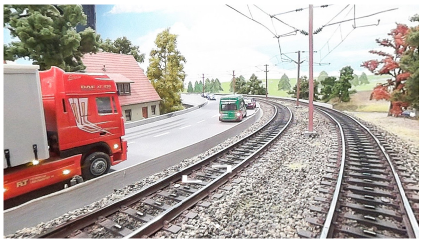
1.3.2 Depth perception from vision

When analyzing the context in which this thesis matured (see Section 1.2), we mentioned that most UAVs are equipped with at least one camera and that visual data can be used to extract different types of high-level information, including depth. Indeed, visual data conveys information about depth in three distinct ways that are the defocus, the visual cues and prior knowledge of the observed scene, and the multiple observations of the scene from different points of view. Hereafter, we explain how each source can be used to infer depth along with its advantages and drawbacks before discussing their potential interest for small UAVs.

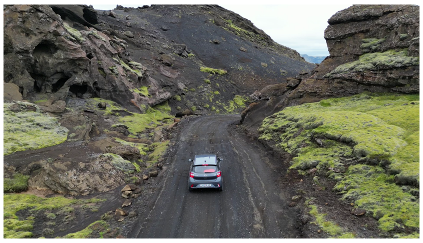
Depth from defocus

Defocus is a visual artifact in images captured by a camera that is due to the camera lens and consists of objects appearing blurred because of their distance to the lens. The purpose of the lens is to make the light rays coming from objects in the scene converge on the camera sensor. However, the physics of light and optics is such that perfect convergence can only be achieved for objects located at a single controllable distance of the lens called the *focus distance*. Static objects appear blurred in an image when the rays of light they emit or reflect do not converge perfectly on the sensor of the camera because they are not at the focus distance of the lens. The defocus effect is stronger for objects that are farther away from the focus distance, resulting in a depth effect that varies depending on the distance of the objects to the camera, as illustrated in Fig. 1.3. With everything else being equal, the magnitude of this effect increases with three parameters of the camera setup that are the focal length of the lens, the aperture of the camera, and the size of the camera sensor.

Depth estimation from defocus works by analyzing the defocus effect in a single image or in a series of images taken from a single point of view with different focus or aperture settings [86, 89, 132]. Single image depth from defocus works by analyzing the defocus pattern created by a lens whose aperture has been modified to produce



(a) A well-structured environment. © Google Street View



(b) An unstructured environment. © Michaël Fonder

Figure 1.4: Estimating the relative depth of objects in a scene from a single image relies on multiple visual cues and prior knowledge of the elements making the scene. Some environments, such as well-structured ones (a), provide strong visual cues about the relative depth of objects making the scene. Others, such as unstructured ones (b), display little usable clues, which makes the task of estimating depth much more complicated.

a specific blur pattern, which is referred to as a coded aperture lens [98, 103, 106, 132]. Methods working with multiple images compare the defocus in the different images to infer depth [32, 149]. For all methods, lower depth of field, *i.e.,* stronger defocus effect, allows for better depth discrimination. However, this comes at the cost of degraded image quality, as a strong blur leads to a significant loss of image frequencies.

Advantage. Since the magnitude of the defocus effect is directly related to the joint properties of the camera lens and sensor, this method is able to estimate the absolute depth of objects in the scene without requiring any external sensor or information.

Drawbacks. Depth from defocus has multiple drawbacks. First, multi-image methods require a perfectly still camera and scene, while methods based on a coded aperture lens require a modified camera. Second, depth from defocus does not work on areas with uniform colors or low-contrasted textures, as defocus is not measurable in these areas. Finally, since this method requires a measurable defocus effect to work, the camera setup requires a large sensor, a large aperture or a long focal length. As larger sensors are more expensive to manufacture and require more complex optical lenses, the ideal hardware required for depth from defocus is heavy and expensive. Additionally, a long focal length implies a narrow field of view, which reduces the area of the scene seen by the camera.

Visual cues and prior knowledge

The visual information in a single image can be rich in cues about the 3-D structure of the scene. They can be as varied as texture gradients, perspective effects, occlusions, shadows, or the relative 2-D projected size of objects of known 3-D size [14]. When detected and interpreted appropriately, these cues can be used to estimate the relative depth of the elements of the scene. However, the correct detection and interpretation of these cues is not possible without some level of prior knowledge of the elements making the scene, as they are often contextual.

Roads and urban environments are places where estimating the distance of an object through this method works well because a lot of man-made objects and buildings have purposes, standard sizes, and well-defined shapes. For example, one can easily infer the 3-D position of the objects making the scene captured in Fig. 1.4a by using knowledge about buildings, road vehicles, and trains. For instance, it can be assumed that the truck is closer to the camera than the car preceding it on the road because of their relative size in the image. Similarly, train tracks are parallel. The fact that they appear closer to each other towards the center of the image means that they are further away from the camera in the center of the image than at the bottom edge of the image.

Advantages. The main advantage of this method is that it requires only a single image of the scene to infer depth. Hence, it has no issue with dynamic objects in the scene. Furthermore, this method does not require specific camera properties nor a specific camera setup.

Drawbacks. Estimating depth from a single image comes with two major drawbacks. First, it fails when large areas of the image feature little usable visual cues or when the scene is made of objects of undefined structure and shape. This is, for example, often the case in natural environments, such as illustrated in Fig. 1.4b, where terrain surface can have unpredictable shapes, and where perspectives can lead to misleading assumptions. Second, this method is not able to reliably estimate the scale of the scene. This is because scale information is lost when projecting the 3-D scene on the 2-D camera sensor [49]. Figure 1.4a illustrates perfectly this issue. It was captured in a miniature city[†], and the red truck in this image is a few centimeters rather than meters away from the camera as most would assume just by looking at the image.

Multiple points of view

When observing a scene through a camera lens, objects are projected onto coordinates of the camera sensor that depend on their location in the scene with respect to the camera. If the same scene is captured from multiple points of view, the objects are projected at different coordinates in the images. This displacement between projection coordinates of the same spatial point in images captured from different points of view is known as the stereo effect. As the distance between the projection coordinates is proportional to the distance of the object to the camera, the stereo effect provides valuable information on the 3-D structure of the scene.

[†]Miniatur Wunderland in Hamburg, Germany.

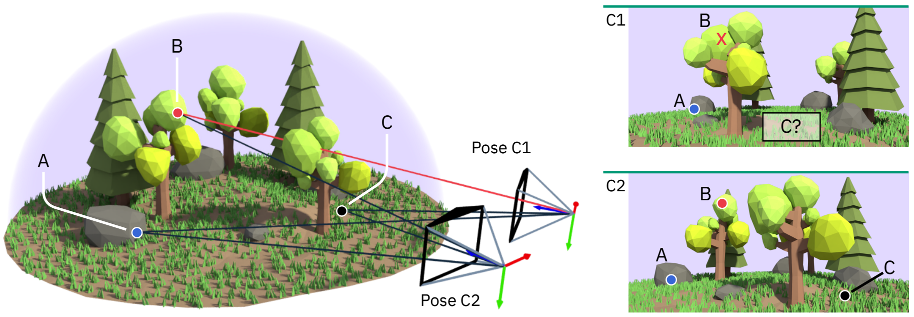


Figure 1.5: Illustration of the concept of depth from multiple points of view. When a same scene is observed from multiple distinct camera poses, it is possible to triangulate the pose of points seen by two cameras from their projected coordinates in the image (A). However, some parts of the scene can be occluded, hidden, to some cameras, which prevents their triangulation (B). This method can also struggle in areas with no or repetitive textures as it makes the image to image point matching, and therefore the triangulation, unreliable (C).

To estimate the depth of the observed point from its projection coordinates in different images, triangulation can be used if the camera poses for the different points of view are known [49]. In this case, the process of estimating the depth from multiple points of view can be divided into two steps. The first step is finding and matching pixels that correspond to the same point in space in the images taken from different points of view. The second step is estimating the depth of this point using an optimization algorithm to perform triangulation.

Advantages. The point matching and triangulation aspects of depth estimation from multiple points of view do not suffer from the drawbacks of single image depth estimation. Indeed, it does no call for prior knowledge of the content or the structure of the scene to work. Additionally, this method can be used with any camera, and does not require an active control on their parameters as opposed to depth from defocus.

Drawbacks. Estimating depth from multiple points of view requires the relative camera pose to be known, which is a drawback because the precision of the depth estimation directly depends on the precision of the relative camera pose estimates. Additionally, as this method relies on image to image point matching, any imprecision in the coordinates of the matched points directly impacts the precision of the estimated depth. This is an issue for two reasons illustrated in Fig. 1.5. First, some areas of the images can be ambiguous for point matching. This is, for example, the case for areas with uniform colors, or repetitive patterns. Second, difference in points of view can lead some points to not being visible in both images because of occlusions. In this case, point matching, and therefore, triangulation is impossible.

Several cameras vs camera motion. Capturing images from different points of view of the scene can either be done by several cameras simultaneously, or by moving a single camera over time. Each method has its own advantages and drawbacks.

Capturing images with several cameras simultaneously is advantageous for two reasons. First, assuming the cameras are rigidly attached to each other, their relative pose can be precisely measured and calibrated, which minimizes depth estimation imprecision due to relative camera pose imprecision. Second, capturing all the images at the same time means that all moving objects in the scene are observed in the same pose, which allows performing triangulation even on moving objects. However, the maximum depth that can be inferred with such a setup increases with the resolution of the camera sensor and with the spacing between the cameras, referred to as the baseline, which is a drawback for long-range depth estimation as the resulting baseline or sensor resolution requirements can be inappropriate for the targeted application.

As opposed to multi-camera setups, long-range depth estimation is not an issue with depth estimation from camera motion, as there is no physical constraint limiting the distance between two temporally spaced camera poses. While this method is perfectly appropriate for any static scene, the temporal spacing of the images makes it unreliable for dynamic objects as triangulation assumes the observed point to be at the same spatial location for all points of view. Additionally, the accuracy of depth estimation from camera motion directly depends on the accuracy of the method used to infer the relative camera poses.

Small UAV constraints vs depth from vision

As discussed hereinabove, there are multiple ways to infer depth information from visual information, with each method having its own advantages and drawbacks. However, given advantages and drawbacks can have an importance that depends on the considered application. In the following, we compare the three methods regarding their potential use for depth estimation in small UAVs.

Depth from defocus is a poor candidate for this task. The cameras used for vehicular sensing typically have a small sensor, a limited aperture, and a wide field of view, which implies a short focal length. This is the complete opposite of the ideal camera for depth from defocus. Hence, the properties of these cameras are the worst for this task. Finally, blurred areas have fewer details, which is a loss of image quality. Although methods exist to partially restore details in areas blurred by a coded aperture lens [86, 132], starting with a reduced signal quality is not ideal for computer vision tasks in general.

Estimating depth from a single image by using structure and prior knowledge and estimating depth from multiple points of view is more appropriate for autonomous vehicle applications since hardware requirements and practical constraints are less limiting. The depth cues given by structure, prior knowledge, and multiple points of view can even be used jointly to benefit from their respective advantages.

Acquiring the images from multiple points of view simultaneously from several cameras has fewer drawbacks than acquiring them over time from a single camera.

However, the spacing of multiple cameras on board a small UAV is limited by the size of the vehicle and therefore limits the maximum reliable depth estimation range. This limitation, which does not apply when capturing the images by using a single moving camera, reduces the comparative interest of multi-camera setups for small UAVs.

1.3.3 Requirements of monocular depth estimation for UAVs

In the previous section, we explain that the visual information given by a single camera could be used to replace dedicated distance sensors by using depth estimation methods. To be useful for drone applications, a monocular depth estimation method should feature several important properties listed hereafter.

Generalizability. Thanks to their ability to fly, drones are able to access a wide variety of places and environments. The depth estimation method should therefore be able to work reliably in all types of environment, even unexpected ones.

Robustness. The depth estimation method is expected to work in a wide variety of conditions. The performance of depth estimation should be invariant to changes in scene lighting and in the visual properties of the environment.

Causality. The purpose of depth estimation being to replace distance sensors for real-time trajectory planing and adaptation, the depth estimation method should provide a depth estimate for the latest frame recorded by the camera.

Computational efficiency. The size and weight constraints of small UAVs prevent the use of power-hungry and heavy computational devices. Therefore, depth estimation methods designed for use in small UAVs should have limited needs in terms of computational resources.

1.3.4 Related works

As just discussed, we are looking for a monocular depth estimation method for UAV application which implies to meet several challenging requirements. Depth estimation from a single camera is a computer vision task that is already well established, and that has experienced a fast-paced evolution since the arrival of convolutional neural networks (CNNs). In this section, we give an overview of the related works, and analyze how they relate to the requirements we need.

Depth estimation from a single image

While the first attempt at estimating depth from a single picture involved a mix of handcrafted properties for the structure of depth and machine learning for the inference of the 3-D information [118], researchers moved to deep learning approaches when CNNs showed their efficiency for solving this task with the works of Eigen *et al.* [24] and Liu *et al.* [77]. The works that followed are well covered by multiple surveys such as the ones of Zhao *et al.* [160], Xiaogang *et al.* [145], Ming *et al.* [97], and Masoumian *et al.* [87].

While the first methods were supervised, Garg *et al.* [35] proposed a semi-supervised method based on a photometric loss on stereo images to overcome the lack of large training datasets. Godard *et al.* [42] improved on this concept by adding a left-right consistency term to the loss to get better results. Their method, called Monodepth, became a significant milestone in the field, and led most following works to adopt a similar semi-supervised training loss. Later, this semi-supervised approach was adapted to work with monocular image sequences instead of stereo images [40, 80, 83, 107, 135].

The surveys [87, 97, 145, 160] show that the state of the art has been held by CNN-based method for a long time. However, the advent of methods based on vision transformers, such as DPT [113] and AdaBins [27], brought a breakthrough in the field in the early 2020s and pushed the performance for single image depth estimation to new levels. These methods were the state of the art in the field in 2022.

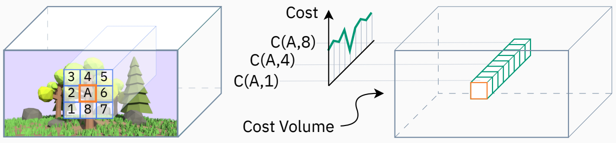
Single depth estimation methods are usually benchmarked against a mix of indoor and outdoor datasets. The most commonly seen datasets for this purpose are the NYU-v2 [124] and the SUN3D [144] indoor datasets. In outdoors, methods are tested on the KITTI [38] dataset that targets autonomous driving applications.

Depth estimation from an image sequence

As explained in Section 1.3.2, most of the shortcomings encountered with monocular image depth estimation can be alleviated by using temporal information. A simple solution that emerged consists of adding time recurrence to specific layers of standard single image depth estimation methods [71, 107, 135, 143, 157]. However, this solution is not optimal as the known geometric constraints that link successive frames are not used explicitly. Watson *et al.* [139] addressed this issue by proposing a method that estimates the successive camera poses of the sequence to build a cost volume with the plane-sweeping method [16, 34]. Xing *et al.* [146] avoided the need for explicit motion modeling by using the planar parallax geometry [60, 117]. The latest method, however, consists of a complex pipeline that explicitly relies on structure in the environment, which makes it unusable in environments where there is no structure to use.

Other methods, such as the one of Luo *et al.* [81], use motion-induced constraints to fine-tune the network at test time to improve the estimates. This is made possible thanks to a self-supervised loss based on motion estimation. These methods achieve outstanding performance, but at the cost of a large computational burden. Furthermore, their design prevents them to estimating depth before the whole

Cost volume 4



Cost volumes are pieces of neural networks architecture used in computer vision methods that have to perform image-to-image point matching for tasks such as optical flow or stereo matching. They provide a measure of similarity or cost between different image patches, each of their elements representing the cost of matching a given pair of patches. For example, computing the cost of matching a given image patch with several candidates from another image allows finding the best matching candidate in this other image. Cost volumes can differ between architectures by the method used to sample the candidates to match and by the cost function used to make the comparison.
In the illustration hereinabove, the cost volume is used to compare the patches making a picture to their neighbors in the same picture. At the spatial position corresponding to a patch A in the image, the cost volume gives the vector of costs $C(A, i)$ for matching the patch A to its neighbor i. In this example, the neighbor that is the most similar to the patch A is the patch 4 because the matching cost is the lowest.

image sequence is available, meaning that it only operates in an offline mode and makes it inappropriate for autonomous vehicle applications.

Since not all benchmark provide images sequences, depth estimation methods that work with image sequences are benchmarked against a sightly different selection of datasets when compared to single image depth estimation methods. The default benchmark used for these methods is the KITTI [38] dataset. In addition, methods are sometimes tested on some indoor datasets such as SUN3D [144], TUM RGB-D [127], or ScanNet [19] for example.

3-D reconstruction from an unordered set of images

Structure from motion (SfM) and multi-view stereo (MVS) are two research fields that have developed in parallel with depth estimation. The idea is to reconstruct 3-D shapes from a set of RGB images that capture the scene from different points of view under specific hypothesis (MVS requires camera poses to be known, SfM does not). Reconstruction is achieved by explicitly expressing the relative camera position between the images of the set. Surveys, such as the ones of Huang *et al.* [57] and Özyeşil *et al.* [105], show that approaches for performing this task are varied and are, by their nature, often unsuitable for real-time depth estimation. However, some methods, such as the ones of Gu *et al.* [45], Ummenhofer *et al.* [133], and Yao *et al.* [153], are adaptable for depth estimation on sequences.

Other approaches, such as the ones proposed by Düzçeker *et al.* [23] and the method called DeepV2D by Teed and Deng [131], are specifically designed to work on image sequences in real time. These approaches are similar and both propose a three-stage network. Their stages are an image-encoding network followed by the computation of a cost volume that is finally processed by a depth estimation network. The cost volume of both methods is built by a plane-sweeping method [16, 34].

SFM and MVS methods are sometimes benchmarked against the same datasets as the ones used for depth estimation from an image sequence. However, these fields also have their own dedicated datasets consisting of unordered set of images for 3-D reconstruction. In this category, the "Tanks and Temples" [67] and the "DTU benchmark" [1] datasets are the most commonly used for comparing the performance of different methods.

Limitations of existing methods

While showing interesting properties, existing methods also have some limitations when considering autonomous vehicle applications. Hereafter, we develop the three main limitations identified for existing monocular depth estimation methods.

Transferability to UAVs. As explained in the previous section, existing depth estimation methods are only tested on benchmarks targeting autonomous driving applications or indoor depth estimation. With these benchmarks featuring mostly structured environments and limited perspective variations, it is unsure how their performance transfers to data captured by a drone flying with 6 degrees of freedom, especially for flights in unstructured environments.

Generalizability. The surveys covering single image depth estimation methods [87, 97, 145, 160] observe that estimating depth from a single image remains difficult, especially for autonomous vehicle applications. Since the problem is ill-posed, networks have to heavily rely on priors to compute a suitable proposal. Such dependency on priors leads to a lack of robustness and generalization. Therefore, methods of this family need to be fine-tuned for every new scenario or environment encountered in order to produce good estimates. Despite their massive parameter count, transformers are no exception to these observations.

Scale estimation. Reconstructing 3-D scenes only from 2-D images can be done up to a projective ambiguity [49]. As a result, methods estimating depth only from visual information are unable to estimate the proper scale for depth without relying on any prior knowledge about the structure of the scene. This not only means that the scale estimated for the outputs may be arbitrary, but also that it is likely to drift over the sequence, which is problematic for autonomous vehicles.

Cost volumes. As highlighted by Schröppel *et al.* [120], the multi-frame methods that rely on plane-sweeping cost volumes share a common shortcoming. They build

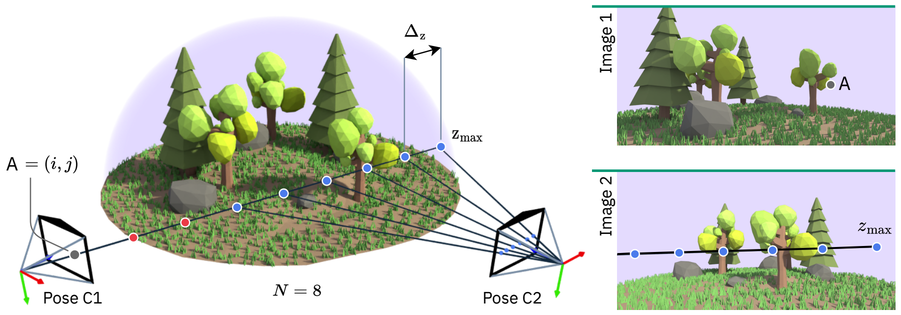


Figure 1.6: A plane-sweeping cost volume gives the cost of matching the pixels of two different images captured from distinct camera poses. Each cost vector of length N at coordinates (i, j) of the cost volume gives the cost of matching the image patch A at the same location (i, j) in one of the two images (Image 1 in our example) with N pixels of the other image. The pixels of Image 2 used for computing the cost values (in blue in the illustration) correspond to the projection coordinates of the point corresponding to the patch A reprojected in space at regularly spaced depth intervals Δ_z along the light ray, up to a maximum depth z_{max}. Note that some points may be projected outside of the other image (red points in the illustration) depending on the relative camera pose and the plane-sweeping parameters, and that a default cost value has to be used for these points.

their cost volumes from depth or disparity intervals, which requires two arbitrary parameters illustrated in Figure 1.6: (1) the maximum range, and (2) the quantization step along this range. The maximum range has to be known at the time of training or fine-tuning, which prevents dynamic adaptations to new depth distributions. The quantization step, that is the range divided by the number of samples along the range, is an important parameter to determine the performance of the network. A large quantization step degrades the depth resolution, hence the performance of a network, while taking a smaller quantization step will increase the inference time without any guarantee of improving the final result.

1.4 Thesis outline and original contributions

In the next parts of this work, we develop the research carried to propose answers to the task of reliable depth estimation from a monocular RGB camera within the context of autonomous UAVs. The three following chapters are each articulated around the work presented in a specific scientific publication. Therefore, the content of the chapters is, for the most part, based on the content of the related publication.

It should be noted that, despite primarily targeting small UAV applications, our work can be used for depth estimation on any device that features a camera, an IMU, and sufficient computational power. Indeed, drones have a lot of constraints that other devices may not have. As a result, transferring our work to less constrained devices is not a problem, whereas the opposite is not true. Besides, we had the will to make our work as useful to the community and as reproducible as possible. This is why we open-sourced most of our work. We give the public link to the data related to the presented research at the beginning of the corresponding chapter. Hereafter, we summarize the content of each chapter and their respective contributions.

Chapter 2 gives an overview of datasets that were available in 2019 and suitable to train deep learning methods for depth estimation for autonomous drones in outdoors. As we noted a clear lack of appropriate datasets, we introduced Mid-Air, a new multipurpose synthetic dataset of low altitude drone flights in unstructured environments. It features synchronized data of multiple sensors for a total of 54 trajectories and more than 420k video frames simulated in various climate conditions. With the intention of making this dataset as useful as possible, we provide data to enable the training of deep neural networks for various tasks such as visual odometry, semantic segmentation, surface normal estimation, etc. In Chapter 2, we motivate the design choices made for this dataset, explain how the data was simulated, and detail its content. We then propose a benchmark for positioning and a benchmark for image generation tasks, and show how Mid-Air can be used to set up a standard evaluation method for assessing computer vision algorithms in terms of robustness and generalization. Finally, we showcase the benefits of Mid-Air by pointing out a few works that made active use of it.

Contributions 1

- We propose the first multi-modal dataset for deep learning dedicated to UAVs flying in unstructured environments;
- Our dataset features ground-truth data, such as surface normals, not present in other major datasets;
- We propose a benchmark to test the robustness of computer vision methods to visual changes;
- Mid-Air has proven its value to the community as various original works made active use of it.

Chapter 3 addresses the challenge of estimating depth in unstructured environments from RGB images captured by a camera that has a motion with six degrees of freedom. We first formalize this task and introduce the metrics used to evaluate the performance of a depth estimation method. We then propose a new method, called M4Depth, that is designed to address the shortcomings observed in existing methods when considering depth estimation in outdoors and in unstructured environments. Finally, we perform an extensive set of experiments which allow us to assess the performance of our method in various setups, and to compare it to a representative baseline of existing methods. This baseline implied to retrain and test existing methods on UAV-oriented datasets, including Mid-Air, for assessing their transferability to depth estimation for UAV applications. This set of experiments, which allows us to highlight the strengths and limitations of our method, shows that M4Depth is superior to other methods on UAV datasets while performing similarly to the baseline on standard depth estimation baseline.

Contributions 2

- We define a notion of visual parallax between two frames from a generic six-degree-of-freedom (6-DoF) camera motion, and present a way to build cost volumes with this parallax;
- We present a novel lightweight multi-level architecture, called M4Depth, that is based on these cost volumes, designed to perform end-to-end depth estimation on video streams acquired in unstructured environments, and suitable for real-time applications;
- It is shown that M4Depth, is state-of-the-art on our Mid-Air dataset, that it has good performances on the KITTI dataset [38], and that it outperforms existing methods in zero-shot transfer on the TartanAir dataset [137].

Chapter 4 explores the possibility to predict the uncertainty on depth estimates produced by M4Depth and presents M4Depth+U, a method to jointly estimate depth and uncertainty based on the architecture of M4Depth. In a first part, we formalize the problem of uncertainty estimation, and explore the related works. We then explain why getting an uncertainty estimate related to depth from the parallax values produced by M4Depth is not trivial, and why the purely probabilistic approach, which would be the natural solution to this problem, is suboptimal. We then proceed to identify and detail a new custom-tailored strategy, called M4Depth+U, to convert the uncertainty estimates related to parallax generated by the network into uncertainty estimates related to depth. Our tests in various conditions, including zero-shot cross-dataset transfer, and on various publicly available datasets show that M4Depth+U consistently outperforms the purely probabilistic approach. In addition, the proposed method performs consistently on the robustness benchmark of the Mid-Air dataset, therefore showing its low sensitivity to visual changes. Finally, the performance obtained on a zero-shot cross-dataset transfer benchmark for multi-view depth (MVD) estimation methods shows the value of our method, as it performs similarly to existing multi-view stereo methods while being 2.5 times faster and causal.

Contributions 3

- We propose the first method that addresses joint monocular depth and uncertainty estimation for the specific constraints of autonomous vehicles;
- We test our method on three public datasets and show that the uncertainty estimate performs consistently in zero-shot transfer in

Contributions continued 3

different environments, therefore showing its generalization capability to unseen environments;

- We test our method on a benchmark for MVS and show that, despite being causal as opposed to other methods, its performance is on par with existing MVS methods for joint depth and uncertainty estimation. In addition, our method is 2.5 times faster.

Chapter 5 summarizes the research presented in this work and concludes on its contributions. This chapter also discusses some further research ideas and open questions.

1.5 Publications

This thesis is based on the three following publications:

- M. Fonder and M. Van Droogenbroeck, "Mid-Air: A Multi-Modal Dataset for Extremely Low Altitude Drone Flights," in *IEEE Int. Conf. Comput. Vis. Pattern Recognit. Work. (CVPRW), UAVision*, Long Beach, CA, USA: Inst. Electr. Electron. Eng. (IEEE), 2019, pp. 553–562. DOI: 10.1109/cvprw.2019.00081
  The content of Chapter 2 is adapted from this publication.
- M. Fonder *et al.*, "Parallax Inference for Robust Temporal Monocular Depth Estimation in Unstructured Environments," *Sensors*, vol. 22, no. 23, pp. 1–22, 2022. DOI: 10.3390/s22239374
  The content of Chapter 3 is adapted from this publication.
- M. Fonder and M. Van Droogenbroeck, "A technique to jointly estimate depth and depth uncertainty for unmanned aerial vehicles," in *IEEE Int. Conf. Syst. Signals Image Process. (IWSSIP)*, Ohrid, North Macedonia, 2023, pp. 1–5
  The content of Chapter 4 is an extended version of this submitted article.

2 Mid-Air: A multi-modal dataset for extremely low altitude drone flights

Summary

Overview

In this chapter, we introduce Mid-Air, a multipurpose synthetic dataset of low altitude drone flights in unstructured environments which features synchronized data of multiple sensors for a total of 54 trajectories and more than 420k video frames simulated in various climate conditions.We first present related and existing datasets and show how they compare to Mid-Air. Then, we motivate our design choices, explain how the data was simulated, and detail the content of our dataset. Finally, we propose recommended train/test splits for standardized method evaluations. We also introduce benchmarks for positioning and for image generation tasks, and show how Mid-Air can be used to set up a standard evaluation method for assessing the performance of computer vision algorithms in terms of robustness and generalization. Finally, we show the value Mid-Air has already brought to the scientific community by highlighting works that made active use of it.

Data availability. The dataset presented in this chapter is publicly available under the CC BY-NC-SA 4.0 license and can be downloaded on the following website: https://midair.ulg.ac.be/.

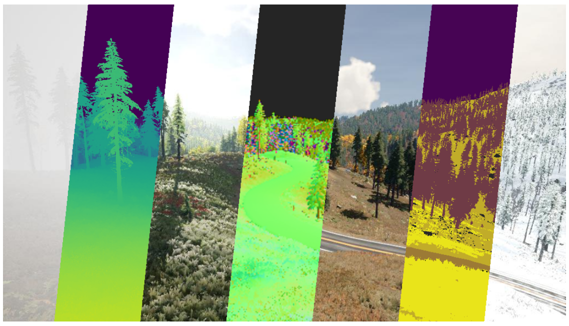


Figure 2.1: Sample from our Mid-Air dataset. The data simulated when flying our drone in a scene includes, among others and from left to right in the illustration, an RGB image under foggy weather, the depth map, an RGB image for a spring sunset, the map of surface normals, an RGB image of a clear sky weather during fall, the semantic segmentation map, and an RGB image in a cloudy winter.

2.1 Introduction

Currently, the best methods for flying drones autonomously rely on machine learning algorithms and, for vision-related tasks, involve deep neural networks. The success of machine learning methods is often due to their ability to learn complex patterns from examples by bypassing the need for an explicit analytic model. However, a common shortcoming of deep learning methods lies in the direct relation that exists between their final performance and the quality of the data used for their training. Indeed, if the training data does not completely represent the task, there is a risk of experiencing poor performance in practice. Stated otherwise, a deep neural network trained with insufficiently varied samples will not generalize properly.

At the start of this thesis, no dataset was available for training deep learning methods for depth estimation on outdoor data recorded by a drone, and we began our research by using datasets that target autonomous car applications. However, our first results hinted at probable generalization issues to drone data. Indeed, as opposed to drones, cars are non-holonomic vehicles that move in constrained environments, *i.e.,* roads. Therefore, sensors mounted on them only encounter a limited subset of motion types, and cameras do not completely explore their 3D environment. As a result, these datasets do not represent drone flights well enough and depth estimation methods trained on these datasets seemed to be strongly biased towards autonomous cars applications. At this point, the need to be able to train and test methods on drone data became blatant and asked for the development of a dataset specific to flying drones.

This motivated us to build a new dataset, named Mid-Air, with has a particular focus on applications for autonomous flight. Our main motivation for its design was to provide a large multi-modal dataset, as illustrated in Table 2.1, that enable the training of most machine learning tasks required for autonomous piloting such as visual odometry, simultaneous localization and mapping, depth estimation, semantic segmentation, or stereo disparity estimation. The synchronicity of the data allows targeting these tasks individually or simultaneously in a multitask setup. In addition to providing a basis for training and benchmarking algorithms, we also wanted to push innovation forward, by providing a new type of data, surface normals,

Datasets	Mid-Air [30]	Kitti [31, 37, 38, 94]	Virtual Kitti [33]	Synthia [51, 116]	RGB-D SLAM [127]	EuRoC MAV [11]
Number of trajectories	54	71	50	7	19	11
Number of frames	119k* (@25Hz)	44k (@10Hz)	21k* (@10Hz)	7k* (@5Hz)	48k (@30Hz)	27k (@20Hz)
Total duration	79 min	73 min	35 min	23 min	27 min	22 min
Resolution	1024 × 1024	1382 × 512	1242 × 375	960 × 720	640x480	752 × 480
Data type	synthetic	real	synthetic	synthetic	real	real
Camera motion	**drone flight**	car drive	car drive	car drive	hand-held	drone flight
Environment type	**unstructured**	city	city	city	indoor	indoor
Climate variations	yes	no	yes	yes	no	no
IMU data	yes	yes	no	no	yes	yes
GPS	yes	yes	no	no	no	no
Depth map	dense	sparse	dense	dense	dense	sparse
Stereo disparity map	yes	yes	no	no	no	no
Surface normals	yes	no	no	no	no	no
Semantic segmentation	yes	yes	yes	yes	no	no
Instances segmentation	no	yes	yes	yes	no	no
Optical flow	no	yes	yes	no	no	no

Table 2.1: Comparison between our Mid-Air dataset and similar datasets usable for multi-task learning. The symbol * denotes that several additional RGB videos, for different conditions, are available for a same trajectory.

that is not present in any other similar dataset.

In this section, we first present the datasets that existed when we designed Mid-Air, and that are related to our need. We then briefly introduce our dataset and explain how it compares to existing ones. Finally, we detail the content of this chapter and highlight our main contributions.

Existing datasets. In 2019, various datasets existed for developing computer vision tasks [12, 17, 90, 104, 118, 119, 123], but only a few of them were large enough for machine learning algorithms or provide the data of several sensors for multitask learning. The KITTI dataset [31, 37, 38, 94] was probably the most complete one and has been a reference in the field for training and benchmarking methods addressing various computer vision tasks. Due to a lack of alternatives, most methods that jointly estimate two types of information, such as image depth and camera ego-motion [83, 155] for example, had to be trained and/or tested on this dataset alone.

Table 2.1 summarizes some common multi-modal datasets that were available at the time and compares them to our Mid-Air dataset (first column), according to the number of samples, the acquisition conditions, and types of data. This table also shows that the largest and most complete datasets were mainly designed for autonomous car applications, as they provide data from sensors mounted on a car driving in urban environments.

Mid-Air: a new synthetic dataset. The dataset we introduce features synthetic, *i.e.,* simulated, data of navigation and vision sensors mounted on board of a quadcopter flying in unstructured environments. Its main characteristics are summarized in Table 2.1. Our aim was not only to offer an alternative to existing datasets for flying drones, but also to provide material to develop algorithms for autonomous vehicles which could be generalizable to a wide variety of situations. For example, the choice of recording the dataset in unstructured environments was guided by the fact that these environments are the most challenging for some computer vision tasks, such as depth estimation. In addition, using a drone allows recording a variety of camera motions and poses which would not be possible with a car, which makes tasks such as visual odometry or SLAM more complex.

The content of the dataset was generated using the Unreal Engine 4 [25], a game engine software with realistic-looking real-time image rendering, with Airsim [121], a plugin for this engine that offers an API-controlled multi-rotor drone simulation framework. As opposed to real data capture, working with a simulator offers a full control on the data to be recorded. We leveraged this advantage in two different ways. First, we recorded each flight trajectory several times with varying climate conditions (different seasons, time of day, and weather conditions), which allows training networks for robustness to visual changes. Second, generating data synthetically also allows recording ground-truth data, which is more reliable or even impossible to capture with real sensors.

In Mid-Air, we do not only provide common dense ground-truth visual maps such as depth maps, but also introduce surface normal maps, a new type of dense ground-truth data previously unseen in a dataset. This innovates by paving the way for the development of methods for surfaces normal estimation tasks. In addition, we introduce two unique benchmarks, a first for positioning tasks and a second for image generation tasks, to test the robustness and generalization capabilities of computer vision methods.

This chapter first presents the models used to simulate the sensors whose data are given in the dataset. We then detail the content of the dataset itself. For this, we explain the setup used for simulating the drone and the visual environments. We also explain the methodology used to generate the data making the dataset, and the limitations induced by our setup. Next, we make a train/test split suggestion for our dataset and introduce two distinct benchmarks to test the robustness of methods trained on our dataset. Finally, we conclude this chapter and make a concise retrospective on the value brought by Mid-Air to the scientific community. Our main contributions for this chapter are as follows:

- We propose the first public multi-modal dataset for deep learning dedicated to UAVs flying in unstructured environments;
- Our dataset features ground-truth data, such as surface normals, not present in other major datasets;
- We propose a benchmark to test the robustness of computer vision methods to visual changes;
- Mid-Air has proven its value to the community as various original works made active use of it.

2.2 Sensors simulation

Building a synthetic dataset for flying drones in unstructured environments requires a precise model of the drone and its sensors. For the physics of the drone, we rely on the default quadcopter drone model provided by the Airsim simulator. However, due to some shortcomings of the simulator, we had to re-implement some sensor models from scratch or to tweak others to gain more control on the generated data and increase their accuracy. Hereafter, we detail the modifications which were made to the default models provided by Airsim and elaborate on the reasons which led us to make these modifications.

2.2.1 Accelerometer and gyroscope

The accelerometer and the gyroscope, two sensors constituting the core of the inertial measurement unit (IMU), are essential for stabilization and more generally for flying a drone. Unfortunately, both are not free of imperfections; they are prone to bias, bias drift, and measurement noise. It is commonly assumed that the bias, b_t, behaves as a Gaussian random walk process, and that the measurement noise follows a Gaussian (or normal) distribution with a zero mean. Accordingly, for a sampling period dt, the relationship between the ground-truth measurement m_{GT} and the sensor measurement m_{sens} for one axis is as follows:

$$\begin{aligned} m_{\text{sens}} &= m_{\text{GT}} + \nu_n + b_t \text{ where } \nu_n \sim \mathcal{N}(0, n) \text{ and} \\ b_t &= b_{t-1} + \nu_b \text{ where } \nu_b \sim \mathcal{N}\left(0, b_0\sqrt{\frac{dt}{t_a}}\right), \end{aligned} \tag{2.1}$$

where n, b_0 and t_a are parameters that can be determined by carrying an Allan diagram analysis (see [142]) and which differ for each individual sensor.

These parameters can, however, not be modified by the API of Airsim. Since we wanted to generate trajectories with different IMU settings, we reimplemented this model. Before each flight, a new set is drawn randomly within bounds that are representative of the variations typical for different IMU models. The choice of the order of magnitudes to use was guided by experimental measurements given in different studies of various sensor models [58, 100, 101, 112].

It is important to note that both the accelerometer and the gyroscope are also prone to axis misalignment and scaling factor issues. Unfortunately, these two imperfections appear to be far less studied experimentally. For this reason, we preferred to rely on a model commonly adopted for control applications, even though it neglects these parameters.

2.2.2 GPS receiver

GPS receivers are heavily used for positioning in the context of autonomous driving and driving assistance. They can indeed regress their absolute position based on the satellites that are in line of sight. The regression process is in three steps. First, the sensor computes the delay between the time of emission of GPS signals by each satellite and their time of arrival to the sensor. This delay is derived from the data encoded in the signals. After that, the sensor uses these delays to estimate the distance separating it from each satellite. These distances are called pseudo-ranges. Eventually, the sensor can triangulate its position with an optimization method based on the positioning knowledge it has about GPS satellite positions and the pseudo-ranges estimated during the previous step.

A GPS model. GPS positioning would be perfect if pseudo-ranges could be computed precisely. Perfect distances would be obtained if all clocks were perfectly synchronized and if the speed of light was known all along the path between a satellite and the receiver. However, this is not the case in practice. Clocks have indeed small, but measurable offsets. Furthermore, the atmosphere is composed of several layers which interact differently with the light and therefore modify its speed and path. Without loss of generality, the relationship between the estimated and real pseudo-range for a given satellite and a given carrier frequency, ρ_e and ρ_r respectively, can be expressed as follows (see [3]):

$$\rho_e = \rho_r + c\,(\delta_i - \delta_R) + \Delta_I + \Delta_T + \nu\,, \tag{2.2}$$

where c is the speed of light, δ_i and δ_R are the satellite signal inaccuracy and the receiver clock offset respectively, Δ_I and Δ_T are the delays induced by the ionosphere and the troposphere respectively, and ν is the receiver measurement noise. Since the ionosphere induces a delay which is inversely proportional to the frequency [3], Δ_I can be estimated only by using several carrier frequencies. Other inaccuracies and delays cannot be estimated by the receiver. For those reasons, a good GPS receiver simulation can only be obtained when these imperfections are accurately modeled.

Shah *et al.* [121] do not provide any detail on the model built in Airsim, which makes it hardly reliable when trying to simulate real-world conditions. For this reason, we developed our own model and had to choose between a model for a single-frequency or a dual-frequency receiver. As the latter are currently not widespread, we built a custom single-frequency receiver model derived from the dual-frequency model and implementation given by Agarwal and Hablani [3]. The adaptation is straightforward and consists of simply removing all the corrections enabled by the use of several carrier frequencies. More precisely, we removed the ionosphere delay correction term from the estimated pseudo-ranges used for the optimization process.

Satellites visibility. To assess the satellites which are in line of sight, Agarwal and Hablani [3] make the assumption that the Earth is a perfect, smooth ellipsoid and that there are no obstacles. This is hardly acceptable for drones that fly at low altitudes. Hence, we decided to refine the model by considering the shape of the 3D environment generated by the simulator. Our idea consists in adding a wide-angle depth camera to the drone that points upwards. This creates a map of areas of the sky which are occluded by obstacles. Assuming that the starting position of the drone was mapped to an arbitrary location on the Earth surface, and since we know perfectly the position and the attitude of the drone, it is then possible to project

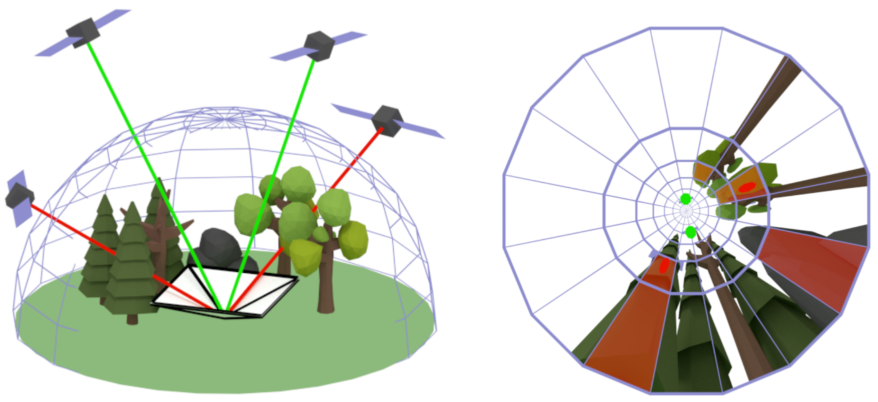


Figure 2.2: Illustration of the method used to determine which GPS satellites are in line of sight. The left picture shows a 3D perspective of the scene. The right picture shows what is seen by the receiver. Green and red dots correspond to projected satellites positions, which are respectively in line of sight or not according to our method. Areas overlaid with red do not contain any pixel belonging to the sky.

the satellite positions on this map and determine if some satellites are occluded by the presence of obstacles. A simple rule would be to discard a satellite as soon as it is occluded by an obstacle along the path. This rule therefore assumes that all obstacles are perfectly stopping the signals. However, several studies [8, 73] contradict this assumption for trees. Since the environments used for our database contain a lot of them, another rule was developed. Because sky occlusion maps are missing information about the amount of GPS signal absorption and availability, we defined an empirical rule, illustrated in Fig. 2.2. It basically consists of dividing the sky in several areas. A satellite is considered as occluded if no sky portion is visible in the whole area on which it is projected.

2.2.3 RGB camera image rendering

As mentioned previously, we used the Unreal Engine for rendering the images of our dataset because it offers a good trade-off between rendering time and visual accuracy. Hereafter, we discuss the consequences on the visual accuracy in terms of shading, geometrical limitations, and level of details. It is important to elaborate on the consequences of our choices, as they might have an impact on the generalization of learning algorithms to real-life scenarios. We also explain why we believe they are acceptable.

Rendering. The Unreal Engine 4 uses an enhanced version of the deferred rendering algorithm. This means that shadings are not rendered with physically accurate equations, but rather with simplified models which were targeting a good visual likelihood. Such models are well suited for diffuse surfaces, but have

Deferred rendering 5

Deferred rendering is a technique used in computer graphics to render 3D scenes efficiently, often in real time. As optically accurate rendering, as done with ray tracing, is computationally expensive, deferred rendering techniques use approximations and simplifications to speed up renderings. Therefore, the target of deferred rendering is good visual likelihood at the expense of physical accuracy.
Initially, such techniques were only designed for surfaces with diffuse, also called Lambertian, reflectance, but have later been improved to accommodate for more complex surface properties. Any material with reflective or refractive properties has to be simulated with arbitrary tricks. For example, the preferred method for faking reflections in this context consists of projecting and capturing the surrounding of an arbitrary area on a cube map, and of projecting this cube map on the surface of all objects present within this area. With this method, reflections are skewed and light rays do not bounce on reflective objects. Refractions, on the other side, are often approximated by simple transparent materials and will not deflect light rays as in the real world.

difficulties dealing with surfaces which interact with light rays in other fashions. This is especially true for reflective and refractive materials. Therefore, these inaccuracies could lead to biases in deep learning methods when used as training data, and hinder their generalization capabilities.

Fortunately, most natural elements, with the notable exception of water bodies, can be faithfully approximated by deferred rendering. Water is an element with complex optical properties which are challenging for all rendering techniques, and dynamic water is probably the natural feature that is the most poorly approximated by deferred rendering. Large bodies of still water such as lakes can nonetheless be decently approximated by a mix of reflective and transparent material. Therefore, synthetic datasets featuring still natural environments should induce limited risks of poor generalization for deep neural networks trained on them.

Geometrical limitations. In addition to shading limitations, there are some geometrical limitations. Since computers have a finite amount of memory, it is unrealistic to populate a virtual world with an infinite amount of different objects. What is done in practice is to use a limited subset of assets and to replicate it with basic modifications (scale and orientation) over the environment where needed. This has obvious implications for semantic segmentation algorithms, and must therefore be kept in mind when using synthetic datasets in general.

Eventually, game engines come with a further memory and computation cost-saving feature that is called the “Level of Details” (LoD). This feature reduces the number of faces to be displayed by simplifying the geometry of objects more and more aggressively as the distance to the camera increases. This simplification is dynamic and leads to object re-instantiation during runtime. It could therefore be harmful

Algorithm: Occlusion mask generation 1

```
1 For each pixel $p_{i,j}$ of depth$_{\text{Camera 1}}$:
2     Compute 3D position of $p_{i,j}$ with respect to Camera 1;
3     Express 3D position of $p_{i,j}$ relatively to Camera 2;
4     Project $p_{i,j}$ on the sensor plane of Camera 2;
5     Get coordinates $[k,l]$ of the corresponding projection;
6     if depth$_{\text{Camera 1}}[i,j] >$ depth$_{\text{Camera 2}}[k,l]$:
7         The surface corresponding to $p_{i,j}$ is hidden to Camera 2;
```

for computer vision learning algorithms relying on video streams. As this feature is mandatory for large environments, we took special care in tuning it such that the re-instantiation only occurs past a distance at which the visual impact on the RGB capture becomes minimal.

2.2.4 Synthetic sensors / 3D and semantic sensors

Working with a simulator enables to gather information about the environment and the 3D world scene which is not possible to capture with real sensors. For example, we can think about perfect dense depth maps, perfect and automatically annotated segmentation maps or even normal maps.

They can be created by exploiting features implemented for deferred shading and gathered through the Airsim API. The only major difficulty is that objects using transparency do not appear on normal maps. This is due to the intrinsic design of the deferred rendering pipeline. Since the only transparent objects present in our datasets are water planes, we decided to solve this issue by assigning a perfectly vertical normal to all pixels of the map belonging to a water plane.

Stereo ground truths. Since we know perfectly the virtual environment, it is also possible to generate ground truths for stereo disparity maps and occlusion masks. The disparity d expressed in pixels can indeed be inferred from a single planar depth map Z expressed in meters by using the following equation:

$$d = \frac{fb}{Z}, \tag{2.3}$$

where f is the focal length of the corresponding camera in pixels, and b is the baseline between the two cameras expressed in meters.

To calculate the occlusion mask, we have to determine the areas of a picture taken by one of the two cameras which are not visible by the second camera. This mask can be inferred using the planar depth maps corresponding to the scene seen by each camera and is detailed in Algorithm 1. It is important to note that the disparity map and occlusion mask are always specific to one of the two cameras.

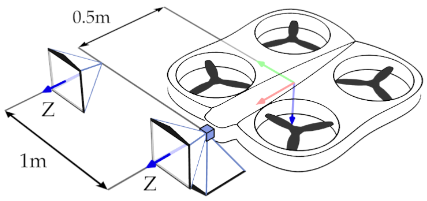


Figure 2.3: Sensor locations on the drone used to generate our dataset. Cameras are represented by the pyramids; the blue cube shows the IMU and the GPS receiver locations.

2.3 Dataset presentation

After the details related to the simulation environment, we now present the design and content of our Mid-Air dataset. In particular, this section explains the different design rules. We first introduce the drone setup that has been used to record the trajectories. Then, we provide some details about the environments used for the flights. Finally, we present an overall view of the dataset content.

2.3.1 Drone setup

The physical drone model used for the flights is the default quadcopter model of Airsim with customized sensors types and placements. We used three different cameras: a front-looking camera placed on the X-axis of the drone, another front-looking camera with a 1-meter baseline compared to the first for stereo applications, and a last camera looking downward. The latter can be especially useful for visual odometry and SLAM algorithms [6], and was therefore added to the common front-looking cameras. The left-stereo camera, the IMU, the GPS receiver and the down-looking camera are placed exactly at the same location (see Fig. 2.3 for a schematic view of the drone setup). This choice, even if unrealistic, should not have any impact on the learning algorithms and greatly eases the use of the dataset since no additional translations are required when working with several sensors at the same time.

The cameras use the pinhole camera model to capture images. They are therefore perfectly calibrated by default. This is once again an advantage since it enables to remove the camera calibration method out of the equation when comparing different algorithms. Due to render engine limitations, cameras act as global shutter cameras and do not present any motion blur. They are all set to capture images at a rate of 25 Hz with a field of view of 90 degrees. In addition to RGB data, the left stereo camera captures a semantic segmentation map, a depth map, a normal map, a stereo disparity map, and a stereo occlusion mask. All images have a size of 1024×1024 pixels, except the normal maps that have a size of 512×512 pixels.

The IMU measurements refresh rate is set to 100 Hz and the GPS receiver updates its position every second. The parameters of the IMU are randomly drawn before each flight, and the initial bias is logged for each trajectory. The same yields for the

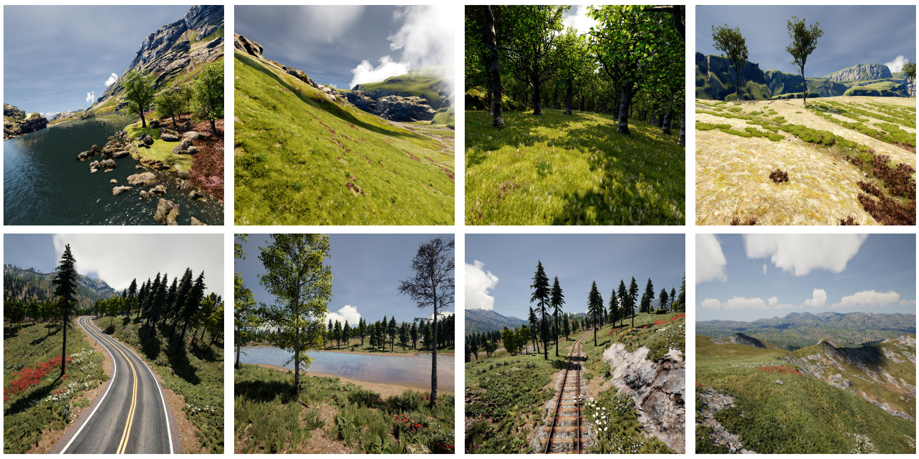


Figure 2.4: Samples of our dataset illustrating the variety of environments used to generate the visual information.

initial GPS position. The latitude, longitude and altitude are drawn uniformly in the ranges of $[0, 60]$ degrees, $[-180, 180]$ degrees and $[-500, 500]$ meters respectively. The GPS satellites having an orbital period of 12 hours, we also randomized the initial trajectory time to get different satellite positions.

2.3.2 Environments setup

As stated earlier, our dataset aims to provide data to train and test algorithms for robustness. This goal can only be achieved if the data is varied. To guarantee enough variety, we used two different large-scale environments displaying varied features (see Fig. 2.4) and different climate setups. The following subsections present the setups in which we flew our drone.

Landscapes

The first environment used is the map given in the Kite demo of the Unreal Engine. It features a mountain landscape with several lakes and forests (see the first row of Fig. 2.4). Its size, which reaches almost 100 km^2, guarantees to find places with varied characteristics and features. Its topography makes it perfect for training algorithms to deal with uneven grounds and height variations.

The second environment is made up of the two demo maps provided by the PLE plugin for the Unreal Engine. Together, they cover an area of roughly 10 km^2 and feature a hilly landscape with forests and some lakes, as for the first environment. In addition, both maps are crossed by a road (with signs, barriers, ...) and a railway track. The specificity of these maps lies in the possibility to change the season and

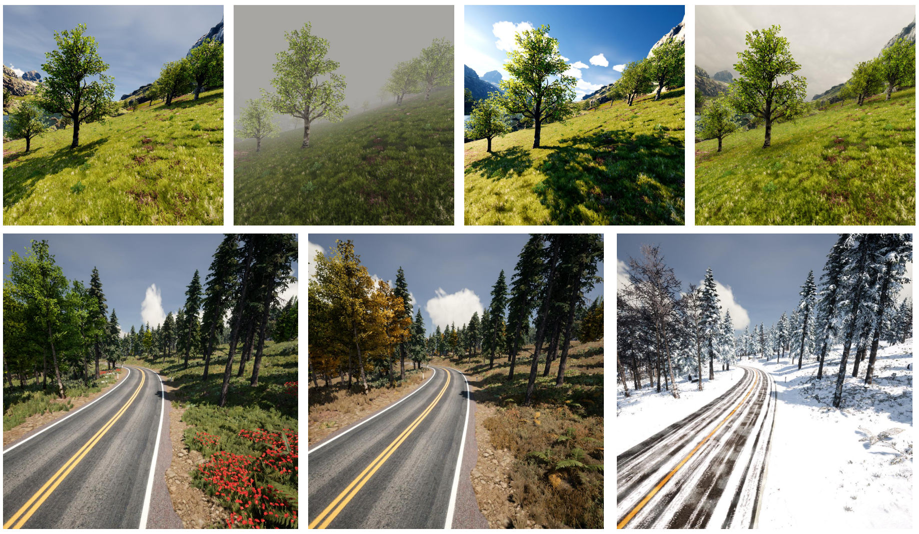


Figure 2.5: Samples of our dataset showing the different climate setups. The top row shows the four simulated weathers. The bottom row illustrates the seasons.

therefore to change the visuals of the environment. Samples of these maps are shown in the second row of Fig. 2.4.

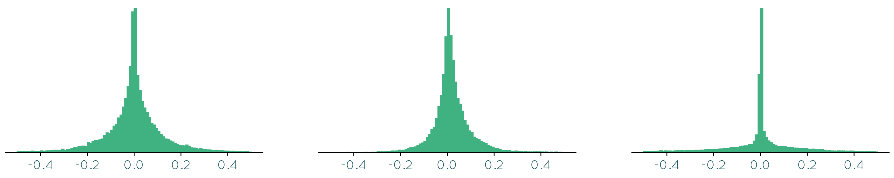
Climate conditions

For the climate settings, we have two tunable modalities: the weather and the season. The weather parameter mainly affects the sky color as well as the illumination of the scene. The season parameter, on the other hand, mostly affects the colors present in the environment. Since nature is extremely varied, the possibilities of configurations are endless. In order to keep a tractable size for the dataset, we restricted ourselves to a carefully chosen subset of scenarios.

We chose to simulate four distinct weathers and three seasons. To generate the different weathers, we use the TrueSky plugin for the Unreal Engine. It allows creating volumetric and dynamic clouds able to cast shadows on the map. They therefore have a realistic behavior, which is important for video applications. For the season setups, we relied on the presets of the PLE plugin. Hereafter, we give a brief description of the visual properties of each setup:

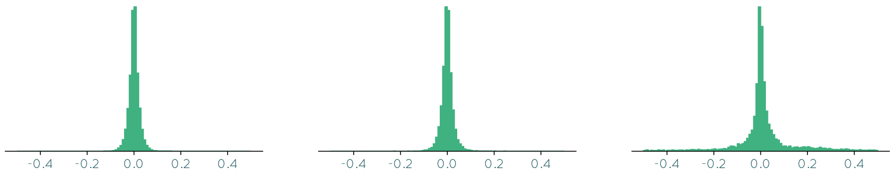
Clear sky at midday. The illumination is typical for a normal use case in sunny weather. The shadows are harsh, and the color of the sky is mainly blue.

Overcast sky. The illumination is dim, and the shadows are almost absent. This and the gray sky color provide a realistic representation of cloudy weather.



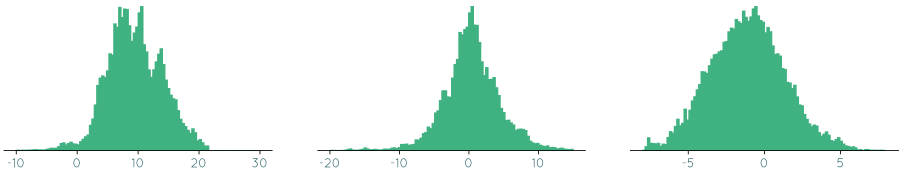
(a) Roll angular velocity [rad/s] **(b)** Pitch angular velocity [rad/s] **(c)** Yaw angular velocity [rad/s]

Figure 2.6: Distribution of the angular velocities of the drone recorded in our dataset for each axis of the drone.



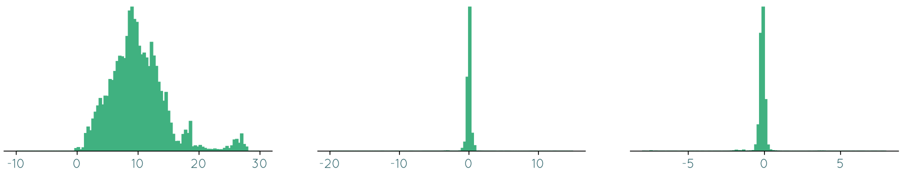
(a) Roll angular velocity [rad/s] **(b)** Pitch angular velocity [rad/s] **(c)** Yaw angular velocity [rad/s]

Figure 2.7: Distribution of the angular velocities of the car recorded in the KITTI dataset for each axis of the car.



(a) Forward velocity $[ms^{-1}]$ **(b)** Lateral velocity $[ms^{-1}]$ **(c)** Vertical velocity $[ms^{-1}]$

Figure 2.8: Distribution of the velocities of the drone recorded in our dataset for each axis of the drone.



(a) Forward velocity $[ms^{-1}]$ **(b)** Lateral velocity $[ms^{-1}]$ **(c)** Vertical velocity $[ms^{-1}]$

Figure 2.9: Distribution of the velocities of the car recorded in the KITTI dataset for each axis of the car.

Sunset. This weather was chosen for its challenging illumination conditions. The shadows are indeed heavily elongated. This creates areas which are completely shadowed and therefore require good illumination robustness to be parsed correctly. In addition, the sun is low and can enter the field of view, which creates simulated sun glares.

Fog. This weather is challenging for algorithms targeting visual odometry. Since visual features fade with distance, algorithms have less visual cues to rely on to correct the state estimation, leading to less accurate state corrections. This weather is also valuable for testing the robustness of image generation tasks, since it tends to desaturate colors.

Seasons. For the season setups, we relied on the presets of the PLE plugin. We found out that all seasons do not contribute equally to the diversity of the dataset. Summer was discarded due to its resemblance with spring and fall. Including it would not have added any significant variety to the dataset, while discarding it allows reducing the dataset size. We kept the spring, fall, and winter seasons. Spring features trees with green leaves and luxuriant ground vegetation. Fall differentiates itself from spring with trees with yellow leaves and dried ground vegetation. Finally, winter is interesting for its trees without leaves and an environment covered with snow.

2.3.3 Scenarios and format

After having properly defined the environments and setups, we manually flew the drone in the simulator using an RC controller connected to the computer through USB and recorded 5 hours of flight. We then extracted 79 minutes out of it. These 79 minutes correspond to 54 trajectories of equal length, *i.e.,* 1.47 minute each. The first 30 are captured in the Kite demo environment and the remaining ones in the PLE environment. As visible in Figures 2.6 to 2.9, the motion of the drone recorded in our dataset is much more varied than the motion typically found in dataset captured with a car. Indeed, with the exception of the yaw angular velocity and the forward velocity, which are similar, the motion of the drone displays a much wider distribution of velocity values than the one of the car.

Each trajectory record is rendered several times, once for every climate scenario. In practice, this translates into rendering trajectories belonging to the Kite demo environment once for each weather setup and once for each season for those belonging to the PLE environment. Since there can be some differences due to objects animation between each render, all data streams are recorded simultaneously at each run.

The data includes the ground-truth positioning information, *i.e.,* the position, velocity, acceleration, attitude and angular velocity, the IMU sensor measurements, the estimated GPS position along with complementary information on the GPS signal such as its dilution of precision and the number of visible satellites, and finally the camera data, *i.e.,* the left, right, and down-looking RGB images, and the segmentation, depth, normals, disparity, and occlusion maps corresponding to the left camera. Our semantic segmentation dataset contains a total of 12 different classes (animal, tree, dirt ground, rocky ground, ground vegetation, boulder, water plane, man-made construction, road, train track, road sign, other man-made objects).

Sensor data is stored in a common `hdf5` dataset file, while the pictures are saved independently in several subdirectories. This enables a good dataset handling and eases data access. RGB pictures are stored in JPEG files and maps are stored as 16-bit float matrices encoded with a lossless PNG format. On average, the set of 8 images recorded at each frame weights less than 2.5 MB. This is roughly 6 times less than the space required for the same data stored in uncompressed raw format. All additional information about data layout and organization is given on the website of our dataset. To ease the data layout understanding and parsing process, we provide several example scripts with the dataset.

2.3.4 Overall limitations

Despite all our efforts, we did not address two specific situations which might be important for generalization.

The first one is that our dataset contains only a few moving objects. Motion is present only through the agitation of the vegetation due to the wind and through the few animals present in the environments. It means that a learning algorithm working on video sequences will be poorly prepared for image changes which are not due to perspective and camera motion.

A second limitation is that the used simulator does not model wind nor drone vibrations. However, these two parameters have a significant impact on the IMU. The former creates accelerations which are not induced by the drone propellers, while the latter adds additional noise terms to the measurements made by the sensors. Both can have an impact on the generalization of algorithms relying on IMU data. Our dataset should nonetheless be useful to get a reliable performance score for simple scenarios, and therefore to get a first overview of the potential of any tested method.

2.4 Benchmarking modalities

Since comparing and interpreting the performance of different methods on a dataset can only be achieved when they all have been developed and tested using the same data split, it is important to clearly define the train and test splits to be used with Mid-Air. Therefore, we propose to allocate one in three of the 192 trajectories featured in the dataset to the test set. Practically, we allocate each trajectory whose last two id numbers make a multiple of 3 to the test set. This creates a two-third/one-third split for the train and test sets, which should guarantee similar data distribution between sets. It should be said that other splits between training, validation and test data are possible, but any use of a different split should be clearly documented to allow for research reproducibility.

Besides the definition of the split, we propose two additional sets for benchmarking methods. The first set was recorded in an environment not available in other sets and aims at providing a benchmark to test the performance of positioning estimation methods such as visual odometry in generalization. The second set aims at testing the robustness of visual map generation methods to visual changes. These additional sets use the same sensor models and provide the same data as the one defined for the main dataset.

2.4.1 Benchmark for positioning tasks

To test the performance on positioning tasks such as visual-inertial odometry or SLAM algorithms in generalization, we provide three additional trajectories recorded in an unseen environment. To assess the robustness of tested algorithms, all trajectories were recorded for three weather conditions, *i.e.,* clear sky, fog, and sunset. As explained in the next paragraphs, each trajectory has its own specificity so that performance scores should ideally be reported independently for all nine possible scenarios.

Ideally, generic positioning algorithms have to be robust to visual novelties. That is why we recorded our trajectories in an environment with strong visual differences compared to the training data. For this, we used a modified version of the Landscapes Mountain demo map for the Unreal Engine.

The first trajectory has a length of 5 minutes and was generated by manually flying the drone, as for the training data. The two other trajectories have a length of 10 minutes and were generated synthetically. They follow both the same path except that, for one of them, the drone is looking towards where it is going and, on the other, the yaw of the drone is shifted by 90 degrees. This allows to have a scenario where the visual features are moving towards the camera, and another where they are scrolling from left to right in the frame.

The synthetic path was chosen to be challenging and consists of a circular trajectory with a varying height, radius and angular velocity with no periodicity. This is an extreme case because the aircraft experiences almost constant acceleration. Therefore, failing to use visual cues properly to correct the state estimate will lead to significant drift when loop closure is not used to periodically correct the state estimation.

2.4.2 Robustness benchmark for visual map generation tasks

For most computer vision tasks, different visual inputs can lead to the same intended output. Take a picture of a city scene on a sunny summer day, for example. A picture of the same scene could have been taken during a gloomy winter day, therefore leading to radically different visual information being captured. However, a computer vision algorithm such as a semantic segmentation or a depth estimation method is expected to produce exactly the same output map for both pictures as they represent the same scene, but may not do so consistently because of the differences in the visual data. To be trusted for autonomous vehicle applications, a computer vision algorithm ideally has, among other factors, to be insensitive to such visual changes.

In order to test the robustness of computer vision methods to visual changes in their input, we provide eleven additional 13-second long trajectories recorded in unseen parts of our virtual environments, with five of them belonging to the Kite map and the six remaining ones to the PLE maps. The trajectories recorded in the Kite environment were rendered four times, once for each weather condition, and the ones in the PLE environment three times, once for each season. To analyze the robustness of a method to visual changes, we recommend testing and reporting its performance separately for each weather and season setup. If the scores are similar for all scenarios, the method will be assumed to be robust to visual changes. On the other hand, a difference in performance will highlight some robustness deficiencies.

Baseline example 1

We actively used Mid-Air for in our following works to train and test depth estimation methods, and used the robustness benchmark to compare some properties of different methods. Therefore, a typical example of the intended use for this benchmark can be found in Section 3.5.4.

2.5 Conclusion

In 2019, we introduced a new synthetic dataset, named Mid-Air, featuring 79 minutes of drone flight recorded several times with different climate conditions. The content of our dataset consists of multiple synchronized modalities providing data for positioning tasks such as SLAM or visual odometry as well as for pure computer vision tasks such as depth estimation, semantic segmentation, or surface normal estimation. While being specifically designed for flying drones in unstructured environments, its size (more than 420k individual frames) and content makes it also useful for training and testing machine learning algorithms for other applications than UAVs.

Large datasets such as ours pave the way for building new benchmarks to evaluate single or multitasks algorithms in complex environments. Hence, after discussing the sensor models and elaborating on the generated data, we propose a train/test split for this dataset alongside two distinct benchmarks. The purpose of the first proposed benchmark is to test visual odometry or SLAM methods in an environment different from the ones present in the training set to test their generalization capabilities. The purpose of the second is to test the robustness of computer vision methods to changes in visual inputs.

The dataset and all relevant practical details regarding its use are publicly available on the following website: https://midair.ulg.ac.be/

Uses of Mid-Air by the scientific community

The Mid-Air dataset has been available for download since June 2019, and has been downloaded by more than 500 researchers around the world by the beginning of 2023. Its multi-modal aspect led it to be used for various original contributions, which confirms its value to the scientific community.

Among the varied uses of our dataset, we find the following contributions. Miclea and Nedevschi [95, 96] used it to train methods aiming at very long-range depth estimation for UAVs. Ercolino *et al.* [26] used it to test the robustness of visual transformers to adversarial attacks on depth estimation. Song *et al.* [125] targeted a different use by training a deep visual odometry method with our dataset. Haggart and Aitken [46] made use of the different climate conditions to test the dependence of a SLAM method on the visual quality of the image. A last example is the use of Mid-Air by Mason *et al.* [85] to test their method for remote tracking of UAVs swarms using Long Range Wide Area Network (LoRaWAN).

3 M4Depth: a network for robust and generalizable depth estimation

Summary

Overview

In this chapter, we consider the task of estimating depth in challenging environments, such as unstructured ones, from RGB images captured by a camera that has a known motion with six degrees of freedom. We first mathematically formalize this task. We then propose a new method, called M4Depth, that is designed to address the shortcomings observed in existing methods when considering depth estimation in unstructured environments, and in generalization. Finally, we perform an extensive set of experiments which allows us to assess the performance of our method in various setups, including both structured and unstructured environments, and to compare it to existing methods. This set of experiments also allows us to discuss the strengths and limitations of our method.

Data availability All the code written to get the results presented in this chapter, which includes the implementation of our method, has been made publicly available and can be found on the following GitHub repository: https://github.com/michael-fonder/M4Depth.

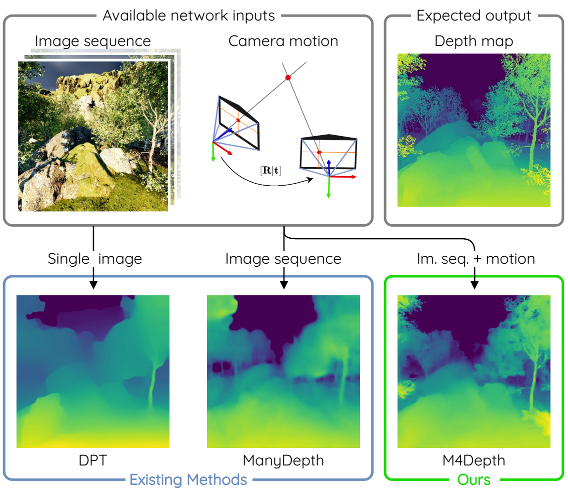


Figure 3.1: State-of-the-art depth estimation methods such as DPT [113, 114] or ManyDepth [139] struggle to produce accurate estimates in cluttered and natural environments. Our method, called M4Depth, outperforms existing methods in these instances and generalizes well to unknown environments.

3.1 Introduction

One of the key advantages of UAVs over other vehicles is their flight capability that allows them to reach places inaccessible by other means. As a result, UAVs are expected to work in a wide variety of environments, and to see these environments from a multitude of different points of view. However, due to a lack of alternatives, existing depth estimation methods targeting outdoor applications are exclusively benchmarked against datasets designed for autonomous driving in urban environments, which makes their transferability to UAV-related depth estimation unknown for two distinct reasons.

First, as explained in Section 1.3.2, some environments display richer visual cues about depth than others. This is particularly the case of urban areas, which contain many objects with a specific structure (cars, roads, signs, buildings, *etc.*) that indirectly provide cues about depth. In contrast, natural environments display much weaker visual cues about depth by their inherent lack of structure. As a result, estimating depth in unstructured environments, such as natural landscapes, is more challenging than in structured environments. Since existing methods were exclusively trained and tested in structured environments with strong visual cues about depth, their ability to deal with weaker visual cues is unknown.

Second, the motion of a car on a road is strongly constrained when compared to that of a drone flight. Therefore, a camera mounted on a drone, which moves with six degrees of freedom (6-DoF), sees its surrounding with points of view never seen by a camera mounted on a car. As a result, it is unsure if existing depth estimation methods that perform well on autonomous driving benchmarks perform equally well when the camera is allowed to see its surrounding from more points of views.

We suspect that two design characteristics of existing methods could hinder their generalization capability. First, existing methods are all designed to directly infer depth or its inverse, known as disparity. As such, networks are trained to reproduce an output distribution that is directly related to the content of the training data, which is problematic as the distribution of depth can change from one place to another. As a result, existing methods have a limited generalization capability to environments where the depth distribution differs from the one used for training. Second, methods relying on plane-sweeping cost volumes build them from depth or disparity intervals, which requires choosing two arbitrary parameters during the design of the method. These parameters are chosen according to the expected depth distribution of the environment, which constraints the performance and the generalization properties of the underlying network, as highlighted by Schröppel *et al.* [120].

In this chapter, we present a method, called M4Depth, designed to address the task of generalizable depth estimation in challenging environments such as unstructured ones, where visual cues about depth are scarce. To overcome one of the shortcomings of existing methods, that is their generalization capability to different depth distributions, we introduce a deep neural network that is designed to infer a notion of visual parallax, with the visual parallax being the perceived frame-to-frame displacement of a pixel in an image sequence. In our formulation, we decouple the visual parallax from the depth values by using the known camera motion. This allows us to decouple the distribution learned by the network from the specific depth distribution of the dataset used for its training, which proved effective to achieve better performance in generalization. In addition, we base our architecture on plane-sweeping cost volumes built from parallax intervals, which contributes to further improve the generalization capability of our network in two distinct ways. First, cost volumes are architectural elements that convert visual information into spatial information. Second, building our cost volumes from parallax intervals instead of depth intervals frees us from the limiting need to choose the arbitrary parameters highlighted by Schröppel *et al.* [120].

This chapter is structured as follows. In Section 3.2, we formalize the task of depth estimation for UAVs. Then, in Section 3.3, we present the notion of visual parallax used to build a new depth estimation method, M4Depth. In Section 3.4, we describe our M4Depth method. We test M4Depth with several experiments presented in Section 3.5. This section describes the experimental setups which are aimed at evaluating methods on unstructured environments as illustrated in Fig. 3.1, in generalization, and on the visual robustness benchmark introduced with Mid-Air. In that section, we also present our results, and discuss our method. Finally, Section 3.6 concludes the chapter.

The main contributions of this chapter can be summarized as follows:

1. We define a notion of visual parallax between two frames from a generic six-degree-of-freedom (6-DoF) camera motion, and present a way to build cost volumes with this parallax;

2. We present a novel lightweight multi-level architecture, called M4Depth, that is based on these cost volumes, designed to perform end-to-end depth estimation on video streams acquired in unstructured environments, and suitable for real-time applications;
3. It is shown that M4Depth, is state-of-the-art on our Mid-Air dataset, that it has good performances on the KITTI dataset [38], and that it outperforms existing methods in a generalization setup on the TartanAir dataset [137].

3.2 Problem Statement

We now present the technicalities of the problem we want to solve. We consider a camera rigidly attached to a vehicle moving within an unknown static environment. The intrinsic parameters of the camera are supposed to be known and constant. We introduce the following components and notations:

- I_t is an RGB image of size $H \times W$ recorded by the camera at time step t. Images have the following properties: (1) motion blur and rolling shutter artifacts are negligible; (2) the camera focal length f is known and constant during a flight; (3) the camera shutter speed and gain are unknown, and can change over time.
- $\mathbf{T}_t$ is the transformation matrix encoding the motion of the optical center of the camera from time step $t-1$ to t. As this matrix is computed for monitoring the state of the vehicle, we assume that it is available for our method as well.
- $\mathrm{z}_{ij,t}$ is the z coordinate (in meters) of the point recorded by the pixel at coordinates (i,j) of the frame I_t with respect to the camera coordinate system.

Using these notations, a depth map $\mathbf{d}_t$ is an array of $\mathrm{z}_{ij,t}$ values with $ij \in \{1,...,W\} \times \{1,...,H\}$.

We denote by h_t the complete series of image frames and camera motions up to time step t. We define a set $\mathcal{D}$ of functions D that are able to estimate a depth map $\mathbf{d}$ from h_t, that is $\hat{\mathbf{d}}_t = D(h_t)$, such that $D \in \mathcal{D}$, with $h_t = [I_0, [I_1, \mathbf{T}_1], \ldots, [I_t, \mathbf{T}_t]]$. Our objective is to find a function D in this set that best estimates $\mathbf{d}_t$. This definition implies that depth has to be estimated for the latest available image, and prevents the use of upcoming information. As a result, our problem has a causality constraint that generic SfM or MVS methods do not have.

Since collision avoidance is essential for autonomous vehicle applications, errors in the estimate of distance for closer objects should have a higher impact than equivalent errors occurring for objects in the background of the scene. This is taken care of by constructing a dedicated loss function for training and by minimizing the error relative to the distance of the object.

3.2.1 Performance metrics

The performance of a depth estimation method can be summarized by a set of seven metrics that were introduced by Eigen *et al.* [24] and that have been widely adopted by the field of research. These metrics are the absolute relative error (Abs Rel), the squared relative error (Sq Rel), the root-mean-square-error (RMSE), the root-mean-square error on the logarithm (RMSE log), and three threshold values. We will use the same set of metrics for assessing the performance of depth estimation methods in this work. Each metric of this set is defined for ground-truth depths z_{ij} and depth estimates $\hat{z}_{ij}$ belonging to a depth map $\mathbf{d}$ of size $H \times W$ pixels. When considering multiple depth maps, the global performance is averaged over all the maps.

For all error metrics, lower scores mean better performances —which we later remind by putting the symbol $\downarrow$ next to a metric— since the distance between estimates and their respective ground truths is lower. The expression of the error metrics are as follows:

$$\text{Abs Rel} = \frac{1}{HW} \sum_{z_{ij} \in \mathbf{d}} \frac{\left| z_{ij} - \hat{z}_{ij} \right|}{z_{ij}} \,, \tag{3.1}$$

$$\text{Sq Rel} = \frac{1}{HW} \sum_{z_{ij} \in \mathbf{d}} \frac{\left(z_{ij} - \hat{z}_{ij} \right)^2}{z_{ij}} \,, \tag{3.2}$$

$$\text{RMSE} = \sqrt{\frac{1}{HW} \sum_{z_{ij} \in \mathbf{d}} \left(z_{ij} - \hat{z}_{ij} \right)^2} \,, \tag{3.3}$$

$$\text{RMSE log} = \sqrt{\frac{1}{HW} \sum_{z_{ij} \in \mathbf{d}} \left(\log(z_{ij}) - \log(\hat{z}_{ij}) \right)^2} \,. \tag{3.4}$$

A threshold metric counts the fraction of pixels for which the ratio δ between the ground-truth and the estimate is below a given threshold. By construction, such a metric provides scores that are always between zero and one, with scores closer to one meaning better performances. Eigen *et al.* [24] propose to use three distinct thresholds that are $\delta < 1.25$, $\delta < 1.25^2$, and $\delta < 1.25^3$. The threshold metric, as defined by Eigen *et al.* [24], is formalized as follows:

$$[\delta < \text{thresh}] = \frac{1}{HW} \sum_{z_{ij} \in \mathbf{d}} F\left(z_{ij}, \hat{z}_{ij} \right) \,, \tag{3.5}$$

with $F(z, \hat{z}) = 1$ if $\max\left(\frac{z}{\hat{z}}, \frac{\hat{z}}{z} \right) < \text{thresh},\ 0$ otherwise.

During our experiments, we noticed that some of these metrics are redundant to compare the performance of different methods. For a purely comparative purpose,

Method	Supervision	Multi-frame	Recurrent	Cam. pose	Pre-trained on KITTI
Monodepth [42]	Self-sup.	No	No	No	Available
Monodepth2 [40]	Self-sup.	No	No	No	Available
ST-CLSTM [157]	Self-sup.	No	Yes	No	Not available
Wang [135]	Self-sup.	No	Yes	No	Not available
ManyDepth [139]	Self-sup.	Yes	No	Self-est.	Available
DeepV2D [131]	Supervised	Yes	No	Self-est.	Available
Our method (M4Depth)	Supervised	Yes	Yes	Given	Not applicable

Table 3.1: Main characteristics of a selection of depth estimation methods used for comparison in this thesis.

a subset of three metrics is sufficient. However, to be representative of the whole set of metrics, this subset of metrics must include a relative error metric, a root-mean-square error metric, and a threshold. In the following, we use such a subset to present performances in instances where appropriate. The subset we use in such instances comprises (1) the absolute relative error (Abs rel), (2) the root-mean-square-error on the logarithm (RMSE log), and (3) the $\delta < 1.25$ threshold.

3.2.2 Our Baseline

Based on the related work presented in Section 1.3.4, we have selected a representative set of existing methods for which the training code is available, as given in Table 3.1; they will constitute the baseline for our performance tests. In this table, we indicate for each method, respectively, the nature of its supervision mode, if it is based on a single or multiple frames, if it is recurrent, how it deals with the camera pose, and if weights for the KITTI are provided by the authors.

3.3 Deriving the bijective relation between depth and visual parallax

The apparent displacement of static objects between images captured with a moving camera is the parallax effect. Parallax is defined for a generic frame-to-frame transformation, and degenerates into standard stereo disparity when the frame-to-frame transformation amounts to a translation along the camera x or y axis. Like disparity, parallax conveys information about the distance of objects from the camera, but for an unconstrained camera baseline.

Previous works using parallax geometry [60, 117, 146] assume that frame-to-frame point correspondence, hence parallax, is known to derive 3-D rigidity constraint between pairs of points for recovering the 3-D scene structure without using the camera motion. In this work, we want to do the opposite; we want to use the constraints imposed by motion, whose parameters are provided by the on-board inertial measurement unit, and geometry to guide the inference of parallax along the epipolar lines. In the following, we establish that depth is a linear function of the inverse of the parallax when the latter is defined in a specific way.

3.3.1 Camera model

In this work, we formalize the concept of parallax for the standard pinhole camera model. We remind its main characteristics in this section.

In the pinhole camera model, the camera is represented by a sensor plane and a focal point, which is the optical center of the camera, taken as the origin (see Fig. 3.2). The focal point is located somewhere along the principal axis that intersects the sensor plane perpendicularly at its central point. The distance separating the focal point from the sensor plane is the focal length; this is expressed as a multiple of a sensor pixel width.

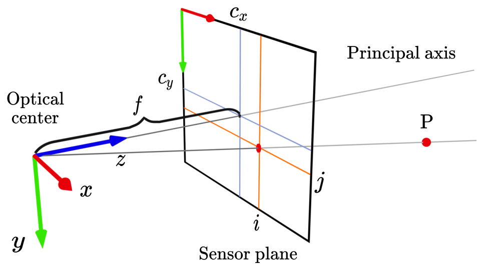


Figure 3.2: A diagram of the pinhole camera model with axes and notations.

In its simple expression, the pinhole model of a camera is characterized by five intrinsic parameters:

- f_x and f_y, the focal lengths along the x and y axes, respectively;
- s, the skew factor of a pixel, and
- (c_x, c_y), the coordinates of the principal point on the camera sensor.

These parameters can be regrouped in a matrix $\mathbf{K}$, called the *projection matrix*, as follows

$$\mathbf{K} = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}. \tag{3.6}$$

The pixel coordinates (i, j), in the camera plane, of the projection of a point located at (x, y, z) in the 3-D scene are obtained by using the camera intrinsic matrix $\mathbf{K}$ and

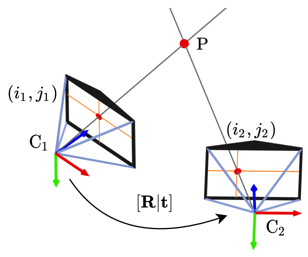


Figure 3.3: Illustration of moving the camera from position C_1 to C_2.

the right-angle theorem as follows:

$$(i, j) = \left(\frac{\alpha}{z}, \frac{\beta}{z}\right) \text{ with } \begin{bmatrix}\alpha\\\beta\\z\\1\end{bmatrix} = \begin{bmatrix}\mathbf{K} & \mathbf{0}\\\mathbf{0} & 1\end{bmatrix}\begin{bmatrix}x\\y\\z\\1\end{bmatrix}. \tag{3.7}$$

When the camera moves, the 3-D coordinates of a point can be expressed with respect to the coordinate system of the first frame instead of the one of the current frame. In this system, if we express the current camera position by a vector **p** of size 3 and its orientation by a 3×3 rotation matrix **R**, then, the position of a point (X, Y, Z) in space can be obtained by the following transformation

$$\begin{bmatrix}X\\Y\\Z\\1\end{bmatrix} = \underbrace{\begin{bmatrix}\mathbf{R} & \mathbf{p}\\\mathbf{0} & 1\end{bmatrix}}_{\mathbf{T}}\begin{bmatrix}x\\y\\z\\1\end{bmatrix}, \tag{3.8}$$

3.3.2 The geometry of a multi-view setup

The visual parallax is only computable when a moving camera sees a same point from different poses. It is therefore necessary to define the constraints that link the projection coordinates on the camera sensor of this same point seen from two distinct camera poses before defining our notion of visual parallax.

Let us assume that some visual information, a point P for instance, is visible from two different camera points of view. We denote the pose corresponding to each point of view by C_1 and C_2, and express the pose C_2 relative to the pose C_1, which is encoded by the transformation matrix $\mathbf{T}_2$ with

$$\mathbf{T}_2 = \begin{bmatrix}\mathbf{R} & \mathbf{t}\\\mathbf{0} & 1\end{bmatrix}. \tag{3.9}$$

As illustrated in Fig. 3.3, the point P seen by C_1 is projected on the sensor plane in (i_1, j_1) while being projected at a different location on the sensor of C_2, that is (i_2, j_2).

As we are only considering rigid camera motions, the camera-intrinsic parameters are the same for C_1 and C_2. Therefore, it is possible to simplify the notations by expressing the coordinates (i, j) relative to the principal point (c_x, c_y). This leads the expression of the camera-intrinsic matrix **K** to simplify into

$$\mathbf{K} = \mathrm{diag}(f_x, f_y, 1)\,, \tag{3.10}$$

if we also assume that the skew parameter s is negligible, which is common in practice.

Our notion of visual parallax requires to first express the relation that links (i_1, j_1) to (i_2, j_2). From Eq. (3.7), it can be seen that recovering the 3-D coordinates of a point whose projection coordinates (i, j) and depth z are known is simple if the intrinsic matrix is known (which is a common hypothesis in computer vision as long as the camera is not zooming). Assuming that P is located at a depth z_2 of C_2, its 3-D coordinates with respect to C_2 are given by

$$\mathbf{P}_{C_2} = \begin{bmatrix} i_2/f_x \\ j_2/f_y \\ 1 \end{bmatrix} z_2\,, \tag{3.11}$$

These coordinates are expressed with respect to the C_2 referential. Their expression in C_1 is given by

$$\mathbf{P}_{C_1} = [\mathbf{R}|\mathrm{t}] \begin{bmatrix} \mathbf{P}_{C_2} \\ 1 \end{bmatrix} = \mathbf{R}\mathbf{P}_{C_2} + \mathbf{t}\,. \tag{3.12}$$

Computing the projection coordinates (i_1, j_1) in C_1 for P is obtained by applying the camera projection equation (Eq. (3.7)), that is

$$z_1 \begin{bmatrix} i_1 \\ j_1 \\ 1 \end{bmatrix} = \mathbf{K}\mathbf{P}_{C_1}\,. \tag{3.13}$$

By combining Eq. (3.11), (3.12), and (3.13), we have

$$z_1 \begin{bmatrix} i_1 \\ j_1 \\ 1 \end{bmatrix} = \mathbf{K} \left(\mathbf{R}\, z_2 \begin{bmatrix} i_2/f_x \\ j_2/f_y \\ 1 \end{bmatrix} + \mathbf{t} \right)\,. \tag{3.14}$$

Equation (3.14) gives us the relation between the two coordinates (i_1, j_1) and (i_2, j_2) for a given location in the 3-D space

3.3.3 Definition of the visual parallax

Our notion of parallax denoted by ρ is established as follows. The transformation matrix $\mathbf{T}_t$ formalizing the known physical camera motion with 6 DoF between consecutive frames of the video stream can be broken down into a rotation matrix $\mathbf{R}_t$ and a 3-D translation vector $\mathbf{t}_t = \begin{bmatrix} \mathrm{t}_x & \mathrm{t}_y & \mathrm{t}_z \end{bmatrix}^T$. Thanks to Eq. (3.14), we know that a point P in space seen by the camera at two different time instants t and $t-1$, and projected at coordinates (i_t, j_t) in the current frame t, is linked to its previous coordinates (i_{t-1}, j_{t-1}) in frame at time $t-1$ by the motion $\mathbf{T}_t$ as follows

$$\mathrm{z}_{i_{t-1}j_{t-1}} \begin{bmatrix} i_{t-1} \\ j_{t-1} \\ 1 \end{bmatrix} = \mathbf{K}\left(\mathbf{R}_t \mathrm{z}_{i_t j_t} \begin{bmatrix} i_t/f_x \\ j_t/f_y \\ 1 \end{bmatrix} + \mathbf{t}_t\right), \tag{3.15}$$

where $\mathrm{z}_{i_t j_t} = \mathbf{d}_t(i_t, j_t)$ is the depth of the point P at time t, and $\mathbf{K}$ is the simplified version of the camera-intrinsic matrix described in Eq. (3.10).

To define our visual parallax, we first rewrite Eq. (3.15) as

$$\mathrm{z}_{i_{t-1}j_{t-1}} \begin{bmatrix} i_{t-1} \\ j_{t-1} \\ 1 \end{bmatrix} = \mathrm{z}_V\ \mathrm{z}_{i_t j_t} \begin{bmatrix} i_V \\ j_V \\ 1 \end{bmatrix} + \begin{bmatrix} f_x \mathrm{t}_x \\ f_y \mathrm{t}_y \\ \mathrm{t}_z \end{bmatrix}, \tag{3.16}$$

with

$$[\mathrm{z}_V i_V,\ \mathrm{z}_V j_V,\ \mathrm{z}_V]^T = \mathbf{K}\,\mathbf{R}\,[i_t/f_x,\ j_t/f_y,\ 1]^T . \tag{3.17}$$

From this equation, we can see that (i_V, j_V) are the coordinates of the point P in the plane of a virtual camera V whose origin is the same as the camera at time t but with the orientation of the camera at time $t-1$.

We now introduce our the parallax map ρ_t , where the pixelwise parallax $\rho_{i_t j_t} = \rho_t(i_t, j_t)$ is defined as the Euclidean norm

$$\rho_{i_t j_t} = \sqrt{\Delta_{i_t}^2 + \Delta_{j_t}^2} \tag{3.18}$$

where

$$\begin{bmatrix} \Delta_{i_t} \\ \Delta_{j_t} \end{bmatrix} = \begin{bmatrix} i_{t-1} - i_V \\ j_{t-1} - j_V \end{bmatrix}. \tag{3.19}$$

With this definition, the parallax is only a function of perceived pixel motion. It is therefore invariant to the particular combination of depth and camera motion.

After reorganization, using Eq. (3.16) and simplification, we get:

$$\begin{bmatrix} \Delta_{i_t} \\ \Delta_{j_t} \end{bmatrix} = \frac{1}{\mathrm{z}_{i_t j_t}\ \mathrm{z}_V + \mathrm{t}_z} \begin{bmatrix} f_x \mathrm{t}_x - \mathrm{t}_z i_V \\ f_y \mathrm{t}_y - \mathrm{t}_z j_V \end{bmatrix} . \tag{3.20}$$

It can be shown that $\mathrm{z}_{i_t j_t}\ \mathrm{z}_V + \mathrm{t}_z$ should rarely be negative when considering the physics of a scene and the camera motion of an autonomous vehicle. As a result, the parallax $\rho_{i_t j_t}$ can be computed as follows:

$$\rho_{i_t j_t} = \frac{\sqrt{(f_x \mathrm{t}_x - \mathrm{t}_z i_V)^2 + \left(f_y \mathrm{t}_y - \mathrm{t}_z j_V\right)^2}}{\mathrm{z}_{i_t j_t}\ \mathrm{z}_V + \mathrm{t}_z} . \tag{3.21}$$

This bijective expression links the parallax for a pixel to the depth of the corresponding point in space. Since parallax can be estimated from the RGB content of two consecutive images, we have a mean to estimate the depth by inverting the equation, yielding:

$$\mathrm{z}_{i_t j_t} = \frac{\sqrt{(f_x \mathrm{t}_x - \mathrm{t}_z i_V)^2 + \left(f_y \mathrm{t}_y - \mathrm{t}_z j_V\right)^2}}{\rho_{i_t j_t}\ \mathrm{z}_V} - \frac{\mathrm{t}_z}{\mathrm{z}_V} . \tag{3.22}$$

As expected, this expression becomes identical to the definition of the standard stereo disparity when the camera only moves along the x or y axis.

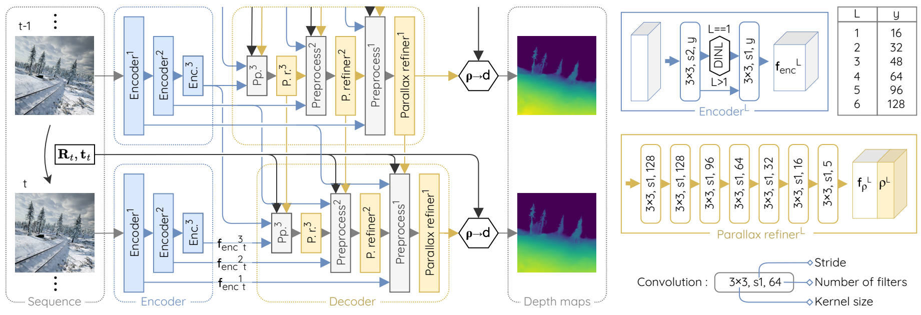
In practice, there are different ways to estimate $\rho_{i_t j_t}$, and in the method proposed in the next section, we build various proposals for $\rho_{i_t j_t}$ and let the network use them to compute the best estimate. Note that, once a parallax map ρ_t has been estimated, the (i_{t-1}, j_{t-1}) coordinates are given by a function Ψ, parametrized as follows

$$(i_{t-1}, j_{t-1}) = \Psi(i_t, j_t, \mathbf{T}_t, \rho_t) . \tag{3.23}$$

These (i_{t-1}, j_{t-1}) coordinates are defined on a continuous space instead of a discrete grid.

3.4 Description of a new method for depth estimation

Like other previous works, our method, named *Motion for Depth* (aka M4Depth), is based on a multi-level architecture that relies on cost volumes and that is trainable in an end-to-end fashion. The key novelty is that the network is designed to infer a parallax map, which is converted into a depth map by using motion information. The parallax map has several interesting properties that should make M4Depth more robust in generalization. This is described in the next section.



L	γ
1	16
2	32
3	48
4	64
5	96
6	128

Figure 3.4: Architecture overview of M4Depth (with three levels here), fed by two consecutive frames and the camera motion. Each parallax refiner produces a parallax estimate and learnable parallax features. All convolutions are followed by a leaky ReLU activation unit [147], except for the ones producing a parallax estimate. To ease the convergence, parallax values are encoded in the log-space. Details of the preprocessing unit are given in Fig. 3.5.

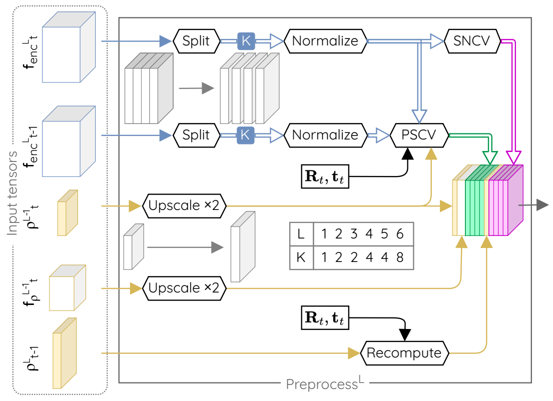
3.4.1 Definition of the architecture

Our primary motivation for inferring parallax rather than depth directly is driven by the need to produce a method that is robust, even in unseen environments. Training a network to infer depths drawn from a given data distribution will tie it to this distribution. Our formulation for the parallax consists of decoupling the value to infer from depth thanks to motion, and allows one to map many depth values to a same parallax value. As a result, a single learned parallax distribution can represent many depth distributions, which is a desirable ability for robustness and generalization.

As for optical flow, estimating the parallax can be done iteratively. Instead of simply iterating on a full network as proposed by Ummenhofer *et al.* [133], we approach the iterative process as a multi-scale pyramidal network, as PWC-Net [129]. By doing so, we embed the iterative process in the architecture itself. This architecture is an adaptation of the U-Net encoder-decoder with skip connections [115], where each level l of the decoder has to produce an estimate for the desired output, which in our case is a parallax map. In the decoder, the estimate produced at one level is forwarded to feed the next level to be refined. The levels of this type of architecture are generic and can be stacked at will to obtain the desired network depth.

Our architecture, illustrated in Fig. 3.4, uses the same standard encoder network as PWC-Net [129] with the only exception that we add a Domain-Invariant Normalization layer (DINL) [156] after the first convolutional layer. We use it to increase the robustness of the network to varied colors, contrasts and luminosity conditions without increasing the number of convolutional filters.

At each level L of the decoder, a small convolutional subnetwork is in responsible for refining the parallax map. We named it the parallax refiner. Its inputs are the



L	1	2	3	4	5	6
K	1	2	2	4	4	8

Figure 3.5: Details of the operations performed by our preprocessing units. Its building blocks do not feature any learnable parameters; they are detailed in Section 3.4.2. The split layer subdivides feature vectors into K sub-vectors to be processed in parallel in subsequent steps. We give the value of K that we use for an architecture with six levels.

upscaled parallax estimate made by the previous decoder level in the architecture and a series of preprocessed data generated by a preprocessing unit.

The preprocessing unit is illustrated in Fig. 3.5. It is made of fixed operations and has no learnable parameters. Its purpose is to prepare the input for the next parallax refiner.

In short, the preprocessor has two main purposes. First, it adapts the vectors of the feature maps produced by the encoders to make the network robust to unknown visual inputs. For that, it uses these data alongside camera motion to build two distinct cost volumes, the Parallax Sweeping Cost Volume (PSCV) and the Spatial Neighborhood Cost Volume (SNCV). Second, it recomputes the parallax estimate obtained for the previous time by adjusting it to the camera motion. These data are then concatenated and forwarded to the parallax refiner.

3.4.2 Building blocks of the preprocessing unit

In the following, we describe the components of the preprocessing unit and motivate their use.

Split and Normalize layers. The use of leaky ReLU activation units in the encoder can lead to feature maps containing plenty of small values. While classification or segmentation networks rely on the raw value of each entry in a feature vector, our network relies on the relative differences between neighboring feature vectors through the use of cost volumes. To achieve good generalization properties, this relative difference should remain significant in all situations. The split and normalize layers ensure that this is the case.

The split layer subdivides feature vectors into K sub-vectors to be processed in parallel in subsequent layers. It provides the network with the ability to decouple the relative importance of specific features within a same vector by assigning them to different sub-vectors.

The normalize layer normalizes the features of a same sub-vector and therefore levels the difference in magnitude of different sub-vectors. This is beneficial for the parallax refiner layers as this normalization leads the outputs of the cost volumes to span to a known pre-defined range. It also allows a full use of the information embedded in sub-vectors whose magnitude is small because of the leaky ReLU activation units.

Recompute layer. The parallax values estimated by the network are specific to the motion occurring between two given frames. By using the set of equations developed in Section 3.3.3 and if the camera motion is known, it is possible to compute the parallax values that should be observed at a given time step from a previous parallax estimate. The purpose of the recompute layer is to update the parallax values estimated for the previous frame to provide a hint in the form of a first estimate of the parallax values for the current frame.

Spatial Neighborhood Cost Volume (SNCV). This cost volume is computed from a single feature map $\mathbf{f}$ and is a form of spatial autocorrelation. Each pixel of the cost volume is assigned the cost of matching the feature vector located at the same location in the feature map with the neighboring feature vectors located within a given range r of the considered location

$$\mathrm{SNCV}_r(\mathbf{f})(i,j) = [\,\mathrm{cost}\,(\mathbf{f}(i,j), \mathbf{f}(i+p, j+q))\ \forall\, p,q \in \{-r,\ldots,r\}\,]\,, \tag{3.24}$$

where the cost of matching two vectors $\mathbf{x}_1$ and $\mathbf{x}_2$ of dimension N is defined as their correlation [22, 148]

$$\mathrm{cost}(\mathbf{x}_1, \mathbf{x}_2) = \frac{1}{N}\mathbf{x}_1^T\mathbf{x}_2\,. \tag{3.25}$$

The SNCV gives an indication about the two-dimensional spatial structure of the scene captured by the features of the encoder. Such a design makes it impossible to recover the feature vectors that led to a given cost value. Network parameters trained with this cost metric will therefore be invariant to changes in the input feature vectors if they lead to the same cost value. This can help us to obtain a robust and generalizable depth estimation network, which was not achievable by forwarding the feature map directly.

Parallax Sweeping Cost Volume (PSCV). This cost volume is computed from two consecutive feature maps $\mathbf{f}_{t-1}$ and $\mathbf{f}_t$, and a parallax map estimate $\hat{\rho}_t$ (see left of Fig. 3.5). For each pixel, the cost volume assigns the cost of matching the feature vector located at the same place in $\mathbf{f}_t$ with the corresponding reprojected feature vectors from $\mathbf{f}_{t-1}$ according to:

$$\mathrm{PSCV}_\delta(\mathbf{f}_t, \mathbf{f}_{t-1}, \hat{\rho}_t)(i,j) = \left[\mathrm{cost}\left(\mathbf{f}_t(i,j), \mathbf{f}_{\mathrm{reproj}}(i,j,\Delta_\rho)\right)\ \forall\, \Delta_\rho \in \{-\delta,\ldots,\delta\}\right]\,. \tag{3.26}$$

In that expression, the feature map $\mathbf{f}_{t-1}$ is reprojected for a range δ of parallax values equally distributed around a given estimate, that is

$$\mathbf{f}_{\text{reproj}}(i, j, \Delta_\rho) = \mathbf{f}_{t-1}\left(\Psi\left(i, j, \mathbf{T}_t, \max\left(\varepsilon, \hat{\rho}_t + \Delta_\rho\right)\right)\right) , \tag{3.27}$$

where Ψ is given by Eq. (3.23), and $\varepsilon > 0$. In this expression, $\mathbf{f}_{\text{reproj}}$ is interpolated from $\mathbf{f}_{t-1}$ since Ψ returns real coordinates , and $\max\left(\varepsilon, \hat{\rho}_t + \Delta_\rho\right)$ ensures the positiveness of the parallax used for computing the reprojection. Each vector element of the cost volume corresponds to one given parallax correction with respect to the provided estimate. Browsing through a range of parallax values for each pixel creates a series of candidates for the corresponding reprojected point. By searching for the reprojected candidate that is the most similar to the visual information observed at time step t, it is possible to assess which parallax is the most likely to be associated with each pixel.

Building the cost volumes from parallax intervals eliminates the contextual and arbitrary choices required otherwise (*i.e.* the range and the quantization step). As the parallax is defined within the image space, it is indeed bound to the image resolution. A good step size is therefore always 1 (pixel) and the maximum parallax value that can be encountered is equal to the diagonal of the image (in pixels). Since we are using a pyramidal architecture, a range δ at the L-th level is equivalent to a range $\delta 2^L$ in the original image. By stacking N levels, our architecture can theoretically cover a range of $\delta(2^{N+1} - 2)$ pixels while needing a total of only $N(2\delta + 1)$ samples to obtain a pixelwise resolution.

3.4.3 Definition of the loss function

Since the levels of our architecture are stackable at will, the architecture can have any depth. We now detail our loss function for a network that is made of M levels.

As in previous works [15, 56, 78], we use a multi-scale loss function. For each frame and each level, we compute the L_1 distance on the logarithm of the depths resulting from the conversion of parallax estimates using Eq. (3.22). The logarithm leads to a scale invariant loss function [24] and the use of an L_1 distance is motivated by its good convergence properties [13]. Since intermediate depth maps have a lower resolution, ground truths are resized by bilinear interpolation to match the dimensions of the estimates. The resulting terms are aggregated through a weighted sum, yielding

$$\mathcal{L}_t = \frac{1}{HW} \sum_{l=1}^{M} \sum_{z_{ij} \in \mathbf{d}_t^l} 2^{l+1} \left|\log(z_{ij}) - \log(\hat{z}_{ij})\right| . \tag{3.28}$$

The total loss for the sequence is defined as the average of the loss computed for each of its time steps.

3.5 Experiments

In this section, we present four experiments to analyze the performance of our method. For each of them, we detail the chosen dataset, the training (if appropriate), and discuss the results. Our first and second series of experiments aim to assess the performance in unstructured environments, as driven by our problem statement, and on a standard benchmark, respectively. For our last experiment, we test all the methods on the visual robustness benchmark introduced with Mid-Air in the previous chapter. The experiments present comparisons with baseline methods that might significantly differ, in terms of training procedure, number of parameters, etc. In order to disentangle the intrinsics of the training phase, we have devoted our third series of experiments to generalization tests.

We use the metrics from [24] for depth maps capped at $80m$ to compare the performance of each method. Additionally, we replicate the experiments performed on M4Depth with PWC-Net [129] to evaluate the benefits of our proposal over its parent. As PWC-Net is an optical flow network, we use Eq. (3.15) to obtain the frame-to-frame optical flow from depth and motion for training the network. During testing, we compute depth by assuming that the length of the optical flow vectors corresponds to the visual parallax.

Please note that we provide the codes used for baseline methods on GitHub[*], both for training and testing modes. Besides, we use the default test code given with the methods, which often rescale the estimated output depth maps to match the scale of the corresponding ground truth. We do not use such rescaling during the evaluation of our method. Finally, we made other multi-view methods causal by evaluating them on the last frame of the input sequence rather than on the frame in the middle.

3.5.1 Unstructured Environments

For our first experiment, we compare the performance of our method with those of the state of the art on a dataset featuring unstructured environments.

Mid-Air dataset. For this experiment, we use our Mid-Air dataset since it is the only one that meets all the assumptions of our problem statement (see Section 3.2) and whose train and test splits feature different places. This latter point is important because this will allow us to test methods in places that have the same data distribution than the train set, but that were not seen during their training. The first performance reported on Mid-Air for depth estimation was provided in 2022 by Miclea and Nedevschi [95]. However, they did not provide the details required to reproduce their train and test splits, and their results. As a result, we will use the splits defined in the previous chapter (see Section 2.4). Nonetheless, we subsample

[*]See https://github.com/michael-fonder/M4Depth-Baselines.

Method	Test size	Abs Rel ↓	SQ Rel ↓	RMSE ↓	RMSE log ↓	$\delta < 1.25$ ↑	$\delta < 1.25^2$ ↑	$\delta < 1.25^3$ ↑
Monodepth [42]	384×384	0.314	8.713	13.595	0.438	0.678	0.828	0.895
Monodepth2 [40]	384×384	0.394	5.366	12.351	0.462	0.610	0.751	0.833
ST-CLSTM [157]	384×384	0.404	6.390	13.685	0.438	<u>0.751</u>	0.865	0.911
Wang [135]	384×384	0.241	5.532	12.599	0.362	0.648	0.831	0.911
ManyDepth [139]	384×384	0.203	3.549	10.919	0.327	0.723	<u>0.876</u>	<u>0.933</u>
PWCDC-Net [129]	384×384	**0.095**	**2.087**	<u>8.351</u>	<u>0.215</u>	0.887	0.938	0.962
M4Depth-d6 (Ours)	384×384	<u>0.105</u>	<u>3.454</u>	**7.043**	**0.186**	**0.919**	**0.953**	**0.969**

Table 3.2: Performance comparison on our test set of Mid-Air. Here, a 6-level version of M4Depth is compared to the baseline methods. Scores correspond to the best performance obtained out of five individual network trainings. The best score for a metric is highlighted in bold and the second best is underlined.

the frame rate by a factor of four (from 25 to 6.25 fps) to increase the apparent motion between two frames. Besides, for all our experiments, images and depth maps were resized to a size of 384×384 pixels. We used bilinear interpolation to resize color images and the nearest-neighbor method for depth maps.

Training. We use the He initialization [50] for our variables and the Adam optimizer [65] for the training itself. We used the default moment parameters for the latter ($\beta_1 = 0.9$, $\beta_2 = 0.999$). The learning rate was set to 10^{-4}. We trained our network with six levels. All our trainings are performed on sequences of four time steps and with a batch size of three sequences. The network was trained on a GPU with 16 GB of VRAM for 220 k iterations. After each epoch, we computed the performance of the network on the validation set of the KITTI dataset to avoid any overfitting, and kept a copy of the best set of weights to be used for the tests after the training.

A series of data augmentation steps were performed on each sequence during the training to boost the robustness of our trained network to visual novelties. More precisely, we applied the same random brightness, contrast, hue, and saturation change to all the RGB images of a sequence and the colors of a sequence were inverted with a 50% probability. Finally, we randomly rotated the data of the sequence by a multiple of 90 degrees around the z-axis of the camera when training on Mid-Air. With these settings, a training takes approximately 30 hours.

Because of the lack of reproducible performances reported on Mid-Air, we had to train a selection of state-of-the-art methods drawn in Table 3.1 to build a baseline. The training details for the chosen methods are given in Section B.1 of the Appendix. We could not guarantee to get the best performance out of DeepV2D [131] because of the importance of its hyperparameters and the excessive duration of its training time. We, therefore, decided to discard it for this experiment.

Results. The results are reported in Table 3.2. In this table and following tables, the best score for a metric is highlighted in bold, and the second best is underlined.

Globally, it appears that M4Depth outperforms the baseline methods. However, it

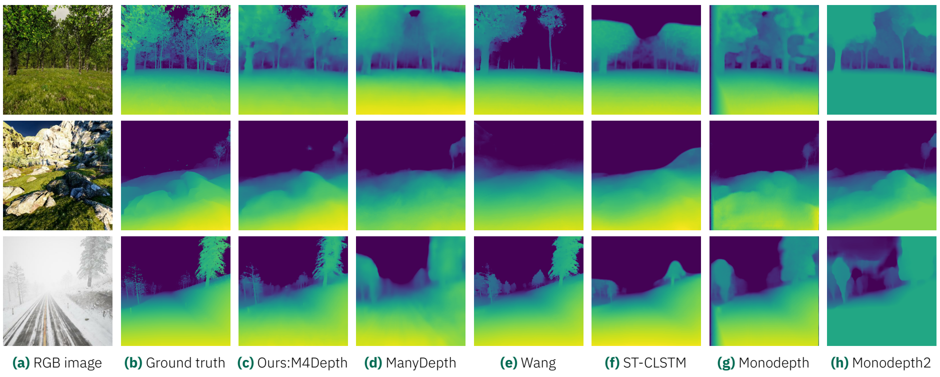


Figure 3.6: Comparison of the depth maps estimated by M4Depth and baseline methods. M4Depth recovers depth details more accurately than baseline methods.

slightly underperforms on the relative performance metrics when compared to PWC-Net. This observation, compared with the excellent performances on other metrics, indicates that our network tends to overestimate depth more often than other methods. A qualitative comparison of the outputs of the different methods is shown in Fig. 3.6. From this figure, we observe that although M4Depth lacks details in areas with sharp depth transitions, it recovers depth details more accurately than baseline methods, even for challenging scene elements such as forests or unstructured terrain.

3.5.2 Standard Depth Estimation Benchmark

The purpose of the second experiment is to assess the performance on a standard depth estimation benchmark.

KITTI dataset [38]. Most real datasets that provide RGB+D and motion data focus on cars driving in partially dynamic urban environments [38, 116, 127]. In this field, KITTI is the reference benchmark dataset when evaluating the performance of a depth estimation method. KITTI is not fully compliant with our problem statement: it has incomplete depth maps, there are some moving objects, and the camera has only three degrees of freedom, etc. Despite that, it is a good practical choice for performing tests on real data.

We used the dataset split proposed by [24]. The camera pose is estimated by a combined GPS-inertial unit and is therefore subject to measurement imperfections. Since a few samples were recorded in urban canyons where poor GPS reception induced erratic pose estimates, and as our method requires reliable pose estimates, we discarded these problematic samples from the splits. Additionally, we also subsampled the frame rate by a factor of two (from 10 to 5 fps) to roughly match the one of our Mid-Air sets. Finally, images were resized to 256×768 pixels.

Method	Test size	Abs Rel ↓	SQ Rel ↓	RMSE ↓	RMSE log ↓	$\delta < 1.25$ ↑	$\delta < 1.25^2$ ↑	$\delta < 1.25^3$ ↑
Monodepth [42]	256×512	0.114	0.898	4.935	0.206	0.861	0.949	0.976
Monodepth2 [40]	320×1024	0.106	0.806	4.630	0.193	0.876	0.958	0.980
ST-CLSTM [157]	375×1240	0.104	N/A	4.139	0.131	0.833	0.967	0.988
Wang [135]	128×416	0.077	0.205	**1.698**	0.110	0.941	0.990	**0.998**
ManyDepth [139]	320×1024	0.087	0.685	4.142	0.167	0.920	0.968	0.983
DeepV2D [131]	300×1088	**0.037**	**0.174**	2.005	**0.074**	**0.977**	**0.993**	0.997
PWCDC-Net [129]	256×768	0.152	2.015	5.883	0.251	0.828	0.920	0.956
M4Depth-d6 (Ours)	256×768	0.095	0.7084	3.515	0.146	0.898	0.962	0.982

Table 3.3: Performance of M4Depth (best of 5 trainings) on the KITTI dataset. The scores reported for reference methods are the ones published by their respective authors.

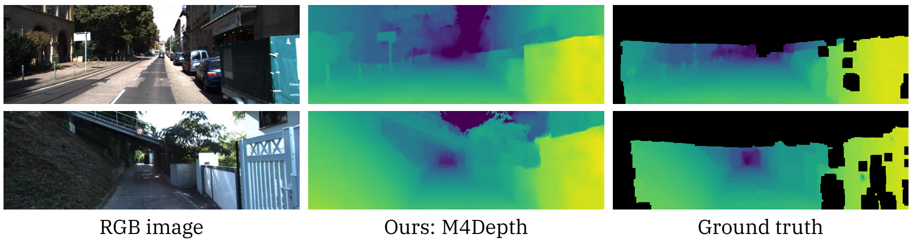
Training. For tests on KITTI, we reuse the weights of the network with 6 levels trained on Mid-Air and fine-tune them for 20 k additional iterations on a 50 – 50% mix of KITTI and Mid-Air samples. The fine-tuning is required to train our network to deal with large areas with poor textures and frame-to-frame illumination changes, as these characteristics are not present in Mid-Air. As the ground-truth depth maps for KITTI were generated from Lidar measurements, they are sparse and fine details are missing in the ground truths. Shortcomings created by these imperfections can be mitigated by fine-tuning on both datasets. During the fine-tuning, we also performed random color augmentation on the sequences. With these settings, the fine-tuning takes three hours.

Results. The performance of M4Depth with six levels on the KITTI dataset is reported in Table 3.3.

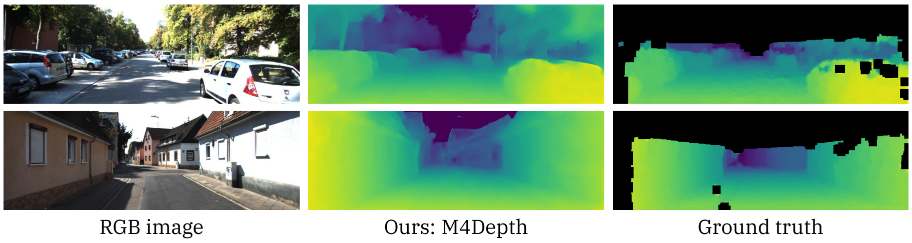
We observe that M4Depth has similar performances to current state-of-the-art methods. As expected, instances with dynamic elements or poor GPS localization lead to degraded performances. These results, however, prove that M4Depth also works with real data despite their imperfections. An overview of the outputs of our method on KITTI is shown in Fig. 3.7. Despite being trained in a supervised fashion on sparse data, our network manages to make accurate predictions on the whole image.

Fig. 3.7a shows the quality of the outputs of M4Depth in environments that meet all our hypotheses. M4Depth appears to preserve fine details and sharp edges. Fig. 3.7b shows that the network can also correctly estimate the depth of shiny or reflective surfaces and textureless areas. However, these surfaces remain challenging for the network and are not always handled as well as shown in these examples.

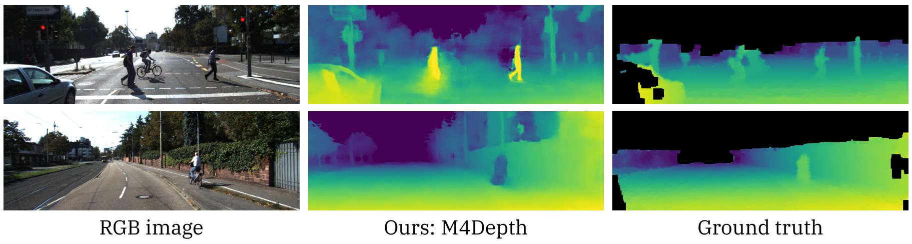
In our problem statement, we made the hypothesis that environments are static. Fig. 3.7c shows the behavior of M4Depth when this hypothesis is not met. The depth estimated for mobile objects is either largely under- or over-estimated depending on the relative motion perceived by the camera. As we use a pyramidal architecture, the reduction of the spatial dimension in the deeper layers of the architecture can lead to depth estimation artifacts that bleed around mobile objects, as seen in the fifth row.



(a) Example of depth estimate produced by M4Depth in environments meeting all hypothesis.



(b) Example of environment featuring adverse visual properties.



(c) Effect of the presence of camera independent motion in the RGB image on the depth estimate.

Figure 3.7: Comparison of depth maps estimated by M4Depth on the KITTI dataset with the corresponding interpolated ground truth.

3.5.3 Generalization

In this next experiment, we want to evaluate the generalization capabilities of all the methods. For this, we want to use static scenes that are semantically close to either the Mid-Air dataset (natural unstructured environments) or the KITTI dataset (urban environments), and test the performance of the method trained on Mid-Air (respectively KITTI) on the selected unstructured (respectively urban) scenes without any fine-tuning.

As we wanted to focus only on the generalization performance for depth estimation, we bypassed the pose estimation network for ManyDepth and DeepV2D, and used the ground-truth motion to generate the depth maps with these methods. Additionally, the depth maps produced by baseline methods are not guaranteed to be at the correct scale. To alleviate this issue in performance tests, we applied a median scaling to the depth maps of baseline methods.

Method	Test size	Abs Rel ↓	SQ Rel ↓	RMSE ↓	RMSE log ↓	$\delta < 1.25$ ↑	$\delta < 1.25^2$ ↑	$\delta < 1.25^3$ ↑
Monodepth [42]	384 × 512	0.929	21.950	19.116	0.992	0.231	0.430	0.590
Monodepth2 [40]	384 × 384	0.922	19.274	18.527	0.799	0.310	0.507	0.651
ST-CLSTM [157]	384 × 384	2.967	51.305	32.453	0.978	0.375	0.517	0.626
Wang [135]	384 × 512	0.761	25.459	31.875	1.482	0.209	0.313	0.411
ManyDepth [139]	384 × 384	0.776	16.551	16.822	0.746	0.326	0.538	0.684
PWCDC-Net [129]	384 × 512	0.343	7.645	17.731	0.684	0.584	0.716	0.786
M4Depth-d6 (Ours)	384 × 512	**0.281**	**5.348**	**11.875**	**0.524**	**0.715**	**0.806**	**0.856**

Table 3.4: Performance for the generalization test on the gascola unstructured environment from TartanAir. Scores were generated by using the same network weights as the ones used to report the performance on Mid-Air in Table 3.2.

Method	Test size	Abs Rel ↓	SQ Rel ↓	RMSE ↓	RMSE log ↓	$\delta < 1.25$ ↑	$\delta < 1.25^2$ ↑	$\delta < 1.25^3$ ↑
Monodepth [42]	384 × 512	1.765	147.3026	33.162	1.118	0.224	0.384	0.516
Monodepth2 [40]	384 × 384	1.651	67.815	24.543	1.058	0.286	0.437	0.561
ST-CLSTM [157]	384 × 384	2.552	40.452	27.338	0.878	0.370	0.518	0.609
Wang [135]	384 × 512	0.776	29.138	28.332	1.260	0.259	0.353	0.442
ManyDepth [139]	384 × 384	1.383	63.285	23.607	0.974	0.374	0.544	0.654
PWCDC-Net [129]	384 × 512	**0.516**	18.459	30.028	1.160	0.463	0.568	0.632
M4Depth-d6 (Ours)	384 × 512	0.537	**17.040**	**16.937**	**0.694**	**0.663**	**0.746**	**0.798**

Table 3.5: Performance for the generalization test on the season forest winter unstructured environment from TartanAir. Scores were generated by using the same network weights as the ones used to report the performance on Mid-Air in Table 3.2.

Method	Test size	Abs Rel ↓	SQ Rel ↓	RMSE ↓	RMSE log ↓	$\delta < 1.25$ ↑	$\delta < 1.25^2$ ↑	$\delta < 1.25^3$ ↑
Monodepth [42]	384 × 512	1.041	54.683	30.957	0.843	0.261	0.465	0.620
Monodepth2 [40]	320 × 1024	0.810	27.904	21.011	0.732	0.412	0.603	0.715
ManyDepth [139]	320 × 1024	0.942	42.846	22.508	0.757	0.432	0.607	0.714
DeepV2D [131]	384 × 512	1.5157	77.063	21.546	0.769	0.335	0.527	0.641
PWCDC-Net [129]	384 × 512	**0.376**	**12.66**	23.782	0.788	0.535	0.652	0.723
M4Depth (Ours)	384 × 512	0.509	24.283	**13.150**	**0.502**	**0.749**	**0.827**	**0.872**

Table 3.6: Performance for the generalization test on the neighborhood structured environment from TartanAir. Scores were generated by using the same network weights as the ones used to report the performance on KITTI in Table 3.3.

Method	Test size	Abs Rel ↓	SQ Rel ↓	RMSE ↓	RMSE log ↓	$\delta < 1.25$ ↑	$\delta < 1.25^2$ ↑	$\delta < 1.25^3$ ↑
Monodepth [42]	384 × 512	0.909	30.616	19.203	0.782	0.267	0.489	0.655
Monodepth2 [40]	320 × 1024	0.775	17.973	15.800	0.727	0.322	0.540	0.692
ManyDepth [139]	320 × 1024	0.759	20.895	15.604	0.649	0.351	0.583	0.730
DeepV2D [131]	384 × 512	0.694	18.777	7.551	0.498	0.494	0.722	0.830
PWCDC-Net [129]	384 × 512	0.338	8.679	16.760	0.703	0.627	0.741	0.800
M4Depth (Ours)	384 × 512	**0.256**	**6.759**	**7.211**	**0.370**	**0.804**	**0.880**	**0.918**

Table 3.7: Performance for the generalization test on the old town structured environment from TartanAir. Scores were generated by using the same network weights as the ones used to report the performance on KITTI in Table 3.3.

TartanAir dataset [137]. For this experiment, we use TartanAir. It is a synthetic dataset consisting of trajectories recorded by a free-flying camera in a series of different environment scenes. With each scene being relatively small in size, there is a lot of overlap in the visual content recorded for different trajectories within a same scene. As such, assembling clearly separated train and test sets drawn from the same data distribution is not possible. Despite this drawback, the diversity of the scenes makes TartanAir an interesting choice for testing the generalization capabilities of methods.

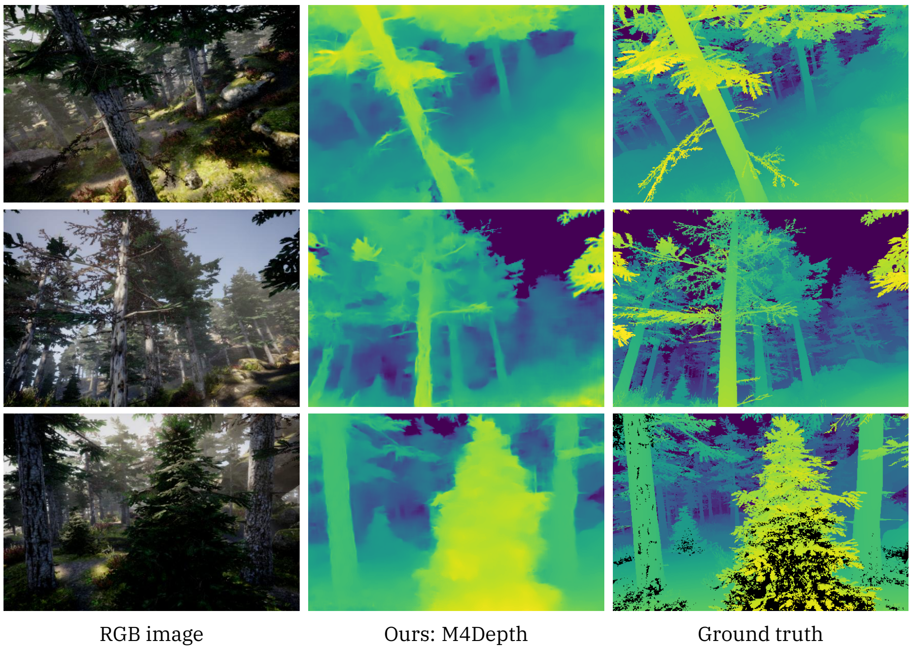
For the generalization test from the Mid-Air dataset, we selected the “Gascola” and “Season forest winter” scenes of TartanAir and used the weights trained for the baseline. For the one from the KITTI dataset, we selected the “Neighborhood” and “Old Town” scenes and used the pre-trained weights released by the authors of the methods.

We resized the images of this dataset to 384×576 pixels and subsampled the frame rate by a factor of two. Additionally, some scenes appear to have large, underexposed areas where there is no color information in the RGB frames. Having large pitch-black areas in an RGB image is unrealistic in practice, since cameras dynamically adapt their shutter speed depending on the exposure of the scene. To prevent the errors made by depth estimation methods in these areas from dominating the performance analysis, we discarded all the pixels for which the color in the RGB image had a value equal to zero.

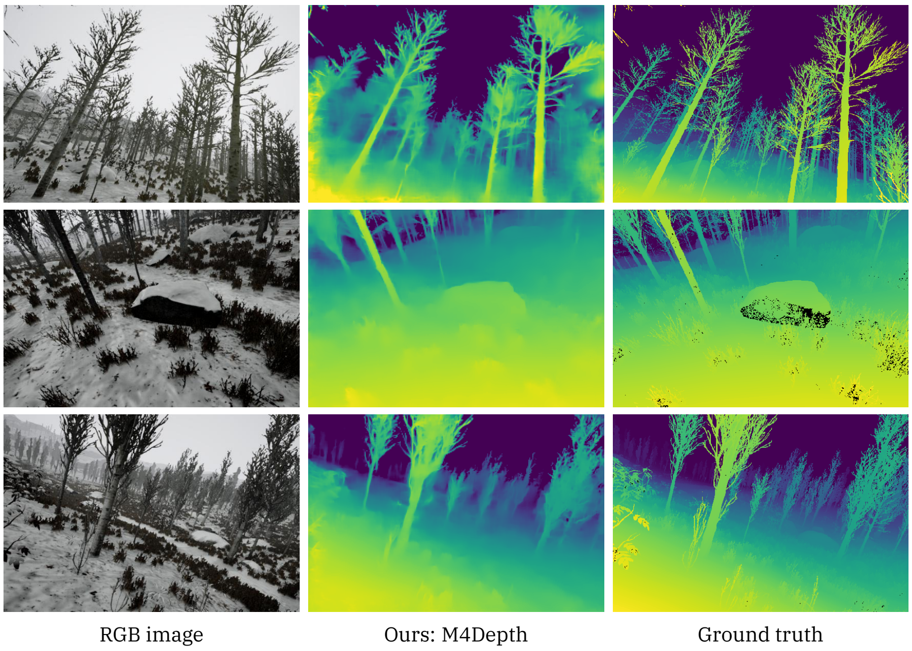
Results. The results of our experiments are reported in Tables 3.4 to 3.7. Overall, M4Depth outperforms the other methods with a significant margin both for structured and unstructured environments. As on Mid-Air, PWC-Net slightly outperforms M4Depth on some relative metrics, but not for both sequences. It is worth noting that the hierarchy of the performances has completely changed between the test on KITTI and the one in generalization, since our method outperforms DeepV2D [131] on the latter. These results therefore show the better generalization capability of M4Depth when compared to state-of-the-art methods.

Images obtained in generalization on TartanAir are shown in Fig. 3.8 and 3.9. It can be seen that the visual quality of the outputs in generalization is similar to that of the outputs produced on the dataset used for training. This confirms the strong performance of M4Depth in generalization as established with metrics.

Some general observations on the weaknesses of M4Depth can also be made from these outputs. First, the network cannot resolve all the details when the scene is too cluttered. This is especially visible in forest environments where tree branches overlap. Second, sometimes there are issues with sky recognition. This, however, is to be expected as our network mostly relies on perceived frame-to-frame pixel displacement to produce estimates. Finally, small and isolated structures such as cables are not always detected (see the outputs on urban scenes).

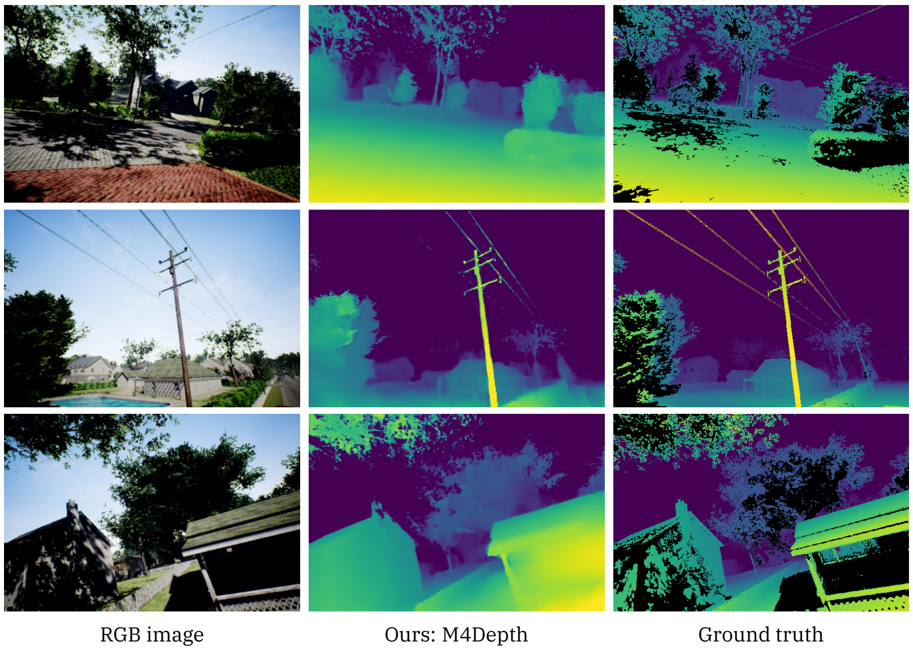


(a) Gascola scene.

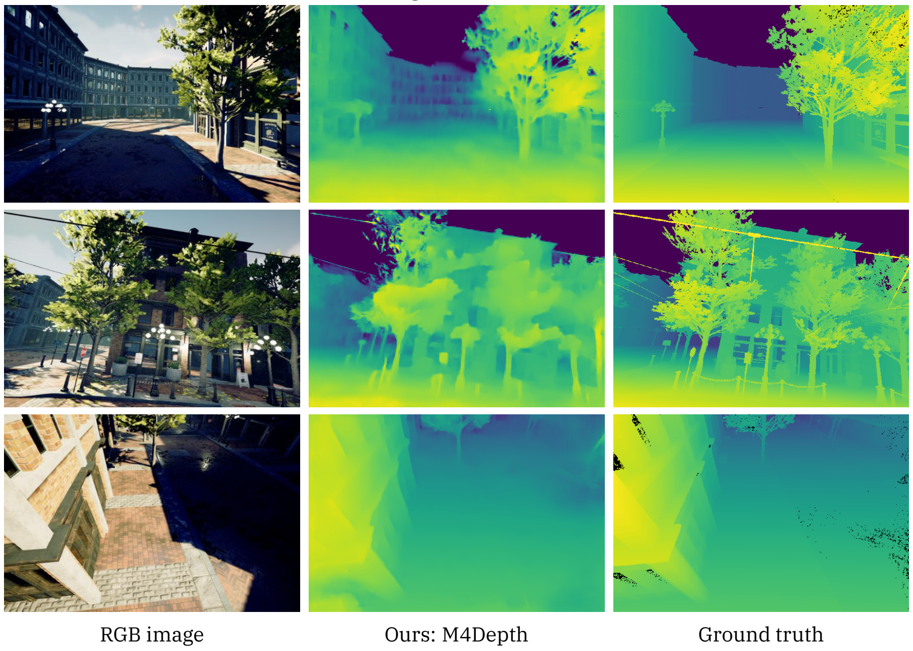


(b) Season forest (winter) scene.

Figure 3.8: Samples of depth maps produced in generalization on unstructured scenes of the TartanAir dataset by M4Depth with six levels trained on Mid-Air. Black areas in the ground truths correspond to pixels with no color information in the RGB image.



(a) Neighborhood scene.



(b) Old town scene.

Figure 3.9: Samples of depth maps produced in generalization on urban scenes of the TartanAir dataset by M4Depth with six levels trained on Mid-Air and fine-tuned on KITTI. Black areas in the ground truths correspond to pixels with no color information in the RGB image.

Method	Test size	Abs Rel		RMSE log		$\delta < 1.25$	
		Mean ↓	Rel. std ↓	Mean ↓	Rel. std ↓	Mean ↑	Rel. std ↓
Monodepth [42]	384×384	0.329	0.064	0.365	0.036	0.666	0.019
Monodepth2 [40]	384×384	0.281	0.039	0.340	0.030	0.637	0.044
ST-CLSTM [157]	384×384	0.428	0.097	0.553	**0.005**	0.522	0.022
Wang [135]	384×384	0.241	0.075	0.418	0.065	0.602	0.053
ManyDepth [139]	384×384	0.177	**0.008**	0.266	$\underline{0.006}$	0.739	0.010
PWCDC-Net [129]	384×384	0.083	0.036	0.192	0.027	0.885	$\underline{0.006}$
M4Depth-d6 (Ours)	384×384	0.097	$\underline{0.022}$	0.166	0.011	0.887	**0.003**

Table 3.8: Performance comparison on the weather variation sets of the visual robustness benchmark introduced with Mid-Air. The scores were generated by using the same network weights as the ones used to report the performance on Mid-Air in Table 3.2. The best score for the relative standard deviations of each metric is highlighted in bold, and the second best is underlined.

Method	Test size	Abs Rel		RMSE log		$\delta < 1.25$	
		Mean ↓	Rel. std ↓	Mean ↓	Rel. std ↓	Mean ↑	Rel. std ↓
Monodepth [42]	384×384	0.424	0.152	0.553	0.033	0.629	0.030
Monodepth2 [40]	384×384	0.491	$\underline{0.013}$	0.626	0.025	0.520	0.012
ST-CLSTM [157]	384×384	0.524	**0.010**	0.598	**0.001**	0.496	$\underline{0.007}$
Wang [135]	384×384	0.399	0.066	0.496	0.141	0.504	0.165
ManyDepth [139]	384×384	0.186	0.063	0.334	0.015	0.720	0.020
PWCDC-Net [129]	384×384	0.075	0.015	0.187	$\underline{0.006}$	0.912	**0.002**
M4Depth-d6 (Ours)	384×384	0.110	0.020	0.184	$\underline{0.015}$	0.913	**0.002**

Table 3.9: Performance comparison on the season variation sets of the visual robustness benchmark introduced with Mid-Air. The scores were generated by using the same network weights as the ones used to report the performance on Mid-Air in Table 3.2. The best score for the relative standard deviations of each metric is highlighted in bold, and the second best is underlined.

Ablation	Abs Rel ↓		SQ Rel ↓		RMSE ↓		RMSE log ↓		$\delta < 1.25$ ↑	
	MA	OT	MA	OT	MA	OT	MA	OT	MA	OT
SNCV	0.118	0.609	4.392	26.870	7.730	9.469	0.203	0.608	0.912	0.738
Normalize	0.099	0.583	3.179	23.495	7.032	9.019	0.185	0.496	0.920	0.770
DINL	0.104	0.521	3.480	19.378	7.182	8.826	0.189	0.536	0.915	0.763
$\mathbf{f}^{l-1}_{\rho,t}$	0.113	0.435	4.007	14.445	7.382	8.732	0.196	0.517	0.916	0.771
Split	0.113	0.366	4.074	9.937	7.424	7.900	0.196	0.478	0.914	0.762
ρ^{l}_{t-1}	0.107	0.435	3.482	15.071	7.201	8.836	0.197	0.437	0.911	0.788
M4Depth-d2	0.108	0.660	3.164	30.091	8.141	13.743	0.230	0.618	0.903	0.742
M4Depth-d4	0.114	0.330	4.124	12.569	7.405	8.391	0.196	0.399	0.916	0.809
M4Depth-d6	0.109	0.434	3.724	14.087	7.169	8.875	0.190	0.494	0.917	0.778

Table 3.10: Performance of M4Depth (trained on Mid-Air, averaged over 4 runs) for various architecture depths and ablations (on a network with 6 levels), and for a full architecture with 2, 4, and 6 levels, when tested on Mid-Air (MA) as well as in generalization on the old town scene (OT) of TartanAir.

3.5.4 Visual robustness benchmark

In this last experiment, we test the robustness of all the methods to visual changes. For this, we test all the methods trained on the Mid-Air dataset on the robustness benchmark for visual tasks introduced in the previous chapter (see Section 2.4.2). All the methods are tested on each climate setup separately. In order to get readable reports, we summarize the scores obtained for the sets belonging to a same environment (Kite for the weather variation, and PLE for the season variation) with a mean and a relative standard deviation. While the mean simply gives us the average performance of a method on a series of sets, the relative standard deviation gives an indication of the stability of the scores obtained over different sets, with higher stability translating into lower relative standard deviation.

Results. We report the results obtained for the tests on the weather variation sets and the season variation sets in Tables 3.8 and 3.9 respectively. No method emerges as being clearly better than all the others regarding the robustness to visual changes. With the exception of Monodepth [42] and the method of Wang *et al.* [135] that display comparatively poor robustness, we can note that the relative standard deviation of the other methods is relatively contained, as it is usually lower than 5%. In the best instances, it even drops below 1%, which can be considered as a sign of strong robustness to visual changes.

With relative standard deviations ranging between 0.2% and 2.2% over all the scores, M4Depth is the method that shows the best consistency over all metrics. These scores also show an overall low sensitivity to visual changes, which is a further strength of our method.

3.5.5 Discussion on the Architecture

Ablation study. We report the average performance over four trainings for ablated versions of our architecture in Table 3.10. The results indicate that the SNCV is the block that leads to the best performance boost. This highlights the benefits of giving some spatial information to the parallax refiners. The other blocks contribute to improving either test or generalization performances, but not both at the same time. As expected, the main contributors to generalization performances are the DINL and the normalization layer.

Increasing the number of levels in the architecture improves the performance. It should be noted, however, that the network tends to overfit the training dataset, therefore leading to worse generalization performance if the network gets too deep.

Overall, this ablation study indicates that a compromise between performance on the training dataset and performance in generalization has to be made.

Limitations. With our approach, large areas with no repetitive textures are prone to poor depth estimates. The feature matching performed by our cost volumes matching can indeed become unstable if large areas share the same features. This can therefore lead to bad depth estimates.

We mitigated this issue by using a multi-scale network and by including an SNCV at each of its levels, but these solutions do not make our network totally immune to this issue.

Inference speed. Our network has 4.5 million parameters and requires up to 500 MB of GPU memory to run with six levels. At inference time on Mid-Air, an NVidia Tesla V100 GPU needs 17 ms to process a single frame for a raw TensorFlow implementation. This corresponds to 59 frames per second which is roughly twenty-times faster than DeepV2D, the best-performing method on KITTI. According to NVidia's technical overview [102], this should translate to 5 fps, at least, on their Jetson TX2 embedded GPU. Such inference speed is compatible with the real-time constraints required for robotic applications, and can even be improved with inference optimizers such as TensorRT.

Interpretation of the results. As opposed to other methods, our network is designed exclusively to use the relative difference between feature vectors rather than relying on the raw semantic cues, *i.e.,* the raw value of the feature vectors, to estimate depth. All reference methods, even the ones based on cost volumes, forward the feature maps generated by their encoder directly to their depth estimation subnetwork. Doing so gives networks the ability to use semantic cues directly to estimate depth. This ability is only valuable for instances where the set of features possibly encountered can be encoded by the network and associated to a specific depth.

Our experiments indicate that reference methods perform well —better than M4Depth for some— on KITTI, the dataset with constrained and structured scenes. However, they fall behind in unstructured environments when the link between semantic cues and depth is weak, and in generalization when semantic cues are different from the reference. This tends to imply that baseline networks rely on the raw feature values to derive depth.

All these observations lead us to believe that severing the direct link between the encoder and the decoder of the architecture while proposing relevant substitute data through the preprocessing unit is the key factor that allows M4Depth to perform so well overall in our experiments.

3.6 Conclusion

In this chapter, we introduced M4Depth, a novel method for estimating depth from RGB image sequences acquired in unknown environments using a camera moving with six degrees of freedom. M4Depth is designed to be motion- and feature-invariant by relying on a notion of visual parallax which we have introduced and defined for generic camera motion. We showed how the visual parallax can be used to estimate depth without tying our network to a specific depth distribution, and presented an alternate way of building plane-sweeping cost volumes that uses the properties of parallax to solve shortcomings of traditional plane-sweeping cost volumes. Additionally, we introduced the Spatial Neighborhood cost volume to decouple the depth estimation from the specific values of the feature vectors encoding the images.

Our experiments on the Mid-Air and TartanAir datasets showed that the performance of M4Depth is superior to the baseline both in unstructured environments and in generalization. This proves its interest for monocular depth estimation in unstructured and unknown environments. Testing M4Depth on the standard KITTI dataset showed that it performs similarly to existing methods, which highlights its ability to also perform well in structured environments despite being primarily designed for unstructured ones. Furthermore, the results on the visual robustness benchmark of Mid-Air showed that the design of our method is robust to visual changes. Finally, the memory requirements and inference speed of our method are such that M4Depth can be considered for real-time applications on embedded GPUs, which is an asset for real-world applications.

4 M4Depth+U: a network for joint depth and uncertainty estimation

Summary

Overview

In this chapter, we show how M4Depth can be enhanced to perform joint depth and uncertainty estimation. As the enhanced method, referred as M4Depth+U, also works with parallax values, we present a custom-tailored solution to convert the uncertainty estimates related to the generated parallax values into uncertainty estimates related to depth, and show that it outperforms the standard probabilistic approach. Our experiments on various public datasets demonstrate that our method performs consistently, even in zero-shot cross-dataset transfer, and is robust to visual changes. Besides, our method offers compelling value when compared to existing multi-view depth estimation methods, as it performs similarly on a multi-view depth estimation benchmark despite being 2.5 times faster and causal, as opposed to other methods.

Data availability. All the code written to get the results presented in this chapter, which includes the implementation of our method, will be made publicly available on the following GitHub repository: https://github.com/michael-fonder/M4DepthU.

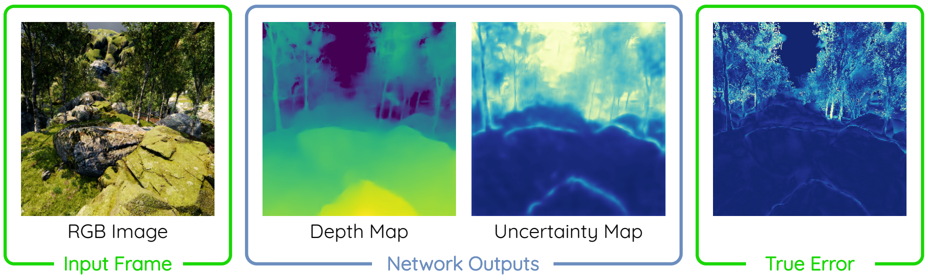


Figure 4.1: In this chapter, we present M4Depth+U, an adaptation of M4Depth, to jointly output an estimate for the most likely depth value and an uncertainty on this estimate for each point of the input frame. The purpose of the estimating uncertainty is to get a mean to anticipate erroneous depth estimates by having a function that behaves similarly to a performance metric without having access to ground-truth data. In the displayed maps, lighter colors correspond to higher uncertainty and error values.

4.1 Introduction

Since one of the many applications of depth estimation is to replace depth sensors in autonomous vehicles for path planning [99] or obstacle avoidance [134, 152], it is essential to anticipate potentially erroneous data in order to take action accordingly. Indeed, unpredicted errors on depth estimates can lead to collisions resulting in damage to the vehicle, or even harm people. While no method is completely error free, several works [59, 62, 120, 158] showed that anticipating the error on estimates produced by a deep neural network can be achieved by estimating the uncertainty on the outputs.

In 2017, Kendall and Gal [63] explained how the global uncertainty on the outputs of deep neural networks arises from two independent factors: (1) the uncertainty linked to the learned weight distribution, called the epistemic uncertainty, and (2) the uncertainty linked to noise in the input data, called the aleatoric uncertainty. They then proceeded to show that it is possible to get a pixel-wise estimate of these uncertainties for deep neural computer vision methods. Following works showed that uncertainty estimates are well correlated to the error observed on the outputs for various tasks [59, 62, 120, 158], therefore meaning that the uncertainty is a good estimate of the imprecision of the method. As a result, uncertainty can be used to anticipate potentially erroneous data and take action accordingly. For example, Yang *et al.* [150] leveraged uncertainty to perform visual odometry, and Yang *et al.* [151] proposed an algorithm that uses depth maps with the corresponding uncertainty as inputs to perform obstacle avoidance with UAVs. However, to the best of our knowledge, the task of joint depth and uncertainty estimation for the specific constraints of autonomous vehicles such as being robust to a wide variety of conditions and environments while being computationally lightweight enough to be able to run in real-time on limited hardware has not been addressed yet.

In Chapter 3, we presented M4Depth, a depth estimation method specifically designed for unstructured environments and UAV applications. In this chapter, we propose a strategy to perform a joint depth and uncertainty estimation, as shown in Fig. 4.1, by adapting the architecture of M4Depth. We detail the underlying problem statement in Section 4.2. Section 4.3 discusses the related works for uncertainty estimation. In Section 4.4, we detail our adaptation of M4Depth for joint depth and uncertainty estimation, referred to as M4Depth+U, and a strategy to convert the uncertainty estimates related to the parallax inferred by the network into uncertainty estimates related to depth. Our experiments are presented in Section 4.5. Our proposal is tested in various conditions including zero-shot cross-dataset transfer, and is compared to multi-view depth (MVD) estimation methods on an existing benchmark. Section 4.6 concludes this chapter.

Our main contributions are:

1. We propose the first method that addresses joint monocular depth and uncertainty estimation for the specific constraints of autonomous vehicles;
2. We test our method on three public datasets and show that the uncertainty estimate performs consistently in zero-shot cross-dataset transfer in different environments;
3. We test our method on a benchmark for MVS and show that its performance is on par with existing MVS methods for joint depth and uncertainty estimation. In addition, our method is causal, as opposed to these methods, and is 2.5 times faster.

4.2 Problem statement

In the previous section, we expressed the need to assess the quality of the estimate during inference, and explained that uncertainty maps are used for this purpose. We now formulate the related problem statement.

As for depth estimation, we consider a camera rigidly attached to a vehicle moving in an unknown static environment. The intrinsic parameters of the camera are supposed to be known and constant. Before formalizing the notion of uncertainty, we remind the following contextual elements that were introduced for depth estimation (see Section 3.2):

- I_t is an RGB image of size $H \times W$ recorded by the camera at time step t. Images have the following properties: (1) motion blur and rolling shutter artifacts are negligible; (2) the camera focal length f is known and constant during a flight; (3) camera shutter speed and gain are unknown, and can change over time.
- $\mathbf{T}_t$ is the transformation matrix encoding the motion of the optical center of the camera from time step $t-1$ to t. This matrix is assumed to be known, which is realistic when the camera motion is monitored, such as with drones.
- $\mathrm{z}_{ij,t}$ is the z coordinate (in meters) of the point recorded by the pixel at

coordinates (i, j) of the frame I_t with respect to the camera coordinate system.

- $\mathbf{d}_t$ is the depth map corresponding to the array of $\mathrm{z}_{ij,t}$ values with $ij \in \{1, \ldots, W\} \times \{1, \ldots, H\}$. With M4Depth, we have a function D that is able to estimate a depth map $\mathbf{d}$ from h_t, the complete series of image frames and camera motions up to time step t:

$$\hat{\mathbf{d}}_t = D(h_t),\ \text{with}\ h_t = [I_0, [I_1, \mathbf{T}_1],\ \ldots,\ [I_t, \mathbf{T}_t]]\ . \tag{4.1}$$

Without any loss of generality, we can assume that the performance of the depth estimator D can be evaluated by a set $\mathcal{E}$ of error metrics E that are functions of the ground-truth depth z_{ij} and the estimate $\hat{\mathrm{z}}_{ij}$, and that increase with the real error in each pixel location ij.

Using these notations, an uncertainty map $\mathbf{u}_t$ is an array of real positive values $\mathrm{u}_{ij,t}$ with $ij \in \{1, \ldots, W\} \times \{1, \ldots, H\}$. We want to find a function $\mathbf{u}_t = U(h_t)$ that has the same properties as these error metrics E in that they increase with the real error in each pixel location ij, but that does solely rely on the inputs of the method without knowing the ground truth. By doing so, the uncertainty map can be computed during inference to provide an estimate of the network performance.

In practice, the requirement for an uncertainty estimate to be strictly increasing with the local error is hardly achievable because several phenomena that affect depth estimation behave differently on a local basis. Therefore, it is more realistic to require the average error to increase with uncertainty. Mathematically, for two positive thresholds μ_a and μ_b such that $0 < \mu_a \leqslant \mu_b$, we require that:

$$\varepsilon\left\{ij \in \mathbf{d}_t | \mathrm{u}_{ij} \leqslant \mu_a\right\} \leqslant \varepsilon\left\{ij \in \mathbf{d}_t | \mathrm{u}_{ij} \leqslant \mu_b\right\}, \tag{4.2}$$

where $\mathcal{P} = \{ij \in \mathbf{d}_t | \ldots\}$ denotes a set of pixel $\mathcal{P}$ on which we calculate the average error $\varepsilon\{\mathcal{P}\}$ according to a metric E, as defined by

$$\varepsilon\{\mathcal{P}\} = \frac{1}{\#(\mathcal{P})} \sum_{ij \in \mathcal{P}} E(\mathrm{z}_{ij}, \hat{\mathrm{z}}_{ij}), \tag{4.3}$$

with $\#(\mathcal{P})$ being the cardinality of the set.

Uncertainty estimate performance metric

In order to exploit an uncertainty map in a practical application, we need to trust the given values. The quality of an uncertainty estimate is computed by using so-called sparsification plots. While having been first introduced for optical flow uncertainty analysis [68, 72, 82, 138], sparsification plots have become a standard tool for analyzing the performance of uncertainty estimates for other tasks including depth estimation. They show how a performance metric for the considered task, depth estimation in our case, changes when pixels with the largest uncertainty are progressively removed from its computation, as illustrated on Fig. 4.2. Formally, a sparsification curve $S()$ is parameterized by a fraction factor x applied to an

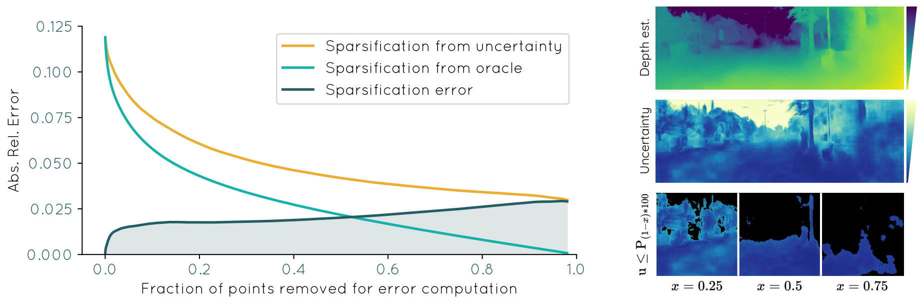


Figure 4.2: Example of sparsification curves obtained from experiments with the KITTI dataset. The area highlighted in the graph on the left is the Area under the Sparsification Error (AuSE), a value commonly used to assess the performance of a given uncertainty estimate. The Absolute Relative (Abs. Rel.) error is one of the performance metrics introduced by Eigen *et al.* [24] for depth estimation. The pictures on the bottom right illustrate how pixels are selected in the uncertainty map according to a given sparsification fraction.

uncertainty map, and is defined as:

$$S(\mathbf{d}_t, x, \mathbf{u}_t) = \varepsilon\{ij \in \mathbf{d}_t | \mathrm{u}_{ij} \leqslant \mathrm{P}_{(1-x)*100}(\mathbf{u}_t)\}, \text{ with } x \in [0, 1[\ , \tag{4.4}$$

where $\mathrm{P}_i(\mathbf{u}_t)$ denotes the i-th percentile of the histogram of the uncertainty map $\mathbf{u}_t$.

As a result of our requirement on $\mathbf{u}_t$, S should be decreasing with the parameter x for a given uncertainty map $\mathbf{u}_t$ (see top curve of Fig. 4.2), and the best uncertainty measure is the one with the lowest area under its sparsification plot. However, it must be noted that this area is always lower bounded by the sparsification curve of the ground-truth error on the estimate. This curve is given by $S(\mathbf{d}_t, x, \mathbf{o}_t)$ where $\mathbf{o}_t$ is referred to as the oracle map in the literature and gives the ground-truth error on the depth estimate for each of its pixels o_{ij}:

$$\mathrm{o}_{ij} = E(\mathrm{z}_{ij}, \hat{\mathrm{z}}_{ij})\,. \tag{4.5}$$

Ilg *et al.* [59] proposed to summarize the performance of an uncertainty estimate with a single number to be minimized, the Area under the Sparsification Error (AuSE), that is the area between the sparsification plots of an uncertainty map and the oracle:

$$\text{AuSE} = \int_0^1 \left(S(\mathbf{d}_t, x, \mathbf{u}_t) - S(\mathbf{d}_t, x, \mathbf{o}_t)\right) dx\,. \tag{4.6}$$

As the evaluation of the performance of a depth estimation method usually relies on the set of metrics defined by Eigen *et al.* [24], the AuSE should be computed for each of them. The best uncertainty estimate for a given depth estimation network is then the one with the smallest AuSE for the largest number of depth estimation performance metrics. An uncertainty estimate that performs and generalizes well should have a low AuSE even on datasets that have a data distribution different from the training set.

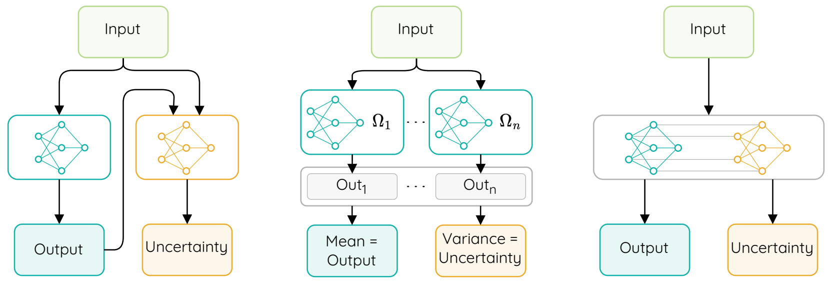
4.3 Related works

As uncertainty in neural networks in general is already well covered in the survey of Gawlikowski *et al.* [36] , we focus this section on works addressing uncertainty assessment for pixel-wise computer vision regression tasks. With the architecture of M4Depth being close to some methods for optical flow or stereo disparity estimation, uncertainty estimation methods developed for these tasks can find their use in our setup. Hence, we also consider them when reviewing the related works. This section is structured in three parts that each present distinct strategies, illustrated in Fig. 4.3, to get uncertainty estimates.

4.3.1 Posterior uncertainty assessment

The first proposals for quantifying uncertainty in optical flow or stereo matching [5, 7] long preceded AlexNet Krizhevsky *et al.* [70] and the deep learning revolution that followed. The first uncertainty measures proposed were handcrafted and consisted in comparing the maps produced by the algorithms to the inputs and to deriving an uncertainty measure from their combined features. For instance, Hu and Mordohai [54] conducted a review and comparison of 17 confidence measures for stereo matching, all based on image patches, that are computed directly from the cost of matching RGB values between the two input images. In another work, Bruhn and Weickert [10] analyzed confidence measures for variational optical flow and proposed an energy-based confidence measure. This measure is based on the use of the inverse of a global energy function to detect violations, and by extension low confidence areas, of their flow computation model. They showed that this measure works better than the then-standard image-gradient-based confidence measure introduced by Barron *et al.* [7]. Kondermann *et al.* [68] noticed that confidence measures for optical flow were often tied to a specific flow computation model and proposed a confidence measure based on sub-space projections of the flow fields that does not rely on a specific flow computation model.

While these methods showed concrete results, they all share a common drawback: they were crafted based on assumptions on the properties of the expected outputs, which makes them specific to a particular use case. Furthermore, the modeling of these properties, while being based on observations, remains arbitrary and may be incomplete or inaccurate. As a result, researchers turned towards machine learning to learn confidence measures [82, 109, 110]. As opposed to some handcrafted measures, learned ones are not built for a specific algorithm, which makes them generic and usable for any method addressing the same task. The extensive review of confidence measures for stereo disparity observation performed by Poggi *et al.* [110] showed that learned measures generally yield better results than handcrafted ones on specific datasets, but have no clear advantage over handcrafted methods in generalization.



(a) Posterior uncertainty **(b)** Epistemic uncertainty **(c)** Aleatoric uncertainty

Figure 4.3: Schematic overview of the different categories of uncertainties used in the literature. Posterior uncertainty is inferred from both the input and output of the method. Epistemic uncertainty models the uncertainty of the underlying neural network and is computed from different weight distributions Ω_i of a same network. Finally, aleatoric uncertainty models the uncertainty on the input data, and is inferred solely from the input of the method.

Assessing the uncertainty only from the inputs and outputs of a method comes, however, with a computational cost. It indeed requires waiting for the output to be available before computing the uncertainty, which can be an issue for real-time applications depending on the joint complexity of the chosen methods.

4.3.2 Multiple model samplings for epistemic uncertainty

Deep learning-based methods led to significant performance improvements for various computer vision tasks when compared to previous algorithms. One key aspect of deep learning is that, unlike previous deterministic algorithms, the output of a deep learning model for a given input can change depending on the training setup. This variation of the outputs at inference can be leveraged to obtain the uncertainty of the model on its output, known as *epistemic uncertainty*. In practice, estimating the epistemic uncertainty is achieved by performing variational inference, *i.e.,* by using different weight distributions of the same architecture to generate varying outputs for a same input. The variance between outputs then serves as a direct indicator of the model uncertainty [59, 63, 108]. Variational inference can be done either with ensemble methods or with Bayesian networks.

Ensemble methods use several distinct instances of the same architecture to get different weight distributions. A first example of an ensemble method is the Monte Carlo dropout [126], which consists of sampling multiple networks from the weights of a same trained architecture by applying random dropouts during inference. Another ensemble method is bootstrapping [75], which involves training multiple randomly initialized instances of the same architecture on different subsets of a same dataset. The variation in initialization and training sets yields multiple networks with different weights, therefore leading to variational inference. Finally, the snapshot ensemble method [55] takes advantage of cyclic learning rate schedules for obtaining several snapshots of a single a network during its training.

This allows to obtain multiple pre-converged instances of the network without having to train them all from scratch, as with bootstrapping.

Bayesian Neural Networks (BNN) replace the deterministic network's weight parameters with probabilistic distributions over these parameters and are optimized by marginalizing over all possible weights rather than choosing a point estimate [63]. In such networks, the epistemic uncertainty can theoretically be directly derived from the posterior probabilistic distribution of the output, but it unfortunately cannot be computed analytically in practice. However, Graves [44] showed that it can be approximate by a simpler distribution thanks to variational inference by using the probabilistic nature of the parameters.

As these methods can be applied to all neural networks, the epistemic uncertainty can be estimated in the same way for all architectures. However, getting the epistemic uncertainty is computationally expensive as it requires generating multiple output samples at inference. Therefore, the time required to get an uncertainty measure increases linearly with the number of samples to generate, which is unpractical for real-time applications.

4.3.3 Training a model for aleatoric uncertainty inference

While part of the uncertainty on the output of a method can be attributed to the parameters of the underlying model, it can also be due to noise in the input data. Kendall and Gal [63] refer to this part of the uncertainty as the *aleatoric uncertainty*. Assuming that the observation noise can vary depending on the input, its impact on the output can be captured by parametrizing the output as a probabilistic distribution that models the possible output values for a given input. For example, the outputs for a regression task can be modeled as being corrupted by Gaussian random noise. In this case, the output distribution is represented by its mean and variance. The task of a regression method then shifts into inferring the two parameters simultaneously to capture this distribution, the uncertainty on the output being directly linked to the variance of the estimated distribution.

Unlike epistemic uncertainty assessment, estimating the aleatoric uncertainty does not rely on variational inference. Instead, it can be estimated by teaching a network to learn the parameters of a probabilistic distribution that is assumed to represent the noise in the output. A network trained to estimate these parameters infers them in a single forward pass, which is more computationally efficient than variational inference. As a result, most works addressing uncertainty for computer vision tasks focus on the aleatoric part of the uncertainty.

Usually, regression methods rely on a L1 loss function to train their neural network because of its better performance. The output noise of a method trained on the L1 distance is assumed to follow a Laplace distribution [59, 62, 120, 158] as it allows learning the parameters of this distribution with a Maximum Log-Likelihood loss function [59, 63, 108]. The main factor differentiating methods that infer aleatoric uncertainty with this assumption is the architecture used to estimate the parameters of the Laplace distribution, *i.e.,* the location and the scale. A common practice consists of estimating the location with an existing method suitable for a standard Maximum A Posteriori (MAP) regression and of adding a distinct additional output for the scale. The simplest way to achieve this is to simply add a channel for the second parameter at the output of the network [76, 93, 108, 158, 161]. In this case, the layers of the network have to learn parameters that are relevant for the inference of both the location and the scale. This can potentially lead to sub-optimal performance if the number of parameters available in the last layers is not sufficient to learn the additional features required for the new output. To avoid this issue, other methods create separated heads for the location and the scale by duplicating the last layers of their network [53, 66, 151]. Zhang *et al.* [159] even go as far as allocating a full sub-network to learn the scale back from the cost volume of their method. Ke *et al.* [62] proceed differently by using a two-step method that first estimates an optical flow map along several confidence measures before post-processing them alongside the original input to produce the distribution estimate. Finally, Su *et al.* [128] propose a multi-level MVS architecture that embeds uncertainty at its core to adapt the computation of its cost volumes.

4.4 Uncertainty estimation using M4Depth

Existing works showed that inferring the aleatoric uncertainty is an efficient way to get an uncertainty estimate when it is integrated within an existing method. In this section, we first show how the architecture of M4Depth, the depth estimation method presented in Chapter 3, can be adapted to perform joint depth and aleatoric uncertainty estimation. Then, we explain how uncertainty estimates related to depth can be obtained from this architecture, and how the network should be trained for this purpose.

4.4.1 M4Depth working principles and adaptation

M4Depth is a multi-level pyramidal architecture, illustrated in Fig. 4.4, where each level has the same structure and outputs a parallax estimate. The parallax ρ is a real positive value defined on the camera sensor plane that gives the distance between the projection coordinates of a same point P in the scene seen from two successive camera poses. This value is linked to the depth z of the point P by the motion of the camera between the two poses:

$$\mathrm{z} = \frac{\sqrt{(f_x \mathrm{t}_x - \mathrm{t}_z i_V)^2 + \left(f_y \mathrm{t}_y - \mathrm{t}_z j_V\right)^2}}{\rho\ \mathrm{z}_V} - \frac{\mathrm{t}_z}{\mathrm{z}_V}\,, \tag{4.7}$$

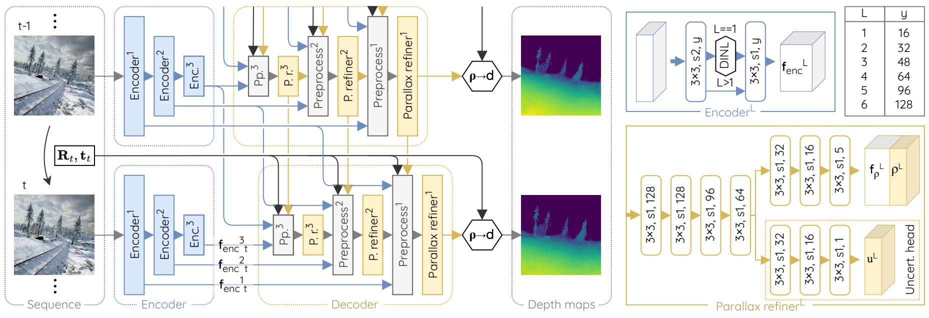


Figure 4.4: Architecture overview of M4Depth (with three levels here) fed by two consecutive frames and the camera motion. Each level L of the decoder has to upscale and refine an estimate for the parallax, ρ. The preprocessing units take the feature maps of two consecutive frames to compute two cost volumes used as input for the parallax refiner alongside the parallax estimate of the previous level. Each parallax refiner produces a parallax estimate ρ and learnable parallax features $\mathfrak{f}_\rho$. For uncertainty inference, we modified the parallax refiner part of the network by creating a separate head for the uncertainty u at each level.

where f_x and f_y are the respective focal lengths along the x and y camera axes, $\begin{bmatrix} \mathrm{t}_x & \mathrm{t}_y & \mathrm{t}_z \end{bmatrix}$ expresses the known translation of the camera between the two poses, and where i_V, j_V and z_V are solely functions of the projection coordinates (i, j) of P and the rotation of the camera between the two poses (see Section 3.3.3). By design, the network starts with a first rough low-resolution parallax estimate and then refines it progressively at higher resolutions to get the final estimate. Each intermediate parallax map can be converted into a depth map using Eq. (4.7) for each pixel.

As we need a dedicated output for the uncertainty, the architecture of M4Depth has to be adapted. For this purpose, we create a distinct head dedicated to the uncertainty in the parallax refiner sub-network of each level, the uncertainty being passed to the next level for refinement alongside depth. As shown in Fig. 4.4, the structure of the heads for depth and uncertainty is identical, but each can specialize for its own output. The number of layers to be assigned to different heads is a hyperparameter that needs to be chosen.

4.4.2 Correspondence between depth and parallax uncertainties

In the previous chapter, we trained M4Depth on a weighted sum of the L_1 distance of the logarithm of the depth for each architecture level l:

$$\mathcal{L}_t = \frac{1}{HW} \sum_{l=1}^{M} \sum_{\mathrm{z}_{ij} \in \mathbf{d}_t^l} 2^{-l} \left| \log(\mathrm{z}_{ij}) - \log(\hat{\mathrm{z}}_{ij}) \right| , \tag{4.8}$$

where depth estimates $\hat{\mathrm{z}}_{ij}$ are obtained from parallax estimates $\hat{\rho}_{ij}$ using Eq. (4.7).

This loss function needs to be adapted for training the network to jointly infer depth and an associated uncertainty estimate. To be able to get uncertainty estimates related to depth, we need to find the equivalent of Eq. (4.7) for uncertainty values. Stated otherwise, we need to find the relation linking uncertainty estimates on the parallax to uncertainty estimates on the depth.

In this section, we first detail the baseline approach, referred as M4Depth+U$_B$, which relies on the standard probabilistic framework to get depth uncertainties from M4Depth. We then present a new and more elaborate method, referred as M4Depth+U, to get depth uncertainty estimates from the parallax ones which, as confirmed by experiments, is better at evaluating the depth uncertainty.

To simplify the notations in the following, we rewrite Eq. (4.7) for a given pixel and a given camera motion as:

$$\mathrm{z} = Z\left(\rho\right) = \frac{a}{\rho} + c\,, \tag{4.9}$$

where

$$a = \frac{\sqrt{\left(f_x \mathrm{t}_x - \mathrm{t}_z i_V\right)^2 + \left(f_y \mathrm{t}_y - \mathrm{t}_z j_V\right)^2}}{\mathrm{z}_V} > 0 \quad \text{and} \quad c = -\frac{\mathrm{t}_z}{\mathrm{z}_V} \tag{4.10}$$

are independent of the depth of the considered point and can, therefore, be considered as constants in the upcoming mathematical reasoning.

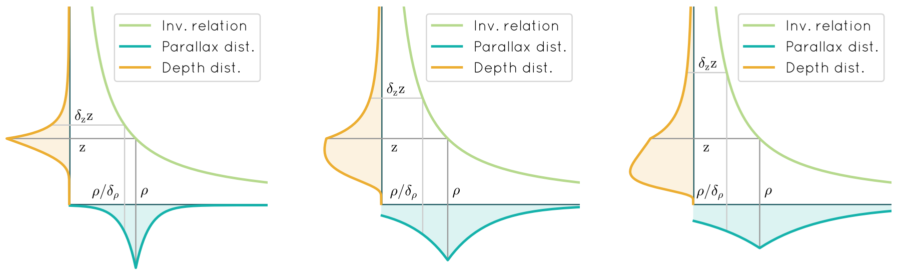
Baseline: the probabilistic framework

Since the aleatoric uncertainty is assumed to be proportional to the variance of the estimated output distribution, the natural solution to get the uncertainty on depth is to find the relation between the variance of the parallax output distribution and the variance of depth. From the literature, we know that training a network as the log-likelihood of the L_1 distance makes the assumption that its outputs follow a Laplace distribution. We can translate this to our depth estimation problem by training the network directly on depth with the loss function $\mathcal{L}_{U_\mathrm{z}}$:

$$\mathcal{L}_{U_\mathrm{z}} = \frac{\left|\mathrm{z} - \hat{\mu}\left(\mathrm{z}\right)\right|}{\hat{\sigma}\left(\mathrm{z}\right)} + \log\left(\hat{\sigma}\left(\mathrm{z}\right)\right)\,, \tag{4.11}$$

which amounts to assume that the depth z follows a Laplace distribution whose mean and standard deviation are equal to $\hat{\mu}\left(\mathrm{z}\right)$ and $\hat{\sigma}\left(\mathrm{z}\right)$, respectively.

Since M4Depth works with parallax values, we are looking for a relation linking the variance of the Laplace distribution on depth, $\mathbb{V}\left(\mathrm{z}\right) = \sigma^2\left(\mathrm{z}\right)$, and the variance $\mathbb{V}\left(\rho\right) = \sigma^2\left(\rho\right)$ of the corresponding distribution on the parallax. Unfortunately, parallax and depth are linked by an inverse relation, which creates issues when converting a distribution from one domain to the other. Indeed, the variance may not be finite in both domains at the same time when inverting a distribution. In addition, the



(a) Small standard deviation **(b)** Intermediate standard deviation **(c)** Large standard deviation

Figure 4.5: Illustration of the correspondence between a Laplace distribution (blue curve) and its inverse (orange curve) when applied to the relation linking parallax to depth for different standard deviations of the Laplace distribution. The modes of the distributions do not match for large standard deviation values. Therefore, we propose to use δ_ρ as an uncertainty measure on the parallax whose correspondence for depth is δ_z.

mode of the corresponding distributions may not be the same in both domains, as illustrated in Fig. 4.5, which is an issue for training the network. However, with the variable change $\zeta = {}^1/_\rho$, Eq. (4.9) becomes:

$$z = a\zeta + c\,, \tag{4.12}$$

and leads to a trivial link between the variance of the distribution on z and the one on ζ. Indeed, the variance $\mathbb{V}$ of a random variable affected by an affine transformation is given by:

$$\mathbb{V}(z) = \mathbb{V}(a\zeta + c) = a^2\mathbb{V}(\zeta) \Leftrightarrow \mathbb{V}(\zeta) = \frac{\mathbb{V}(z)}{a^2}\,. \tag{4.13}$$

This relation is convenient because of its simplicity. Training the network directly on depth with the loss function $\mathcal{L}_{U_z}$ then amounts to assuming that the inverse of the parallax, ζ, also follows a Laplace distribution whose mean and standard deviation are equal to $\frac{\hat{\mu}(z)-c}{a}$ and $\frac{\hat{\sigma}(z)}{a}$ respectively. With this workaround, the network simply needs to jointly infer the inverse of the parallax ζ and the standard deviation $\sigma(\zeta)$ to compute the distribution on the depth estimate.

Training loss function. During our experiments, we found out that M4Depth learns poorly when trained directly on the L_1 distance in the linear space. Therefore, training M4Depth+U directly on $\mathcal{L}_{U_z}$, as expressed by Eq. (4.11), also leads to poor results. Consequently, we propose a modified loss function that consists of learning the maximum a posteriori value for depth using Eq. (4.8), while learning the uncertainty in a separate term of the loss function by using Eq. (4.11). Since the maximum a posteriori value for the output of a network should correspond to the mode of the underlying distribution, we can assume that it corresponds to the mean of the Laplace distribution.

In practical terms, we train the network to infer the uncertainty related to the inverse parallax using the following loss function:

$$\mathcal{L}_{z,t} = \mathcal{L}_t + \frac{1}{HW}\sum_{l=1}^{M}\sum_{z_{ij}\in\mathbf{d}_t^l} g_{ij}2^{-l}\left[\frac{\oslash\left(\left|z_{ij}-\hat{z}_{ij}\right|\right)}{a u_{ij}} + \beta \log\left(a u_{ij}\right)\right], \tag{4.14}$$

where gradients are not propagated to the variables enclosed in the $\oslash()$ expression to avoid interference with the gradients generated by the $\mathcal{L}_t$ term of the loss, and where β is an arbitrary weighting factor for the uncertainty (we use the value $\beta = 0.02$ in our experiments). We noticed that the network does not converge properly if the uncertainty is trained on pixels belonging to the sky because of the large induced error terms. For this reason, we exclude these pixels from the uncertainty part of the loss thanks to g_{ij} where:

$$g_{ij} = 1 \text{ if } z_{ij} \leqslant 400, \quad 0 \text{ otherwise}. \tag{4.15}$$

In the following, we will refer at our modified version of M4Depth trained on this loss function as M4Depth+U$_B$.

Custom-tailored conversion of the uncertainty

One of the strengths of M4Depth is the direct link that exists between the parallax and the disparity sweeping cost volumes, which are the main sources of information available to infer the parallax. We assume that, as the cost volumes provide valuable information on the parallax, they should also provide valuable information on the related uncertainty. In addition, we assume that this information is best used if there is a trivial relation between the distribution to learn and the cost volumes. However, such trivial relation does not exist when learning the distribution of the inverse parallax because of the inversion process. As a result, the probabilistic approach is not well suited for M4Depth, and we propose another approach to get depth uncertainty estimates from the parallax domain.

In order to make the link between the uncertainty and the cost volumes, we propose, as an alternative to $\mathcal{L}_{U_z}$, to learn the uncertainty directly from the parallax as :

$$\mathcal{L}_{U_\rho} = \frac{|\rho - \hat{\mu}(\rho)|}{\hat{\sigma}(\rho)} + \log(\hat{\sigma}(\rho)) . \tag{4.16}$$

Similar to $\mathcal{L}_{U_z}$, this amounts to assume that the underlying data, parallax in this case, follows a Laplace distribution. In this case, $\hat{\mu}(\rho)$ is the mode of the distribution and corresponds to the estimated parallax $\hat{\rho}$. The corresponding estimated depth $\hat{z}$ can be computed thanks to Eq. (4.9). We then need to convert the uncertainty estimate represented by $\hat{\sigma}(\rho)$ to the depth domain. More precisely, we want to find a value $\Delta_z > 0$ in the depth domain that represents the uncertainty carried by $\hat{\sigma}(\rho)$ or, stated otherwise, for any corresponding pair $(\sigma(\rho), \Delta_z)$ and with everything else being equal, we want:

$$\sigma_1(\rho) < \sigma_2(\rho) \Leftrightarrow \Delta_{z1} < \Delta_{z2}\,. \tag{4.17}$$

Assuming that $\hat{\sigma}(\rho)$ is a valid indicator for the uncertainty, we derive a notion of relative uncertainty Δ_ρ defined as follows:

$$\Delta_\rho = \frac{\hat{\sigma}(\rho)}{\hat{\rho}} > 0\,. \tag{4.18}$$

This relative uncertainty can be used to define a factor δ_ρ such that:

$$\hat{\rho} + \sigma(\rho) = \delta_\rho\hat{\rho},\ \text{with}\ \delta_\rho = 1 + \Delta_\rho > 1\,, \tag{4.19}$$

which allows us to derive a range of values that is representative of the uncertainty as it monotonously increases with uncertainty:

$$\left[\frac{\hat{\rho}}{\delta_\rho}, \hat{\rho}\right] = \left[\frac{\hat{\rho}}{1+\Delta_\rho}, \hat{\rho}\right]\,. \tag{4.20}$$

As shown in Fig. 4.5, the equivalent of this range in the depth domain can be defined as $\left[\hat{z}, \delta_\rho\hat{z}\right]$ where $\hat{z} = Z(\hat{\rho})$. With the expression $\delta_\rho\hat{z}$ being similar to Eq. (4.19), we pose:

$$\delta_z = 1 + \Delta_z > 1\,. \tag{4.21}$$

With this definition, Δ_z has properties similar to that of Δ_ρ, and is also representative of the uncertainty on the parallax since Eq. (4.17) is verified.

To find the relation between Δ_z and Δ_ρ, we use Eq. (4.9) by substituting its variables values as follows:

$$\delta_z\hat{z} = \frac{\delta_\rho a}{\hat{\rho}} + c\,. \tag{4.22}$$

By rewriting this equation and taking into account that $\frac{a}{\rho} = z - c$, we derive the following relation between δ_ρ and δ_z, that also gives a relation between the relative uncertainties Δ_ρ and Δ_z:

$$\delta_z = \frac{c}{\hat{z}} + \delta_\rho\left(1 - \frac{c}{\hat{z}}\right) > 1\,. \tag{4.23}$$

Since $\delta_\rho > 1$ and $\hat{z} > 0$, the inequality is verified if:

$$\frac{c}{\hat{z}} = -\frac{t_z}{z_V\hat{z}} < 1 \Leftrightarrow z_V\hat{z} + t_z > 0\,, \tag{4.24}$$

which is the same condition of existence than for the parallax itself (see Section 3.3.3). Therefore, our δ_z and, by extension, Δ_z exist and are defined for any value of the parallax. The uncertainty on depth can then be simply associated to Δ_z. As such, the pair of values $\hat{z}$ and Δ_z has lost the probabilistic interpretation provided by $\mu(\rho)$ and $\sigma(\rho)$ since $\hat{z}$ is not always the mode of the inverse Laplace distribution (see Fig. 4.5). However, Δ_z still represents perfectly the uncertainty information carried by $\sigma(\rho)$ since Eq. (4.17) is verified.

Training loss function. Similarly to M4Depth+U$_B$, training the network directly with $\mathcal{L}_{U_z}$ yields poor results in practice, and for the same reasons, we use an adapted loss function. Here, we train the network using the following loss function:

$$\mathcal{L}_{\rho,t} = \mathcal{L}_t + \frac{1}{HW} \sum_{l=1}^{M} \sum_{\rho_{ij} \in \rho_t^l} 2^{-l} \left[\frac{\oslash\left(\left|\rho_{ij} - \hat{\rho}_{ij}\right|\right)}{u_{ij}} + \beta \log\left(u_{ij}\right) \right] , \tag{4.25}$$

where gradients are not propagated to the variables enclosed in the $\oslash()$ expression, and where β is an arbitrary weighting factor for the uncertainty (we chose $\beta = 0.05$ in our experiments). The ground-truth parallax ρ_{ij} used in this equation is obtained from the ground-truth depth using Eq. (4.7). In the following, we will refer to our modified version of M4Depth trained on this loss function as M4Depth+U.

In a nutshell, getting joint depth and uncertainty estimates with M4Depth+U amounts to training the network to infer the parallax and its related uncertainty, then to convert them into depth z and its related uncertainty Δ_z by using Eq. (4.9) and (4.23) respectively.

4.5 Experiments

In this section, we test the performance of M4Depth+U, our joint depth and uncertainty estimation method. We start by introducing the setup used for our performance analysis. We then assess the impact of the number of dedicated layers in the heads of the network, and compare our method to the probabilistic baseline, M4Depth+U$_B$, in zero-shot cross-dataset transfer on various datasets. Finally, we test it on two benchmarks proposed in the literature. The first aims at comparing our method to an existing baseline for MVS methods, and the second aims at testing the robustness of our method to visual changes.

4.5.1 Experimental setup

Datasets. The experiments presented in this section are all based on the same selection of datasets as the one used for M4Depth, that is Mid-Air [30], KITTI [38], and TartanAir [137]. Unless stated otherwise, we use strictly the same train and test splits, as well as the same resolution for the input images. Here, we use Mid-Air to train and test the method in unstructured environments, KITTI for zero-shot cross-dataset transfer tests on real data in urban environments, and TartanAir for further tests either in urban or unstructured environments.

Performance evaluation. We base our performance analysis on the subset of the metrics from Eigen *et al.* [24], namely the absolute relative error (Abs Rel), the root-mean-square error in the log domain (RMSE log), and the $\delta < 1.25$ threshold. Similar

Method	# head layers	Abs Rel		RMSE log		$\delta < 1.25$	
		Perf. ↓	AuSE ↓	Perf. ↓	AuSE ↓	Perf. ↑	AuSE ↓
M4Depth	N/A	0.127	–	0.185	–	0.907	–
M4Depth+U_B	1	0.145	0.028	0.190	0.084	0.906	0.009
M4Depth+U_B	2	0.132	0.027	0.186	0.084	0.908	0.008
M4Depth+U_B	3	0.137	0.046	0.185	0.131	0.907	0.011
M4Depth+U_B	4	0.132	0.039	0.183	0.103	0.908	0.011
M4Depth+U	1	0.134	0.007	0.188	0.020	0.906	0.006
M4Depth+U	2	0.141	0.007	0.193	0.019	0.901	0.006
M4Depth+U	3	0.138	0.006	0.191	0.018	0.905	0.006
M4Depth+U	4	0.141	0.007	0.193	0.019	0.901	0.006

Table 4.1: Ablation study of the impact on the number of layers dedicated to different heads. Reported performances correspond to the average of five networks trained and tested on the Mid-Air dataset.

to related works, distant points (points for which ground-truth depth is greater than $80m$) are excluded from the performance metric computations. This allows to exclude pixels that belong to the sky, which is desired since they take up a large part of the image while being of little interest for the analysis of uncertainty performance.

As mentioned in the problem statement (Section 4.2, the quality of an uncertainty estimate can be computed by using so-called *sparsification plots* [68, 72, 82, 138] and summarized with a single value to minimize, the Area under the Sparsification Error (AuSE). In the following, we report the AuSE for the three chosen depth estimation performance metrics.

Network training. All the performance tests reported and analyzed in this section are based on networks with six levels trained on the train set of the Mid-Air dataset. We use the same hyperparameters and the same data augmentation steps as the ones used for M4Depth. However, we let the network train on more iterations (250 k steps). We compute the performance of the network in validation after each epoch and use the set of weights that performed the best in validation for our performance analysis.

Ablation study. In order to review the performance with the best possible architecture, we first analyze the impact of the number of layers allocated to distinct heads on the performance of the network. We train five instances of both M4Depth+U_B and M4Depth+U with six levels and for heads with up to four dedicated layers. The averaged performances are reported in Table 4.1. Results show that the number of dedicated layers allocated to each head has no significant impact on the performance of the network. Therefore, in the rest of this work, we proceed with the performance analysis for architectures with a single dedicated head layer since it is the most computationally efficient.

Inference statistics. In this configuration, both methods have 5.7M parameters, and require up to 840Mo of VRAM to run. On a NVidia V100 GPU, depth and

Set	Method	Abs Rel		RMSE log		$\delta < 1.25$	
		Perf. ↓	AuSE ↓	Perf. ↓	AuSE ↓	Perf. ↑	AuSE ↓
KITTI	M4Depth	0.193	–	0.224	–	0.849	–
	M4Depth+U$_B$	0.140	0.025	0.195	0.046	0.858	0.021
	M4Depth+U	0.147	**0.021**	0.195	**0.041**	0.858	**0.019**
TtA-G	M4Depth	0.292	–	0.433	–	0.726	–
	M4Depth+U$_B$	0.274	0.049	0.448	0.173	0.725	**0.029**
	M4Depth+U	0.274	**0.041**	0.466	**0.170**	0.718	**0.029**
TtA-W	M4Depth	0.614	–	0.593	–	0.652	–
	M4Depth+U$_B$	0.618	0.176	0.597	0.217	0.636	0.031
	M4Depth+U	0.478	**0.058**	0.592	**0.157**	0.646	**0.028**
TtA-N	M4Depth	0.658	–	0.537	–	0.699	–
	M4Depth+U$_B$	0.748	0.101	0.573	0.176	0.688	0.021
	M4Depth+U	0.614	**0.039**	0.530	**0.144**	0.700	**0.020**
TtA-T	M4Depth	0.446	–	0.355	–	0.793	–
	M4Depth+U$_B$	0.468	0.077	0.410	0.155	0.776	**0.020**
	M4Depth+U	0.268	**0.032**	0.382	**0.122**	0.789	**0.020**

Table 4.2: Comparison of the performance of M4Depth+U$_B$ and M4Depth+U when trained on the Mid-Air dataset and tested in zero shot transfer on various datasets. The four following environments of the TartanAir dataset are used: gascola (TtA-G), seasons forest winter (TtA-W), neighborhood (TtA-N), and old town (TtA-T). Performances are reported for each set and they correspond to the best of five trained networks. The best AuSE scores for each set are highlighted in bold.

uncertainty maps are jointly estimated in 18ms for input samples with a size of 384×384 pixels, which is 1ms slower than M4Depth alone. The overhead of uncertainty estimation on the inference speed of the network is therefore limited and negligible for the joint task.

4.5.2 M4Depth+U vs the baseline

In Section 4.4, we propose a custom-tailored method, M4Depth+U, to train a modified architecture of M4Depth for joint depth and uncertainty estimation. We also explain why this method should perform better at estimating uncertainty than the standard probabilistic baseline, referred as M4Depth+U$_B$. We trained our network with both methods on the Mid-Air dataset. In this section, we compare their performance for various datasets.

Performance analysis

The performance of both M4Depth+U$_B$ and M4Depth+U on the test set of Mid-Air is reported in Table 4.1. Their performance in zero-shot cross-dataset transfer on KITTI and the chosen environments of TartanAir are reported in Table 4.2. As hypothesized, the probabilistic framework underpinning the M4Depth+U$_B$ baseline is sub-optimal while our elaborate uncertainty conversion method, M4Depth+U, consistently performs better. Also, the AuSE score for M4Depth+U varies less

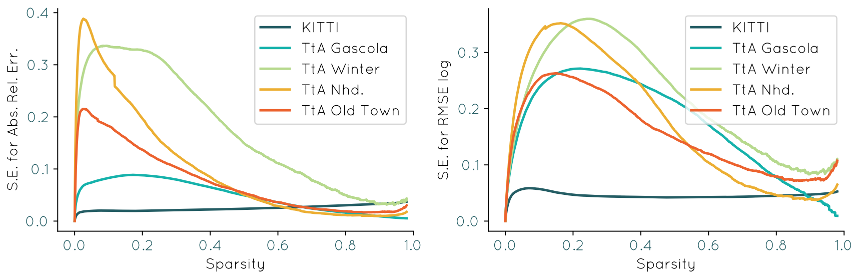


Figure 4.6: Sparsification error (S.E.) curves on the Absolute Relative and the RMSE log performance metrics for the best of five M4Depth+U_B networks tested in zero-shot cross-dataset transfer on the KITTI dataset and four environments of the TartanAir (TtA) dataset.

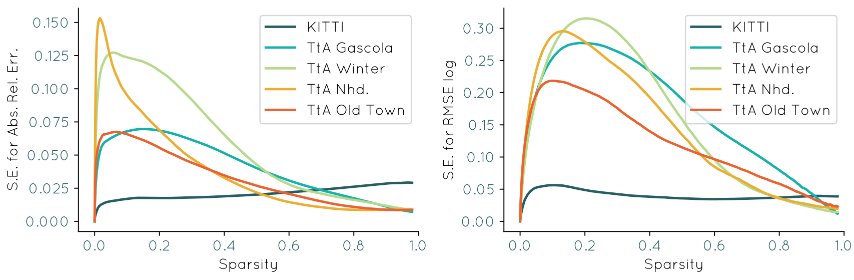
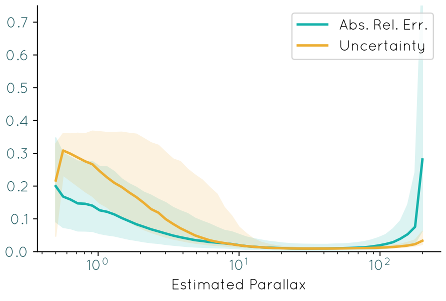


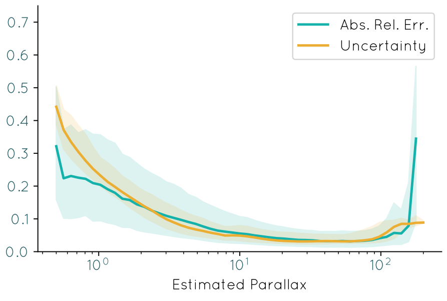
Figure 4.7: Sparsification error (S.E.) curves on the Absolute Relative and the RMSE log performance metrics for the best of five M4Depth+U networks tested in zero-shot cross-dataset transfer on the KITTI dataset and four environments of the TartanAir (TtA) dataset.

between datasets when compared to M4Depth+U_B, therefore hinting at more consistent generalization performances. Finally, it is worth noting that both approaches for estimating depth and its uncertainty preserve the raw performance for depth estimation of M4Depth. Further performance reports for the two strategies, given in Section B.2 of the Appendix, lead to the same observations.

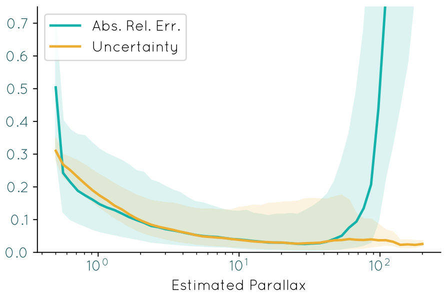
Sparsification error curves for all datasets used in zero-shot cross-dataset transfer in our experiments, displayed in Fig. 4.6 and Fig. 4.7 for M4Depth+U_B and M4Depth+U respectively, show that the sparsification errors for M4Depth+U_B are much higher than for M4Depth+U. For most datasets, the sparsification error is large for high uncertainty values and low for low uncertainty values, which means that the network is better at accurately discriminating depths with a low error than depths with a higher error. The upward trend at the very end of the sparsification error curve for M4Depth+U_B on the TartanAir set hints that the network is very confident in some areas with higher errors, which is not desired. This behavior is not observed with M4Depth+U which further motivates its interest over the baseline.



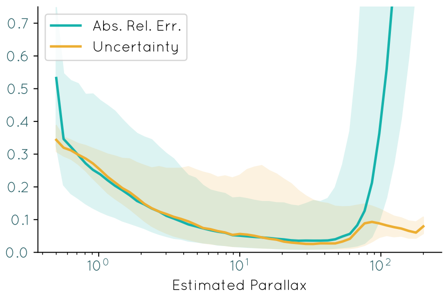
(a) Mid-Air



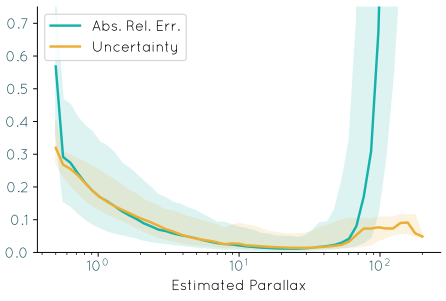
(b) KITTI



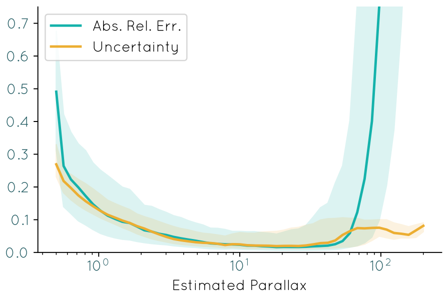
(c) TartanAir- Gascola



(d) TartanAir- Forest winter

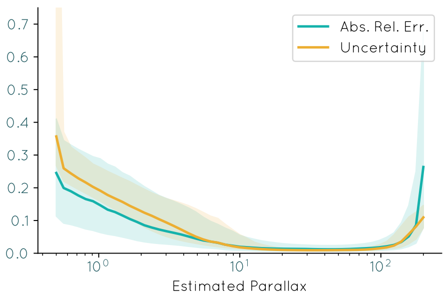


(e) TartanAir- Neighborhood

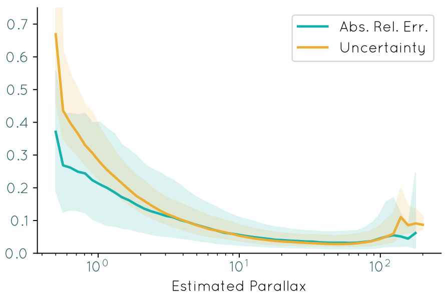


(f) TartanAir- Old town

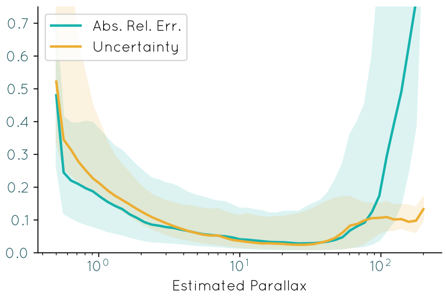
Figure 4.8: Joint overview of the observed Absolute Relative (Abs. Rel.) error on the depth estimate and of the estimated uncertainty produced by M4Depth+U$_B$ on different datasets when sorted depending on the estimated parallax. All graphs are generated using a network trained on the Mid-Air dataset. The lines highlight the median value for the samples falling within the considered parallax range. The areas surrounding the lines give the envelope that contains these samples. In order to ease the comparison between the uncertainty and the error, uncertainty values are rescaled according to the median of absolute relative errors observed in the considered dataset.



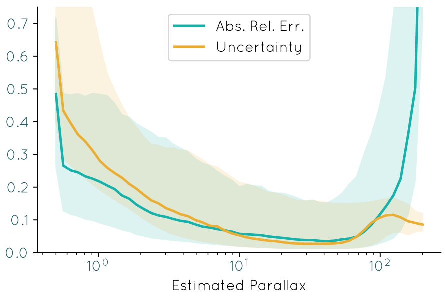
(a) Mid-Air



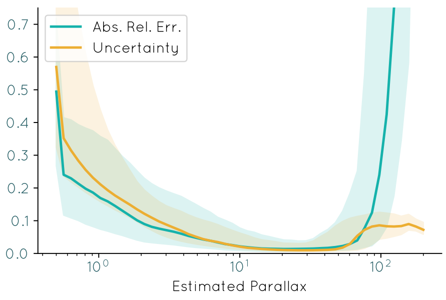
(b) KITTI



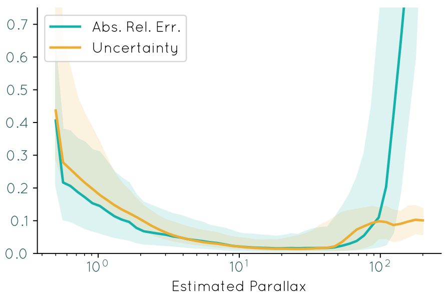
(c) TartanAir- Gascola



(d) TartanAir- Forest winter



(e) TartanAir- Neighborhood



(f) TartanAir- Old town

Figure 4.9: Joint overview of the observed Absolute Relative (Abs. Rel.) error on the depth estimate and of the estimated uncertainty produced by M4Depth+U on different datasets when sorted depending on the estimated parallax. All graphs are generated using a network trained on the Mid-Air dataset. The lines highlight the median value for the samples falling within the considered parallax range. The areas surrounding the lines give the envelope that contains these samples. In order to ease the comparison between the uncertainty and the error, uncertainty values are rescaled according to the median of absolute relative errors observed in the considered dataset.

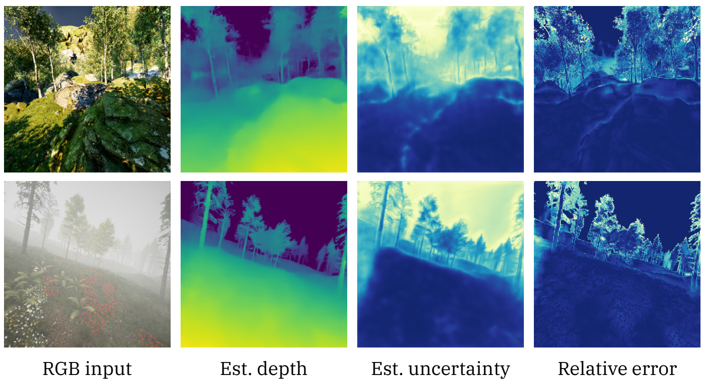


Figure 4.10: Illustrations of outputs produced by M4Depth+U on the Mid-Air dataset. Lighter colors correspond to higher uncertainty and error values.

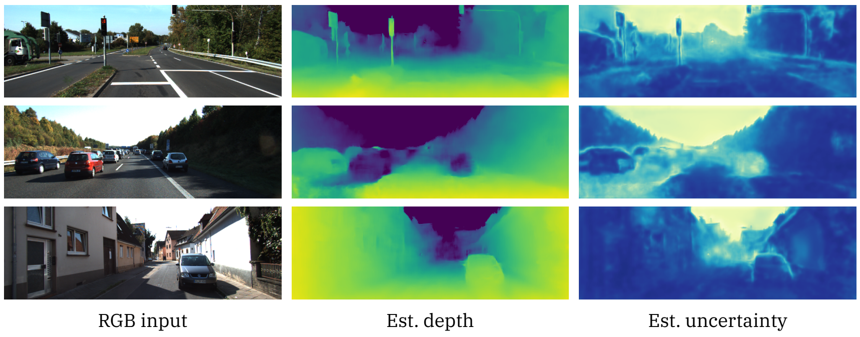


Figure 4.11: Illustrations of outputs produced by M4Depth+U in zero-shot cross-dataset transfer on the KITTI dataset. Lighter colors correspond to higher uncertainty values.

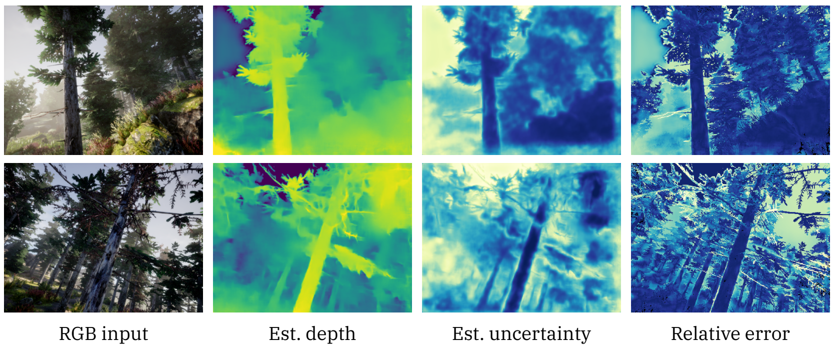


Figure 4.12: Illustrations of outputs produced by M4Depth+U in zero-shot cross-dataset transfer on the "Gascola" environment of the TartanAir dataset. Lighter colors correspond to higher uncertainty and error values.

Error distribution analysis

In order to get further insights on the performance of the network, we plotted absolute relative errors and uncertainty estimate values as a function of estimated parallax values. The resulting graphs are shown in Fig. 4.8 and 4.9 for M4Depth+U$_B$ and M4Depth+U respectively.

The distribution of the absolute relative error on depth is similar for both M4Depth+U$_B$ and M4Depth+U, which validates the similarities seen in Table 4.2 for depth estimation. In addition, the shape of these distributions is similar for all datasets. The relative error, high for small parallax values, decreases to reach a low error plateau for parallax values larger than 10 pixels, and then increases steeply at the end of this plateau, which is located between parallax values of 50 and 100 pixels depending on the considered dataset. This typical U shape shows that M4Depth works best over a given subset of parallax values, which can be explained by the discrete nature of pixels and the finite dimensions of pictures. These observations suggest that implementing a dynamic temporal sampling of reference points to be used for parallax estimation could lead to a reduction of the average error on depth estimates produced by M4Depth+U.

The uncertainty curves highlight the differences between M4Depth+U$_B$ and M4Depth+U. While the median uncertainty values closely follows the median of the absolute relative errors for most datasets, the shape of the envelope of estimated uncertainties significantly differs from the shape of the envelope of relative errors for M4Depth+U$_B$ whereas both shapes match for M4Depth+U. Also, neither M4Depth+U$_B$ or M4Depth+U is able to accurately estimate uncertainty for pixels with high parallax values, even though the latter performs slightly better than the former in this regard.

Overall, these graphs show that the uncertainty estimated with our proposed strategies is well correlated with the absolute relative error. In addition, they confirm that the uncertainties estimated by M4Depth+U are better estimates of the error than the ones estimated by M4Depth+U$_B$.

Qualitative overview of M4Depth+U uncertainty samples

We provide illustrations of the outputs produced by M4Depth+U on the datasets used for our experiments in Figures 4.10 to 4.15. A general observation that can be made is that the uncertainty is visually well correlated with the relative error. The network is typically uncertain around sharp depth transitions, especially in cluttered places such as foliage, and in areas that are not visible simultaneously from the two camera point of views (see the left side in the last row of Figure 4.10). The network is also generally less certain of its outputs for distant objects, where parallax values are smaller. However, Figures 4.12 and 4.13 show that the network can erroneously state to be certain of its output in places with many details, both in the RGB and the depth domains, such as for tree branches. On the another hand,

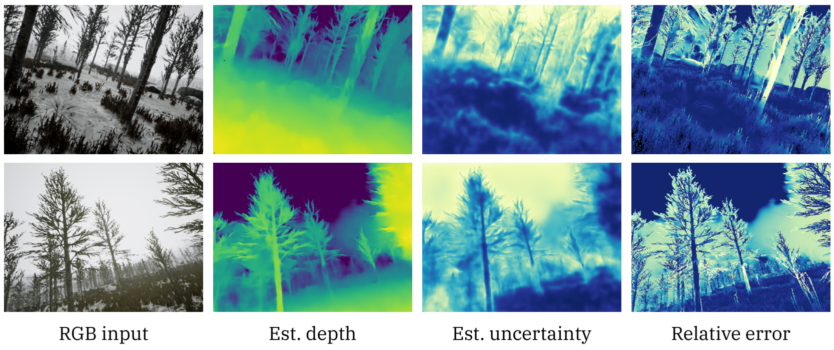


Figure 4.13: Illustrations of outputs produced by M4Depth+U in zero-shot cross-dataset transfer on the “Seasons forest winter” environment of the TartanAir dataset. Lighter colors correspond to higher uncertainty and error values.

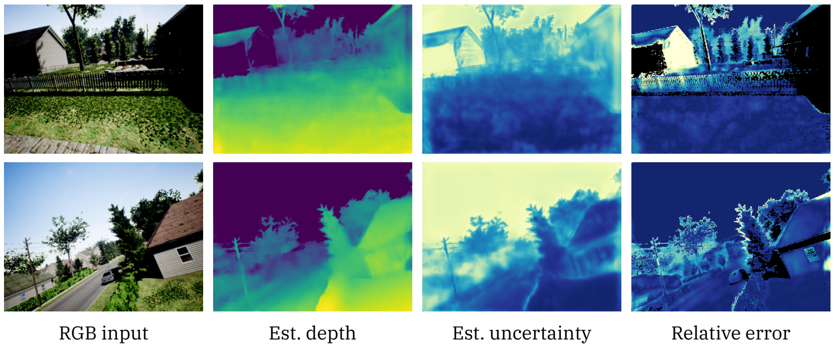


Figure 4.14: Illustrations of outputs produced by M4Depth+U in zero-shot cross-dataset transfer on the “Neighborhood” environment of the TartanAir dataset. Lighter colors correspond to higher uncertainty and error values.

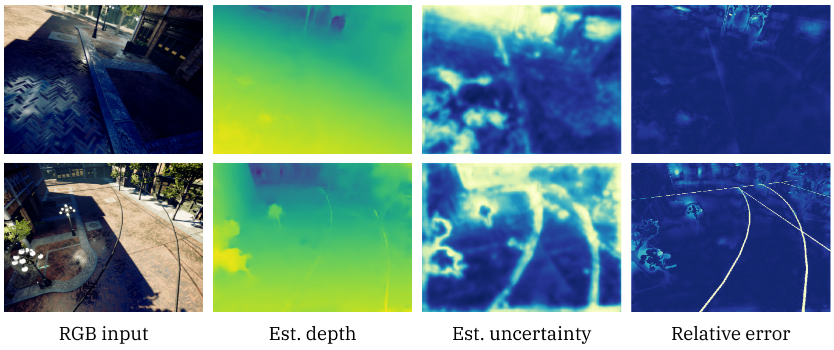


Figure 4.15: Illustrations of outputs produced by M4Depth+U in zero-shot cross-dataset transfer on the “Old Town” environment of the TartanAir dataset. Lighter colors correspond to higher uncertainty and error values.

Method	Causal	Abs Rel ↓	AuSE ↓	Time [ms] ↓
MVSNet [153]	✗	0.140	0.025	150
Fast-MVSNet [154]	✗	0.121	0.034	350
Vis-MVSNet [158]	✗	0.103	0.028	820
Robust MVD [120]	✗	0.071	0.017	60
M4Depth+U	✓	0.086	0.020	26

Table 4.3: Comparison of the performance of M4Depth+U on the uncertainty benchmark proposed by Schröppel *et al.* [120] for MVS methods. Performances are reported in zero-shot cross-dataset transfer on the 93-images test set for the KITTI dataset used for this benchmark. Inference timings are reported for full-size KITTI images. Note that M4Depth+U is causal and only uses a sequence of frames that precedes the frame considered for depth inference, while SFM methods are anti-causal as they also use upcoming frames.

the second row of Figure 4.11 shows that the network is able to correctly detect errors induced by dynamic objects in most instances, which is unexpected since the method assumes static scenes. Finally, Figure 4.14 and 4.15 show that the network can reliably evaluate its own error in problematic areas such as low-textured surfaces (see the barn in the first row of Figure 4.14), and fine transitional details (see the wires in the second row of Figure 4.15).

4.5.3 M4Depth+U on benchmarks

In this section, we test our network on two benchmarks proposed in the literature. We only focus on the performance of M4Depth+U since it systematically outperforms the baseline.

Robust MVD

As our method targets autonomous UAV applications, it has to produce estimates for the latest available frame. Therefore, it cannot use future information as opposed to generic multi-view depth estimation methods which can use past as well as upcoming frames of the sequence. Since we are the first to target this specific use case, there is no existing baseline to compare to directly. Nonetheless, we assess the value proposition of M4Depth+U over some other existing methods on the benchmark proposed by Schröppel *et al.* [120] for joint multi-view depth and uncertainty estimation. Results on the KITTI set of the benchmark are reported in Table 4.3. Despite working with fewer data than other methods, M4Depth+U outperforms most of the baseline and comes close to the state of the art on this benchmark. This, combined with the fact that M4Depth+U is at least 2.5 times faster than other methods, leads us to conclude that our method performs on par with existing methods tested on this benchmark, and that M4Depth+U has a real benefit for practical use.

Visual variation	Method	Abs Rel		RMSE log		$\delta < 1.25$	
		Mean ↓	Rel Std ↓	Mean ↓	Rel Std ↓	Mean ↑	Rel Std ↓
Weather	M4Depth	0.097	0.022	0.166	0.011	0.887	0.003
	M4Depth+U	0.087	0.015	0.159	0.014	0.890	0.001
Season	M4Depth	0.110	0.020	0.184	0.015	0.913	0.002
	M4Depth+U	0.098	0.027	0.176	0.019	0.917	0.002

Visual variation	Method	Abs Rel AuSE		RMSE log AuSE		$\delta < 1.25$ AuSE	
		Mean ↓	Rel Std ↓	Mean ↓	Rel Std ↓	Mean ↓	Rel Std ↓
Weather	M4Depth+U	0.006	0.006	0.019	0.058	0.005	0.002
Season	M4Depth+U	0.007	0.024	0.022	0.179	0.007	0.028

Table 4.4: Performance of M4Depth+U on the weather and season robustness benchmarks of the Mid-Air dataset. Results correspond to the performance of the network that had the best validation performance over five trained candidates. The relative standard deviation should be as small as possible.

Mid-Air robustness benchmark

Similar to the analysis carried in Section 3.5.4, we test the robustness of our method on the benchmark we introduced with Mid-Air. The results obtained by M4Depth+U on this benchmark are given in Table 4.4. First, we note that the performance of M4Depth on depth estimation is not affected by our modifications. Second, we note that the uncertainty estimates feature a comparable level of robustness to visual changes than the depth estimate for the absolute relative error and on the threshold metrics. However, their performance is comparably worse on the RMSE log metric. This difference in robustness could be explained by the fact that our uncertainty metric is a relative metric that may be best compared to the absolute relative error. Overall, these results indicate that the uncertainty has a good robustness to visual changes since its performance on the benchmark is similar to one of depth estimates, which was shown to be good in Section 3.5.4.

4.6 Conclusion

In this chapter, we showed that it is possible to adapt M4Depth, the depth estimation method presented in the previous chapter, for joint depth and uncertainty estimation at minimal cost. We also demonstrated that converting the uncertainty values produced by the network into uncertainty values related to depth is better done with a custom-tailored conversion method, referred as M4Depth+U, than with the standard probabilistic approach. Our performance tests on the Mid-Air dataset and in zero-shot cross-dataset transfer on the KITTI dataset and four environments of the TartanAir dataset show that M4Depth+U emerges as an excellent uncertainty estimator. These tests also show that this new method has the same performance for depth estimation than M4Depth, while having a negligible computational overhead when compared to the fast inference speed of M4Depth.

Besides, the results obtained by M4Depth+U on two benchmarks show that a perfect fit for depth uncertainty estimation for autonomous vehicle applications. Indeed, the robustness benchmark of Mid-Air shows that our method is robust to visual changes, while the Robust MVD benchmark in zero-shot cross-dataset transfer shows that it generalizes well when compared to other methods. M4Depth+U has, indeed, similar performance than multi-view depth estimation methods on this benchmark, while being 2.5 times faster and causal, as opposed to existing methods.

5 Conclusion

About our research

Throughout this thesis, we explored the challenge of achieving reliable monocular depth estimation to replace dedicated distance sensors for UAV flight automation. In our introduction, we explain why we believe that any depth estimation method should meet four important requirements for such application. These requirements are (1) generalizability, (2) robustness to visual changes, (3) causality, and (4) computational efficiency to allow for real-time operation. From our review of the scientific literature, it appeared that methods existing prior to our work made poor candidates for applications in UAVs as they fail one or several requirements, and that research was still needed to address depth estimation for UAVs.

The first expected requirement for a depth estimation method aimed at UAV applications, that is generalizability, expects the method to work in a wide variety of environments. The review of datasets available to train and test depth estimation methods showed that the most significant datasets for outdoor applications are designed for autonomous cars in urban environments. Therefore, training and testing depth estimation methods aimed at UAV applications in a wide diversity of environments was not possible with existing datasets. As a result, we created and publicly released our own dataset called Mid-Air. This synthetic dataset, presented in Chapter 2, features 79 minutes of drone flight recorded several times with different climate conditions in unstructured environments. Its content consists of multiple synchronized modalities providing data for positioning tasks such as SLAM or visual odometry as well as for pure computer vision tasks such as depth estimation, semantic segmentation, or surface normal estimation. While being specifically designed for flying drones, its size (more than 420k individual frames) and content makes it also useful for training and testing machine learning algorithms for other applications than UAVs. The choice of unstructured environments was guided by the challenge they represent for depth estimation. We introduce two benchmarks alongside our dataset. The purpose of the first one being to test visual odometry or SLAM methods in an environment different from the ones present in the training set to test their generalization capabilities. The purpose of the second is to test the robustness of computer vision methods to changes in visual inputs. Our dataset has been available for download since June 2019, and has been downloaded by more than 500 researchers around the world by the beginning of 2023. Its multi-modal aspect already led it to be used for various original contributions, which confirms its value to the scientific community.

In Chapter 3, we addressed the challenging task of depth estimation. First, we proceeded to identify the weaknesses of existing depth estimation methods when considering generalizability environments, even unseen ones. We used this analysis to propose a new causal multi-view depth estimation method, called M4Depth, designed to be motion- and feature-invariant by relying on a notion of visual parallax which we have introduced and defined for generic camera motion. We showed how the visual parallax can be used to estimate depth without tying our network to a specific depth distribution, and presented alternate ways of building cost volumes to increase the generalization capability of our method. Our experiments showed that the performance of M4Depth is superior to the baseline both in unstructured environments and in generalization, while performing similarly to existing methods in structured environments. Furthermore, tests on the visual robustness benchmark of Mid-Air showed that the design of our method is robust to visual changes. Finally, the memory requirements and inference time of M4Depth are low enough to be considered for real-time inference on embedded GPUs. Thus, M4Depth meets all the requirements needed when considering embedded depth estimation for UAV applications.

One of the main purposes of depth estimation methods in UAVs is to be used as tools for planning trajectories or avoiding obstacles. For such application, where any unpredicted error can lead to collisions, it is essential to anticipate potentially erroneous data in order to take appropriate action. In Chapter 4, we investigated the use of uncertainty as a mean to anticipate erroneous data in the depth estimates, and presented M4Depth+U, an adaption of M4Depth for joint depth and uncertainty estimation at minimal additional cost. Our M4Depth+U method encompasses both the adaptation of the architecture, and the method needed to convert the uncertainty values produced by the network into uncertainty values related to depth. Tests on multiple data datasets, including zero-shot cross-dataset transfer, showed that M4Depth+U works better than the standard probabilistic approach for uncertainty conversion. Besides, our network adaptation keeps the good robustness properties of M4Depth. More generally, M4Depth+U emerged as an excellent joint depth and uncertainty estimation method when compared to other methods. Its performance on a public benchmark for multi-view depth estimation is, indeed, comparable to the best existing method while, while being 2.5 times faster and causal, as opposed to other methods that do not meet this requirement of flying drones.

In conclusion, this thesis proposes (1) Mid-Air, a new dataset for developing computer vision methods for UAV applications, (2) M4Depth, a new robust and generalizable depth estimation method, and (3) M4Depth+U, the first method designed to jointly estimate depth and depth uncertainty while meeting the requirements needed for UAV applications. With these contributions, we believe to have achieved concrete steps towards truly reliable monocular depth estimation for unmanned aerial vehicles.

Future works

While our contributions already show strong results, we identify three research questions that could be investigated to further improve our proposal for reliable depth estimation.

Dynamic frame sampling. As detailed in Chapter 3, our depth estimation method relies on a specific notion of parallax which is the visible frame-to-frame pixel displacement of a static object. The architecture of the network is such that our method works best for a specific range of parallax values, as shown in Section 4.5.2. Therefore, we assume that the performance of the method could be improved by working with parallax values that are optimal for the network. As the parallax is function of the frame-to-frame camera motion, adapting the frame-to-frame camera motion could be a solution to achieve this. Without controlling the vehicle, adapting the perceived frame-to-frame motion can be achieved by choosing the time that separates the two frames used for depth estimation. As a result, it could be interesting to find a strategy to dynamically sample past frames for each pixel depending on the needs of the method.

Handling dynamic scenes. In this thesis, we considered the temporal multi-view approach of depth estimation because of its better potential for long-range depth estimation when compared to a stereo setup with a small baseline, as the ones typically found in small UAVs. However, the main drawback of the temporal approach when compared to the stereo setup is its inability to deal with dynamic objects in the scene. Combining both approaches would allow combining their strengths but requires a strategy to fuse the available information adaptively for each pixel of the image. While such a strategy is relatively straightforward in a generic multi-view stereo method relying on a single plane-sweeping cost volume, the parallax and parallax sweeping cost volumes approach of our method induces some challenges that require a custom solution.

Visual odometry integration. Throughout this thesis, we assumed the pose of the camera corresponding to each recorded frame to be known, since it can be estimated by standard visual odometry or SLAM algorithms. However, a part of the work performed by these algorithms, which is triangulating feature points, is partially redundant with depth estimation, therefore leading to a waste of computational resources. We believe that the next natural step would be to get rid of the need for an external algorithm by jointly estimating depth and camera pose by solely relying on input images and IMU measurements.

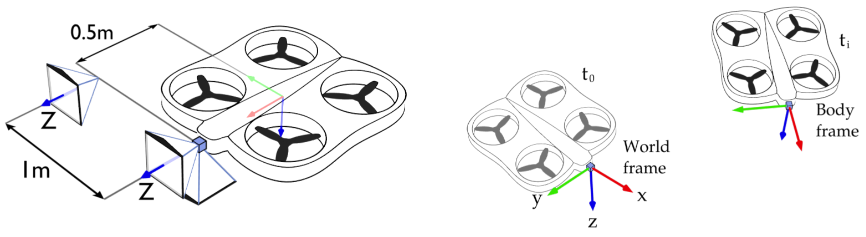
Appendix

A Mid-Air: technical specifications and practical information

Summary

Overview

In this part, we provide the practical information needed for using Mid-Air. We give the characteristics and specifications of all simulated sensors, we cover the structure of the dataset itself, and we explain the procedure to download it.



(a) Sensors position

(b) Frames of reference definition

Figure A.1: Illustration of the axis definition and of the placement of the sensors on the drone used to record the Mid-Air dataset.

A.1 Technical specifications

Hereafter, we present the details about the sensors used in Mid-Air, which include their type, their capture rate, their unit, their reference frame, and sensor positions.

A.1.1 Sensors positioning and frames of reference definition

Figure A.1a shows the sensor locations on the drone used to generate our dataset. Cameras are represented by the pyramids. The blue cube shows the IMU and the GPS receiver locations.

Figure A.1b illustrates the frames of reference used for all position-related data. The World frame is defined at the starting point of the trajectory and oriented such that the drone yaw is equal to zero. The other axes are horizontal. The Body frame is rigidly attached to the drone with its origin corresponding to the position of the IMU and GPS receiver. All frames use the North, East, Down (NED) axes convention.

A.1.2 Positioning data

For all position-related data, we use the North, East, Down (NED) axes convention. Distances are expressed in meters, rotations in quaternions, angles in radians, and time in seconds. The positioning information stored in the dataset is as follows:

- Ground truths for the position, speed, acceleration, and attitude are expressed in the World frame;
- Angular velocity ground truths and IMU data, *i.e.,* acceleration and angular velocity are expressed in the Body frame;
- GPS position and speed are given in meters in the World frame.

It is important to note that the GPS position is not given by the standard longitude, latitude, and altitude information, but by a simple position in meters expressed in the World frame. This position in meters is obtained by projecting the position given with the longitude/latitude/altitude format relative to the first point of the trajectory.

Additionally, our dataset stores some information about the state of the sensors. The following sensor data are made available:

- An estimate of the initial bias for the accelerometer and the gyroscope;
- The GPS signal quality estimates (*i.e.,* GDOP, PDOP, HDOP, VDOP) for each measurement;
- The number of satellites visible by the GPS receiver for each of its measurements.

A.1.3 Visual data

Each trajectory record comes with eight video streams corresponding to the (1) left, (2) right and (3) down-looking RGB camera views and the (4) segmentation, (5) depth, (6) normals, (7) disparity and (8) occlusion maps seen by the left camera. Each video stream consists of a set of successively numbered pictures stored in a dedicated directory. The image formats and content are the following:

- RGB pictures are stored in JPEG images.
- Occlusion masks are stored as lossless 1-channel PNGs.
- Surface normals are stored as RGB lossless PNG files.
- Normal vectors are tri-dimensional and are expressed with respect to the Body frame.
- Red color corresponds to the Y-axis, blue to the X-axis, and green to the Z-axis (but with reverse direction).
- All vectors were normalized to have a unit norm. In order to fit in an RGB picture, the range of possible element values, *i.e.,* $[-1; 1]$, was scaled and shifted to fit a range of $[0; 1]$. For example, with this convention, a perfectly flat and horizontal surface will have an RGB color corresponding to (0:5; 1; 0:5) if the drone does not have any pitch nor roll angle.
- Depth and stereo disparity maps are expressed in meters and in pixels respectively, and are stored as 16-bit float matrices in lossless 1-channel PNGs. One of the provided example scripts shows how to decode them.
- Semantic segmentation maps are stored as lossless 1-channel 8-bit unsigned int PNGs. The value of a pixel indicates a label number. Correspondences between label numbers and classes are given below.

Camera intrinsic matrix

The cameras used to record visual data all share the same intrinsic matrix. For an image of height h and width w, this matrix is given by:

$$\mathbf{K} = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \quad \text{with} \quad f_x = c_x = \frac{w}{2} \quad \text{and} \quad f_y = c_y = \frac{h}{2}.$$

This corresponds to a visual field of view of 90 degrees.

Semantic segmentation classes

Mid-Air provides, among others, semantic segmentation maps. Each pixel of these maps has an Id number that corresponds to a specific class. The correspondence between an Id number and its class is given in Table A.1.

Id	Class content	Id	Class content
1	Animals	8	Water plane
2	Trees	9	Man-made construction
3	Dirt ground	10	Road
4	Ground vegetation	11	Train track
5	Rocky ground	12	Road sign
6	Boulders	13	Other man-made objects
7	[empty]	14	[empty]

Table A.1: Correspondence between Id numbers and classes in the semantic segmentation map.

A.1.4 Sensors summary table

We list all the data available in the dataset in Table A.2.

Data	Sampling freq.	Reference	Unit
Ground-truth position	100 Hz	World	[m]
Ground-truth velocity	100 Hz	World	[m/s]
Ground-truth acceleration	100 Hz	World	$[m/s^2]$
Ground-truth attitude	100 Hz	World	[rad/s]
Ground-truth angular velocity	100 Hz	World	[quaternion]
IMU acceleration	100 Hz	Body	$[m/s^2]$
IMU angular velocity	100 Hz	Body	[rad/s]
GPS position	1 Hz	World	[m]
GPS velocity	1 Hz	World	[m/s]
GPS signal information	1 Hz	n/a	n/a
Downward-looking RGB picture	25 Hz	n/a	n/a
Right stereo RGB picture	25 Hz	n/a	n/a
Left stereo RGB picture	25 Hz	n/a	n/a
Stereo disparity map	25 Hz	n/a	[pix]
Stereo occlusion map	25 Hz	n/a	n/a
Depth map	25 Hz	n/a	[m]
Surfaces normal map	25 Hz	n/a	n/a
Semantic segmentation map	25 Hz	n/a	n/a

Table A.2: List of all the data available in the dataset with their respective capture frequency. For each data, we also give the related frame of reference and unit when applicable.

```
Directories
Climate setup x
  ├ color_left
  │  ├ Trajectory i
  │  │  ├ img_j.jpg
  ├ color_right
  │  ├ Trajectory i
  │  │  ├ img_j.jpg
  ├ color_down
  │  ├ Trajectory i
  │  │  ├ img_j.jpg
  ├ depth
  │  ├ Trajectory i
  │  │  ├ img_j.png
  ├ normals
  │  ├ Trajectory i
  │  │  ├ img_j.png
  ├ segmentation
  │  ├ Trajectory i
  │  │  ├ img_j.png
  ├ stereo_disparity
  │  ├ Trajectory i
  │  │  ├ img_j.png
  ├ stereo_occlusion
  │  ├ Trajectory i
  │  │  ├ img_j.png
  └ HDF5 dataset
```

(a) Directories hierarchy for image storage.

```
HDF5 dataset
Trajectory i
  ├ camera_data
  │  ├ color_left
  │  ├ color_right
  │  ├ color_down
  │  ├ depth
  │  ├ normals
  │  ├ segmentation
  │  ├ stereo_disparity
  │  └ stereo_occlusion
  ├ gps
  │  ├ GDOP
  │  ├ HDOP
  │  ├ PDOP
  │  ├ VDOP
  │  ├ no_vis_sats
  │  ├ position
  │  └ velocity
  ├ groundtruth
  │  ├ attitude
  │  ├ angular_velocity
  │  ├ position
  │  ├ velocity
  │  └ acceleration
  └ imu
     ├ accelerometer
     └ gyroscope
```

(b) Data hierarchy inside of the hdf5 dataset file.

Table A.3: Graphical overview of the data organization in the Mid-Air dataset.

A.2 Dataset files organization

Our dataset contains two major types of files; images (PNG or JPEG) and `hdf5` dataset files. The former ones are used to store the visual sensor streams, while the latter ones contain the records for all positioning information (ground truths and sensors). Additionally, our `hdf5` files also provide information about the time synchronization between positional and visual records.

A.2.1 Directories hierarchy

Each climate setup is considered as an individual dataset part. They all have their own `hdf5` dataset file and are the root of a directory hierarchy which is the same for all of them. This hierarchy is illustrated in Table A.3b. In this figure, i corresponds to each trajectory number and j is the image frame number (it ranges from 0 to the number of images recorded for the corresponding trajectory).

A.2.2 HDF5 files content

The `hdf5` data structure is illustrated in Table A.3a. Its root contains one group for each trajectory. Each of those sub-dataset then has the same content organization, which is detailed in the same table.

Since visual sensors are sampled at 25 Hz, GPS at 1 Hz and positioning information at 100 Hz, and assuming that `trajectory_i` as a length of N seconds, the groups in the `hdf5` dataset have the following content:

- Groups in the `camera_data` group are a list of strings of length $25N$. Entry `k=floor(25*t)` of a list gives the relative path to the frame captured at time `t`, where `t` is expressed in seconds.
- Groups in the `gps` group are matrices of size $N \times m$, where m is equal to 3 for the position and velocity groups and 1 for the others. Line `k=floor(t)` of a matrix gives the sensor measurement captured at time `t`, where `t` is expressed in seconds.
- Groups in the `groundtruth` group are matrices of size $100N \times m$, where m is equal to 4 for the attitude group and 3 for the others. Line `k=floor(100*t)` of a matrix gives the sensor measurement captured at time `t`, where t is expressed in seconds.
- Groups in the `imu` group are matrices of size $100N \times 3$. Line `k=floor(100*t)` of a matrix gives the sensor measurement captured at time `t`, where `t` is expressed in seconds. Additionally, these groups have an attribute giving the initial bias estimate.

A.3 Dataset download procedure

Our dataset can be downloaded on the following webpage: https://midair.ulg.ac.be/download.html. Due to the sheer volume of data, we had to fragment the dataset in several individual compressed archives to prevent the need for extracting data for a huge single archive. This also allows us to let the user choose the exact files he wants to download. The download procedure works as follows:

1. Select the desired visual sensors (left RGB, right RGB, down RGB, surface normals, semantic segmentation map, range/distance map, stereo disparity map, stereo occlusion map);
2. Select the desired trajectories and climate setups;
3. Select desired benchmarks;
4. Request download links through a form to guarantee the user agreement to the dataset license.

Once the form is submitted, the website will generate a text file containing all links to the archives containing the requested data. Archives can then be downloaded using wget and the received text file as follows:

```
wget --content-disposition -x -nH -i path_to/download_config.txt
```

This command line will take care of downloading all the archives whose link is in the file and of storing them in the correct directory (in order to respect paths encoded in the hdf5 files). Once all archives are downloaded, they can just be uncompressed in place. This can be done by running the following command line at the root of the dataset:

```
find . -name "*.zip" | while read filename; do unzip -o -d $(dirname
↪  "$filename") "$filename"; done;
```

B Complementary details on experiments

Summary

Overview

In this chapter, we provide some details about our experiments that were left out of the body of this document to preserve the flow of the text. In the first section, we give the training details of the methods making the baseline for depth estimation on the Mid-Air dataset. In the second section, we provide complementary results to compare the performance of the joint depth and uncertainty estimation method introduced in Chapter 4, M4Depth+U, to the baseline and M4Depth that allow us to reinforce the observations made in this chapter. More specifically, we give the mean and the standard deviation of the performance obtained by five networks in various configurations.

B.1 Mid-Air baseline methods training details

In this section, we provide the training details needed to reproduce the results of all the methods used in our baseline for depth estimation on the Mid-Air dataset. These details complement the code made available on the following GitHub repository: https://github.com/michael-fonder/M4Depth-Baselines. In this dissertation, we have chosen six methods for our baseline, namely: Monodepth [42, 43], Monodepth2 [40, 41], ST-CLSTM [141, 157], the method of [135, 136], ManyDepth [139, 140], and PWCDC-Net [129].

To get a baseline that is true to the work of the authors and that is coherent between different methods, we proceeded as follows. We kept the original default parameter values of each method. Next, we adjusted the batch size so that every learning step contained around 18 frames, and we trained each network five times. As some method have specific input pipelines, we adjusted the training epoch count of each method to guarantee that a network sees every training sample at least 50 times during its training. After a first round of training, it appeared that some methods did not converge. This has led us to adapt the training setup for these methods in order to obtain a representative performance.

The training of PWCDC-Net [129] had to be done differently as it is an optical flow network. To make it work, we had to convert depth maps to optical flow maps by using Equation 3.14. It also required more steps during the training to reach a steady-state on the validation set.

Method	Train epoch count	Batch size	Sequence length
Monodepth [42, 43]	50	18	1*
Monodepth2 [40, 41]	17	6	3*
ST-CLSTM [141, 157]	50	3	5*
Wang *et al.* [135, 136]	50	3	8
Manydepth [139, 140]	25	6	3*
PWCDC-Net [129]	100	8	2*

Table B.1: Main parameters used to train baseline methods. The asterisk denotes default parameters of methods suggested by their respective authors.

The values reported for the baseline methods correspond to the best results obtained out of five runs. The most important parameters of the baseline setup are given in Table B.1. We kept all other parameters unchanged to a large extent. However, adjustments were necessary for some methods, as explained hereafter.

- With the proposed setup, Monodepth failed to produce any output for three trainings out of the five.
- The lower epoch count for Monodepth2 is due to the fact that the training pipeline sees each sample three times during a single epoch.
- Performances obtained with the default learning rate for the ST-CLSTM method

were extremely poor. We obtained better results by reducing it to 10^{-4}.

- The code written by Wang *et al.* [135, 136] worked as expected. However, the length of the training sequence had to be downsized from 10 to 8 frames to accommodate our internal pipeline constraints.
- Finally, for Manydepth, we had to select the encoder architecture; we chose the ResNet-50 encoder.

B.2 Complementary performance comparison for M4Depth+U vs M4Depth+U$_B$

In this section, we provide complementary results to compare the performance of the joint depth and uncertainty estimation method introduced in Chapter 4, M4Depth+U, to the baseline and M4Depth. More specifically, we give the mean and the standard deviation of the performance obtained by five networks in various configurations.

In Tables B.2 to B.13, we present the mean and the standard deviation of the performance of five networks for depth and uncertainty estimation. Similar to observations made in the body of this document, all tables show that the number of layers allocated to distinct heads in the network has no significant impact on the performance of the architecture. The results also confirm that the modifications made to M4Depth have no significant impact on the performance of depth estimation. Tables B.8 to B.13 give more insight on the comparative performance of M4Depth+U$_B$ and M4Depth+U for uncertainty estimation, and lead to two important observations. First, on average, M4Depth+U systematically outperforms M4Depth+U$_B$. Second, the performance of M4Depth+U$_B$ for uncertainty estimation varies from one run to the other much more than M4Depth+U. This could indicate that an approximate for the uncertainty on the inverse of the parallax is less reliable than one for the parallax. All of these additional observations further motivate the superiority of the conversion method developed for M4Depth+U over the baseline represented by M4Depth+U$_B$.

Method	# head layers	Abs Rel ↓		RMSE log ↓		δ < 1.25 ↓	
		Mean	STD	Mean	STD	Mean	STD
M4Depth	N/A	0.127	0.012	0.185	0.007	0.907	0.003
M4Depth+U$_B$	1	0.145	0.009	0.190	0.006	0.906	0.001
M4Depth+U$_B$	2	0.132	0.004	0.186	0.003	0.908	0.003
M4Depth+U$_B$	3	0.137	0.008	0.185	0.006	0.907	0.003
M4Depth+U$_B$	4	0.132	0.005	0.183	0.002	0.908	0.002
M4Depth+U	1	0.134	0.008	0.188	0.006	0.906	0.004
M4Depth+U	2	0.141	0.011	0.193	0.013	0.901	0.008
M4Depth+U	3	0.138	0.008	0.191	0.004	0.905	0.002
M4Depth+U	4	0.141	0.010	0.193	0.013	0.901	0.008

Table B.2: Ablation study giving the performance of networks with different numbers of layers dedicated to the depth head on the test set of the Mid-Air dataset. The reported performances correspond to the ones of five networks trained on the Mid-Air dataset.

Method	# head layers	Abs Rel ↓		RMSE log ↓		δ < 1.25 ↓	
		Mean	STD	Mean	STD	Mean	STD
M4Depth	N/A	0.205	0.014	0.229	0.004	0.841	0.006
M4Depth+U$_B$	1	0.181	0.032	0.220	0.023	0.848	0.009
M4Depth+U$_B$	2	0.173	0.014	0.214	0.011	0.853	0.005
M4Depth+U$_B$	3	0.229	0.087	0.251	0.047	0.835	0.016
M4Depth+U$_B$	4	0.172	0.022	0.211	0.012	0.854	0.012
M4Depth+U	1	0.182	0.038	0.245	0.020	0.848	0.009
M4Depth+U	2	0.192	0.007	0.227	0.013	0.844	0.009
M4Depth+U	3	0.172	0.018	0.212	0.010	0.850	0.005
M4Depth+U	4	0.192	0.007	0.227	0.013	0.844	0.009

Table B.3: Ablation study giving the performance of networks with different numbers of layers dedicated to the depth head on the test set of the KITTI dataset in zero-shot cross-dataset transfer. The reported performances correspond to the ones of five networks trained on the Mid-Air dataset.

Method	# head layers	Abs Rel ↓		RMSE log ↓		δ < 1.25 ↓	
		Mean	STD	Mean	STD	Mean	STD
M4Depth	N/A	0.298	0.009	0.493	0.050	0.709	0.013
M4Depth+U$_B$	1	0.302	0.022	0.458	0.009	0.712	0.009
M4Depth+U$_B$	2	0.297	0.017	0.586	0.086	0.687	0.024
M4Depth+U$_B$	3	0.285	0.025	0.495	0.028	0.702	0.015
M4Depth+U$_B$	4	0.289	0.022	0.545	0.058	0.692	0.014
M4Depth+U	1	0.289	0.010	0.482	0.058	0.710	0.014
M4Depth+U	2	0.302	0.025	0.528	0.062	0.698	0.017
M4Depth+U	3	0.298	0.021	0.510	0.051	0.703	0.011
M4Depth+U	4	0.302	0.026	0.528	0.062	0.698	0.017

Table B.4: Ablation study giving the performance of networks with different numbers of layers dedicated to the depth head on the Gascola set of the TartanAir dataset in zero-shot cross-dataset transfer. The reported performances correspond to the ones of five networks trained on the Mid-Air dataset.

Method	# head layers	Abs Rel ↓		RMSE log ↓		$\delta < 1.25$ ↓	
		Mean	STD	Mean	STD	Mean	STD
M4Depth	N/A	0.592	0.030	0.637	0.042	0.630	0.019
M4Depth+U_B	1	0.674	0.057	0.612	0.014	0.634	0.009
M4Depth+U_B	2	0.555	0.095	0.695	0.075	0.613	0.028
M4Depth+U_B	3	0.545	0.126	0.643	0.048	0.626	0.013
M4Depth+U_B	4	0.488	0.047	0.646	0.062	0.627	0.016
M4Depth+U	1	0.542	0.072	0.627	0.067	0.627	0.067
M4Depth+U	2	0.581	0.062	0.686	0.067	0.611	0.021
M4Depth+U	3	0.550	0.165	0.647	0.087	0.629	0.020
M4Depth+U	4	0.581	0.062	0.686	0.067	0.610	0.021

Table B.5: Ablation study giving the performance of networks with different numbers of layers dedicated to the depth head on the Seasons Forest Winter set of the TartanAir dataset in zero-shot cross-dataset transfer. The reported performances correspond to the ones of five networks trained on the Mid-Air dataset.

Method	# head layers	Abs Rel		RMSE log		$\delta < 1.25$	
		Mean	STD	Mean	STD	Mean	STD
M4Depth	N/A	0.637	0.039	0.598	0.049	0.677	0.019
M4Depth+U_B	1	0.930	0.174	0.619	0.041	0.672	0.017
M4Depth+U_B	2	0.575	0.055	0.669	0.093	0.660	0.034
M4Depth+U_B	3	0.789	0.110	0.618	0.028	0.672	0.028
M4Depth+U_B	4	0.589	0.091	0.632	0.065	0.669	0.017
M4Depth+U	1	0.701	0.191	0.585	0.062	0.585	0.063
M4Depth+U	2	0.664	0.094	0.636	0.092	0.665	0.026
M4Depth+U	3	0.683	0.140	0.591	0.058	0.675	0.016
M4Depth+U	4	0.664	0.094	0.636	0.092	0.665	0.026

Table B.6: Ablation study giving the performance of networks with different numbers of layers dedicated to the depth head on the Neighborhood set of the TartanAir dataset in zero-shot cross-dataset transfer. The reported performances correspond to the ones of five networks trained on the Mid-Air dataset.

Method	# head layers	Abs Rel ↓		RMSE log ↓		$\delta < 1.25$ ↓	
		Mean	STD	Mean	STD	Mean	STD
M4Depth	N/A	0.373	0.043	0.419	0.056	0.772	0.018
M4Depth+U_B	1	0.543	0.123	0.444	0.034	0.765	0.014
M4Depth+U_B	2	0.430	0.073	0.555	0.098	0.741	0.030
M4Depth+U_B	3	0.447	0.060	0.451	0.030	0.762	0.012
M4Depth+U_B	4	0.430	0.048	0.486	0.071	0.754	0.018
M4Depth+U	1	0.371	0.099	0.418	0.058	0.772	0.020
M4Depth+U	2	0.491	0.162	0.524	0.116	0.745	0.035
M4Depth+U	3	0.350	0.058	0.430	0.064	0.770	0.014
M4Depth+U	4	0.491	0.162	0.524	0.116	0.745	0.035

Table B.7: Ablation study giving the performance of networks with different numbers of layers dedicated to the depth head on the Old Town set of the TartanAir dataset in zero-shot cross-dataset transfer. The reported performances correspond to the ones of five networks trained on the Mid-Air dataset.

Method	# head layers	Abs Rel ↓		RMSE log ↓		$\delta < 1.25$ ↓	
		Mean	STD	Mean	STD	Mean	STD
M4Depth+U_B	1	0.028	0.0070	0.084	0.0199	0.009	0.0024
M4Depth+U_B	2	0.027	0.0105	0.084	0.0338	0.008	0.0011
M4Depth+U_B	3	0.046	0.0155	0.131	0.034	0.011	0.0025
M4Depth+U_B	4	0.039	0.0261	0.103	0.0543	0.011	0.0032
M4Depth+U	1	0.007	0.0006	0.020	0.0015	0.006	0.0007
M4Depth+U	2	0.007	0.0006	0.019	0.0017	0.006	0.0006
M4Depth+U	3	0.006	0.0003	0.018	0.0004	0.006	0.0003
M4Depth+U	4	0.007	0.0008	0.019	0.0009	0.006	0.0008

Table B.8: Ablation study giving the performance of networks with different numbers of layers dedicated to the uncertainty head on the test set of the Mid-Air dataset. The reported performances correspond to the ones of five networks trained on the Mid-Air dataset.

Method	# head layers	Abs Rel ↓		RMSE log ↓		$\delta < 1.25$ ↓	
		Mean	STD	Mean	STD	Mean	STD
M4Depth+U_B	1	0.028	0.0071	0.054	0.0131	0.021	0.0032
M4Depth+U_B	2	0.026	0.0029	0.050	0.0101	0.019	0.0005
M4Depth+U_B	3	0.039	0.0196	0.078	0.041	0.021	0.0017
M4Depth+U_B	4	0.029	0.0067	0.059	0.0199	0.020	0.0016
M4Depth+U	1	0.022	0.0007	0.040	0.0017	0.019	0.0004
M4Depth+U	2	0.022	0.0011	0.041	0.0029	0.018	0.0007
M4Depth+U	3	0.022	0.0010	0.041	0.0019	0.020	0.0008
M4Depth+U	4	0.022	0.0010	0.040	0.0018	0.019	0.0009

Table B.9: Ablation study giving the performance of networks with different numbers of layers dedicated to the uncertainty head on the test set of the KITTI dataset in zero-shot cross-dataset transfer. The reported performances correspond to the ones of five networks trained on the Mid-Air dataset.

Method	# head layers	Abs Rel ↓		RMSE log ↓		$\delta < 1.25$ ↓	
		Mean	STD	Mean	STD	Mean	STD
M4Depth+U_B	1	0.049	0.003	0.159	0.014	0.029	0.001
M4Depth+U_B	2	0.052	0.007	0.257	0.057	0.034	0.004
M4Depth+U_B	3	0.046	0.006	0.180	0.030	0.030	0.003
M4Depth+U_B	4	0.051	0.012	0.214	0.049	0.031	0.003
M4Depth+U	1	0.041	0.002	0.158	0.025	0.028	0.002
M4Depth+U	2	0.038	0.004	0.187	0.049	0.027	0.003
M4Depth+U	3	0.037	0.002	0.170	0.029	0.027	0.002
M4Depth+U	4	0.036	0.002	0.170	0.030	0.026	0.002

Table B.10: Ablation study giving the performance of networks with different numbers of layers dedicated to the uncertainty head on the Gascola set of the TartanAir dataset in zero-shot cross-dataset transfer. The reported performances correspond to the ones of five networks trained on the Mid-Air dataset.

Method	# head layers	Abs Rel ↓		RMSE log ↓		$\delta < 1.25$ ↓	
		Mean	STD	Mean	STD	Mean	STD
M4Depth+U_B	1	0.162	0.028	0.208	0.016	0.031	0.001
M4Depth+U_B	2	0.109	0.049	0.255	0.043	0.036	0.006
M4Depth+U_B	3	0.128	0.081	0.216	0.056	0.031	0.003
M4Depth+U_B	4	0.095	0.013	0.223	0.055	0.032	0.004
M4Depth+U	1	0.070	0.013	0.162	0.034	0.029	0.003
M4Depth+U	2	0.062	0.007	0.197	0.059	0.030	0.004
M4Depth+U	3	0.062	0.007	0.197	0.059	0.030	0.004
M4Depth+U	4	0.056	0.001	0.179	0.032	0.029	0.003

Table B.11: Ablation study giving the performance of networks with different numbers of layers dedicated to the uncertainty head on the Seasons Forest Winter set of the TartanAir dataset in zero-shot cross-dataset transfer. The reported performances correspond to the ones of five networks trained on the Mid-Air dataset.

Method	# head layers	Abs Rel ↓		RMSE log ↓		$\delta < 1.25$ ↓	
		Mean	STD	Mean	STD	Mean	STD
M4Depth+U_B	1	0.222	0.116	0.245	0.061	0.025	0.003
M4Depth+U_B	2	0.125	0.034	0.286	0.051	0.029	0.006
M4Depth+U_B	3	0.217	0.089	0.262	0.041	0.024	0.002
M4Depth+U_B	4	0.170	0.056	0.264	0.050	0.025	0.003
M4Depth+U	1	0.048	0.006	0.150	0.039	0.021	0.004
M4Depth+U	2	0.042	0.008	0.192	0.074	0.023	0.006
M4Depth+U	3	0.042	0.008	0.192	0.074	0.023	0.006
M4Depth+U	4	0.033	0.002	0.149	0.029	0.021	0.002

Table B.12: Ablation study giving the performance of networks with different numbers of layers dedicated to the uncertainty head on the Neighborhood set of the TartanAir dataset in zero-shot cross-dataset transfer. The reported performances correspond to the ones of five networks trained on the Mid-Air dataset.

Method	# head layers	Abs Rel ↓		RMSE log ↓		$\delta < 1.25$ ↓	
		Mean	STD	Mean	STD	Mean	STD
M4Depth+U_B	1	0.139	0.068	0.208	0.046	0.023	0.003
M4Depth+U_B	2	0.093	0.022	0.238	0.019	0.025	0.002
M4Depth+U_B	3	0.150	0.061	0.231	0.041	0.023	0.003
M4Depth+U_B	4	0.102	0.016	0.207	0.039	0.021	0.002
M4Depth+U	1	0.039	0.005	0.125	0.017	0.020	0.001
M4Depth+U	2	0.037	0.006	0.162	0.061	0.021	0.004
M4Depth+U	3	0.037	0.006	0.162	0.061	0.021	0.004
M4Depth+U	4	0.031	0.001	0.122	0.015	0.018	0.001

Table B.13: Ablation study giving the performance of networks with different numbers of layers dedicated to the uncertainty head on the Old Town set of the TartanAir dataset in zero-shot cross-dataset transfer. The reported performances correspond to the ones of five networks trained on the Mid-Air dataset.

Bibliography

[1] H. Aanæs, R. R. Jensen, G. Vogiatzis, E. Tola, and A. B. Dahl, “Large-Scale Data for Multiple-View Stereopsis,” *Int. J. Comput. Vis.*, vol. 120, no. 2, pp. 153–168, 2016. DOI: 10.1007/s11263-016-0902-9.

[2] I. Abaspur Kazerouni, L. Fitzgerald, G. Dooly, and D. Toal, “A survey of state-of-the-art on visual SLAM,” *Expert Syst. Appl.*, vol. 205, p. 117 734, 2022. DOI: 10.1016/j.eswa.2022.117734.

[3] S. Agarwal and H. Hablani, “Automatic Aircraft Landing over Parabolic Trajectory using Precise GPS Measurements,” *Int. Conf. & Work. Emerg. Trends Technol. (ICWET)*, vol. 1, no. 7, pp. 38–45, 2011.

[4] L. R. Agostinho, N. M. Ricardo, M. I. Pereira, A. Hiolle, and A. M. Pinto, “A Practical Survey on Visual Odometry for Autonomous Driving in Challenging Scenarios and Conditions,” *IEEE Access*, vol. 10, pp. 72 182–72 205, 2022. DOI: 10.1109/access.2022.3188990.

[5] P. Anandan, “Computing Dense Displacement Fields With Confidence Measures In Scenes Containing Occlusion,” in *Intelligent Robots and Computer Vision*, D. P. Casasent and E. L. Hall, Eds., ser. Proc. SPIE, vol. 521, SPIE, 1985. DOI: 10.1117/12.946179.

[6] M. Aqel, M. Marhaban, M. Saripan, and N. Ismail, “Review of visual odometry: types, approaches, challenges, and applications,” *SpringerPlus*, vol. 5, no. 1, pp. 1897–1923, 2016. DOI: 10.1186/s40064-016-3573-7.

[7] J. L. Barron, D. J. Fleet, and S. S. Beauchemin, “Performance of optical flow techniques,” *Int. J. Comput. Vis.*, vol. 12, no. 1, pp. 43–77, 1994. DOI: 10.1007/BF01420984.

[8] A. Bastos and H. Hasegawa, “Behavior of GPS Signal Interruption Probability under Tree Canopies in Different Forest Conditions,” *Eur. J. Remote Sens.*, vol. 46, no. 1, pp. 613–622, 2013. DOI: 10.5721/EuJRS20134636.

[9] J. Billington, *The Prometheus project: The story behind one of AV’s greatest developments*, A. A. V. International, Ed., https://www.autonomousvehicleinternational.com/features/the-prometheus-project.html, Accessed: 2023-03-28, 2018.

[10] A. Bruhn and J. Weickert, “A Confidence Measure for Variational Optic flow Methods,” in *Geometric Properties for Incomplete data*, ser. Computational Imaging and Vision. Springer, 2006, vol. 31, pp. 283–298. DOI: 10.1007/1-4020-3858-8_15.

[11] M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Rehder, S. Omari, M. W. Achtelik, and R. Siegwart, “The EuRoC micro aerial vehicle datasets,” *Int. J. Robot. Res.*, vol. 35, no. 10, pp. 1157–1163, 2016. DOI: 10.1177/0278364915620033.

[12] D. Butler, J. Wulff, G. Stanley, and M. Black, “A naturalistic open source movie for optical flow evaluation,” in *Eur. Conf. Comput. Vis. (ECCV)*, ser. Lect. Notes Comput. Sci. Vol. 7577, Springer, 2012, pp. 611–625. DOI: 10.1007/978-3-642-33783-3_44.

[13] M. Carvalho, B. Le Saux, P. Trouvé-Peloux, A. Almansa, and F. Champagnat, "On Regression Losses for Deep Depth Estimation," in *IEEE Int. Conf. Image Process. (ICIP)*, Athens, Greece: Inst. Electr. Electron. Eng. (IEEE), 2018, pp. 2915–2919. DOI: 10.1109/icip.2018.8451312.

[14] D. Cheda, "Monocular Depth Cues in Computer Vision Applications," PhD thesis, Universitat Autònoma de Barcelona, 2012.

[15] X. Chen, X. Chen, and Z.-J. Zha, "Structure-Aware Residual Pyramid Network for Monocular Depth Estimation," in *Int. Jt. Conf. Artif. Intell. (IJCAI)*, Macao, China: AAAI Press, 2019, pp. 694–700.

[16] R. Collins, "A space-sweep approach to true multi-image matching," in *IEEE Int. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, San Francisco, CA, USA, 1996, pp. 358–363. DOI: 10.1109/CVPR.1996.517097.

[17] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The Cityscapes Dataset for Semantic Urban Scene Understanding," in *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, Las Vegas, Nevada, USA, 2016, pp. 3213–3223. DOI: 10.1109/CVPR.2016.350.

[18] Y. Cui and S. S. Ge, "Autonomous vehicle positioning with GPS in urban canyon environments," *IEEE Trans. Robot. Autom.*, vol. 19, no. 1, pp. 15–25, 2003. DOI: 10.1109/tra.2002.807557.

[19] A. Dai, A. X. Chang, M. Savva, M. Halber, T. Funkhouser, and M. Nießner, "ScanNet: Richly-Annotated 3D Reconstructions of Indoor Scenes," in *IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Honolulu, Hawaii, USA: Inst. Electr. Electron. Eng. (IEEE), 2022, pp. 2432–2443. DOI: 10.1109/CVPR.2017.261.

[20] Defense Advanced Research Projects Agency (DARPA), *The Grand Challenge*, DARPA, Ed., https://www.darpa.mil/about-us/timeline/-grand-challenge-for-autonomous-vehicles, Accessed: 2023-03-28, 2005.

[21] D. Dissanayaka, T. R. Wanasinghe, O. D. Silva, A. Jayasiri, and G. K. I. Mann, "Review of Navigation Methods for UAV-Based Parcel Delivery," *IEEE Trans. Autom. Sci. Eng.*, pp. 1–15, 2023. DOI: 10.1109/tase.2022.3232025.

[22] A. Dosovitskiy, P. Fischer, E. Ilg, P. Hausser, C. Hazirbas, V. Golkov, P. v. d. Smagt, D. Cremers, and T. Brox, "FlowNet: Learning Optical Flow with Convolutional Networks," in *IEEE Int. Conf. Comput. Vis. (ICCV)*, Santiago, Chile: Inst. Electr. Electron. Eng. (IEEE), 2015, pp. 2758–2766. DOI: 10.1109/iccv.2015.316.

[23] A. Düzçeker, S. Galliani, C. Vogel, P. Speciale, M. Dusmanu, and M. Pollefeys, "DeepVideoMVS: Multi-View Stereo on Video with Recurrent Spatio-Temporal Fusion," *CoRR*, vol. abs/2012.02177, 2020. DOI: 10.48550/ARXIV.2012.02177. arXiv: 2012.02177.

[24] D. Eigen, C. Puhrsch, and R. Fergus, "Depth Map Prediction from a Single Image using a Multi-Scale Deep Network," in *Adv. Neural Inf. Process. Syst. (NeurIPS)*, 2014, pp. 2366–2374.

[25] Epic Games, *Unreal Engine web site*, https://www.unrealengine.com.

[26] S. Ercolino, A. Devoto, L. Monorchio, M. Santini, S. Mazzaro, and S. Scardapane, "On the robustness of vision transformers for in-flight monocular depth estimation," *Industrial Artificial Intelligence*, vol. 1, no. 1, pp. 1–14, 2023. DOI: 10.1007/s44244-023-00005-3.

[27] S. Farooq Bhat, I. Alhashim, and P. Wonka, "AdaBins: Depth Estimation Using Adaptive Bins," in *IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Nashville, TN, USA: Inst. Electr. Electron. Eng. (IEEE), 2021, pp. 4008–4017. DOI: 10.1109/cvpr46437.2021.00400.

[28] M. Fonder, D. Ernst, and M. Van Droogenbroeck, "Parallax Inference for Robust Temporal Monocular Depth Estimation in Unstructured Environments," *Sensors*, vol. 22, no. 23, pp. 1–22, 2022. DOI: 10.3390/s22239374.

[29] M. Fonder and M. Van Droogenbroeck, "A technique to jointly estimate depth and depth uncertainty for unmanned aerial vehicles," in *IEEE Int. Conf. Syst. Signals Image Process. (IWSSIP)*, Ohrid, North Macedonia, 2023, pp. 1–5.

[30] ——, "Mid-Air: A Multi-Modal Dataset for Extremely Low Altitude Drone Flights," in *IEEE Int. Conf. Comput. Vis. Pattern Recognit. Work. (CVPRW), UAVision*, Long Beach, CA, USA: Inst. Electr. Electron. Eng. (IEEE), 2019, pp. 553–562. DOI: 10.1109/cvprw.2019.00081.

[31] J. Fritsch, T. Kuehnl, and A. Geiger, "A New Performance Measure and Evaluation Benchmark for Road Detection Algorithms," in *IEEE Int. Conf. Intell. Transp. Syst. (ITSC)*, The Hague, Netherlands, 2013, pp. 1693–1700. DOI: 10.1109/ITSC.2013.6728473.

[32] Y. Fujimura, M. Iiyama, T. Funatomi, and Y. Mukaigawa, "Deep Depth from Focal Stack with Defocus Model for Camera-Setting Invariance," *CoRR*, vol. abs/2202.13055, 2022. DOI: 10 . 48550 / arXiv . 2202 . 13055. arXiv: 2202.13055.

[33] A. Gaidon, Q. Wang, Y. Cabon, and E. Vig, "VirtualWorlds as Proxy for Multi-object Tracking Analysis," in *IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Las Vegas, NV, USA: Inst. Electr. Electron. Eng. (IEEE), 2016, pp. 4340–4349. DOI: 10.1109/CVPR.2016.470.

[34] D. Gallup, J.-M. Frahm, P. Mordohai, Q. Yang, and M. Pollefeys, "Real-Time Plane-Sweeping Stereo with Multiple Sweeping Directions," in *IEEE Int. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Minneapolis, MN, USA, 2007, pp. 1–8. DOI: 10.1109/CVPR.2007.383245.

[35] R. Garg, V. K. B.G., G. Carneiro, and I. Reid, "Unsupervised CNN for Single View Depth Estimation: Geometry to the Rescue," in *Eur. Conf. Comput. Vis. (ECCV)*, ser. Lect. Notes Comput. Sci. Vol. 9912, Springer Int. Publ., 2016, pp. 740–756. DOI: 10.1007/978-3-319-46484-8_45.

[36] J. Gawlikowski, C. R. N. Tassi, M. Ali, *et al.*, “A Survey of Uncertainty in Deep Neural Networks,” *CoRR*, vol. abs/2107.03342, 2021. DOI: 10.48550/arXiv.2107.03342. arXiv: 2107.03342.

[37] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, “Vision meets robotics: The KITTI dataset,” *Int. J. Robot. Res.*, vol. 32, no. 11, pp. 1231–1237, 2013. DOI: 10.1177/0278364913491297.

[38] A. Geiger, P. Lenz, and R. Urtasun, “Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite,” in *IEEE Int. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Providence, RI, USA, 2012, pp. 3354–3361. DOI: 10.1109/CVPR.2012.6248074.

[39] S. Giancola, M. Valenti, and R. Sala, “A Survey on 3D Cameras: Metrological Comparison of Time-of-Flight, Structured-Light and Active Stereoscopy Technologies,” *SpringerBriefs in Computer Science*, 2018. DOI: 10.1007/978-3-319-91761-0.

[40] C. Godard, O. Mac Aodha, and G. J. Brostow, “Digging Into Self-Supervised Monocular Depth Estimation,” in *IEEE Int. Conf. Comput. Vis. (ICCV)*, Seoul, South Korea, 2019, pp. 3827–3837. DOI: 10.1109/ICCV.2019.00393.

[41] ——, *Digging into Self-Supervised Monocular Depth Prediction*, https://github.com/nianticlabs/monodepth2, 2019.

[42] ——, “Unsupervised Monocular Depth Estimation with Left-Right Consistency,” in *IEEE Int. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Honolulu, HI, USA: Inst. Electr. Electron. Eng. (IEEE), 2017, pp. 6602–6611. DOI: 10.1109/CVPR.2017.699.

[43] ——, *Unsupervised Monocular Depth Estimation with Left-Right Consistency*, https://github.com/mrharicot/monodepth, 2017.

[44] A. Graves, “Practical variational inference for neural networks,” in *Adv. Neural Inf. Process. Syst. (NeurIPS)*, Granada, Spain, 2011, pp. 2348–2356.

[45] X. Gu, Z. Fan, S. Zhu, Z. Dai, F. Tan, and P. Tan, “Cascade Cost Volume for High-Resolution Multi-View Stereo and Stereo Matching,” in *IEEE Int. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Seattle, WA, USA, 2020, pp. 2495–2504. DOI: 10.1109/CVPR42600.2020.00257.

[46] R. Haggart and J. Aitken, “Online Scene Visibility Estimation as a Complement to SLAM in UAVs,” in *Towards Autonomous Robotic Systems Conference (TAROS)*, ser. Lect. Notes Comput. Sci. Vol. 13054, Springer, 2021, pp. 365–369. DOI: 10.1007/978-3-030-89177-0_38.

[47] A. Halin, J. G. Verly, and M. Van Droogenbroeck, “Survey and synthesis of state of the art in driver monitoring,” *Sensors*, vol. 21, no. 16, pp. 1–48, 2021. DOI: 10.3390/s21165558.

[48] M. Hansard, S. Lee, O. Choi, and R. Horaud, *Time of Flight Cameras: Principles, Methods, and Applications*, ser. SpringerBriefs in Computer Science. Springer, 2012.

[49] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, second. Cambridge, UK: Cambridge University Press, 2004. DOI: 10.1017/CBO9780511811685.

[50] K. He, X. Zhang, S. Ren, and J. Sun, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification," in *IEEE Int. Conf. Comput. Vis. (ICCV)*, Santiago, Chile, 2015, pp. 1026–1034. DOI: 10.1109/ICCV.2015.123.

[51] D. Hernandez-Juarez, L. Schneider, A. Espinosa, D. Vázquez, A. López, U. Franke, M. Pollefeys, and J. Moure, "Slanted Stixels: Representing San Francisco's Steepest Streets," in *Br. Mach. Vis. Conf. (BMVC)*, London, England, UK, 2017, pp. 1–12.

[52] HEXAGON, *LiDAR Comparison Chart*, HEXAGON, Ed., https://autonomoustuff.com/lidar-chart, Accessed: 2023-03-31, 2023.

[53] C. Homeyer, O. Lange, and C. Schnörr, "Multi-view Monocular Depth and Uncertainty Prediction with Deep SfM in Dynamic Environments," in *Int. J. Pattern Recognit. Artif. Intell.*, ser. Lect. Notes Comput. Sci. Vol. 13363, Springer Int. Publ., 2022, pp. 373–385. DOI: 10.1007/978-3-031-09037-0_31.

[54] X. Hu and P. Mordohai, "A Quantitative Evaluation of Confidence Measures for Stereo Vision," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 11, pp. 2121–2133, 2012. DOI: 10.1109/tpami.2012.46.

[55] G. Huang, Y. Li, G. Pleiss, Z. Liu, J. E. Hopcroft, and K. Q. Weinberger, "Snapshot Ensembles: Train 1, get M for free," in *Int. Conf. Learn. Represent. (ICLR)*, Toulon, France, 2017, pp. 1–14.

[56] L. Huang, J. Zhang, Y. Zuo, and Q. Wu, "Pyramid-Structured Depth MAP Super-Resolution Based on Deep Dense-Residual Network," *IEEE Signal Process. Lett.*, vol. 26, no. 12, pp. 1723–1727, 2019. DOI: 10.1109/lsp.2019.2944646.

[57] P.-H. Huang, K. Matzen, J. Kopf, N. Ahuja, and J.-B. Huang, "DeepMVS: Learning Multi-view Stereopsis," in *IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Salt Lake City, UT, USA: Inst. Electr. Electron. Eng. (IEEE), 2018, pp. 2821–2830. DOI: 10.1109/cvpr.2018.00298.

[58] A. Hussen and I. Jleta, *Low-Cost Inertial Sensors Modeling Using Allan Variance*, 2015. DOI: 10.5281/zenodo.1105987.

[59] E. Ilg, Ö. Çiçek, S. Galesso, A. Klein, O. Makansi, F. Hutter, and T. Brox, "Uncertainty Estimates and Multi-hypotheses Networks for Optical Flow," in *Eur. Conf. Comput. Vis. (ECCV)*, ser. Lect. Notes Comput. Sci. Vol. 11211, Springer Int. Publ., 2018, pp. 677–693. DOI: 10.1007/978-3-030-01234-2_40.

[60] M. Irani and P. Anandan, "Parallax geometry of pairs of points for 3D scene analysis," in *Eur. Conf. Comput. Vis. (ECCV)*, ser. Lect. Notes Comput. Sci. Vol. 1064, Springer Berlin Heidelberg, 1996, pp. 17–30. DOI: 10.1007/bfb0015520.

[61] P. Ji, L. Ruan, Y. Xue, L. Xiao, and Q. Dong, "Perspective, Survey and Trends: Public Driving Datasets and Toolsets for Autonomous Driving Virtual Test," *CoRR*, vol. abs/2104.00273, 2021. DOI: 10.48550/arXiv.2104.00273. arXiv: 2104.00273.

[62] T. Ke, T. Do, K. Vuong, K. Sartipi, and S. I. Roumeliotis, “Deep Multi-view Depth Estimation with Predicted Uncertainty,” in *IEEE Int. Conf. Robot. Autom. (ICRA)*, Xian, China: Inst. Electr. Electron. Eng. (IEEE), 2021, pp. 9235–9241. DOI: 10.1109/icra48506.2021.9560873.

[63] A. Kendall and Y. Gal, “What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision?” In *Adv. Neural Inf. Process. Syst. (NeurIPS)*, Long Beach, CA, USA, 2017, pp. 5574–5584.

[64] J. Kim, S. Kim, C. Ju, and H. I. Son, “Unmanned Aerial Vehicles in Agriculture: A Review of Perspective of Platform, Control, and Applications,” *IEEE Access*, vol. 7, pp. 105 100–105 115, 2019. DOI: 10.1109/access.2019.2932119.

[65] D. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization,” in *Int. Conf. Learn. Represent. (ICLR)*, San Diego, CA, USA, 2015, pp. 1–15.

[66] M. Klodt and A. Vedaldi, “Supervising the New with the Old: Learning SFM from SFM,” in *Eur. Conf. Comput. Vis. (ECCV)*, ser. Lect. Notes Comput. Sci. Vol. 11214, Springer Int. Publ., 2018, pp. 713–728. DOI: 10.1007/978-3-030-01249-6_43.

[67] A. Knapitsch, J. Park, Q.-Y. Zhou, and V. Koltun, “Tanks and temples: benchmarking large-scale scene reconstruction,” *ACM Trans. Graph.*, vol. 36, no. 4, pp. 1–13, 2017. DOI: 10.1145/3072959.3073599.

[68] C. Kondermann, D. Kondermann, B. Jähne, and C. Garbe, “An Adaptive Confidence Measure for Optical Flows Based on Linear Subspace Projections,” in *Pattern Recognit.*, ser. Lect. Notes Comput. Sci. Vol. 4713, Springer, 2007, pp. 132–141. DOI: 10.1007/978-3-540-74936-3_14.

[69] M. Krishnakumar, *The 6 Autonomous Driving Levels Explained*, W. Biases, Ed., https://wandb.ai/mukilan/autonomous-cars/reports/The-6-Autonomous-Driving-Levels-Explained--VmlldzoyNTcwOTQ1, Accessed: 2023-03-28, 2022.

[70] A. Krizhevsky, L. Sutskever, and G. G. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” in *Adv. Neural Inf. Process. Syst. (NeurIPS)*, vol. 25, 2012, pp. 1097–1105.

[71] A. C. S. Kumar, S. M. Bhandarkar, and M. Prasad, “DepthNet: A Recurrent Neural Network Architecture for Monocular Depth Prediction,” in *IEEE Int. Conf. Comput. Vis. Pattern Recognit. Work. (CVPRW)*, Salt Lake City, UT, USA, 2018, pp. 396–404. DOI: 10.1109/CVPRW.2018.00066.

[72] J. Kybic and C. Nieuwenhuis, “Bootstrap optical flow confidence and uncertainty measure,” *Comput. Vis. Image Underst.*, vol. 115, no. 10, pp. 1449–1462, 2011. DOI: 10.1016/j.cviu.2011.06.008.

[73] G. Lachapelle, J. Henriksen, and T. Melgard, “Seasonal Effecf of Tree Foliage on GPS Signal Availability and Accuracy for Vehicular Navigation,” in *Int. Techn. Meeting of the Satellite Division of the Institute of Navigation*, 1994, pp. 527–532.

[74] C.-É. N. Laflamme, F. Pomerleau, and P. Giguère, “Driving Datasets Literature Review,” *CoRR*, vol. abs/1910.11968, 2019. DOI: 10.48550/arXiv.1910.11968. arXiv: 1910.11968.

[75] B. Lakshminarayanan, A. Pritzel, and C. Blundell, "Simple and scalable predictive uncertainty estimation using deep ensembles," in *Adv. Neural Inf. Process. Syst. (NeurIPS)*, Long Beach, CA, USA: Curran Assoc. Inc., 2017, pp. 6405–6416.

[76] C. Liu, J. Gu, K. Kim, S. G. Narasimhan, and J. Kautz, "Neural RGB-D Sensing: Depth and Uncertainty From a Video Camera," in *IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Long Beach, CA, USA: Inst. Electr. Electron. Eng. (IEEE), 2019, pp. 10 978–10 987. DOI: 10.1109/cvpr.2019.01124.

[77] F. Liu, C. Shen, and G. Lin, "Deep convolutional neural fields for depth estimation from a single image," in *IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Boston, MA, USA: Inst. Electr. Electron. Eng. (IEEE), 2015, pp. 5162–5170. DOI: 10.1109/cvpr.2015.7299152.

[78] J. Liu, X. Zhang, Z. Li, and T. Mao, "Multi-Scale Residual Pyramid Attention Network for Monocular Depth Estimation," in *IEEE Int. Conf. Pattern Recognit. (ICPR)*, Milan, Italy: Inst. Electr. Electron. Eng. (IEEE), 2021, pp. 5137–5144. DOI: 10.1109/icpr48806.2021.9412670.

[79] W. Liu, Q. Dong, P. Wang, G. Yang, L. Meng, Y. Song, Y. Shi, and Y. Xue, "A Survey on Autonomous Driving Datasets," in *8th International Conference on Dependable Systems and Their Applications (DSA)*, Yinchuan, China: Inst. Electr. Electron. Eng. (IEEE), 2021, pp. 399–407. DOI: 10.1109/dsa52907.2021.00060.

[80] C. Luo, Z. Yang, P. Wang, Y. Wang, W. Xu, R. Nevatia, and A. Yuille, "Every Pixel Counts ++: Joint Learning of Geometry and Motion with 3D Holistic Understanding," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 42, no. 10, pp. 2624–2641, 2020. DOI: 10.1109/TPAMI.2019.2930258.

[81] X. Luo, J.-B. Huang, R. Szeliski, K. Matzen, and J. Kopf, "Consistent video depth estimation," *ACM Transactions on Graphics (TOG)*, vol. 39, no. 4, 71:1–71:13, 2020. DOI: 10.1145/3386569.3392377.

[82] O. Mac Aodha, A. Humayun, M. Pollefeys, and G. J. Brostow, "Learning a Confidence Measure for Optical Flow," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, no. 5, pp. 1107–1120, 2013. DOI: 10.1109/tpami.2012.171.

[83] R. Mahjourian, M. Wicke, and A. Angelova, "Unsupervised Learning of Depth and Ego-Motion from Monocular Video Using 3D Geometric Constraints," in *IEEE Int. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Salt Lake City, UT, USA, 2018, pp. 5667–5675. DOI: 10.1109/CVPR.2018.00594.

[84] L. Masello, B. Sheehan, F. Murphy, G. Castignani, K. McDonnell, and C. Ryan, "From Traditional to Autonomous Vehicles: A Systematic Review of Data Availability," *Transportation Research Record: Journal of the Transportation Research Board*, vol. 2676, no. 4, pp. 161–193, 2021. DOI: 10.1177/03611981211057532.

[85] F. Mason, M. Capuzzo, D. Magrin, F. Chiariotti, A. Zanella, and M. Zorzi, "Remote Tracking of UAV Swarms via 3D Mobility Models and LoRaWAN Communications," *IEEE Transactions on Wireless Communications*, vol. 21, no. 5, pp. 2953–2968, 2022. DOI: 10.1109/twc.2021.3117142.

[86] M. Masoudifar and H. R. Pourreza, “Image and Depth from a Single Defocused Image Using Coded Aperture Photography,” *CoRR*, vol. abs/1603.04046, 2016. DOI: 10.48550/arXiv.1603.04046. arXiv: 1603.04046.

[87] A. Masoumian, H. A. Rashwan, J. Cristiano, M. S. Asif, and D. Puig, “Monocular Depth Estimation Using Deep Learning: A Review,” *Sensors*, vol. 22, no. 14, pp. 1–24, 2022. DOI: 10.3390/s22145353.

[88] K. Máthé and L. Buşoniu, “Vision and Control for UAVs: A Survey of General Methods and of Inexpensive Platforms for Infrastructure Inspection,” *Sensors*, vol. 15, no. 7, pp. 1–30, 2015. DOI: 10.3390/s150714887.

[89] S. Matsui, H. Nagahara, and R.-i. Taniguchi, “Half-sweep imaging for depth from defocus,” *Image Vis. Comput.*, vol. 32, no. 11, pp. 954–964, 2014. DOI: 10.1016/j.imavis.2014.09.001.

[90] N. Mayer, E. Ilg, P. Häusser, P. Fischer, D. Cremers, A. Dosovitskiy, and T. Brox, “A Large Dataset to Train Convolutional Networks for Disparity, Optical Flow, and Scene Flow Estimation,” in *IEEE Int. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Las Vegas, NV, USA, 2016, pp. 4040–4048. DOI: 10.1109/CVPR.2016.438.

[91] V. Mazzari, *How to select the right LiDAR?* L. B. G. Robots, Ed., https://www.generationrobots.com/blog/en/how-to-select-the-right-lidar/, Accessed: 2023-03-31, 2019.

[92] P. F. McManamon, *LiDAR Technologies and Systems*. SPIE Press, 2019, ISBN: 9781510625419. DOI: 10.1117/3.2518254.

[93] M. Mehltretter and C. Heipke, “Aleatoric uncertainty estimation for dense stereo matching via CNN-based cost volume analysis,” *ISPRS J. Photogramm. Remote Sens.*, vol. 171, pp. 63–75, 2021. DOI: 10.1016/j.isprsjprs.2020.11.003.

[94] M. Menze and A. Geiger, “Object Scene Flow for Autonomous Vehicles,” in *IEEE Int. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Boston, MA, USA, 2015, pp. 3061–3070. DOI: 10.1109/CVPR.2015.7298925.

[95] V.-C. Miclea and S. Nedevschi, “Monocular Depth Estimation With Improved Long-Range Accuracy for UAV Environment Perception,” *IEEE Trans. Geosci. Remote Sens.*, vol. 60, pp. 1–15, 2022. DOI: 10.1109/tgrs.2021.3060513.

[96] ——, “Temporal Attention for Monocular Depth Estimation in Aerial Scenarios,” in *International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME)*, Maldives, Maldives: Inst. Electr. Electron. Eng. (IEEE), 2022, pp. 1–6. DOI: 10.1109/iceccme55909.2022.9988383.

[97] Y. Ming, X. Meng, C. Fan, and H. Yu, “Deep learning for monocular depth estimation: A review,” *Neurocomputing*, vol. 438, pp. 14–33, 2021. DOI: 10.1016/j.neucom.2020.12.089.

[98] N. Mishima, T. Kozakaya, A. Moriya, R. Okada, and S. Hiura, “Physical Cue based Depth-Sensing by Color Coding with Deaberration Network,” *CoRR*, vol. abs/1908.00329, 2019. DOI: 10.48550/arXiv.1908.00329. arXiv: 1908.00329.

[99] F. Mumuni, A. Mumuni, and C. K. Amuzuvi, “Deep learning of monocular depth, optical flow and ego-motion with geometric guidance for UAV navigation in dynamic environments,” *Mach. Learn. Appl.*, vol. 10, pp. 2–15, 2022. DOI: 10.1016/j.mlwa.2022.100416.

[100] C. Naranjo, *Analysis and Modeling of MEMS based Inertial Sensors*, 2008.

[101] K. Nirmal, A. Sreejith, J. Mathew, M. Sarpotdar, A. Suresh, A. Prakash, M. Safonova, and J. Murthy, “Noise modeling and analysis of an IMU-based attitude sensor: improvement of performance by filtering and sensor fusion,” in *Proceedings of the SPIE*, vol. 9912, 2016, pp. 1–10. DOI: 10.1117/12.2234255.

[102] NVidia, *NVidia Deep Learning Inference Plateform Performance Study*, Technical Overview, 2018.

[103] C. A. Ochotorena, C. N. Ochotorena, and E. Dadios, “Learning filtered color planes for depth-extraction in coded aperture images,” in *International Conference on Humanoid, Nanotechnology, Information Technology,Communication and Control, Environment and Management (HNICEM)*, Cebu, Philippines: Inst. Electr. Electron. Eng. (IEEE), 2015, pp. 1–5. DOI: 10.1109/hnicem.2015.7393264.

[104] D. Olid, J. M. Facil, and J. Civera, “Single-View Place Recognition under Seasonal Changes,” *CoRR*, vol. abs/1808.06516, 2018. arXiv: 1808.06516.

[105] O. Özyeşil, V. Voroninski, R. Basri, and A. Singer, “A survey of structure from motion,” *Acta Numerica*, vol. 26, pp. 305–364, 2017. DOI: 10.1017/S096249291700006X.

[106] V. Paramonov, I. Panchenko, V. Bucha, A. Drogolyub, and S. Zagoruyko, “Depth Camera Based on Color-Coded Aperture,” in *IEEE/CVF Conf. Comput. Vis. Pattern Recognit. Work. (CVPRW)*, Las Vegas, NV, USA: Inst. Electr. Electron. Eng. (IEEE), 2016, pp. 910–918. DOI: 10.1109/cvprw.2016.118.

[107] V. Patil, W. Van Gansbeke, D. Dai, and L. Van Gool, “Don’t Forget The Past: Recurrent Depth Estimation from Monocular Video,” *IEEE Robot. Autom. Lett.*, vol. 5, no. 4, pp. 6813–6820, 2020. DOI: 10.1109/LRA.2020.3017478.

[108] M. Poggi, F. Aleotti, F. Tosi, and S. Mattoccia, “On the Uncertainty of Self-Supervised Monocular Depth Estimation,” in *IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Seattle, WA, USA: Inst. Electr. Electron. Eng. (IEEE), 2020, pp. 3224–3234. DOI: 10.1109/cvpr42600.2020.00329.

[109] M. Poggi, F. Aleotti, F. Tosi, G. Zaccaroni, and S. Mattoccia, “Self-adapting Confidence Estimation for Stereo,” in *Eur. Conf. Comput. Vis. (ECCV)*, ser. Lect. Notes Comput. Sci. Vol. 12369, Springer Int. Publ., 2020, pp. 715–733. DOI: 10.1007/978-3-030-58586-0_42.

[110] M. Poggi, S. Kim, F. Tosi, S. Kim, F. Aleotti, D. Min, K. Sohn, and S. Mattoccia, “On the confidence of stereo matching in a deep-learning era: a quantitative evaluation,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 9, pp. 5293–5313, 2021. DOI: 10.1109/tpami.2021.3069706.

[111] D. A. Pomerleau, “ALVINN, an autonomous land vehicle in a neural network,” Carnegie Mellon University, 1990. DOI: 10.1184/R1/6603146.V1.

[112] L. Pupo, *Characterization of Errors and Noises in MEMS Inertial Sensors Using Allan Variance Method*, 2016.

[113] R. Ranftl, A. Bochkovskiy, and V. Koltun, “Vision Transformers for Dense Prediction,” in *IEEE Int. Conf. Comput. Vis. (ICCV)*, Montréal, Can.: Inst. Electr. Electron. Eng. (IEEE), 2021, pp. 12 159–12 168. DOI: 10.1109/iccv48922.2021.01196.

[114] R. Ranftl, K. Lasinger, D. Hafner, K. Schindler, and V. Koltun, “Towards Robust Monocular Depth Estimation: Mixing Datasets for Zero-Shot Cross-Dataset Transfer,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 3, pp. 1623–1637, 2022. DOI: 10.1109/tpami.2020.3019967.

[115] O. Ronneberger, P. Fischer, and T. Brox, “U-Net: Convolutional Networks for Biomedical Image Segmentation,” in *Medical Image Comput. Comput. Assist. Interv. (MICCAI)*, ser. Lect. Notes Comput. Sci. Vol. 9351, Springer, 2015, pp. 234–241. DOI: 10.1007/978-3-319-24574-4_28.

[116] G. Ros, L. Sellart, J. Materzynska, D. Vazquez, and A. M. Lopez, “The SYNTHIA Dataset: A Large Collection of Synthetic Images for Semantic Segmentation of Urban Scenes,” in *IEEE Int. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Las Vegas, NV, USA: Inst. Electr. Electron. Eng. (IEEE), 2016, pp. 3234–3243. DOI: 10.1109/cvpr.2016.352.

[117] H. S. Sawhney, “3D geometry from planar parallax,” in *IEEE Int. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Seattle, WA, USA: IEEE Comput. Soc. Press, 1994, pp. 929–934. DOI: 10.1109/cvpr.1994.323927.

[118] A. Saxena, S. Chung, and A. Ng, “Learning Depth from Single Monocular Images,” in *Adv. Neural Inf. Process. Syst. (NeurIPS)*, vol. 18, MIT Press, 2005, pp. 1161–1168.

[119] D. Scharstein, R. Szeliski, and R. Zabih, “A taxonomy and evaluation of dense two-frame stereo correspondence algorithm,” in *IEEE Workshop on Stereo and Multi-Baseline Vision*, Kauai, HI, USA, 2001, pp. 131–140. DOI: 10.1109/SMBV.2001.988771.

[120] P. Schröppel, J. Bechtold, A. Amiranashvili, and T. Brox, “A Benchmark and a Baseline for Robust Multi-view Depth Estimation,” *CoRR*, vol. abs/2209.06681, 2022. DOI: 10.48550/arXiv.2209.06681. arXiv: 2209.06681.

[121] S. Shah, D. Dey, C. Lovett, and A. Kapoor, “AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles,” in *Fields and Service Robotics*, ser. Proceedings in Advanced Robotics, vol. 5, 2017, pp. 621–635, ISBN: 978-3-319-67360-8. DOI: 10.1007/978-3-319-67361-5_40.

[122] M. Shan, Y. Bi, H. Qin, J. Li, Z. Gao, F. Lin, and B. M. Chen, “A brief survey of visual odometry for micro aerial vehicles,” in *IECON - 42nd Annual Conference of the IEEE Industrial Electronics Society*, Florence, Italy: Inst. Electr. Electron. Eng. (IEEE), 2016, pp. 6049–6054. DOI: 10.1109/iecon.2016.7793198.

[123] N. Silberman and R. Fergus, “Indoor scene segmentation using a structured light sensor,” in *IEEE Int. Conf. Comput. Vis. Work. (ICCV Work.)*, Barcelona, Spain, 2011, pp. 601–608. DOI: 10.1109/ICCVW.2011.6130298.

[124] N. Silberman, D. Hoiem, P. Kohli, and R. Fergus, “Indoor Segmentation and Support Inference from RGBD Images,” in *Eur. Conf. Comput. Vis. (ECCV)*, ser. Lect. Notes Comput. Sci. Vol. 7576, Firenze, Italy, 2012, pp. 746–760, ISBN: 978-3-642-33714-7. DOI: 10.1007/978-3-642-33715-4_54.

[125] S.-Y. Song, S.-W. Park, H.-G. Kim, and S.-H. Choi, “Deep Learning based Visual-Inertial Drone Odomtery Estimation,” *Proceedings of the Korea Information Processing Society Conference*, pp. 842–845, 2020. DOI: 10.3745/PKIPS.y2020m11a.842.

[126] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A Simple Way to Prevent Neural Networks from Overfitting,” *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014.

[127] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, “A Benchmark for the Evaluation of RGB-D SLAM Systems,” in *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*, Vilamoura, Portugal, 2012, pp. 573–580. DOI: 10.1109/IROS.2012.6385773.

[128] W. Su, Q. Xu, and W. Tao, “Uncertainty Guided Multi-View Stereo Network for Depth Estimation,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 32, no. 11, pp. 7796–7808, 2022. DOI: 10.1109/tcsvt.2022.3183836.

[129] D. Sun, X. Yang, M. Liu, and J. Kautz, “PWC-Net: CNNs for Optical Flow Using Pyramid, Warping, and Cost Volume,” in *IEEE Int. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Salt Lake City, UT, USA, 2018, pp. 8934–8943. DOI: 10.1109/CVPR.2018.00931.

[130] T. Taketomi, H. Uchiyama, and S. Ikeda, “Visual SLAM algorithms: a survey from 2010 to 2016,” *IPSJ Trans. Comput. Vis. Appl.*, vol. 9, no. 1, 2017. DOI: 10.1186/s41074-017-0027-2.

[131] Z. Teed and J. Deng, “DeepV2D: Video to Depth with Differentiable Structure from Motion,” *CoRR*, vol. abs/1812.04605, 2018. DOI: 10.48550/ARXIV.1812.04605. arXiv: 1812.04605.

[132] P. Trouvé-Peloux, F. Champagnat, G. Le Besnerais, G. Druart, and J. Idier, “Performance model of depth from defocus with an unconventional camera,” *J. Opt. Soc. Am. A*, vol. 38, no. 10, p. 1489, 2021. DOI: 10.1364/josaa.424621.

[133] B. Ummenhofer, H. Zhou, J. Uhrig, N. Mayer, E. Ilg, A. Dosovitskiy, and T. Brox, “DeMoN: Depth and Motion Network for Learning Monocular Stereo,” in *IEEE Int. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Honolulu, HI, USA, 2017, pp. 5622–5631. DOI: 10.1109/CVPR.2017.596.

[134] D. Wang, W. Li, X. Liu, N. Li, and C. Zhang, “UAV environmental perception and autonomous obstacle avoidance: A deep learning and depth camera combined solution,” *Comput. Electron. Agric.*, vol. 175, pp. 1–11, 2020. DOI: 10.1016/j.compag.2020.105523.

[135] R. Wang, S. M. Pizer, and J.-M. Frahm, “Recurrent Neural Network for (Un-)Supervised Learning of Monocular Video Visual Odometry and Depth,” in *IEEE Int. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Long Beach, CA, USA, 2019, pp. 5550–5559. DOI: 10.1109/CVPR.2019.00570.

[136] ——, *Recurrent Neural Network for (Un-)supervised Learning of Monocular VideoVisual Odometry and Depth*, https://github.com/wrlife/RNN_depth_pose, 2019.

[137] W. Wang, D. Zhu, X. Wang, Y. Hu, Y. Qiu, C. Wang, Y. Hu, A. Kapoor, and S. Scherer, “TartanAir: A Dataset to Push the Limits of Visual SLAM,” in *IEEE/RSJ Int. Conf. Intell. Robot. Syst. (IROS)*, Las Vegas, NV, USA: Inst. Electr. Electron. Eng. (IEEE), 2020, pp. 4909–4916. DOI: 10.1109/iros45743.2020.9341801.

[138] A. S. Wannenwetsch, M. Keuper, and S. Roth, “ProbFlow: Joint Optical Flow and Uncertainty Estimation,” in *IEEE Int. Conf. Comput. Vis. (ICCV)*, Venice, Italy: Inst. Electr. Electron. Eng. (IEEE), 2017, pp. 1182–1191. DOI: 10.1109/iccv.2017.133.

[139] J. Watson, O. Mac Aodha, V. Prisacariu, G. Brostow, and M. Firman, “The Temporal Opportunist: Self-Supervised Multi-Frame Monocular Depth,” in *IEEE Int. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Nashville, TN, USA: Inst. Electr. Electron. Eng. (IEEE), 2021, pp. 1164–1174. DOI: 10.1109/cvpr46437.2021.00122.

[140] ——, *The Temporal Opportunist: Self-Supervised Multi-Frame Monocular Depth*, https://github.com/nianticlabs/manydepth, 2021.

[141] X. Weihao, *Exploiting Temporal Consistency for Real-Time Video Depth Estimation - Unofficial Implementation*, https://github.com/weihaox/ST-CLSTM, 2020.

[142] O. Woodman, “An introduction to inertial navigation,” University of Cambridge, Tech. Rep. 696, 2007.

[143] Z. Wu, X. Wu, X. Zhang, S. Wang, and L. Ju, “Spatial Correspondence With Generative Adversarial Network: Learning Depth From Monocular Videos,” in *IEEE Int. Conf. Comput. Vis. (ICCV)*, Seoul, South Korea, 2019, pp. 7494–7504. DOI: 10.1109/ICCV.2019.00759.

[144] J. Xiao, A. Owens, and A. Torralba, “SUN3D: A Database of Big Spaces Reconstructed Using SfM and Object Labels,” in *IEEE Int. Conf. Comput. Vis. (ICCV)*, Sydney, NSW, Australia: Inst. Electr. Electron. Eng. (IEEE), 2013, pp. 1625–1632. DOI: 10.1109/iccv.2013.458.

[145] R. Xiaogang, Y. Wenjing, H. Jing, G. Peiyuan, and G. Wei, “Monocular Depth Estimation Based on Deep Learning: A Survey,” in *Chin. Autom. Congr. (CAC)*, Shanghai, China, 2020, pp. 2436–2440. DOI: 10.1109/CAC51589.2020.9327548.

[146] H. Xing, Y. Cao, M. Biber, M. Zhou, and D. Burschka, “Joint prediction of monocular depth and structure using planar and parallax geometry,” *Pattern Recognit.*, vol. 130, p. 108 806, 2022. DOI: 10.1016/j.patcog.2022.108806.

[147] B. Xu, N. Wang, T. Chen, and M. Li, “Empirical Evaluation of Rectified Activations in Convolutional Network,” *CoRR*, vol. abs/1505.00853, 2015. DOI: 10.48550/ARXIV.1505.00853. arXiv: 1505.00853.

[148] J. Xu, R. Ranftl, and V. Koltun, “Accurate Optical Flow via Direct Cost Volume Processing,” in *IEEE Int. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Honolulu, HI, USA: Inst. Electr. Electron. Eng. (IEEE), 2017, pp. 5807–5815. DOI: 10.1109/cvpr.2017.615.

[149] F. Yang, X. Huang, and Z. Zhou, “Deep Depth from Focus with Differential Focus Volume,” in *IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, New Orleans, LA, USA: Inst. Electr. Electron. Eng. (IEEE), 2022, pp. 12 632–12 641. DOI: 10.1109/cvpr52688.2022.01231.

[150] N. Yang, L. von Stumberg, R. Wang, and D. Cremers, “D3VO: Deep Depth, Deep Pose and Deep Uncertainty for Monocular Visual Odometry,” in *IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Seattle, WA, USA: Inst. Electr. Electron. Eng. (IEEE), 2020, pp. 1278–1289. DOI: 10.1109/cvpr42600.2020.00136.

[151] X. Yang, J. Chen, Y. Dang, H. Luo, Y. Tang, C. Liao, P. Chen, and K.-T. Cheng, “Fast Depth Prediction and Obstacle Avoidance on a Monocular Drone Using Probabilistic Convolutional Neural Network,” *IEEE Trans. Intell. Transp. Syst.*, vol. 22, no. 1, pp. 156–167, 2021. DOI: 10.1109/tits.2019.2955598.

[152] X. Yang, H. Luo, Y. Wu, Y. Gao, C. Liao, and K.-T. Cheng, “Reactive obstacle avoidance of monocular quadrotors with online adapted depth prediction network,” *Neurocomputing*, vol. 325, pp. 142–158, 2019. DOI: 10.1016/j.neucom.2018.10.019.

[153] Y. Yao, Z. Luo, S. Li, T. Fang, and L. Quan, “MVSNet: Depth Inference for Unstructured Multi-view Stereo,” in *Eur. Conf. Comput. Vis. (ECCV)*, ser. Lect. Notes Comput. Sci. Vol. 11212, Springer, 2018, pp. 785–801. DOI: 10.1007/978-3-030-01237-3_47.

[154] Z. Yu and S. Gao, “Fast-MVSNet: Sparse-to-Dense Multi-View Stereo With Learned Propagation and Gauss-Newton Refinement,” in *IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Seattle, WA, USA: Inst. Electr. Electron. Eng. (IEEE), 2020, pp. 1946–1955. DOI: 10.1109/cvpr42600.2020.00202.

[155] H. Zhan, R. Garg, C. S. Weerasekera, K. Li, H. Agarwal, and I. M. Reid, “Unsupervised Learning of Monocular Depth Estimation and Visual Odometry with Deep Feature Reconstruction,” in *IEEE Int. Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Salt Lake City, UT, USA, 2018, pp. 340–349. DOI: 10.1109/CVPR.2018.00043.

[156] F. Zhang, X. Qi, R. Yang, V. Prisacariu, B. Wah, and P. Torr, “Domain-Invariant Stereo Matching Networks,” in *Eur. Conf. Comput. Vis. (ECCV)*, ser. Lect. Notes Comput. Sci. Vol. 12347, Springer, 2020, pp. 420–439. DOI: 10.1007/978-3-030-58536-5_25.

[157] H. Zhang, Y. Li, Y. Cao, Y. Liu, C. Shen, and Y. Yan, “Exploiting Temporal Consistency for Real-Time Video Depth Estimation,” in *IEEE Int. Conf. Comput. Vis. (ICCV)*, Seoul, South Korea, 2019, pp. 1725–1734. DOI: 10.1109/ICCV.2019.00181.

[158] J. Zhang, S. Li, Z. Luo, T. Fang, and Y. Yao, “Vis-MVSNet: Visibility-Aware Multi-view Stereo Network,” *Int. J. Comput. Vis.*, vol. 131, no. 1, pp. 199–214, 2022. DOI: 10.1007/s11263-022-01697-3.

[159] Y. Zhang, Y. Chen, X. Bai, S. Yu, K. Yu, Z. Li, and K. Yang, "Adaptive Unimodal Cost Volume Filtering for Deep Stereo Matching," in *AAAI Conf. Artif. Intell.*, vol. 34, Association for the Advancement of Artificial Intelligence (AAAI), 2020, pp. 12 926–12 934. DOI: 10.1609/aaai.v34i07.6991.

[160] C. Zhao, Q. Sun, C. Zhang, Y. Tang, and F. Qian, "Monocular depth estimation based on deep learning: An overview," *Science China Technological Sciences*, vol. 63, no. 9, pp. 1612–1627, 2020. DOI: 10.1007/s11431-020-1582-8.

[161] W. Zhao, S. Liu, Y. Wei, H. Guo, and Y.-J. Liu, "A Confidence-based Iterative Solver of Depths and Surface Normals for Deep Multi-view Stereo," in *IEEE Int. Conf. Comput. Vis. (ICCV)*, Montreal, QC, Canada: Inst. Electr. Electron. Eng. (IEEE), 2021, pp. 6148–6157. DOI: 10.1109/iccv48922.2021.00611.